Unified Web Application Development

A Declarative Approach to Empower End-User Developers

Markku Laine
Unified Web Application Development

A Declarative Approach to Empower End-User Developers

Markku Laine

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall T2 of the school on 26 January 2018 at 12 noon.

Aalto University
School of Science
Department of Computer Science
Abstract

Unified web application development is an umbrella term that encompasses isomorphic, multi-tier, and tierless programming. The common theme of all these efforts is to reduce the complexity of conventional web application development by expanding the concepts and technologies commonly used on a single tier to cover all tiers of a web application. Consequently, a tier-specific expert (professional programmer) can autonomously write an entire multi-tier web application (as a single artifact) in a programming language that s/he is most comfortable with. However, professional programmers are not the only ones building web applications; numerous web designers and end-users also actively engage in web development. These so-called end-user developers are non-professional programmers, whose motivations, needs, and web development skills greatly differ from that of professional programmers.

The aim of this dissertation was to study how to unify web application development and, at the same time, leverage end-user developers’ existing skills in declarative languages, namely (X)HTML and CSS. The study focused on approaches that allow creating and editing today’s modern web applications primarily textually (rather than visually). To accomplish this research aim, a comprehensive survey and comparative analysis of approaches to unified web application development was conducted; a complementary review of approaches to end-user development (EUD) of web applications was carried out; and a novel declarative approach called XFormsDB to empower intermediate level end-user developers in building web applications was designed, implemented, and evaluated. The research followed an iterative, six-step design science research methodology.

As a result of this first-of-its-kind survey, 16 relevant approaches to unified web application development published between 2006 and 2016 were identified and classified into three categories: presentation-centric, logic-centric, and data-centric approaches. The approaches were reviewed and analyzed in detail with respect to a set of evaluation criteria. The comparative analysis revealed a number of new insights, current trends, and research gaps, including a lack of approaches suitable for end-user developers. The complementary review, on the other hand, showed that current EUD approaches lack support for complete web application development. Additionally, a nine-level classification of web-related EUD activities was developed.

The results with XFormsDB demonstrated that the familiar syntax of (X)HTML and CSS can be seamlessly extended far beyond the current state of practice. The proposed extensions covered all common aspects of a web application, including reactive user interfaces, real-time communication, server-side application logic, data access and management, and multi-tier components. The user study, in turn, showed that XFormsDB IDE (XIDE) can help intermediate end-user developers to expand their web application development possibilities.

Keywords web application development, declarative languages, end-user development
Tekijä
Markku Laine

Väittöskirjan nimi
Yhtenäinen web-sovelluskehitys: Deklaratiivinen lähestymistapa käyttäjäkehittäjälle

Julkaisija
Perustieteiden korkeakoulu

Yksikkö
Tietotekniikan laitos

Sarja
Aalto University publication series DOCTORAL DISSERTATIONS 6/2018

Tutkimusala
Mediatekniikka

Käsikirjoituksen pvm
10.10.2017

Väitöspäivä
26.01.2018

Julkaisuluvun myöntämispäivä
05.12.2017

Kieli
Englanti

Monografia
Artikkeliväittöskirja
Esseeväittöskirja

Tiivistelmä


Avainsanat
web-sovelluskehitys, deklaratiiviset kielet, käyttäjäkehitys

ISBN (painettu)
978-952-60-7796-3

ISBN (pdf)
978-952-60-7797-0

ISSN-L
1799-4934

ISSN (painettu)
1799-4934

ISSN (pdf)
1799-4942

Julkaisupaikka
Helsinki

Painopaikka
Helsinki

Vuosi
2018

Sivumäärä
182

urn
TO MY PARENTS
for raising me to believe that
anything is possible

&

TO MY FAMILY
for giving me love and support that
makes everything possible
## Contents

Preface xi  
List of Publications xiii
Author’s Contribution xv
List of Abbreviations xvii

1 Introduction 1  
1.1 Motivation ................................................................. 1  
1.2 Research Problem and Questions .............................. 3  
1.3 Positioning the Research .......................................... 5  
1.4 Research Approach .................................................. 6  
1.5 Research Contributions ............................................ 7  
1.6 Structure of the Dissertation .................................... 8

2 Background 11  
2.1 What is Unified Web Application Development? .......... 11  
2.2 Declarative versus Imperative Programming ......... 12  
2.3 End-User Development ............................................ 14

3 A Survey on Unified Web Application Development 19  
3.1 Presentation-Centric Approaches .............................. 19  
3.2 Logic-Centric Approaches .......................................... 24  
3.3 Data-Centric Approaches .......................................... 33  
3.4 Comparative Analysis ............................................. 36  
3.5 Discussion .............................................................. 40  
3.6 Summary ............................................................... 41

4 A Declarative Approach to Empower End-User Developers 43  
4.1 The XFormsDB Approach in a Nutshell .................. 43  
4.2 The XFormsDB Language .......................................... 47  
4.3 System Implementations ........................................... 51  
4.4 Evaluation ............................................................. 55  
4.5 Related Work ......................................................... 60  
4.6 Summary .............................................................. 64
5 Conclusions 65
  5.1 Answers to the Research Questions..............................................65
  5.2 Future Work..................................................................................68

References 69

Publications 89
Preface

Pursuing a doctoral degree is like setting off on an adventure into uncharted territory. You are not quite sure what lies ahead of you, where the journey will lead you, and what discoveries you will make along the way. Indeed, investigating the unknown is one of the things that makes research so fascinating and engaging! My personal journey into the world of web engineering and end-user development research has been truly exciting and rewarding, but at times, also challenging and tiring. Without the support and encouragement of the following people, this dissertation would not have seen the light of day!

First and foremost, I would like to express my deepest gratitude to my dissertation supervisor Professor Petri Vuorimaa. He gave me the initial opportunity to join his research group at Aalto University, and has kindly and patiently guided me through the entire dissertation process. Professor Petri Vuorimaa is also largely thanked for the success of my research, which culminated with a publication in the prestigious journal of World Wide Web.

My sincere gratitude is also due to all my other co-authors and collaborators for their valuable contribution to the original publications. The contribution of Evgenia Litvinova is especially acknowledged. Without her hard work and expertise in end-user development, this dissertation would not be as complete as it is now. Dr. Denis Shestakov I wish to thank for his mentoring and drive that really pushed me and my research forward especially in the early stages of my doctoral studies. Dr. Jari Kleimola is primarily thanked for his creativity and ability to think outside the box. Each and every discussion with him was a personal learning experience for me. A special thank you goes to Dr. Mikko Honkala and Oskari Koskimies at Nokia Technologies (formerly Nokia Research Center) for introducing me to the research topic of this dissertation and co-authoring my first scientific paper. I also wish to express my warm and sincere thanks to
Professor Emilia Mendes for hosting me during my three-month research visit at the University of Auckland, New Zealand.

I would like to extend my gratitude to the preliminary examiners, Professor Jean Vanderdonckt from the Université catholique de Louvain and Professor Kari Systä from Tampere University of Technology, for their careful and thoughtful reading of the manuscript and providing me with insightful comments and encouraging feedback.

I further wish to thank my fellow doctoral students in the Web Services research group, Dr. Hannu Järvinen, Evgenia Litvinova, and Pia Tukkinen, for sharing not only the room and their passion for research, but also the joys and sorrows of work, table hockey, and life in general with me for several years. Your support and friendship mean a lot to me and I wish nothing but the best in your future endeavors.

Most importantly, I want to thank my mother Hilkka, my late father Veli-Pekka, and my brother Jussi-Pekka from the bottom of my heart for their unconditional love and support, and everything they have done for me. I am also indebted to my dear friends, who have provided welcome distractions and helped me keep my sanity during the writing of this dissertation. Finally, my utmost gratitude goes to my love Jenni and our wonderful children Sonja and Julia. Words simply cannot express how fortunate I am to have you in my life. Thank you, you mean the world to me!

Espoo, December 6, 2017

Markku Laine
List of Publications

This dissertation consists of an overview and of the following publications which are referred to in the text by their Roman numerals.


Author’s Contribution

Publication I: “Toward Unified Web Application Development”

Publication I provides an initial survey and analysis of the existing research literature on unified web application development.

The present author conducted the literature review in collaboration with Dr. Denis Shestakov, evaluated the approaches together with Evgenia Litvinova and Prof. Petri Vuorimaa, and took the lead in writing the article. The present author also revised and significantly extended the survey for this dissertation.

Publication II: “XFormsDB: An Extensible Web Application Framework Built upon Declarative W3C Standards”

Publication II introduces a declarative approach (XFormsDB) to unified web application development. The article is an extension of our prior work on extending XForms with server-side functionality (Laine et al., 2012) and connecting XForms to databases (Honkala et al., 2007).

The present author designed, implemented, and evaluated the entire work. He also wrote the article, excluding Sections 2 and 9, which were written jointly with Dr. Denis Shestakov and Prof. Petri Vuorimaa.

Publication III: “Leveraging Declarative Languages in Web Application Development”

Publication III describes how declarative languages can empower end-user developers in web application development. The article builds upon and extends the work that was carried out in Publications I, II, and V.

The present author designed and implemented the XFormsRTC language extension for adding real-time communication capabilities to XForms. In
addition, he participated in extending the classification of end-user web application development activities and developed the blog web application and its components for the XIDE demonstration. The present author wrote Section 6.2 and also contributed significantly to the editing of all other sections of the article.

**Publication IV: “UniQue: An Approach for Unified and Efficient Querying of Heterogeneous Web Data Sources”**

Publication IV proposes a unified querying (UniQue) approach that provides a uniform and declarative query interface for heterogeneous web data sources.

The present author conceived the idea, designed the query language (UQL) with the help of Dr. Jari Kleimola, and implemented the entire system himself. In addition, he conducted the evaluation, including the development of the web mashup application. The present author wrote approximately 95% of the paper.

**Publication V: “XIDE: Expanding End-User Web Development”**

Publication V presents XIDE, an end-user web application development tool that helps end-user developers to expand their web application development possibilities.

The present author was mainly responsible for designing the Template Language (TL) syntax for components and the architecture for the backend system. In addition, he developed the web application and its components, which were used as part of the evaluation. The present author wrote Section 4.2 of the paper.
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CORS</td>
<td>Cross-Origin Resource Sharing</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain-Specific Language</td>
</tr>
<tr>
<td>EUD</td>
<td>End-User Development</td>
</tr>
<tr>
<td>GPL</td>
<td>General-Purpose Language</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>MVVM</td>
<td>Model-View-ViewModel</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-Time Communication</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WYSIWYG</td>
<td>What You See Is What You Get</td>
</tr>
<tr>
<td>XHTML</td>
<td>Extensible HyperText Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XMPP</td>
<td>Extensible Messaging and Presence Protocol</td>
</tr>
<tr>
<td>XRX</td>
<td>XForms/REST/XQuery</td>
</tr>
</tbody>
</table>
1 Introduction

Unified web application development refers to architectural approaches that enable the development of multi-tier web applications in a single programming language, and potentially as a single artifact. This dissertation provides a comprehensive survey of the research and recent developments on unified web application development, and presents a novel declarative approach specifically targeted at end-user developers.

1.1 Motivation

Over the past 25 years, the web has undergone a number of evolutionary phases. What was originally designed as a relatively simple system for sharing mostly static documents among researchers has now become the dominant target platform for a wide variety of services, including desktop-style applications (Taivalsaari et al., 2008; Taivalsaari and Mikkonen, 2011). Indeed, many existing end-user software systems (e.g., word processors, spreadsheets, presentation programs, and email clients) have been and will continue to be migrated to the web because of the many benefits it offers (Taivalsaari et al., 2011). A recent prediction even goes as far as saying that "any application that can be written as a web app, is likely to eventually be written as a web app" (Mesbah, 2016).

This largely unplanned evolution of the web has also had a direct impact on the complexity of writing web applications (Mikkonen and Taivalsaari, 2008). Consider, for example, the three-tier architecture (Alonso et al., 2004; Schulte, 2009) used in conventional web application development. In such architecture, the functionality of a highly interactive data-driven web application (hereinafter simply referred to as "web application") is split into the presentation, business logic, and data access layers (cf. Figure 1) that run on client, server, and data tiers, respectively. Developing and
maintaining such distributed web applications can be challenging – even for experienced full-stack web developers – and costly for a number of reasons (Rode, Rosson, et al., 2005; Yang et al., 2006; Mikkonen and Taivalsaari, 2008; Fu et al., 2011; Sinha et al., 2015). First, web developers must learn and master a myriad of technologies, programming languages and paradigms, data models, frameworks, and tools to write even a trivial web application. Second, this multiplicity of different programming languages and their respective data models gives rise to the impedance mismatch problem (Copeland and Maier, 1984); to overcome this, extra glue code needs to be written for translating data between the different tiers. Third, the reliance on imperative programming forces web developers to manually deal with tedious low-level implementation details related, but not limited, to DOM (van Kesteren et al., 2015) manipulation, cross-tier communication, data management, and security. Last but not least, as a consequence of all this complexity, multi-tier web applications are typically built by a team of tier-specific experts (professional programmers).

On the other hand, web designers and some end-users also engage in web development (Rode et al., 2006). These so-called end-user developers who are non-professional programmers autonomously create, modify, and extend web-based software systems to support their professional or leisure activities (Lieberman et al., 2006). Examples of these activities include customizing personal blogs, creating online surveys, mashing up open web data.
data, prototyping user interfaces, and writing utility applications. However, only a small fraction of end-user developers possess enough technical skills and motivation to make use of the conventional web development model described above (Lieberman et al., 2006). Thus, there is an obvious need to support end-user developers with approaches, languages, and tools whose level of complexity is more appropriate to their skills and situations.

Over the years, a number of end-user development approaches have been developed that aim to lower the barriers to web development (Ko et al., 2011; Paternò, 2013). While many of these approaches empower end-users as web developers, they often have their limitations, such as high threshold or low ceiling (Myers et al., 2000). Consider visual tools for example. Such tools generally allow end-user developers such as domain experts to build certain types of web applications (e.g., surveys or mashups) from start to finish using wizards, components, and visual manipulations. However, when the desired functionality cannot be achieved using visual programming techniques, end-user developers are faced with a "wall". To make further progress, they must either switch to a more expressive tool (assuming source code editing is not possible) or manually edit the underlying source code implemented in unfamiliar and complex technologies targeted at professional programmers (cf. Figure 1). Similarly, many end-user developers (e.g., web designers) who are comfortable with declarative HTML (Hickson et al., 2014) and CSS (Atkins Jr. et al., 2017) face a major learning barrier when they try to move beyond static web pages toward creating and editing interactive web applications. According to the results from two recent surveys with around 4,000 respondents each, web designers, in fact, prefer HTML and CSS for prototyping (Vinh, 2015) but often struggle to program with imperative JavaScript (Verou, 2016); not to mention implementing the server-side aspects of a web application. To summarize, both visual and textual programming approaches within the reach of most end-user developers currently lack support for complete web application development and/or fail to leverage the full potential of end-user developers' existing web development skills.

