

Developing an Enterprise Web Application in XQuery

Martin Kaufmann and Donald Kossmann

ETH Zürich, Systems Group,
Universitätsstrasse 6, 8092 Zürich, Switzerland
{martinka,donaldk}@ethz.ch

Abstract. XQuery is a declarative programming language which can be used to express queries and transformations of XML data. The goal of this paper is to explore the expressive power of XQuery as a general-purpose programming language. To this end, this paper describes how to build an entire enterprise web application in XQuery. It is shown that it is actually possible and quite effective to implement a web application entirely in XQuery and that there are several advantages in doing so. The resulting code has proven to be very concise and elegant. More importantly, the use of XQuery simplifies the overall application architecture and improves flexibility.

Keywords: XQuery, XML, Enterprise Application, Web Application, Programming.

1 Introduction

Today, enterprise web applications play an important role for e-business software and data integration. Companies have demand for software which is well designed and structured to allow short training periods for new developers. The software should be easy to maintain by separation of concerns and extensible for future scaling.

In order to develop large-scale enterprise web applications, the state-of-the-art is to use frameworks such as the Java Enterprise Edition (J2EE) or Microsoft's .Net. These frameworks take advantage of the vast experience of object-oriented software development with programming languages like Java and C#. Using J2EE, for instance, Java objects can be mapped to a relational database using an object/relational persistence and query service like Hibernate. In the .Net framework LINQ [5] can be used in order to embed SQL-style database access into the object-oriented application code.

Yet, both J2EE and .Net suffer from the impedance mismatch between different data models used at different application tiers and from repeated work for the same tasks at different application tiers (e.g., error handling, etc.). The core of this problem stems from the fact that all these frameworks actually make use of a mix of technologies.

A result of this observation was the development of object-oriented database systems. They mitigate the impedance mismatch by providing a unified data model but introduce additional problems like verbose code for implementing complex queries by means of the programming language (e.g. SODA API for DB4o [3]).

This paper explores a new approach to develop enterprise web applications. The key idea is to leverage the (unified) technology stack developed by the World Wide Web Consortium (W3C). We argue that a unified technology stack can result in simpler, more flexible, and more efficient application code. A central piece of this new approach to build web applications is the use of the XQuery programming language [9]. XQuery was initially designed to query and transform XML data. The main contribution of this paper is to demonstrate that such a uniform technology stack based entirely on W3C standards can be used in order to build and deploy large-scale enterprise web applications.

The remainder of this paper is organized as follows: Section 2 describes a system architecture of a web application written entirely in XQuery. Section 3 sketches the implementation of our example application called PubZone [7], which is a web based repository for scientific publications. Section 4 describes our experience in the development of PubZone with XQuery. Section 5 discusses related work. Section 6 contains conclusions and avenues for future work.

2 Architecture

For this work, we adopt the traditional three-tier architecture for enterprise web applications. As shown in Figure 1(a), this three-tier architecture is typically implemented using different languages and data models at each tier. According to Figure 1(b), we propose to use, as much as possible, one programming language and one data model throughout the entire application stack.

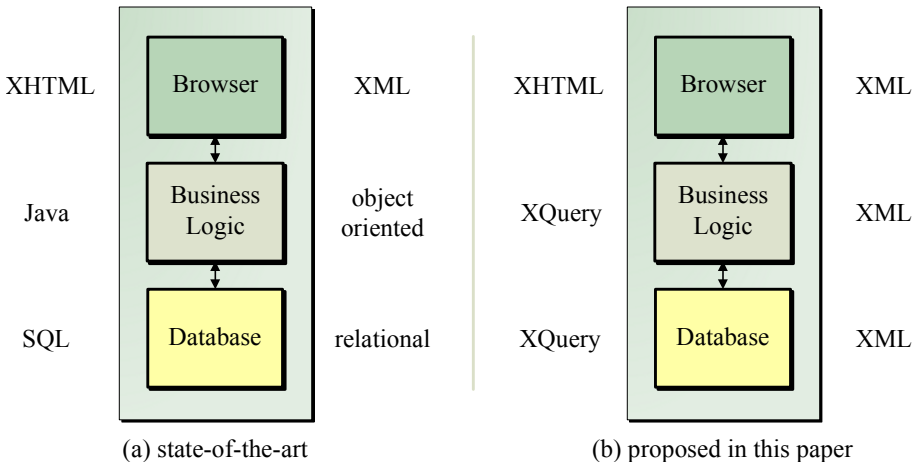


Fig. 1. Three-tier Application Architecture

The advantage of this approach is that the architecture becomes more flexible, more efficient, and simpler. The architecture of Figure 1(b) is more flexible because it allows for moving code from one tier to another. Simplicity and efficiency is achieved by applying the same programming language to all three application tiers.

