University of Konstanz
Department of Computer and Information Science

Master Thesis for the degree
Master of Science (M.Sc.) in Information Engineering

Implementing Web Applications Using XQuery

XML from Front to Back

Michael Seiferle

01/596485

Konstanz, March 16, 2012

1st Referee:	Prof. Dr. Marc H. Scholl
2nd Referee:	Prof. Dr. Marcel Waldvogel
Supervisors:	Alexander Holupirek & Dr. Christian Grün

Zusammenfassung (Deutsch) Entwickler sehen sich heute immer häufiger und mit einer immer grösser werdenden Anzahl an XML-Daten konfrontiert. Es liegt auf der Hand, dass diese mit dafür konzipierten Datenbanksystemen verwaltet werden. Innerhalb eines modernen XML-Datenbank-Management-Systems (XML-DBMS) kann XML effizient gespeichert werden und es stehen domänenspezifische Sprachen, wie XQuery, zur Weiterverarbeitung zur Verfügung. Gleichwohl wird die Konzentration auf die Datenhaltung allein den Herausforderung nicht gerecht; Informationen in Anwendungen „zum Leben zu erwecken" ist mindestens genauso wichtig. Die vorliegende Arbeit untersucht Chancen und Probleme der Webapplikationsentwicklung in einer rein auf XML-Technologie beruhenden Systemarchitektur.

Wir beschäftigen uns mit der Frage, ob sich die Entwicklung grundlegend vereinfachen lässt, indem man konzeptuellen Ballast, den moderne Web-Frameworks wie Ruby on Rails oder CakePHP mit sich bringen, in weiten Teilen obsolet macht. Der reine XML-Stack bringt hierzu die notwendigen Voraussetzungen mit: einheitliche Programmierparadigmen und die durchgängige Verwendung eines Datenmodells in allen Schichten der Applikation.

Zur Klärung der Frage beschreiben wir die Konzeption von BaseX Web, einem XQuery-getriebenen Anwendungsserver. BaseX Web ist eine auf der XML-Datenbank BaseX basierende Technologiestudie, die es erlaubt Web-Applikationen allein unter Verwendung von W3C Standards zu realisieren. Ergebnis ist ein leichtgewichtiges Anwendungsframework mit dessen Hilfe Expertensysteme, wie beispielsweise ein Online Public Access Catalogue (OPAC), implementiert werden können. Eine abschließende Evaluation, basierend auf echten Daten der Bibliothek Universität Konstanz, zeigt die positiven Ergebnisse unserer Studie.

Abstract (English) With the ever growing amount of XML encoded data readily available in many application domains, the need to efficiently store, process & query these data has become evident. Yet, managing these bits of information is only half the story; bringing data to life by means of deploying data centric applications is just as important. This thesis investigates chances and challenges of deploying and implementing Web Applications in a pure XML technology stack, based exclusively on W3C standards.

With this thesis, we claim that application development may be fundamentally simplified by removing the conceptual *baggage* introduced with popular, modern frameworks such as *Ruby on Rails* or *CakePHP*. Inside the pure XML technology stack developers, are neither faced with differing programming paradigms nor will they have to convert their data back and forth between multiple application layers.

We support this claim by presenting BaseX Web, a proof-of-concept application server. Based on the BaseX XML database and XQuery processor, we describe the implementation and components of a lightweight application framework. The thesis concludes with the evaluation of a proof of concept: a library catalog retrieval system, based on real-world data of the Library of University of Konstanz.

Contents

1 Introduction

Web application development has undergone serious paradigm shifts, coming from hyper-linked bits of information, as proposed by Tim Berners-Lee in 1990 [5], to fully fledged applications running inside a user's browser, ideally indistinguishable from desktop applications. In the beginning said bits of information were mainly static and scientific data. This changed significantly in the mid 90s—when public interest in the internet, thanks to the world wide web, rose. The community soon realized that instead of serving only static documents, they could as well use scripts or programs that dynamically generate HTML content and deliver it to the client.

This set way for the *common gateway interface*[1], a de facto standard that allows web servers interfere with external applications. Perl and PHP are particular popular scripting languages in this context, although more recently Python and Ruby joined the company.