1.2 Research Problem and Questions

The research problem of this dissertation, which is motivated by the challenges, needs, and gaps identified above, is stated as follows:
How to unify web application development and, at the same time, leverage end-user developers’ existing skills in declarative languages, namely (X)HTML and CSS?

Thus, this dissertation studies unified web application development both from technical (architectures and declarative languages) and human (end-user developers) perspectives. In order to find answers to the research problem, the following three research questions (RQ) are formulated:

RQ1: What architectural approaches exist to unified web application development?

RQ2: How to extend declarative (X)HTML and CSS to cover common aspects of a web application?

RQ3: How to gently expand end-user developers’ web application development possibilities?

RQ1 is about gaining an in-depth understanding and new insights into the topic of unified web application development. RQ2 explores the possibilities and limitations of declarative (X)HTML and CSS in unified web application development. RQ3 focuses on identifying and overcoming major learning barriers in the end-user development of web applications. Table 1 shows how each of these research questions is related to the publications of this dissertation.

Table 1: Research questions and the publications in which they are addressed.

<table>
<thead>
<tr>
<th>Research Question (RQ)</th>
<th>Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1: What architectural approaches exist to unified web application development?</td>
<td>● ● ○ ○ ○</td>
</tr>
<tr>
<td>RQ2: How to extend declarative (X)HTML and CSS to cover common aspects of a web application?</td>
<td>○ ● ● ● ●</td>
</tr>
<tr>
<td>RQ3: How to gently expand end-user developers’ web application development possibilities?</td>
<td>○ ○ ● ○ ●</td>
</tr>
</tbody>
</table>

Relevance levels: ● = High, ○ = Medium, ○ = Low
1.3 Positioning the Research

The research in this dissertation belongs to the fields of Web Engineering (Murugesan et al., 2001; Deshpande et al., 2002) and End-User Development (Lieberman et al., 2006). The specific research topics covered within these fields include (i) unified web application development, (ii) declarative web application development, and (iii) end-user web application development, with the main research focus lying at their intersection (cf. Figure 2). Moreover, the focus is on building today's modern web applications, i.e., rich, form-based web applications that cover common aspects (presentation, logic, data, communication) and run in standard web browsers without requiring any additional software installation. The type of web applications is not limited to any specific domain. However, covering the architectural aspects of large-scale web applications with complex business logic is left outside the scope of this dissertation. The body of research primarily focused on building web applications visually (rather than textually) is also beyond the scope of this dissertation. This includes, among others, Model-Driven Web Engineering (Koch et al., 2008).

![Figure 2: Research focus and the topics of research covered by this dissertation.](image)
**Unified Web Application Development** refers to architectural approaches that expand the concepts and technologies commonly used on a single tier to cover all three tiers (client, server, data) of a web application. Consequently, an entire multi-tier web application can be written (as a single artifact) in a single programming language. This web application development style is in stark contrast to the conventional practice (cf. Figure 1), which requires different tier-specific experts (professional programmers) because of the various technologies, programming languages and paradigms, and data models used on each tier. The topic of unified web application development is explained thoroughly in Section 2.1.

**Declarative Web Application Development** allows specifying at a high level of abstraction what the web application should do without worrying about the low-level details of how it is actually going to be implemented. This high level of abstraction makes declarative web applications easy to write, reason about, and transform as well as arguably makes their creation more accessible to non-expert programmers (e.g., web designers), just to name a few of their benefits. Examples of widely adopted declarative languages used in conventional web application development include (X)HTML and CSS (user interface definition), JSON (Bray, 2014) and XML (Bray et al., 2008) (data interchange), and SQL (data querying). Declarative programming and its relationship to imperative and other programming paradigms is investigated in Section 2.2.

**End-User Web Application Development** deals with approaches, languages, and tools that aim to empower end-users with various levels of programming experience to create, modify, and extend web applications on their own. The important difference here, compared to professional web application development, is that the web applications they build are primarily intended for personal use and to support specific goals (not related to software development) in their own domain, such as conducting scientific research or building a web-based prototype application. This topic is further considered in Section 2.3.

### 1.4 Research Approach

The research approach employed in this dissertation is *design science research* (Hevner et al., 2004; Peffers et al., 2007). Design science is a
problem-solving research paradigm that seeks to create and evaluate innovative Information Technology (IT) artifacts to meet identified business needs in a certain application domain (Hevner et al., 2004). The produced design artifacts are of four types: constructs, models, methods, and instantiations (March and Smith, 1995; Hevner et al., 2004). Gregor and Jones (2007) further separate them into abstract and material ones. They argue that the abstract artifacts – constructs, models, and methods – are theory or components of theory, whereas the material artifacts (e.g., software or hardware) are their instantiations that have a physical existence in the real world. The design artifacts produced in this dissertation are both abstract and material in nature (cf. Section 1.5 for further details).

The research followed an iterative, six-step design science research methodology process model described by Peffers et al. (2007). First, the research problem and the associated motivation were identified. Next, the objectives for a solution were defined. Furthermore, a thorough literature study, a review of current solutions (if any), and use cases were used to derive requirements. Then, based on the requirements, the artifacts were designed and developed. After that, the feasibility of the artifacts for solving the research problem was demonstrated through proof-of-concept implementations and use case web applications. Besides demonstration, the utility, quality, and efficacy of the artifacts were evaluated in laboratory experiments; both technical and subject-based (Peffers et al., 2012). Finally, the resulting knowledge was communicated via a number of scientific peer-reviewed journal articles and conference papers. In addition, all the material artifacts, i.e., software, were made available to the public under the MIT license.

1.5 Research Contributions

The main contribution of this dissertation is the design, implementation, and evaluation of a novel declarative approach, called XFormsDB, to unified web application development that empowers intermediate level end-user developers (cf. Section 2.3.1 for a definition) in building modern web applications. Specifically, this dissertation makes the following contributions:
• A classification of end-user web application development activities was developed (Publications III and V, Chapter 2)
• A comprehensive survey on unified web application development was conducted (Publication I, revised and significantly extended in Chapter 3)
• A language extension (XFormsRTC) for adding real-time communication capabilities to XForms was designed (Publication III, Chapter 4)
• A language extension (XFormsDB) for adding common server-side and database-related functionalities to XForms was designed (Publication II, Chapter 4)
• A query language (UQL) for improving the expressiveness of CSS Selectors with XPath expressions was designed (Publication IV, Chapter 4)
• A language (TL) for defining reusable multi-tier web application components was designed (Publication V, Chapter 4)
• A declarative web framework (XFormsDB) for unified web application development was designed, implemented, and evaluated; XFormsDB implements the XFormsRTC and XFormsDB language extensions mentioned above (Publications I-II, Chapter 4)
• A web service (UniQue) for unified and efficient querying of heterogeneous web data sources was designed, implemented, and evaluated; UniQue implements the UQL query language mentioned above (Publication IV, Chapter 4)
• A web-based tool (XIDE) for assisting end-user developers in web application development was designed, implemented, and evaluated; XIDE builds upon the XFormsDB framework and implements the TL language mentioned above (Publications III and V, Chapter 4)

1.6 Structure of the Dissertation

The rest of the dissertation is organized as follows: Chapter 2 presents the theoretical foundations on which the research in this dissertation is based. Specifically, the topics covered are unified web application development, programming language paradigms, and end-user development. Next, Chapters 3 and 4 provide answers to the research questions raised earlier in the introduction. Chapter 3 reviews existing approaches to unified web
application development (RQ1). Chapter 4 presents the proposed declarative approach to unified web application development, including its language design, system implementations, and their evaluations (RQ2, RQ3). Finally, Chapter 5 concludes this dissertation and provides suggestions for further research. The original publications are reprinted and attached as appendices at the end of this dissertation.
2 Background

This chapter gives a short overview of the theoretical background relevant to this dissertation. First, a definition of unified web application development is presented. Then, the differences between declarative and imperative programming are discussed. Finally, an introduction to end-user development is given along with a review of end-user web application development activities.

2.1 What is Unified Web Application Development?

Unified web application development (cf. Publication I) is an umbrella term for architectural approaches that enable the development of multi-tier web applications (as a single artifact) in a single programming language. Compared to conventional web application development (cf. Section 1.1), unified web application development reduces the amount of required knowledge, eliminates the impedance mismatch (Copeland and Maier, 1984) between different tier-specific technologies, and abstracts away from the accidental complexity (Brooks Jr., 1987). Additionally, it allows tier-specific experts (e.g., web developers, software engineers, and database administrators) to write entire multi-tier web applications on their own, and can significantly reduce the overall time and cost of web application development.

Several different terms appear in the literature to denote more or less the same concept, including isomorphic programming (Strimpel and Najim, 2016), multi-tier programming (Serrano and Berry, 2012), and tierless programming (Philips et al., 2014). While these different terms are often used interchangeably, there are subtle but noteworthy differences between them, as explained below. For simplicity, however, all the aforementioned
terms are collectively referred to as unified web application development within this dissertation.

The term isomorphic programming is generally associated with client-server web applications that are written entirely in JavaScript (Ecma International, 2017) and use Node.js\(^1\) as a server-side JavaScript runtime environment. In such applications, parts of the code (e.g., views, routes, models) can be shared, rendered, and run on both the browser client and the web application server. Consequently, isomorphic JavaScript applications offer better performance, search engine optimization (SEO), and maintainability compared to their single-page application (SPA) counterparts. (Strimpel and Najim, 2016)

Multi-tier programming and tierless programming take a slightly different architectural approach to unified web application development. Instead of splitting the code base between client and server applications with shared code parts, they enable writing an entire multi-tier web application as a single artifact in a single, unified programming language. From this single source code base (or potentially even a single source file), the language implementation generates deployable code for different target runtime environments, including code that transparently handles cross-tier communication. The difference between the two approaches lies in how tiers are treated. Where multi-tier programming languages promote tiers as first-class values of the language, tierless programming languages aim to fade out the boundary between different tiers to a bare minimum (Serrano and Prunet, 2016).

### 2.2 Declarative versus Imperative Programming

Numerous programming languages have been developed over the years (Cass, 2017; TIOBE, 2017). One way to better understand their similarities and differences is to classify them into programming paradigms, each supporting a different set of programming concepts that shape the way programmers reason about and write computer programs (Van Roy, 2009). Programming languages that realize several paradigms are often referred to as multi-paradigm languages.

Typically, classifications make a basic distinction between declarative and imperative programming paradigms. For example, Appleby and

\(^1\) Node.js, http://nodejs.org/
VandeKopple (1997) organize programming paradigms into a multi-level hierarchy that encompasses two top-level programming paradigms – declarative and imperative – each with its own subparadigms (cf. Figure 3).

In declarative programming, a programmer uses declarations to state what a program should do without necessarily concerning about how to actually do it (Lloyd, 1995). In contrast, imperative programming requires that a programmer also specifies explicit commands on how the program should work (Van Roy and Haridi, 2004). Equivalently, in the terminology of Kowalski’s (1979) equation: "Algorithm = Logic + Control", declarative programming only states the logic component (which determines the meaning) of an algorithm, but not necessarily the control component (which affects the efficiency) as in imperative programming. This higher level of abstraction makes declarative programs generally easier to write, reason about, and transform (Lloyd, 1995). Lloyd also argues that declarative programming improves programmer productivity and makes programming more accessible to ordinary people (i.e., end-user developers). Imperative programming, on the other hand, offers more expressive power, but at the price of making programs more complicated (Van Roy and Haridi, 2004).

![Hierarchy of programming paradigms](image)

**Figure 3:** Hierarchy of programming paradigms, adapted from Appleby and VandeKopple (1997).

This classification of top-level programming paradigms is not absolute though, and thus should rather be thought of as a spectrum with declarative (stateless) programming on one end and imperative (stateful) programming
on the other (Van Roy and Haridi, 2004; Van Roy, 2009). In other words, some programming languages are more declarative than others, and some even support programming in both paradigms.

Declarative and imperative programming languages also differ in their scope of use. Typically, imperative programming languages are more general-purpose and applicable for solving a variety of problems on the client side (JavaScript) and server side (e.g., Java, Python, and C#) of a web application. Declarative programming languages, in turn, are usually more applicable for solving domain-specific problems, such as defining the structure (HTML) or styling (CSS) of a web page, or managing data (SQL) stored in a relational database. This, of course, makes a certain declarative programming 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.

2.3 End-User Development

More and more computer programs are being developed not by professional programmers but by people with expertise in other domains working toward goals for which they need computational support (Ko et al., 2011). This activity where accountants, engineers, designers, teachers, and so forth develop their own software artifacts is commonly referred to as end-user development (EUD). According to Lieberman et al. (2006), EUD can be defined as:

"[...] a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact."

This definition, which is followed in this dissertation, is purposely designed to be broad and inclusive without making any restrictions on end-

---

3 Numerous definitions exist for what counts as a programming language. For example, some definitions (e.g., TIOBE (2017)) require that a programming language should be Turing complete, whereas other definitions (e.g., Cass (2017)) do not have the same restriction. This dissertation adopts the latter — more liberal — view.
4 See a taxonomy of end-user computing concepts in (Korvela, 2014) on how EUD compares with the other main terms used to describe this phenomena.
users' programming experience, intent, identity, or development effort. In practice, however, the programming experience of most end-user developers is limited, especially when it comes to programming practices, languages, and tools used by professional programmers (Lieberman et al., 2006; Ko et al., 2011).

So, why do end-users without formal training in programming engage in software development activities in the first place? One of the main reasons is that EUD empowers end-users to develop and adapt systems themselves (Lieberman et al., 2006). Professional programmers simply lack the domain knowledge and their development cycles are too slow (and expensive) to adapt to end-users' changing requirements (Burnett and Scaffidi, 2013). Another important reason is that there are not enough professional programmers to meet all the software needs of end-users. While the estimates (Scaffidi et al., 2005; Guzdial, 2016) of the number of end-user developers vary, it is nonetheless safe to say that their number far exceeds that of professional programmers. Lastly, recent research (Scaffidi, 2016) also suggests that potential financial considerations could justifiably motivate end-users to engage in programming.