3 Implementation

For the implementation and deployment of the PubZone application, we used the Sausalito XQuery application server [8] which integrates an XML database system, the Zorba XQuery engine [10], and the Apache Web Server into a single XQuery application server. In a nutshell, the XQuery application server maps a URL to an XQuery source file (called module) and an XQuery function. The return value of the function is sent back to the client as a result.

According to best practices, we adopt the MVC pattern in order to separate business logic from program control flow and presentation layer. The *Controller* function is the entry point which is called by a defined URL. This function receives an XML input generated from the HTTP request of the web client and carries out checks on the input. If these tests succeed the business logic of the *Model* is called. The result of the business logic is forwarded by the *Controller* to the *View* function which renders the output. [Code 1] shows an example of a *View* function. The result of the *View* is returned to the client by the *Controller*.

Code 1: Module of the *View* to add a new user

```
declare function def:inputForm () {
  let $text := <form method="post" action="/userNew/submit">
    Username: {form:text("uid", 25)}    Group:
    <select name="groupId" size="1">{form:option("", "select...")}
    {for $group in groups:listAllGroups()
     return form:option($group/@id, $group/name)}
    </select><input type="submit" value="Save " /></form>
  return navigation:showPage("Add user", $text)
};
```

For a large enterprise web application it is important to identify reusable components. This code can be implemented as separated XQuery modules that can be used by many functions of the application.

4 Discussion

A considerable amount of time was required to develop standard libraries like modules for pre-filled HTML form elements. When these reusable components had been finished, the development of web applications turned out to be quite effective and fast. Due to clean design and encapsulation of business logic in XQuery modules, this software architecture is expected to scale up for large applications.

Our measurements in terms of runtime performance shows that the XQuery implementation of our example application [7] in average currently is only about 35% slower than the corresponding Java implementation [6]. This result is basically caused by the fact that the XQuery application server [8] is still in a beta state. There is no principal limitation which prevents an application written in XQuery from reaching performance results achieved by state-of-the art technologies like J2EE.

5 Related Work

There has been a great deal of work on XQuery already. XQuery has been implemented by all major database vendors. Furthermore, XQuery is a popular tool for various tasks in the middleware; for instance, BEA's Enterprise Information Bus is based on XQuery and BEA's ALDSP server for enterprise information integration is based on XQuery [1]. XQuery is also used for data stream processing such as RSS streams [2], [4].

6 Conclusion

This paper reported on our experience to develop a complex and customizable enterprise web application entirely using the XQuery programming language and using related W3C recommendations (e.g., XML as a data format, XML Schema to describe data types, and HTTP and REST for remote communication). Overall, the conclusion is that the W3C family of standards is very well suited for this task and has important advantages over the state-of-the-art (e.g., J2EE, .Net, or PHP). Most importantly, using XQuery and W3C standards only ensures a uniform technology stack and avoids the technology jungle of mixing different technologies and data models. As a result, the application architecture becomes more flexible, simpler, and potentially more efficient. Today, the biggest concern in adopting this approach is that there are no mature application servers available, but we believe that the situation will change soon in this regard.

Obviously, this work was only a first step in order to develop complex and large-scale enterprise web applications entirely in XQuery with the help of W3C standards. In the future, more experience with other applications is needed. One important avenue for future work is to introduce a methodology to develop such applications with XQuery.

References

1. BEA AquaLogic Data Services Platform, <http://edocs.bea.com/aldsp/docs30/>
2. Botan, I., Fischer, P., Florescu, D., et al.: Extending XQuery with Window Functions. In: VLDB 2007, Vienna, Austria (2007)
3. Native Java & .NET Open Source Object Database, <http://www.db4o.com/>
4. Koch, C., Scherzinger, S., Schweikardt, N., Stegmaier, B.: FluXQuery: An Optimizing XQuery Processor for Streaming XML Data. In: VLDB 2004 (2004)
5. LINQ Project, <http://msdn.microsoft.com/en-us/netframework/aa904594.aspx>
6. Kaufmann, M., et al.: PubZone Java implementation, <http://java.pubzone.org/>
7. Kaufmann, M.: PubZone XQuery implementation, <http://xquery.pubzone.org/>
8. Sausalito, XQuery Application Server, <http://sausalito.28msec.com/>
9. Chamberlin, D., et al.: XQuery 1.1, W3C Working Draft, <http://www.w3.org/TR/xquery-11/>
10. The Zorba XQuery Processor, <http://www.zorba-xquery.com/>