2.3.1 End-User Development of Web Applications

With the ever-increasing popularity and importance of the web in our lives, more and more end-user development is now taking place on the web (Paternò, 2013). Moreover, end-users are now more familiar with the possibilities, concepts, and technologies of the web than they were ever before. Indeed, ACM even cites the ability to create web pages to be at the K–12 level\(^5\) of computer science education (Seehorn et al., 2011). It is important to note, however, that not all end-user developers are the same (Segal, 2005) and the skills they possess and can apply to web application development vary from one end-user developer to another. Consequently, different types of end-user developers need to be supported with techniques, languages, and tools whose level of complexity matches their existing skills and mental models.

Figure 4, which is adapted from Publications III and V, presents a spectrum of end-user web application development appearing between web surfing (far left) and professional development (far right). The spectrum

---

\(^5\) K–12 is a term used in the United States and some other countries, to refer to all primary and secondary education that children receive from kindergarten until 12th grade.
identifies nine levels of end-user web application development activities that are classified into three higher-level categories based on their degree of complexity. In the following, each of these categories is described in detail along with examples of end-user developers that belong to them, their tools, and the development activities (levels) they are able to perform.

**Beginner Level EUD.** The first three development activities (levels 1-3) are within the reach of most end-user developers. At these levels, end-user developers who lack programming skills use visual interaction techniques and tools to customize and create (component-based) web applications. The tools that fall within this category are generally easy to learn and use but limited in terms of what can be done with them. Example tools include Wix, Orbeon Form Builder, and EzWeb/FAST (Lizcano et al., 2011).

**Intermediate Level EUD.** The development activities (levels 4-6) in the middle of the spectrum involve markup authoring, direct source code editing, and single language tier-specific programming. With these skills, intermediate level end-user developers who possess knowledge of a textual markup and/or programming language can overcome some of the limitations of visual tools. For example, they can modify the underlying source code of visually generated web applications or implement custom application functionality from scratch. Adobe Dreamweaver and Click (Rode, Bhardwaj, et al., 2005) are two examples of the tools that end-user

---

6 Wix, http://www.wix.com/
7 Orbeon Form Builder, http://www.orbeon.com/
developers at these levels find useful as the tools support both visual development and direct source code editing of web applications.

**Advanced Level EUD.** The last three development activities (levels 7-9) often require advanced programming skills. At these levels, end-user developers are capable of building fully customized multi-tier web applications from scratch, typically with the help of a web application framework/platform and professional Integrated Development Environment (IDE). For example, eXist-db⁹, which implements the XForms/REST/XQuery (XRX) architecture (Nemeș et al., 2012), provides an all-in-one declarative solution for building interactive web applications.

---

⁹ eXist-db, http://exist-db.org/
3 A Survey on Unified Web Application Development

In Section 2.1, the term *unified web application development* was defined and explained in detail. This chapter follows up on this topic by providing a comprehensive survey and analysis of the existing research\(^\text{10}\) approaches to unified web application development that fall within the scope (cf. Section 1.3) of this dissertation. In the following, the approaches are classified into three categories based on their area of focus: presentation-centric approaches, logic-centric approaches, and data-centric approaches. The survey is an extension of a prior work presented in Publication I.

3.1 Presentation-Centric Approaches

Web designers working on the client tier are, as a rule, familiar with declarative (X)HTML and CSS but not skillful in the server-side aspects of a web application. The same applies to web developers, who are specialized in frontend web development and imperative JavaScript programming. Such experts can benefit from an architectural approach expanding a presentation-centric language with common server-side and database-related functionality (cf. Figure 5).

\(^{10}\) GWT (formerly Google Web Toolkit), one of the three pioneers in unified web application development, being the only exception.


3.1.1 Links

Links\(^{11}\) (Cooper \textit{et al.}, 2007) is a novel functional programming language for building three-tiered Ajax (Garrett, 2005) web applications from a single source. Its syntax resembles mostly that of JavaScript, which makes it an attractive option for web developers. Links incorporates several ideas and features present in other languages, including static typing, web interaction, client-side concurrency, Ajax-style communication, and XML and database programming. In addition, it introduces new web-centric features of its own, such as support for client-server code partitioning and abstractions over form components. The former uses location annotations, i.e., the keywords \texttt{client} and \texttt{server} in the definition of top-level functions to denote the execution side. If the location annotation is missing, the function is made available for both the client and the server. Each of these functions may also call any other function regardless of the location annotation used. The latter, in turn, provides a solution called formlets for form abstractions (Cooper \textit{et al.}, 2008). Formlets are reusable form fragments that support input validation and custom error messages to be displayed upon failure. Furthermore, formlets can be combined together to build larger, more complex forms. Finally, Links' native language-integrated query capabilities allow it to perform create, read, update, and delete (CRUD) operations against relational databases (Cooper, 2009). Consequently, a single

\(^{11}\) Links, \url{http://links-lang.org/}
A Survey on Unified Web Application Development

programming language and data model can be used to write an entire web application, thus eliminating the impedance mismatch problem.

Links was originally implemented as a server-side component written in OCaml. The component consisted of (i) a static analysis phase, (ii) a translator that generated HTML and JavaScript to run in the browser, SQL to run against the database, and cross-tier communication using Ajax, and (iii) an interpreter for the Links code that remained on the server. Since then, various efforts have been made to improve Links' performance (Holmes, 2009), security (Baltopoulos and Gordon, 2009; Corcoran et al., 2009), and tool support, namely Links Online Development Environment (LODE) (Andersson, 2008).

3.1.2 Groke

Groke\textsuperscript{12} is a middleware platform for hosting complex web applications, whose application logic code is written entirely in JavaScript. It implements the approach and ideas presented by Kuuskeri and Mikkonen (2009; 2010a; 2010b), namely how to support server-side JavaScript, application code partitioning, and remote function calls in distributed web applications. First, Groke uses the Rhino\textsuperscript{13} JavaScript engine to enable the use of JavaScript on the server. Second, it partitions application code into reusable code fragments using a module system that conforms to the CommonJS Modules specification\textsuperscript{14}. The module system allows defining reusable server-side JavaScript modules (i.e., \texttt{exports}) and consuming existing ones that run on the server (i.e., \texttt{require}). Moreover, the Groke server makes all the modules and their exposed resources (e.g., functions) available through a REST (Fielding, 2000) inspired interface (Kuuskeri and Mikkonen, 2010b). Some examples of available server-side JavaScript modules and their responsibilities include accessing the file system, logging, scheduled tasks, real-time collaboration, and database management. Third, Groke allows both clients and the server to invoke exposed server-side JavaScript functions as if they were local. Thus, it hides all low-level networking details present in deployed code from application developers. Indeed, Groke automatically modifies requested application code and generates the required code at runtime for the client and the server. The generated code

\textsuperscript{12} Groke, http://github.com/wuher/groke
\textsuperscript{13} Rhino, http://www.mozilla.org/rhino/
\textsuperscript{14} CommonJS Modules, http://wiki.commonjs.org/wiki/Modules
then handles all cross-tier communication using Ajax and Comet (Russell, 2006). The presented features have also been integrated into Lively Kernel\textsuperscript{15} (Ingalls et al., 2008) – a platform for developing client-side web applications in JavaScript – to enhance it with peer-to-peer collaboration (Kuuskeri et al., 2010). When building web applications with Groke, existing JavaScript libraries and development tools can be freely utilized.

### 3.1.3 SWAC

SWAC\textsuperscript{16} (Sharing a Web Application’s Codebase) (Ast et al., 2013; Ast et al., 2014) is a JavaScript-based framework for efficiently developing progressively enhanced web applications that share a common code base between the client and the server. The code base is divided into multiple files, with a number of models, views, and routes. The core of the approach revolves around the latter, i.e., routes, which are defined hierarchically. By doing so, SWAC makes it possible to move through a route step by step and execute only the necessary parts of a web application's business logic. Similarly, only the necessary parts of the view can be rendered; on data changes, these view fragments are automatically updated using built-in Ajax support. Consequently, common parts (e.g., validation and rendering) needs to be implemented only once as they are shared between the client and the server. SWAC also provides a means of restricting access to certain code parts. For example, the client side cannot directly access the database (calls are proxied through an automatically provided RESTful API), models and their properties can be declared as server only, and custom logic can be applied to authenticate and authorize user access to selected areas and models’ data.

SWAC is implemented entirely in JavaScript using Node.js\textsuperscript{17} on the server. Therefore, existing JavaScript libraries, APIs, and development tools can be leveraged in the development process. Additionally, the SWAC framework supports the use of Apache CouchDB\textsuperscript{18} as a JSON (Bray, 2014) document store.

\textsuperscript{15} Lively Kernel, http://www.lively-kernel.org/
\textsuperscript{16} SWAC, http://github.com/rkusa/swac
\textsuperscript{17} Node.js, http://nodejs.org/
\textsuperscript{18} Apache CouchDB, http://couchdb.apache.org/
3.1.4 STiP.js

STiP.js\(^1\) (Slicing Tierless JavaScript Programs) (Philips \textit{et al.}, 2014) is a
tier splitting tool that enables tierless web application development in
JavaScript. With STiP.js, application logic code is written akin to single-
tiered JavaScript programs. The only exception is the inclusion of location
annotations, namely \texttt{@client}, \texttt{@server}, and \texttt{@shared} inside
comments, which specify what tier a particular block of code belongs to.
Using the annotations, the tool is able to construct a distributed dependency
graph, which in turn is used to split the code into the client and server tiers
through a variant of \textit{program slicing} (Weiser, 1984). These tier-specific
program slices are subsequently transpiled to produce deployable code that
implements seamless cross-tier function calls of STiP.js, for instance, using
remote procedure calls (RPCs). Currently, STiP.js realizes the tier split by
generating server code for Meteor\(^2\), or Node.js with the asyncCall library
for remote communication. Eventually, the generated client code is
manually included into a separate HTML document that contains element
IDs to bind data to JavaScript variables. When building web applications,
developers can reuse existing JavaScript development tools. STiP.js also
supports the use of existing client-side and server-side JavaScript libraries
(e.g., MongoDB's Node.js driver) to enable the creation of highly interactive
data-driven web applications. Furthermore, it provides a JavaScript library
of its own containing tierless primitives for widely used tasks in web
applications, such as DOM manipulation and two-way cross-tier
communication.

While the approach behind STiP.js has several benefits over Meteor (e.g.,
seamless cross-tier communication and protection of sensitive server-side
code), it is still a work in progress. For instance, STiP.js only supports a
small subset of JavaScript, and lacks distributed failure handling, tierless
primitives for data management and data synchronization, and dedicated
tools to maintain the tierless illusion (Philips \textit{et al.}, 2015).

3.1.5 WebNat

WebNat (Sinha \textit{et al.}, 2015) is a tierless programming language that enables
Model-View-Controller (MVC) (Krasner and Pope, 1988) based design of
interactive web applications \textit{end-to-end}. The WebNat language builds upon

\(^1\) STiP.js, http://soft.vub.ac.be/~lphilips/stip/
\(^2\) Meteor, http://www.meteor.com/
the syntax of JavaScript, which it extends with static typing, built-in data query capabilities, and multiple programming abstractions that are mainly related to end-to-end data binding and reactivity. Together these abstractions reduce the need for writing excessive and error-prone glue code and make web programming oblivious of different tiers. Another key benefit of the language design is that WebNat seamlessly integrates with the JavaScript ecosystem both on the client side (via various libraries) and server side (via Node.js). WebNat also includes a utility library of its own to facilitate, among others, DOM manipulation and invocation of external REST services. The language features presented above, however, only allow defining models and controllers of a web application, whereas accompanied views must be defined separately as HTML & CSS templates with placeholders for dynamic data expressions.

In the current implementation, the WebNat compiler takes a higher-level WebNat program as an input and, as the result of a set of transformations, creates a deployable multi-tier web application using a JavaScript technology stack. Specifically, the deployed web application uses HTML, CSS, and JavaScript (with Backbone\textsuperscript{21} and Handlebars\textsuperscript{22}) on the client, Node.js on the server, and MongoDB\textsuperscript{23} as a database. Client-server communication takes place using asynchronous JavaScript callbacks.

### 3.2 Logic-Centric Approaches

Software engineers, who are responsible for writing back-end server code, possess excellent technical knowledge but do not necessarily have all the skills needed to create a fancy user interface for a web application. A logic-centric architectural approach (cf. Figure 6) based on a general-purpose programming language (e.g., Java, Python, and C#) is the most attractive option in this case.

\textsuperscript{21} Backbone, http://backbonejs.org/
\textsuperscript{22} Handlebars, http://handlebarsjs.com/
\textsuperscript{23} MongoDB, http://www.mongodb.com/
3.2.1 GWT

The GWT\textsuperscript{24} (formerly Google Web Toolkit) open-source project from Google (2006) is a mature development toolkit that enables writing, debugging, testing, and running complex web applications in a unified manner. In GWT, the client-side and server-side (if any) parts of the application code are written in a single programming language – namely, object-oriented, statically typed Java. Both parts reside in the same code base, where they are separated into their own Java packages. The \textit{client} package encompasses the application code that implements the user interface and client-side functionality. The user interface can be constructed either programmatically using widgets and panels, or declaratively using UiBinder XML templates. Client-side functionality can be complemented with JavaScript by integrating JavaScript code directly into the Java source code using GWT's JavaScript Native Interface (JSNI). The optional \textit{server} package contains the application code for implementing server-side functionality, such as business logic and data access. For example, data stored in a relational or object database can be queried using Java Persistence Query Language (JPQL); an object-oriented, SQL-like query language that is defined as part of the Java Persistence API (JPA).

\textsuperscript{24} GWT, http://www.gwtproject.org/
specification\textsuperscript{25}. As both sides are implemented in Java, parts of the application code can be shared between the client and the server. The communication between the two sides is handled using the rather involved GWT RPC mechanism, which allows clients to make asynchronous calls to services that run on the server.

The GWT software development kit (SDK) and tools are available as a standalone download and as a plugin for popular Java IDEs, including Eclipse. The tools include, among others, GWT Designer and the GWT compiler. The former is a visual tool bundled with Google Plugin for Eclipse (GPE) for creating graphical user interfaces, whereas the latter is a Java-to-JavaScript compiler. For deployment, the GWT compiler translates the Java source code within the \textit{client} package into a set of highly optimized, browser-specific files, including the main JavaScript file to be included in a host HTML page to load the application. The server-side code is compiled as usual to Java bytecode. Finally, the compiled web application can be deployed to any servlet container, such as Apache Tomcat\textsuperscript{26}.

Several open-source and research projects port, extend, or leverage GWT in their work. For example, there are a wide variety of GWT-related open-source projects hosted on GitHub\textsuperscript{27}, including the Vaadin\textsuperscript{28} framework. The Swift (Chong \textit{et al.}, 2007; Chong \textit{et al.}, 2009) system from Cornell University allows building secure web applications in Java that are automatically partitioned between the client and the server based on information-flow constraints. Swift uses the GWT Java-to-JavaScript compiler to translate the partitioned client Java code (and the Swift client runtime system) to JavaScript.

3.2.2 Hop

Hop\textsuperscript{29} (Serrano \textit{et al.}, 2006; Serrano and Queinnec, 2010; Serrano and Berry, 2012) is a multi-tier programming language for building interactive multimedia web applications (Serrano, 2007). It is a superset of the functional-first Scheme programming language, that covers both client-side and server-side programming, HTML abstraction, reactive programming (Berry \textit{et al.}, 2011), two-way communication between the client and the

\textsuperscript{25} Java Persistence API (JPA), \url{http://download.oracle.com/otn-pub/jcp/persistence-2_1-fr-eval-spec/JavaPersistence.pdf}
\textsuperscript{26} Apache Tomcat, \url{http://tomcat.apache.org/}
\textsuperscript{27} GitHub, \url{http://github.com/}
\textsuperscript{28} Vaadin, \url{http://vaadin.com/}
\textsuperscript{29} Hop, \url{http://hop.inria.fr/}
server, modules, and access to third-party data sources – all within a single, unified language. To denote on which side certain parts of the code should be executed, the Hop language uses special annotations at the level of individual expressions. Expressions prefixed with the \( \sim \) construct are client-side code and executed by the web browser. With the \$ construct, server-side code can be injected into the client-side code on the server at elaboration time. Unannotated code is executed on the server. The separation between the two sides takes place on the server by compilation. Hop also comes with a set of widgets and APIs to ease the programming of user interfaces and business logic, respectively. For database management, Hop supports the use of embedded SQL statements.

The Hop development kit contains two compilers, an interpreter, and a bootstrapped web server (Serrano, 2009). The first compiler is responsible for producing HTML and JavaScript on the fly for modern web browsers\(^\text{30}\). The second compiler generates native code for the server. The compilers also automatically generate code for two-way cross-tier communication. The interpreter residing on the server is merely for fast prototyping. While Hop offers good tool support including a novel multi-tier debugger (Serrano, 2014), it lacks many of the tools and advantages available in mainstream programming languages. To overcome these limitations, the Hop principles were transposed to JavaScript in its latest incarnation, HopScript (Serrano and Prunet, 2016).

### 3.2.3 F# Web Tools

F# Web Tools\(^\text{31}\) (Petříček and Syme, 2007) is an extension to the functional-first F# programming language for developing type-safe Ajax web applications in an integrated way. In F# Web Tools, user-defined modules and classes are annotated using specific attributes (i.e., \texttt{NeutralSide}, \texttt{ClientSide}, and \texttt{MixedSide} with a monad type \texttt{server} (default) or \texttt{client}) to indicate the intended execution environment. Based on the annotations, the translator determinates which parts of the application logic code should be served to the client as JavaScript and which parts should be run on the server as a regular .NET program. F# Web Tools also provides a means for client-side code to access native JavaScript components and call

---

\(^{30}\) The compiler can optionally produce, for example, HTML4 and Flash for legacy web browsers.

\(^{31}\) F# Web Tools, http://fswebtools.codeplex.com/
server-side functions, and for server-side code to access heterogeneous data sources using the LINQ libraries (Syme, 2006).

WebSharper\textsuperscript{32} takes the core design principles of F# Web Tools to the next level with its mature framework for developing reactive web and mobile applications (Granicz, 2015). The main complementary features include various functional reactive web abstractions (Denuzière and Granicz, 2016b), real-time data synchronization between clients and a server (Denuzière and Granicz, 2016a), end-to-end security (Balliu \textit{et al.}, 2016), source maps, and support for C# as an alternative programming language to F#. Developing WebSharper applications is currently possible with several IDEs, including Visual Studio, Xamarin Studio, and an online IDE called CloudSharper\textsuperscript{33}.

\subsection{3.2.4 WebDSL}
WebDSL\textsuperscript{34} (Visser, 2008) is a novel domain-specific language (DSL) optimized for developing form-based interactive web information systems. It supports \textit{separation of concerns} through a number of specially designed sublanguages that are \textit{linguistically integrated} into a single coherent language (Groenewegen \textit{et al.}, 2010). The sublanguages share a common type system and cover different aspects of a web application, including user interface, data modeling, application logic, data validation, access control, and workflow. In addition to its own domain-specific sublanguages, WebDSL uses a subset of Hibernate Query Language (HQL) for database querying and CSS for visual formatting. Many of these sublanguages are declarative in nature. However, the sublanguage used for defining application logic (action code) relies on an imperative Java-like language. WebDSL also provides advanced features for building web applications, such as static consistency checking (Hemel \textit{et al.}, 2011), asynchronous page updates (Groenewegen and Visser, 2013), and the native Java class interface. Furthermore, HTML and JavaScript escapes allow the use of existing client-side widgets in WebDSL applications.

WebDSL is available as a standalone compiler and as a plugin\textsuperscript{35} for Eclipse IDE. From a WebDSL application, the compiler generates HTML,

\textsuperscript{32} WebSharper, http://websharper.com/
\textsuperscript{33} WebSharper IDEs, http://websharper.com/downloads
\textsuperscript{34} WebDSL, http://webdsl.org/
\textsuperscript{35} WebDSL plugin for Eclipse IDE (abbreviated as WPE in this dissertation), http://webdsl.org/page/WebDSL.plugin
CSS, and JavaScript for the client, and Java or Python via Platform Independent Language (PIL) (Hemel and Visser, 2010) for the server. MySQL is used as a persistence store. The research group has also created mobl, a variation of the WebDSL language, to declaratively construct pure client-side mobile web applications (Hemel and Visser, 2011).

### 3.2.5 ML5

ML5\(^{36}\) (Murphy VII et al., 2008), a variant of ML, is a functional programming language for spatially distributed computing. With ML5, an entire distributed application can be developed and reasoned about as a unified program. This also applies to web applications. Indeed, the current prototype implementation specifically targets web programming by compiling an ML5 source code file into JavaScript for the web browser and bytecode for the ML5 web server. Additionally, the compiler generates code for two-way cross-tier communication and verifies that only well-typed programs are translated; ML5’s type system is based on modal logic.

ML5 uses the special `from address get expression` construct to execute any ML5 expression from another host (i.e., client or server). To support the specific needs of web programming, the language offers a simple interface for DOM manipulation and a library to access a rudimentary key-value persistent database on the server. Interestingly, the DOM interface is used as the only way to define and manipulate user interfaces via embedded HTML strings. Clearly, more advanced features and improved tool support is called for.

Another closely related project is SMLtoJs\(^{37}\) (Elsman, 2011), a compiler from Standard ML (SML) to JavaScript. SMLtoJs provides a type-safe interface for interacting with plain JavaScript (e.g., the DOM API) and allows JavaScript code to call SML functions. Furthermore, it has a library support for reactive programming (Bainomugisha et al., 2013) and most of the SML Basis Library modules. When used together with SMLserver\(^{38}\) (Elsman and Hallenberg, 2003; Elsman and Friis Larsen, 2004), type-safe Ajax-like web applications can be written entirely in SML.

---

\(^{36}\) ML5, [http://www.cs.cmu.edu/~tom7/ml5/](http://www.cs.cmu.edu/~tom7/ml5/)

\(^{37}\) SMLtoJs, [http://www.smlserver.org/smltojs/](http://www.smlserver.org/smltojs/)

\(^{38}\) SMLserver, [http://www.smlserver.org/](http://www.smlserver.org/)
3.2.6 Volta

The discontinued Volta (Manolescu et al., 2008) project from Microsoft Live Labs is a toolset that simplifies the development process of multi-tier applications by leveraging established programming languages, libraries, compilers, tools, and the .NET Framework\(^{39}\). The novelty of the approach revolves around the idea of using declarative annotations (i.e., \texttt{RunAt} and \texttt{Async} custom attributes) and rewriting programs at the .NET Common Intermediate Language (CIL) bytecode level. Together, these features make the development of distributed applications completely source programming language and target execution environment agnostic. For example, a client-server web application can be written in any programming language that compiles to .NET CIL, such as C++, C#, Visual Basic, or Python. The Volta recompiler then takes the initial single-tier CIL as input and tier-splits it into an equivalent multi-tier CIL based on declarative tier-splitting and asynchronous method definitions. Additionally, Volta automatically replaces cross-tier method calls with Ajax RPCs and translates the client-side component into JavaScript for execution in any standards-compliant web browser; the server-side component is executed within the Common Language Runtime (CLR). Volta also provides a foreign function interface for interoperating with JavaScript code and allows making use of the .NET Framework capabilities, including LINQ. The implementation of Volta is available as a plugin\(^{40}\) for Visual Studio 2008 IDE. The Volta extension for Visual Studio also comes with the Rotunda profiler that enables end-to-end profiling of web applications.

Since Volta, a few closely related research projects have emerged. Ripley (Vikram et al., 2009) is a system that automatically preserves the integrity of a distributed computation through replicated execution. Ripley is built on top of Volta but the underlying techniques are generic enough to be applied to any distributed web application. Both IL2JS (Microsoft, 2011) and IL2JS (Kolarz, 2012) are spin-offs of the original Volta project.


\(^{40}\) Volta plugin for Visual Studio 2008 IDE (abbreviated as VPVS in this dissertation)
3.2.7 Ocsigen

Ocsigen\(^{41}\) (Balat \textit{et al.}, 2009; Balat, 2014) is a mature web programming framework for developing client-server web applications (Balat \textit{et al.}, 2012) and widgets (Balat, 2013). It promotes static type checking throughout the web application and abstracts away many technical details from programmers, thus simplifying the development process (Balat, 2006). In Ocsigen, both the client and server parts – database queries included – of a web application are written as a single program mainly by leveraging the functional aspects of a multi-paradigm programming language called OCaml. Indeed, Ocsigen extends OCaml with a number of modules to, among others, generate and manipulate web pages, allow bidirectional communication between the client and the server, and support \textit{functional reactive programming} (Hudak, 1999). It also introduces special constructs to call JavaScript methods from OCaml, and denote the side (i.e., client, server, or shared) on which a certain code section should be executed. In addition, code fragments can be injected into opposite sides.

The implementation of Ocsigen is composed of several components\(^{42}\). The main ones are the full-featured and extensible Ocsigen web server and its module called Eliom (i.e., the framework with a compiler). The Eliom compiler is responsible for extracting the server part of a program and compiling it to OCaml bytecode to be executed on the server. The \texttt{js\_of\_ocaml} compiler (Vouillon and Balat, 2014) in turn extracts and compiles the client part of the program into OCaml bytecode, and finally into JavaScript to be included in an HTML document. An experimental library called MaCaQue\(^{43}\) can be used to perform CRUD operations against PostgreSQL databases. Currently, Ocsigen lacks dedicated development environments but existing IDEs for OCaml can be used to write code.

3.2.8 Haste

Haste\(^{44}\) (Ekblad and Claessen, 2014) is a software development suite for building interactive, distributed web applications in purely functional Haskell. The suite includes a Haskell to JavaScript compiler built on top of

\(^{41}\) Ocsigen, http://ocsigen.org/
\(^{42}\) Ocsigen components, http://ocsigen.org/projects
\(^{43}\) MaCaQue, http://github.com/ocsigen/macaque
\(^{44}\) Haste, http://haste-lang.org/
Glasgow Haskell Compiler (GHC) and a monad-based library that extends Haskell with support for DOM manipulation, event handling, Ajax, local storage, and WebSocket (Fette and Melnikov, 2011), just to name a few. The library also provides a foreign function interface for interoperating with JavaScript code at a high level of abstraction (Ekblad, 2015). Haste itself does not provide means for accessing backend databases, but there are a number of libraries available (e.g., HaskellDB) that allow Haskell to interface with different types of databases. Together these components constitute a seamless programming model that covers all layers and allows the logic of a web application to be written as a single, type-safe program. Finally, the Haste implementation takes care of compiling the program into a server-side binary and a client-side JavaScript file to be linked to a web page. The communication between the two sides is handled via automatically generated, type-safe RPC calls. While Haste itself lacks dedicated development tools and environments, there are a number of Haskell IDEs available.

Like Haste, the Utrecht Haskell Compiler (UHC) JavaScript library (Dijkstra et al., 2013) also provides means for compiling Haskell code to JavaScript and interfacing with JavaScript code from Haskell. UHCJS, however, suffers from poor performance and lacks support for advanced Haskell language features.

### 3.2.9 Ur/Web

Ur/Web (Chlipala, 2010; Chlipala, 2015b; Chlipala, 2016) is a novel statically typed purely functional programming language for writing concise, secure, and high-performance web applications. The language extends Ur – a programming language with an SML-like syntax and a rich type system – with web-specific functionality to support parsing and compilation of inline XML and SQL. The parsing extension allows embedding of HTML and SQL code to the language as first-class, strongly typed values. The optimizing compiler, in turn, transforms high-level Ur/Web source code into low-level native code with extremely competitive performance (Chlipala, 2015a). Specifically, the compiler generates

---

45 Glasgow Haskell Compiler (GHC), http://www.haskell.org/ghc/
46 Applications and libraries/Database interfaces, http://wiki.haskell.org/Applications_and_libraries/Database_interfaces
47 Haskell IDEs, http://wiki.haskell.org/IDEs
JavaScript and HTML for the client, low-level C code for the server, and SQL statements to be run against SQL databases from *location-obliverous* (i.e., unannotated) source code. Other notable features of Ur/Web include a module system, support for building reactive graphical user interfaces (GUIs), abstractions over two-way client-server communication, and interfaces for connecting with C and JavaScript code.

The implementation of Ur/Web consists of two main components: the abovementioned Ur/Web compiler and a runtime system, which handles network communication and data marshaling. The compiler is implemented in SML, whereas the runtime system is implemented in JavaScript (client part) and C (server part). Other than that, there are no dedicated tools available to support Ur/Web based web application development.

### 3.3 Data-Centric Approaches

Database administrators have special needs for designing and developing web applications. For example, they might need to implement a user interface to visualize and analyze available data. To leverage their existing knowledge, they need a development approach based on concepts and technologies commonly used in the data tier. Data-centric architectural approaches extend a query language’s (e.g., declarative SQL and XQuery) capabilities to cover the most common client-side and server-side functionality (cf. Figure 7).

**Figure 7**: Data-centric approach to unified web application development.
3.3.1 XQIB

XQIB\(^{49}\) (Fourny et al., 2007; Fourny et al., 2008; Fourny et al., 2009) is a language extension to the XQuery family of W3C standards (Bamford et al., 2009). It extends functional XQuery (Robie et al., 2017) – a shorthand for XML Query Language – with a set of browser-specific functions to enable DOM access, CSS access, browser context access, event listener management, Ajax, and JavaScript-XQuery communication. Consequently, XQuery can be used in the browser to implement common client-side application logic instead of, or as a complement to, JavaScript. Moreover, by using XQIB in conjunction with an XQuery web application server and a native XML database (e.g., eXist-db\(^{50}\) (Meier, 2003)) enables the use of XQuery and XML on all layers (i.e., presentation, logic, and data) of a web application.

XQIB was originally implemented as a browser extension for Microsoft Internet Explorer, and later for Firefox. Currently, the XQIB extension functions are implemented in the MXQuery\(^{51}\) XQuery engine, which is available, for example, as a multi-browser JavaScript library (Etter et al., 2011). With the library, extended XQuery code can be embedded into XHTML documents using the `<script>` tag for execution inside the web browser. No visual tools are available for the development of XQuery-based web applications. However, several XQuery IDEs exist for editing, debugging, and profiling server-side XQuery code.

Further studies have also confirmed the suitability of XQuery for unified web application development. In (Kaufmann and Kossmann, 2008; Kaufmann and Kossmann, 2009), Kaufmann and Kossmann demonstrated that XQuery is expressive enough for developing even an enterprise-level web application. Gui et al. (2010), in turn, extended XQuery with a set of functionalities for both client-side and server-side programming. Their implementation works differently by translating the extended XQuery syntax into appropriate target language for the client and the server, i.e., JavaScript and Java Servlets, respectively.

\(^{49}\) XQIB, http://www.xqib.org/
\(^{50}\) eXist-db, http://exist-db.org/
\(^{51}\) MXQuery, http://mxquery.org/
3.3.2 FORWARD

FORWARD\textsuperscript{52} (Fu et al., 2010; Fu et al., 2011; Fu et al., 2013) is an SQL-based declarative web framework that enables the rapid development of data-driven Ajax reports and applications. In FORWARD, applications follow the MVC architectural pattern, and thus are modularized into three components: \textit{unified application state} (models), \textit{page specifications} (views), and \textit{action specifications} (controllers). The Unified Application State (UAS) is essentially a virtual database that integrates data from both persistent and transient data sources, such as JSON native stores, relational databases, sessions, URL parameters, the HTML DOM, and JavaScript components. For querying and modeling the data, the UAS uses a single query language called SQL++ (Ong et al., 2015) and SQL++ values, respectively. The former extends SQL with support for semi-structured data, whereas the latter provides a unified data model that subsumes both the relational model and JSON. Page specifications, in turn, specify the user interface using HTML and a template markup. The template markup goes beyond the capabilities of HTML by enabling the generation of dynamic data on web pages. Specifically, it supports the execution of SQL++ queries over the UAS, conditional iteration of query results, event-based user interaction with the data, two-way data binding, and Ajax communication. Moreover, it offers a set of declarative \textit{visual unit wrappers} around popular third-party JavaScript components (e.g., Google Maps and Highcharts) for building rich presentations (Fu et al., 2014). Finally, action specifications map URL patterns to functions, which manipulate the UAS and define the next page to be displayed. Action specifications are implemented in PL/SQL++\textsuperscript{53}, which is an extended version of the procedural language for SQL (PL/SQL).

In the current implementation, the FORWARD middleware translates page specifications into native browser technologies, i.e., HTML, CSS, and JavaScript, capable of automatically handling cross-tier communication and incremental page updates. Similarly, it decomposes SQL++ queries into the underlying databases' native queries, executes them, and finally, combines the query results before returning them to the client. As of now, FORWARD is available only as a cloud service, and the development is done with FORWARD Online IDE\textsuperscript{53}.

\textsuperscript{52} FORWARD, http://forward.ucsd.edu/
\textsuperscript{53} FORWARD Online IDE (abbreviated as FODE in this dissertation), http://forward.ucsd.edu/ide/main
3.4 Comparative Analysis

This section presents a comparative analysis of the presentation-centric, logic-centric, and data-centric approaches reviewed above to identify well-covered topics, emerging trends, current challenges, and existing gaps in research. To facilitate the analysis and comparison, a set of criteria was established. The following criteria are adapted from Toffetti et al. (2011).

**Language.** The base programming language forms the core of this analysis. It determines probable target users and (to some extent) applications as well as the category (i.e., presentation-centric, logic-centric, data-centric) an individual approach belongs to. Additionally, each programming language comes with distinct features, including available data types and structures, and supported programming paradigms.

**Paradigm.** As explained in Section 2.2, programming paradigms can be divided into two top-level categories: declarative and imperative. This criterion indicates the dominant top-level programming paradigm of an individual approach.

**Abstraction.** Richard-Foy et al. (2013) classify approaches providing high-level abstractions for web programming into four categories: fat languages, thin languages, domain-specific languages (DSLs), and deeply embedded languages. Whether deeply embedded languages meet the definition of unified web application development is arguable (Reynders et al., 2014; Ekblad and Claessen, 2014), and thus individual approaches following it were excluded from this review right from the start.

**Communication.** According to a classification by Franklin and Zdonik (1998), data delivery mechanisms can be divided into two top-level categories: client pull and server push. This criterion indicates whether an individual approach supports these communication mechanisms, i.e., provides built-in support for asynchronous, aperiodic, pull-based and push-based communication between the client and the server.

**Tool Support.** Does an individual approach provide any dedicated tools that facilitate unified web application development, and thus maintain the tierless illusion throughout the different phases of the development process?
Reference. A reference to the first peer-reviewed scientific publication that provides an extensive description of the individual approach. In the case of GWT, this criterion refers to the year of its release.

Presentation-centric approaches rely heavily on existing web application user interface technologies. All of the presented approaches (Links, Groke, SWAC, STiP.js, WebNat) are based, either directly or indirectly, on JavaScript. This comes as no surprise, given that JavaScript is the de facto programming language of the web and the only programming language natively supported by all major web browsers. The use of JavaScript also means that all the approaches focus more or less on the same target users, namely web developers. Out of the five approaches in this category, Links is the only one that truly stands out – in almost every aspect, too. Most of the differences are due to historical reasons as back in 2006 when Links was published, the JavaScript programming language and the existing ecosystem around it were not as well-developed as they are today. Moreover, besides WebNat, Links is the only other approach in this category to provide advanced features like static typing and native language-integrated query capabilities.

Logic-centric approaches form the largest category with nine approaches (GWT, Hop, F# Web Tools, WebDSL, ML5, Volta, Ocsigen, Haste, Ur/Web) to choose from. The approaches cover a wide variety of different programming languages and paradigms. Most of these languages are statically typed and follow the functional programming style; a subparadigm of declarative programming. Moreover, seven out of nine approaches provide abstractions for web programming by extending an existing general-purpose programming language. Only WebDSL and Ur/Web are based on a completely new programming language; a design decision that requires a lot of effort in terms of language and compiler design, and tool support (Kossakowski et al., 2012), and can significantly hinder the adoption (Fernández et al., 2000) mainly due to the lack of available debugging, development, and configuration features (Sinha et al., 2015). Many of the approaches support building modern web applications with reactive user interfaces that automatically update their views upon receiving data from the server. GWT and WebSharper (the successor of F# Web Tools) are by far two of the most mature approaches, both being supported by several IDEs via plugins.
Data-centric approaches include an XQuery-based (XQIB) and an SQL-based (FORWARD) solution for building Ajax web applications in a unified manner. Both XQuery and SQL have been standardized, and provide a declarative language for querying XML and relational data sources, respectively. Moreover, FORWARD even allows querying, for example, native JSON stores using its extended SQL language called SQL++. With both XQIB and FORWARD, the development of a web application is done in multiple source code files but using a single, extended query language. FORWARD comes with its own IDE, whereas XQIB relies solely on existing XQuery development tools that lack support for XQIB specific language extensions. The research related to the FORWARD project is still ongoing.

Table 2 summarizes the comparative analysis of the reviewed approaches to unified web application development. Several interesting observations can be derived by examining the table. First, the research in unified application development started in 2006 and was pioneered by three different approaches: GWT from Google, Hop from Inria, and Links from the University of Edinburgh. Second, the current research focus is moving toward presentation-centric approaches based on JavaScript (cf. SWAC, STiP.js, WebNat, and the latest version of Hop). Third, none of the presentation-centric approaches reviewed above are targeted at web designers, who are non-expert programmers. Fourth, declarative, and especially functional, programming paradigm has proven its suitability and usefulness for unified web application development in all three categories (i.e., presentation-centric, logic-centric, data-centric). The same applies to various programming language features, such as static typing, modules, and extensibility. Fifth, recent efforts within the research domain have focused on extending an existing programming language with real-time data synchronization and communication, and reactive programming features. Finally, many of the reviewed approaches lack dedicated tools and development environments, which partly hinder their adoption.
<table>
<thead>
<tr>
<th>Approach</th>
<th>Language</th>
<th>Paradigm</th>
<th>Abstraction</th>
<th>Communication</th>
<th>Tool Support</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Presentation-centric</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Links</td>
<td>Links (new)</td>
<td>Declarative</td>
<td>Fat language</td>
<td>Pull</td>
<td>LODE</td>
<td>Cooper et al. (2007)</td>
</tr>
<tr>
<td>SWAC</td>
<td>JavaScript</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull</td>
<td>–</td>
<td>Ast et al. (2013)</td>
</tr>
<tr>
<td>STiP.js</td>
<td>JavaScript</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Philips et al. (2014)</td>
</tr>
<tr>
<td><strong>Logic-centric</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GWT</td>
<td>Java</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull</td>
<td>GPE</td>
<td>Google (2006)</td>
</tr>
<tr>
<td>Hop</td>
<td>Scheme^c</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>Debugger</td>
<td>Serrano et al. (2006)</td>
</tr>
<tr>
<td>F# Web Tools^d</td>
<td>F#</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull</td>
<td>–</td>
<td>Peťiček and Syme (2007)</td>
</tr>
<tr>
<td>WebDSL</td>
<td>WebDSL (new)</td>
<td>Declarative</td>
<td>DSLs</td>
<td>Pull &amp; Push</td>
<td>WPE</td>
<td>Visser (2008)^g</td>
</tr>
<tr>
<td>ML5</td>
<td>SML</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Murphy VII et al. (2008)^a</td>
</tr>
<tr>
<td>Volta</td>
<td>C++ / C# / ...</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull</td>
<td>VPVS</td>
<td>Manolescu et al. (2008)</td>
</tr>
<tr>
<td>Ocsigen</td>
<td>OCaml</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Balat et al. (2009)</td>
</tr>
<tr>
<td>Ur/Web</td>
<td>Ur (new)</td>
<td>Declarative</td>
<td>Fat language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Chlipala (2015b)</td>
</tr>
<tr>
<td><strong>Data-centric</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XQIB</td>
<td>XQuery</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull</td>
<td>–</td>
<td>Fourmy et al. (2009)</td>
</tr>
<tr>
<td>FORWARD</td>
<td>SQL</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull</td>
<td>FODE</td>
<td>Fu et al. (2010)</td>
</tr>
</tbody>
</table>

^a Revised paper; the original publication appeared one year earlier.
^b Support for server push via third-party libraries; lacks built-in support.
^c Since version 3.x, Hop uses JavaScript (presentation-centric) as its base programming language.
^d WebSharper, the successor of F# Web Tools, supports C# as an alternative programming language, server push communication, and several IDEs.
3.5 Discussion

Several interesting, but out of the scope of this survey, approaches to unified web application development have been proposed both inside and outside of academia that are worth discussing briefly.

Hilda\(^{54}\) (Yang et al., 2006) is a novel high-level language that provides a declarative and unified approach for building data-driven web applications. What makes Hilda unique is its ability to \textit{automatically} partition a web application to optimize the runtime system performance across multiple tiers (Yang et al., 2007). Consequently, Hilda removes the burden of manual tier-splitting decision making from programmers. Hilda also offers a What You See Is What You Get (WYSIWYG) editor called Hilda GUI (Gupta et al., 2007), which allows non-technical users to visually build their web applications. The Hilda client-side runtime system requires the use of a proprietary browser plugin (i.e., Java applet), which is the main reason for its exclusion. QHTML (El-Ansary et al., 2005) is another early approach to unified web application development that also relies on a Java applet in order to run in web browsers. Unlike Hilda, QHTML extends an existing programming language, namely multi-paradigm Oz, with various aspects needed for the development of multi-tier web applications.

Deeply embedded languages (Richard-Foy et al., 2013) provide an alternative but arguable (Reynders et al., 2014; Ekblad and Claessen, 2014) approach to unified web application development, as explained earlier in Section 3.4. The idea behind the approach is to deeply embed domain-specific languages (DSLs) into a general-purpose programming language (GPL). This allows embedded DSLs to exploit the features of a host language, such as static typing. For example, js-scala\(^{55}\) (Kossakowski et al., 2012) embeds JavaScript as a DSL in Scala using an extended version of the lightweight modular staging (LMS) framework (Rompf and Odersky, 2010; Rompf and Odersky, 2012). Sunroof\(^{56}\) (Bracker and Gill, 2014), in turn, embeds JavaScript as a DSL in Haskell using monadic reification.

In addition to GWT (included in this survey), many interesting approaches to unified web application development have recently emerged outside of academia. These include multi-tier/tierless programming

\(^{54}\) Hilda, http://www.cs.cornell.edu/bigreddata/hilda/
\(^{55}\) js-scala, http://github.com/js-scala/js-scala
\(^{56}\) Sunroof, http://ku-fpg.github.io/software/sunroof/
languages and web frameworks like Meteor\textsuperscript{57}, StratifiedJS\textsuperscript{58}, and Opa\textsuperscript{59}; several isomorphic JavaScript libraries and frameworks\textsuperscript{60}; and a vast number of alternatives for programming client-side application logic in a programming language other than JavaScript\textsuperscript{61}. For an excellent insight into how to run your favorite programming language in web browsers, refer to Canou et al. (2012). Finally, the existence of all these different approaches suggests that JavaScript might become the target language, rather than the source programming language for the web at some point in the future.

### 3.6 Summary

This chapter provided a comprehensive, first-of-its-kind survey of the existing research literature on unified web application development (also called isomorphic, multi-tier, or tierless programming) published between 2006 and 2016. In this survey, the identified approaches (16) to unified web application development were classified into three categories: presentation-centric approaches, logic-centric approaches, and data-centric approaches. A thorough review and a comparative analysis of the approaches revealed their diversity, strengths, and weaknesses as well as that the current research focus is moving toward presentation-centric approaches based on JavaScript. In addition, there is a growing interest in real-time data synchronization and communication as well as reactive programming. In contrast, none of the approaches focused on leveraging the existing skills of a vast amount of non-expert programmers (e.g., web designers), let alone would provide the above-mentioned advanced features for them. The next chapter presents a unified web application development approach that addresses this research gap by providing a presentation-centric approach built upon declarative W3C standards.

\textsuperscript{57} Meteor, http://www.meteor.com/
\textsuperscript{58} StratifiedJS, http://onilabs.com/stratifiedjs
\textsuperscript{59} Opa, http://opalang.org/
\textsuperscript{60} Isomorphic JavaScript frameworks and libraries, http://isomorphic.net/libraries
\textsuperscript{61} List of languages that compile to JavaScript, http://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS
4 A Declarative Approach to Empower End-User Developers

This chapter gives a high-level description of the proposed XFormsDB approach to unified web application development, including its language design, system implementations, and their evaluations. The details of the approach and related research aspects are presented in Publications II-V.

4.1 The XFormsDB Approach in a Nutshell

This dissertation proposes the XFormsDB approach to empower intermediate level end-user developers (levels 4-6; cf. Section 2.3.1) in building modern web applications on their own. The idea is to employ a presentation-centric architectural approach (cf. Section 3.1) that seamlessly unifies the client-side (presentation) and server-side (logic and data) aspects of a web application under a single programming model. The unified programming model is based on declarative (cf. Section 2.2) markup languages\(^\text{62}\) that targeted end-user developers are comfortable using and already familiar with. Consequently, the need for imperative client-side scripting and server-side programming languages can be completely eliminated. XFormsDB also provides a supportive, web-based development environment for building, managing, and deploying XFormsDB web applications.

The following subsections provide additional details about the design principles behind the XFormsDB approach as well as a brief overview of the XFormsDB platform architecture.

\(^\text{62}\) Sebesta (2015) uses the term *markup/programming hybrid languages* to describe markup languages that include elements for specifying programming actions, such as control flow and computation.
4.1.1 Design Principles

The XFormsDB approach was architected with certain high-level design principles in mind. These design principles have much in common with the ones defined by the W3C in its 24-point list of design principles (Bos, 2003), including simplicity, learnability, use what is there, extensibility, modularity, interoperability, longevity, and device independency.

**Target Non-Expert Programmers.** Existing approaches to unified web application development are, as a rule, targeted at professional programmers, who possess solid technical skills in at least one general-purpose programming language (level 7). The goal of the XFormsDB approach is to reduce the amount of technical skills required so that end-user developers, such as web designers (levels 6) and domain experts (levels 4-5) can build modern web applications on their own.

**Facilitate Web Development with Appropriate Tools.** When it comes to web development tools, end-user developers have varying needs, skills, and preferences; thus, different end-user developers should be served by different tools (Rode, Howarth, et al., 2005). In the context of XFormsDB, web designers should be able to use their own and familiar HTML text editors for web development. A more diverse and technically less skilled group of domain experts in turn needs a dedicated visual editor that supports several levels of modification (MacLean et al., 1990).

**Leverage Declarative DSLs and Open Web Standards.** Considering the targeted end-user developers, declarative languages have many benefits over imperative ones (cf. Section 2.2). For example, they are easier to write programs with, reason about, and transform. Similarly, domain-specific languages (DSLs), which are usually small and declarative, fit better for non-expert programmers compared to general-purpose languages (GPLs) (Mernik et al., 2005). While adopting a DSL approach brings many benefits (e.g., abstractions and focused expressive power), it also poses some challenges; especially related to their design, implementation, and adoption (van Deursen et al., 2000). To overcome these challenges, the XFormsDB approach aims to leverage existing open web standards (van den Bleeken, 2013) as much as possible in its language design.

**Support Heterogeneous Clients and Data Sources.** Today, modern web applications are expected to run across a wide variety of web browsers and
devices without requiring any additional software installation. In addition, they often consume data in different formats from heterogeneous web data sources. To meet these demands in an efficient way, the implementation of the XFormsDB approach should automatically handle cross-browser inconsistencies (Barskar and Patidar, 2016) and data translations on behalf of end-user developers.

4.1.2 Platform Architecture

Figure 8 illustrates the XFormsDB platform architecture. As mentioned in the design principles section, the platform is primarily targeted at two types of users: web designers and domain experts. Web designers, who are more skilled in web development, can use their favorite HTML text editors to manually write multi-tier web applications in the XFormsDB language (cf. Section 4.2). The language covers common aspects (presentation, logic, data, communication) of a web application, which in turn follows the Model-View-ViewModel (MVVM) design pattern (Syromiatnikov and Weyns, 2014). For domain experts, the platform offers a web-based tool called XIDE (cf. Section 4.3.3), which supports the visual component-based development of XFormsDB web applications. Web designers can, of course, use XIDE as well. Indeed, web designers and other users with more advanced programming skills are encouraged to create reusable components in XIDE for others to consume. The distributed XFormsDB runtime system consists of client applications, the XFormsDB web application server (cf. Section 4.3.1), the UniQue proxy server (cf. Section 4.3.2), and heterogeneous data sources. The XFormsDB web application server hosts XFormsDB web applications, provides generic backend services for the applications, and transforms designed XFormsDB documents at runtime into client applications viewable by modern and legacy web browsers. The transformation also generates proper code for client-server communication. Currently, the platform supports data access to real XML data sources (i.e., XML stored in flat files or databases) directly and virtual XML data sources (e.g., JSON from a web API) via the UniQue web service. Support for additional data sources (e.g., relational databases) can be easily added, for example, by using the DataDirect XQuery\(^63\) middleware.

\(^{63}\) DataDirect XQuery, http://www.progress.com/xquery/xml
Figure 8: High-level overview of the XFormsDB platform architecture.
4.2 The XFormsDB Language

XFormsDB is a declarative domain-specific language (DSL) (van Deursen et al., 2000) that enables writing modern web applications in a unified manner. The language is based on XHTML (Mavrody, 2010; Hickson et al., 2014) and CSS (Atkins Jr. et al., 2017) – the two well-known and widely adopted open web standards by the W3C – whose vocabularies it enhances with linguistically integrated (Groenewegen et al., 2010) sublanguages to cover common aspects of a web application (cf. Figure 9). Consequently, the whole multi-tier web application can be defined within an XFormsDB document using a single paradigm (declarative markup) and data model (XML) (Bray et al., 2008), yet maintaining separation of concerns.

The XFormsDB language also provides high-level abstractions for many common web application development tasks, including DOM manipulation; cross-tier communication; data translations, querying, and synchronization; server programming; session management; and security. Hiding the underlying technical complexities from end-user developers eliminates the need for writing a lot of unnecessary glue code, allows the end-user developers to stay focused on application design, and increases their productivity.

Next, each of the sublanguages (excluding the well-known XHTML and CSS used for defining visual appearance) is briefly described, including the language extensions proposed in this dissertation.

Figure 9: Overview of the XFormsDB language and its sublanguages.
4.2.1 XForms for Reactive User Interfaces

XForms (Boyer, 2009), a W3C recommendation since October 2003, is an XML-based forms technology with Turing complete expressive power. It was originally designed to tackle the most common problems found in HTML forms, but, at present, is flexible enough to be used as a declarative, general-purpose client-side web application framework. Indeed, XForms cleanly separates the template-based user interface from the underlying data and logic by applying the MVVM design pattern in the frontend of a web application. The View (i.e., XForms User Interface) handles user interaction via device and modality independent user interface controls that are declaratively bound to ViewModel data with two-way data bindings. The ViewModel (i.e., XForms Model) in turn handles view data (i.e., XML instance data), data calculations and constraints using spreadsheet-like formulas, data manipulations, data retrieval and submissions, and interaction between data models and their corresponding presentation. Thus, when embedded into XHTML (structure) with CSS (styling), it becomes possible to create reactive web user interfaces with customized look and feel in a declarative manner. XForms also integrates seamlessly with other declarative W3C standards, including XPath (querying), XML Events (events), and XML Schema (validation). Furthermore, using XForms does not preclude the use of imperative JavaScript, but the two can coexist and interact within the same document (cf. Section 4.2.6).

4.2.2 XFormsRTC for Real-Time Communication

Publication III presents the requirements and design of a language extension called XFormsRTC for adding bidirectional real-time communication (RTC) capabilities to the XForms markup language. The language extension defines a high-level API that unifies the way XForms clients communicate with different types of RTC servers. The API supports connection management, sending and receiving data, and handling of related events. Using the API not only enables basic instant messaging between clients connected to the same RTC server but also allows clients to consume any number of other services offered by the server. For example, several RTC servers based on Extensible Messaging and Presence Protocol (XMPP) (Saint-Andre, 2005; Saint-Andre, 2011a; Saint-Andre, 2011b) support a binding for either HTTP (Paterson et al., 2014) or WebSocket (Stout et al., 2014), and come with a wide range of built-in services (e.g., publish-
subscribe and multi-user chat) to facilitate the development of real-time web applications.

4.2.3 XFormsDB for Server-Side Application Logic

Publication II presents the requirements and design of XFormsDB, an extension to the XForms markup language for defining common server-side and database-related functionalities within an XFormsDB document. The core of the language extension revolves around a mechanism, which allows clients to seamlessly call XFormsDB services that run on the server. Currently, XFormsDB offers eight different backend services, out of which authentication, database, and storage are the most prominent ones. By leveraging these generic backend services, end-user developers can stay focused on the presentation aspects (frontend) of a web application without worrying about their application's backend. Other features of the XFormsDB language extension include role-based access control and XML document inclusion. The former allows restricting access within an XFormsDB document to authorized users only, whereas the latter allows modularization of XFormsDB documents.

4.2.4 XPath or UQL for Data Access and Management

XForms uses XPath 1.0 (Clark and DeRose, 1999) – one of the core XML recommendations by the W3C – as an expression language for accessing local XML data (i.e., instance data). Similarly, XFormsDB uses XPath as a query language for accessing and managing remote XML data, whether real (cf. Publication II) or virtual (cf. Publication IV). While parameterized queries written in XPath are primarily used for retrieving data of interest, they can also be used for performing updates with support for conflict resolution; see Honkala et al. (2007) for a more detailed description.

Publication IV presents the requirements and design of Unified Query Language (UQL), which is a superset of CSS Selectors (Çelik et al., 2011) designed by the W3C. In essence, UQL combines Selectors and XPath into a single and flexible query language. Consequently, end-user developers can use the more familiar Selectors to access remote XML data, yet take full advantage of XPath and its function library when needed – all within a single expression. With the introduction of pluggable expression languages in XForms 2.0 (Boyer et al., 2012), UQL could even be used in XForms as
an alternative to XPath 1.0 or XPath 2.0 (van den Bleeken and Boyer, 2012) to access local XML data.

4.2.5 TL for Reusable Multi-Tier Components

Publication V introduces Template Language (TL), an XML-based language for supporting component-based web application development in XIDE (cf. Section 4.3.3). The language provides a means for defining and calling reusable XFormsDB components that encapsulate all the aforementioned aspects (reactive user interfaces, real-time communication, server-side application logic, data access) into a single package. In addition, XFormsDB components can be parameterized to increase their reusability even further. For a detailed description of the language design and its requirements, refer to (Laine, 2009).

4.2.6 Extensibility and Limitations

The XFormsDB language has been designed with extensibility in mind from the very beginning. On the client-side (presentation), XForms offers a wide variety of options for extensibility ranging from defining new data types, user interface controls, actions, and events to interoperating with JavaScript code (Dubinko, 2003). As for real-time communication, XFormsRTC allows using a special application-level subprotocol, such as XMPP over WebSocket (Stout et al., 2014). On the server-side (logic and data), the most elegant way to extend XFormsDB is by defining new XFormsDB backend services (cf. Section 4.2.3). Alternatively, custom server-side application logic and services can be implemented in Turing complete XQuery (Robie et al., 2017) and, optionally, expose them as RESTful APIs (Retter, 2012). XQuery, which is a superset of XPath, can also be used to overcome XPath and UQL’s (cf. Section 4.2.4) limitations in data querying and management, such as lack of sorting, grouping, aggregating, and joining data. For validating the structure and types of data to be collected and transferred, the same XML Schema (Fallside and Walmsley, 2004) definition can be used both on the client side (for responsiveness) and the server side (for security). A more in-depth description of these different extension methods is given in Publication II along with illustrative code examples.

An obvious drawback of each extension method is that they require additional learning effort from end-user developers; only incremental learning is required, though. In addition, the XFormsDB language assumes
knowledge of declarative W3C standards (namely, XHTML, CSS, XForms) plus the proposed language extensions, which may be a barrier for lower-level end-user developers (levels 4-5). Therefore, to widen the group of potential end-user developers utilizing the XFormsDB approach, a dedicated IDE (cf. Section 4.3.3) with special assistance features (e.g., syntax highlighting and component-based development) is called for.

4.3 System Implementations

This section gives a high-level description of three proof-of-concept software systems (XFormsDB, UniQue, XIDE) that, as a whole, implement the proposed XFormsDB approach and its language design. The source code of the system implementations is open sourced under the MIT license.

4.3.1 XFormsDB: A Web Application Development Framework

Publication II presents the requirements and implementation of a full-stack web application development framework called XFormsDB. The framework consists of a set of generic software components running on a web application server (XFormsDB) that enable end-user developers to build and host multi-tier web applications written in the XFormsDB language (cf. Section 4.2).

Figure 10 depicts the XFormsDB web application server. The server is based on Apache Tomcat and encompasses three main software components: an XForms processor (Orbeon Forms), an XFormsDB & XFormsRTC processor, and a native XML database (eXist-db (Meier, 2003)). As the component names indicate, Orbeon Forms' XForms processor implements the XForms language, whereas the XFormsDB & XFormsRTC processor implements the XFormsDB and XFormsRTC language extensions, including backend XFormsDB services. eXist-db, which runs as a web application, implements the XPath and XQuery query languages on top of its native XML database. All three server-side components are mainly written in Java/Scala. In addition, Orbeon Forms

64 Apache Tomcat, http://tomcat.apache.org/
65 Orbeon Forms, http://www.orbeon.com/
66 eXist-db, http://exist-db.org/
67 From a software architecture point of view, XFormsRTC should have been implemented as part of the XForms processor as it extends XForms with client-side functionality only. The decision to implement it as part of the XFormsDB processor was simply made to eliminate/minimize the dependence on any individual XForms processor.
includes a client-side runtime library written in JavaScript. The server itself can be installed on various platforms like Mac, Linux, and Windows.

![Diagram of XFormsDB web application server components](image)

**Figure 10**: Main software components of the XFormsDB web application development server.

When a client requests an XFormsDB document from the server, the document goes through a series of server-side transformations before being sent to the client. Each processor performs its own transformation(s) to the document and embedded queries – one after another – to render the initial view for the target runtime environment. Currently, the processors can generate a fully functional client application for both modern (with JavaScript, Ajax, and WebSocket support) and legacy (without requiring any plugins) web browsers. At runtime, the generated client application may consume backend XFormsDB services (e.g., execute queries against XML data stored in eXist-db), query heterogeneous web data sources via UniQue (cf. Section 4.3.2), and communicate with third-party RTC servers.

### 4.3.2 UniQue: A Web Service for Querying Data on the Web

Publication IV presents the requirements and implementation of UniQue, a complementary web service to XFormsDB that enables unified and efficient querying of heterogeneous web data sources. The web service is exposed as a web API that completely hides the differences between the underlying query techniques and data formats used by different data sources. Moreover, the API supports several standard-based query languages (cf. XPath, UQL, UQL, UQL).

---

68 The current implementation supports communication with WebSocket servers only.
and XQuery in Sections 4.2.4 and 4.2.6) with gradually increasing expressiveness and complexity to allow retrieving only data of interest from a target data source. Consequently, end-user developers need to master only a single data format and query language; CORS (van Kesteren, 2014) issues can be avoided; XFormsDB web applications can consume data in formats other than just XML; and less data needs to be transferred over the network (cf. Publication IV for detailed performance measurements).

Figure 11 gives an overview of the UniQue web service that operates as a proxy server between client applications and data sources. The proxy server is implemented in XQuery on top of the 28.io\(^{69}\) platform, and it consists of three software components. *UniQue processor* is the main component that implements the service interface. The two other components, *Data converters* and *Query converters*, are dedicated utility modules for performing data and query unification processes, respectively. The former converts original JSON, CSV, HTML, or XML data into *friendly* (i.e., easy to consume and query) and *round-trippable* XML by following a set of mapping rules, whereas the latter translates a given query expression into XQuery. The UniQue processor handles all the rest, including reading input parameters, retrieving original data, executing translated queries against unified data, and returning the response, which includes only data of interest.

![Main software components of the UniQue proxy server.](image-url)

---

4.3.3 XIDE: An End-User Web Application Development Tool

Publication V describes the requirements and implementation of *XFormsDB IDE (XIDE)*, a web-based tool that assists level 4-5 end-user developers (e.g., domain experts) in building, managing, and deploying XFormsDB web applications. XIDE supports multiple levels of modification (cf. Section 2.3.1), including the visual composition of web applications from reusable components and direct source code editing of both web pages and components. What really distinguishes XIDE from other end-user web application development tools (Rode, Howarth, *et al.*, 2005) is the combination of its flexibility and features with a component-based (cf. Section 4.2.5) architecture founded on a unified, markup-based background technology (cf. Section 4.3.1). Indeed, XIDE gently leads end-user developers from one level to the next and smooths the transition from visual editing (levels 1-3) to direct source code editing (levels 4-7).

Figure 12 shows the main user interface, i.e., the page editing view of XIDE. The preview (or design view) of the web page currently being edited is shown in the top-left panel, the components used for building the web page are listed in the top-right panel, and the source code of the selected component (*Footer component* in this case) is displayed in the bottom panel. The source code editor supports advanced syntax highlighting and error checking. Moreover, changes made to the source code are immediately reflected in the preview (or design view) – in other words, the user interface supports *design-at-runtime* (Rode and Rosson, 2003; Rode, Bhardwaj, *et al.*, 2005), which speeds up the design-build-test cycle immensely.

The XIDE web application itself is written in Java using GWT (cf. Section 3.2.1). In addition, it uses the MySQL database for storing data about its users, web applications, and components; and the XFormsDB framework for processing XFormsDB web applications written by end-user developers.
4.4 Evaluation

This section evaluates the proposed XFormsDB approach from three different perspectives: XFormsDB as a software product, web application development with XFormsDB, and XIDE as an end-user development tool.

4.4.1 Software Product Quality

Publication I presents an assessment of three different approaches to unified web application development: XFormsDB (presentation-centric), GWT (logic-centric), and XQIB\(^\text{70}\) (data-centric). The assessment was based on the ISO/IEC 25010 standard (ISO/IEC, 2011), which defines a software product quality model composed of eight attributes: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. However, because the standard is not specifically designed to evaluate web applications or unified development approaches, two additional attributes were added: time to market (Offutt, 2007).

\(^{70}\) The publication refers to XQIB as Sausalito, named after the Sausalito XQuery web application server.
2002) and support. The latter includes documentation quality, development community activity, and development tool availability. For brevity and clarity, only the assessment results for XFormsDB are included below.

**Functional Suitability.** XFormsDB's functionality covers common aspects of a web application; end-user developers (e.g., web designers) have full control over the user interface, whereas server-side and database-related functionalities are limited by currently available XFormsDB backend services. Hence, XFormsDB is best suited for developing small and medium-sized web applications rather than enterprise-level applications with complex business logic needs (cf. Section 4.4.2).

**Performance Efficiency.** The response sizes and times of XFormsDB web applications are reasonable but not highly optimized (Laine, 2010). The small amount of network traffic generated by the UniQue proxy server represents the current state of the art in web querying (cf. Publication IV).

**Compatibility.** XFormsDB web applications can consume data in different formats and from various third-party data sources (cf. Section 4.3.2). Additionally, there is a possibility to extend the XFormsDB programming model so that applications can expose their data through RESTful APIs (cf. Section 4.2.6).

**Usability.** For end-user developers with little technical skills (e.g., domain experts), XFormsDB is easy to learn and use due to its visual component-based editor and the use of declarative markup languages (cf. Section 4.4.3). The adoption rate of XForms, however, is still quite low despite being a W3C recommendation.

**Reliability.** XFormsDB is an academic project. However, its reliability has been proven through numerous web applications over several years (cf. Section 4.4.2).

**Security.** XFormsDB provides built-in and easy to use security features for user authentication, authorization, and access control (cf. Section 4.2.3). These features along with server-side processing ensure that sensitive information is never exposed to unauthorized clients.

**Maintainability.** XFormsDB promotes modularity and reusability both at the level of its language design (cf. Section 4.2) and system
implementations (cf. Section 4.3). In addition, XFormsDB is an open-source project, so it can be easily modified and extended to meet specific needs.

**Portability.** XFormsDB SDK (including the XFormsDB web application server) can be downloaded and installed easily on different platforms (Mac, Linux, and Windows), whereas UniQue and XIDE run in the cloud (cf. Section 4.3). XFormsDB web applications are client device and browser independent (cf. Section 4.1.2).

**Time to Market.** XFormsDB allows rapid development of modern web applications, but the language requires some learning up front (cf. Usability).

**Support.** XFormsDB's documentation is good, but it mostly consists of scientific publications. XFormsDB also comes with a dedicated tool to assist end-user developers in their web development activities (cf. Usability).

### 4.4.2 Web Applications

Figure 13 showcases five web applications that were developed to validate the feasibility and effectiveness of the proposed XFormsDB approach. The web applications are described in detail below.

![Figure 13: XFormsDB web applications (from left to right): Climbing Stalker, XFormsRTC Chat, XFormsDB Blog, Wi-Fi Finder, and Weather.](image)

**Climber Stalker** is a mashup web application that provides a central place for climbers to stay updated with all the latest news from elite rock climbers. Its user interface comprises a single view, which is divided into five sections: about, ascents, competition, tweets, and media. The data for
the sections is fetched asynchronously from heterogeneous web data sources and services (eight in total) via the UniQue proxy server.

**XFormsRTC Chat** is a Messenger\(^{71}\) like instant messaging web application with a responsive user interface to support a wide range of devices, such as smartphones and tablets. For simplicity, all discussions within the application take place in a public chat room, where users join with a nickname. Thereafter, users communicate with each other by sending messages that are delivered in real-time via a WebSocket server.

**XFormsDB Blog** is a stripped-down version of the popular Blogger\(^{72}\) platform with two main user interfaces (public and administration), each having multiple views. It allows blog users to read published posts and related comments, leave their own comments, and browse through archives. For authorized blog administrators, the web application offers necessary tools to write new posts as well as to manage published posts and comments. All the data used by the web application is stored in eXist-db. A separate component-based version of the web application has also been developed with XIDE (cf. Publication III).

**Wi-Fi Finder** is a highly interactive web application widget for finding nearby Wi-Fi hotspots and delivering their access information to users through configuration SMS. The widget's user interface comprises two views: *main view* with an interactive map (Google Maps) and *user settings view* with both client-side and server-side data validation (XML Schema). In addition, there is a separate user interface with multiple views for authorized administrators to manage Wi-Fi hotspots. For storing the data, the widget uses two distinct locations; user settings are stored in eXist-db, whereas Wi-Fi hotspots are stored in the cloud. In the latter case, the data is accessed asynchronously via a REST API.

**Weather** is a simple web application widget for showing a 5-day weather forecast for a specific location. It has two views: *main view* showing a weather forecast and *user settings view* for personalizing the widget. The weather information is fetched asynchronously from Google Weather API, whereas user settings are stored in eXist-db.

---

\(^{71}\) Messenger, http://www.messenger.com/
^{72}\) Blogger, http://www.blogger.com/
Other notable XFormsDB web applications include Address Book, Project Management, and Mobile Ticketing. The two first ones are described in Publication II, whereas the description and analysis of the latter can be found in Publication III.

### 4.4.3 User Study

Publication V presents a qualitative user study that was conducted to evaluate the usability and effectiveness of the XIDE tool. Specifically, the aim of the study was to investigate whether level 4 end-user developers can perform higher level *editing* tasks without facing any major learning barriers.

**Participants.** The user study involved 9 participants (5 males and 4 females), aged between 22-31 years, with a mean age of 25 years. All participants were students studying Economics or Computer Science; none of them studied web development as a major. In addition, all participants belonged to the level 4 category of end-user developers; they were active Internet users and they had hands-on experience with a visual tool and an XML-based markup language. None of them had ever used the XIDE tool, or the XForms and XFormsDB markup languages.

**Procedure and Tasks.** Participants completed the study individually in a laboratory setting, with each session lasting between 1.5 and 2 hours. The session started with an interview, where participants were asked about their web application development experience. Next, participants were given a brief introduction to the XIDE tool, including the XFormsDB markup language. After the introduction, participants were asked to think aloud while performing a set of tasks (10 in total), which were designed to resemble the transition from visual editing (level 3) to unified language programming (level 7). First, participants used the visual editor for simple modifications (level 3). Then, they executed the same tasks by manually editing the source code (level 4). After that, participants were asked to modify the page source code (level 5). Next, they had to do first minor (level 5) and then major (levels 6-7) modifications to the components. Finally, participants had to contribute new components (levels 6-7). A researcher observed participants during the whole process and was allowed to ask qualifying questions to better understand the motivations and
decisions made by participants. At the end of the session, a closing interview with participants was conducted.

Results and Discussion. All participants were able to complete all tasks successfully, and thus smoothly transit from the level 4 to the levels 6-7 without facing any major learning barriers. In general, participants said they found the component and unified markup based approach implemented in XIDE promising and would use such a tool to create web applications that they are otherwise unable to build. Moreover, they found it useful and flexible, because it supported several levels of modification (gentle slope).

While performing editing tasks in XIDE, participants widely employed the copy-paste-tweak method. They found it intuitive to search for existing code with similar functionality, copy-and-paste it, and finally modify it according to the task. During the process, the immediate response from the system was highly appreciated as it helped participants to test their code changes very quickly. On the other hand, providing a reference to the XFormsDB vocabulary or supporting automatic code completion in XIDE may have changed participants’ approach to solving tasks.

Advanced syntax highlighting was cited by all participants as one of the most usable features in XIDE. It directed participants’ attention to code that most likely needed editing, while less important parts (e.g., data models) were dimmed with light gray color. Participants were generally able to read the code and guess the functionality of unknown tags based on their names. However, they found some of the tag and ID names not self-descriptive enough. Also, error descriptions from the system left room for improvement.

The XIDE user study provided useful information, but it also holds some threats to its validity. First, participants progressed gradually from one task to another and from one level to the next. In a real-world scenario, users might have problems in decomposing a large task into smaller ones. Second, participants were selected from a narrow group of university students who were about the same age.

4.5 Related Work

Open Web Standards. In recent years, a considerable amount of standardization work has been carried out toward making declarative web
application development more feasible. For example, HTML5 (Hickson et al., 2014) – the latest major revision of the HTML standard – includes many enhancements to web forms, such as new input types, validation constraint attributes, and form elements that eliminate (a great deal of) the need for imperative scripting related to client-side form data validation. Other efforts, such as Web Components\textsuperscript{73} enables component-based, declarative web user interface development by providing a means for defining custom, reusable HTML/DOM elements with encapsulated visual appearance and functionality. A more lightweight alternative to declaratively extending the HTML syntax, however, is to use HTML5 data-* attributes. These custom data attributes are particularly useful for embedding application/framework specific data or annotations into standard HTML elements, while still being valid HTML. The proposed XFormsDB language is especially well suited for declaratively building not only complex web forms and data-intensive applications with accessibility requirements (Pohja, 2010) but also entire multi-tier web applications. The component model in XFormsDB mainly differs from Web Components in that XFormsDB components cover multiple tiers and their definition/modification does not involve imperative scripting, namely JavaScript. Finally, XFormsDB extends XHTML with modular sublanguages separated into different namespaces in order to avoid possible naming collisions.

\textbf{HTML-Centric EUD Approaches.} <angular/> (Hevery and Abrons, 2009), Endev (Kis and Bogdan, 2016a; Kis and Bogdan, 2016b), and Mavo (Verou et al., 2016) are three closely related systems for building reactive CRUD web applications without the need to write any programming code or manage any backend servers. They all use static web pages as templates for their web applications, extend the HTML syntax with declarative annotations (i.e., custom attributes and expressions) to describe application functionality, process these annotations \textit{in the browser}, and bind user interface elements to underlying data stored remotely in the cloud. They also exhibit distinct features of their own to further facilitate the end-user

\textsuperscript{73} Web Components is an umbrella term for a set of specifications, namely Custom Elements (Denicola, 2016), Shadow DOM (Ito, 2017), HTML Imports (Glazkov and Morrita, 2016), and HTML Templates (Glazkov et al., 2014).
development of web applications. For example, `<angular/>`\(^{74}\) uses its declarative annotations, i.e., `<angular/>` directives, to describe the application visual widgets, lightweight computation, form validation, data persistence (two-way data binding) with a proprietary cloud storage, and URL encoding strategy. Endev\(^{75}\) annotations, in turn, allow keeping HTML elements automatically in synchronization (three-way data binding) with the application data stored either locally in the browser or remotely in the cloud. The annotations also support querying heterogeneous web data and conditionally showing and hiding user interface elements. Lastly, Mavo\(^{76}\) annotations describe how to load, transform, manage, and store data; various data formats and storage options are supported out of the box. On top of this, Mavo provides a spreadsheet-like expression language called MavoScript, supports WYSIWYG-style editing of hierarchical, semantically enriched data right in the website, and offers a plugin architecture for extensibility. In contrast to `<angular/>`, Endev, and Mavo, XFormsDB purposefully takes a somewhat different approach to web application development. First, XFormsDB keeps and runs certain parts of the application on the server for security and performance reasons as well as to support heterogeneous clients, including those without JavaScript. Second, the XFormsDB language is designed to cover all common aspects of a web application and is extensible beyond client-side functionality. Third, XFormsDB is not limited to textual development; web applications can also be built, managed, and deployed visually using XIDE directly in the browser.

**Spreadsheet-Centric EUD Approaches.** Quilt\(^{77}\) (Benson et al., 2014) is a recent system for building simple reactive CRUD web applications in a fully declarative manner. The central idea behind Quilt is to connect (three-way data binding) HTML pages to spreadsheets (Google Sheets\(^{78}\)) by placing spreadsheet references inside custom HTML attributes. The advantages of this approach are three-fold: (i) create fully customizable client user interfaces using HTML and CSS, (ii) visually manage data stored in the cloud/spreadsheets, and (iii) perform server-side data computations

\(^{74}\) `<angular/>` has evolved to what is now known as Angular. AngularFire, in turn, binds Angular with Firebase. An archived version of the `<angular/>` website is available at http://web.archive.org/web/20091002201638/getangular.com.

\(^{75}\) Endev, http://www.endevjs.org/

\(^{76}\) Mavo, http://mavo.io/

\(^{77}\) Quilt has evolved to Cloudstitch, http://cloudstitch.com/

\(^{78}\) Google Sheets, http://www.google.com/sheets/about/
using widely adopted spreadsheet formulas. The same idea is further developed in Object Spreadsheets\textsuperscript{79} (McCutchen et al., 2016), which overcomes some of the limitations present in Quilt. First, Object Spreadsheets provides an enhanced spreadsheet tool that supports defining and editing hierarchical, structured data as well as a formula language suited to the data model. Second, Object Spreadsheets integrates with Mavo\textsuperscript{80} to allow describing more advanced client user interfaces. Different from Quilt and Object Spreadsheets, the web application development model in XFormsDB is unified, i.e., it is not split into two parts: HTML client user interfaces and backend spreadsheets. XFormsDB also supports working with heterogeneous data formats and data sources (not only spreadsheets data), including hierarchical, structured data.

Gneiss\textsuperscript{81} (Chang and Myers, 2014; Chang and Myers, 2017) is a live programming environment that extends the spreadsheet metaphor to support rapid prototyping of interactive mashup web applications. The environment consists of a web data extractor, an enhanced spreadsheet editor to work with the extracted hierarchical data, and a visual web interface builder that connects GUI elements with the spreadsheet data using spreadsheet references (two-way data binding). Spreadsheet references are also used to bind parameters of a web API call to values in spreadsheet cells. The main drawback for Gneiss compared to XIDE is that the possibilities in user interface design are rather limited due to a fixed set of available GUI elements and hidden user interface source code. Moreover, Gneiss only supports web data sources returning JSON data; consuming other data formats and storing user-owned data is not possible.

Other EUD Approaches. INTER-Mediator\textsuperscript{82} (Nii et al., 2015; Nii et al., 2016) is a web application development framework that binds (three-way) web pages to relational databases using declarative descriptions. To establish such bindings, one needs to extend the HTML mockup (Benson, 2013) with custom data attributes, describe database access information and related logic in a PHP file, and set up the database with a valid schema. Unlike INTER-Mediator, XFormsDB does not necessitate the use of a pre-defined database schema and its application development can take place in a single file using extended markup syntax.

\textsuperscript{79} Object Spreadsheets, http://sdg.csail.mit.edu/projects/objsheets/
\textsuperscript{80} As of writing, the Mavo integration is still a work in progress.
\textsuperscript{81} Gneiss, http://www.cs.cmu.edu/~shihpinc/gneiss.html
\textsuperscript{82} INTER-Mediator, http://inter-mediator.com/
4.6 Summary

This chapter presented a novel presentation-centric approach, called XFormsDB, to unified web application development that empowers intermediate level end-user developers in building modern web applications. The proposed approach leverages end-user developers' existing skills in W3C-standardized declarative languages (XHTML and CSS), whose expressiveness it seamlessly extends with additional functionalities to cover common aspects (presentation, logic, data, communication) of a web application. The approach and related language extensions were implemented both as an installable framework and web-based tools to support end-user developers with varying needs, skills, and preferences. Finally, the applicability, effectiveness, and usability of the system implementations were evaluated against a standard software quality model, with web applications, and in laboratory experiments; both from technical and human perspectives.
5 Conclusions

The research presented in this dissertation studied how to unify web application development and, at the same time, leverage end-user developers' existing skills in declarative languages, namely (X)HTML and CSS. In particular, the study focused on exploring approaches that primarily allow creating and editing today's modern web applications textually (rather than visually). In this final chapter, the main results from previous chapters are summarized and conclusions are drawn by answering the research questions (RQ1-RQ3) formulated in Section 1.2. The chapter concludes with possible directions for future work.

5.1 Answers to the Research Questions

RQ1: What architectural approaches exist to unified web application development?

In Chapter 3, the existing research literature on approaches to unified web application development was surveyed. A total of 16 relevant approaches were identified, classified into three categories based on their area of focus (Publication I), and reviewed in detail with respect to a set of evaluation criteria. Table 3, which is extended with the proposed XFormsDB approach (cf. Chapter 4) for comparison, summarizes the results of the survey. Based on these results and conducted analysis, conclusions and discussion about the current state and future prospects of unified web application development were drawn. The main observation was that while all of the 16 reviewed approaches can streamline conventional web application development, their level of complexity is still more suitable for professional programmers (tier-specific experts) rather than end-user developers.
Table 3: Comparison of the proposed XFormsDB approach with the other approaches reviewed in this dissertation.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Language</th>
<th>Paradigm</th>
<th>Abstraction</th>
<th>Communication</th>
<th>Tool Support</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Presentation-centric</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Links</td>
<td>Links (new)</td>
<td>Declarative</td>
<td>Fat language</td>
<td>Pull</td>
<td>LODE</td>
<td>Cooper et al. (2007)(^a)</td>
</tr>
<tr>
<td>XFormsDB</td>
<td>XForms</td>
<td>Declarative</td>
<td>DSLs</td>
<td>Pull &amp; Push</td>
<td>XIDE</td>
<td>Laine et al. (2012)</td>
</tr>
<tr>
<td>SWAC</td>
<td>JavaScript</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull(^b)</td>
<td>–</td>
<td>Ast et al. (2013)</td>
</tr>
<tr>
<td>STiP.js</td>
<td>JavaScript</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Philips et al. (2014)</td>
</tr>
<tr>
<td><strong>Logic-centric</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GWT</td>
<td>Java</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull(^b)</td>
<td>GPE</td>
<td>Google (2006)</td>
</tr>
<tr>
<td>Hop</td>
<td>Scheme(^c)</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>Debugger</td>
<td>Serrano et al. (2006)</td>
</tr>
<tr>
<td>F# Web Tools(^d)</td>
<td>F#</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull</td>
<td>–</td>
<td>Petiček and Syme (2007)</td>
</tr>
<tr>
<td>WebDSL</td>
<td>WebDSL (new)</td>
<td>Declarative</td>
<td>DSLs</td>
<td>Pull</td>
<td>WPE</td>
<td>Visser (2008)(^a)</td>
</tr>
<tr>
<td>ML5</td>
<td>SML</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Murphy VII et al. (2008)</td>
</tr>
<tr>
<td>Volta</td>
<td>C++ / C# / ...</td>
<td>Imperative</td>
<td>Thin language</td>
<td>Pull</td>
<td>VPVS</td>
<td>Manolescu et al. (2008)</td>
</tr>
<tr>
<td>Ocsigen</td>
<td>OCaml</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Balat et al. (2009)</td>
</tr>
<tr>
<td>Ur/Web</td>
<td>Ur (new)</td>
<td>Declarative</td>
<td>Fat language</td>
<td>Pull &amp; Push</td>
<td>–</td>
<td>Chlipala (2015b)</td>
</tr>
<tr>
<td><strong>Data-centric</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XQIB</td>
<td>XQuery</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull</td>
<td>–</td>
<td>Fourny et al. (2009)</td>
</tr>
<tr>
<td>FORWARD</td>
<td>SQL</td>
<td>Declarative</td>
<td>Thin language</td>
<td>Pull</td>
<td>FODE</td>
<td>Fu et al. (2010)</td>
</tr>
</tbody>
</table>

\(^a\) Revised paper; the original publication appeared one year earlier.

\(^b\) Support for server push via third-party libraries; lacks built-in support.

\(^c\) Since version 3.x, Hop uses JavaScript (presentation-centric) as its base programming language.

\(^d\) WebSharper, the successor of F# Web Tools, supports C# as an alternative programming language, server push communication, and several IDEs.
RQ2: How to extend declarative (X)HTML and CSS to cover common aspects of a web application?

Chapter 4 presented XFormsDB, a high-level domain-specific language and a framework, that enables writing diverse types of modern web applications in a unified and declarative manner. The experimental results with XFormsDB demonstrated that the familiar syntax of (X)HTML and CSS can be seamlessly extended far beyond the current state of practice to include the following functionalities: reactive user interfaces (XForms, W3C), real-time communication (XFormsRTC, Publication III), server-side application logic (XFormsDB, Publication II), data access and management (XPath, W3C or UQL, Publication IV), and reusable multi-tier components (TL, Publication V). By combining all these innovations together, the XFormsDB language covers all common aspects of a web application (cf. Figure 9) and can benefit especially web designers in prototyping and becoming full-stack web developers. However, as the review of related work in Section 4.5 revealed, recent advancements in web technologies such as HTML5 have paved the way for novel end-user development approaches. Like XFormsDB, these approaches declaratively extend the (X)HTML syntax with custom functionality (mainly related to data binding), making them a viable option to consider in certain cases.

RQ3: How to gently expand end-user developers' web application development possibilities?

Chapter 4 presented XIDE, an end-user web application development tool that helps end-user developers in building modern web applications, and in particular, transitioning from visual editing to direct source code editing. The user study conducted with XIDE showed that combining a component-based visual tool with a unified, markup-based background technology (XFormsDB) can gently lead intermediate level end-user developers (e.g., domain experts) from one level to the next (cf. Section 2.3.1 as well as Publications III and V), and thus allow them to perform higher level editing tasks without facing any major learning barriers. Therefore, the study results and design principles can be considered as guidelines for future tools targeted at intermediate level end-user developers.
5.2 Future Work

Based on the results presented in this dissertation and in response to the latest research and industry trends in web application development, some interesting directions for further research can be suggested.

In recent years, there has been an increasing interest in improving the performance of web applications. New technologies (e.g., WebAssembly\(^{83}\)), JavaScript APIs (e.g., Service Workers\(^{84}\)), patterns (e.g., PRPL\(^{85}\)), architectures (e.g., Progressive Web Apps, PWAs\(^{86}\)), and frameworks (e.g., Preact\(^{87}\)) are constantly being developed to deliver more reliable, faster, and engaging user experiences on the web. Investigating how unified web application development could help programmers to transparently adopt these and future techniques as part of their web applications is one research topic of interest. The experiences gained from Hilda (cf. Section 3.5) suggest that high-level declarative languages, in particular, could provide a good starting point for this research.

In addition to addressing current limitations of XFormsDB, which were discussed throughout Chapter 4, there are several potential areas for further improvement. For example, the review of related end-user web application development approaches in Section 4.5 identified some emerging trends and features that are missing in XFormsDB, such as support for local storage and advanced (three-way and four-way) data bindings. Perhaps the most interesting trend among them, however, is the movement toward serverless development model and utilization of existing cloud services. Therefore, further research should focus on exploring how to outsource XFormsDB backend services, or in general, how to integrate HTML5 or XForms with popular cloud services. Is it possible to design common, high-level declarative abstractions over popular cloud services and their providers?

Currently, XIDE lacks many functional and usability features that are common in popular IDEs and end-user development tools. Therefore, further development efforts should investigate the potential of incorporating XIDE innovations into existing development environments via plugins to have the best of both worlds.

---

83 WebAssembly, http://webassembly.org/
84 Service Workers, http://www.w3.org/TR/service-workers-1/
87 Preact, http://preactjs.com/
References


Cooper, E. (2009). The Script-Writer’s Dream: How to Write Great SQL in your own Language, and be sure it will Succeed. In Gardner, P. and
Geerts, F. (eds.) *Database Programming Languages (DBPL '09)*, LNCS 5708, pp. 36–51. Springer.


Guzdial, M. (2016). End-User Programmers are at Least Half of All Programmers. Available at: http://computinged.wordpress.com/2016/01/20/end-user-programmers-are-at-least-half-of-all-programmers/.


References

Generative Programming and Component Engineering (GPCE ’10), pp. 127–136. ACM.


Unified web application development refers to architectural approaches that enable the development of multi-tier web applications in a single programming language, and potentially as a single artifact. Consequently, the complexity, time, and cost of web application development and maintenance can be significantly reduced. Additionally, these novel web application architectures allow tier-specific experts (professional programmers) to write an entire multi-tier web application on their own. This dissertation provides a comprehensive survey and comparative analysis of approaches to unified web application development; includes a complementary review of approaches to end-user development of web applications; and presents a novel declarative approach and systems that can benefit end-user developers, especially web designers, in prototyping and becoming full-stack web developers.