In the early days, web applications usually have been large, monolithic systems that were neither easy to maintain nor easy to extend. Jazayeri in [19, p. 3] compares these early days of web development to the evolution software engineering made in the late 1960s, yet at a much greater pace. Along with HTML's content model of intermixing data and layout, the inter-weaved scripting languages—not yet forcing a separation of data and business logic—added another *layer of confusion*.

Soon after that, the Model View Controller Pattern (MVC), first described by Trygve Reenskaug [29], was *rediscovered* for web application development and has been extremely successful to date.

In a nutshell, an MVC application is divided in interchangeable parts, consisting of:

Models representing knowledge and encapsulating data access
Views acting as a visual representation of the model
Controllers being the link between the user and the application

[1] `http://www.ietf.org/rfc/rfc3875`

Details on MVC in web applications and state-of-the-art implementations, as well as the application of the MVC pattern in the context of XQuery web application development, will be covered in more depth in the following chapters.

Likewise, interactive HTML applications also gained momentum with the introduction of Ajax in 2005 [14]. This equipped application developers with new concepts to create an even more interactive experience on the web.

Indubitably, developing web applications has not only become more streamlined, but also matured to a much more scalable and maintainable process. However, developers still face a multitude of involved technologies, ranging from HTML, client-side scripting, server side scripting to database query languages. These are the concepts that web application frameworks try to hide from the developers.

Therefore the recent rise of specialized NoSQL data stores not only pushed another technology on the stack, but even more so showed that developers tried to overcome certain flaws, in both performance and flexibility, of traditional relational databases.

Emerging from a database context, the main contribution of this work is to provide implementers and software architects with a development stack, consisting of a lightweight application server in company with a software framework written in XQuery, which is built with XML technologies from front to back. Suggested approach not only takes away the need to master a plenitude of technologies, but also tackles some of the shortcomings relational database management systems face.

2 Concepts: Web Application Frameworks

Web application frameworks usually try to make developers' lives easier by providing functionality commonly needed in developing web applications [31, p.2]. These building blocks assist developers in creating dynamic websites, web services and web applications.

Besides this rather weak definition, there is no common agreement on what a web application framework actually is, but in the course of time, some common features evolved [25] that are widely accepted to play a key role in the web application framework architecture.

Data Persistence

Virtually all applications today need to persistently store and process data. In all but the most trivial applications, pages are generated on request, based on server side stored data. In addition users are often required to change this data in order to administer their applications. These requirements already led to concepts that provide developers with:

> **An API** to access the data storage
> **An Object-Relational-Mapper** to simplify storage and retrieval, this if often combined with an
> **A SQL Builder** that provides developers with an interface that simplifies query generation
>
> Keith et al. in [22]

Security

As web applications often offer services tailored for specific users it is crucial to identify these users and ask them for their credentials, if the resource they are about to request requires authentication. The framework also has to persist their session and may keep track of user groups or roles the logged in users belong to.

Caching

To improve performance many applications cache resources that are either expensive to generate or unlikely to change much over time. As an example, consider navigations: even on dynamically generated web pages they usually remain very stable, so there is no need to regenerate them upon each request.

Templating

It is usually perceived good practice to separate data and representation. In [13] this is even said to be "[…] one of the most fundamental heuristics of good software design". Therefore most web application frameworks offer a templating engine that provides a consistent interface to build HTML (or any presentation language) blueprints with developer defined fields that are subsequently filled with data. Templating engines range from simple logic-less templates[1] to very sophisticated builders[2] that define a DSL[3] themselves and may even contain control structures.

Scaffolding

Scaffolding may be understood as a special case of templating. Scaffolded components provide the user with generated, ready-to-use implementations of CRUD[4] functionality. This is usually done via introspection and allows application developers to gradually finish the application based on the automatically generated code. Scaffolding has been strongly promoted by Ruby on Rails and was since then adopted by many frameworks.

⋆ ⋆ ⋆

Using a web application framework makes development and implementation a faster and more robust process, as the used framework predefines the functionality and terminology. The framework ideally fosters code reuse, thus clearly following the principle of not repeat-

[1] i.e., {{ Mustache }}, `http://mustache.github.com/`
[2] i.e., Smarty, `http://www.smarty.net/`
[3] Domain Specific Language, a language tailored to solve a very specific set of problems
[4] named after: `CREATE, READ, UPDATE, DELETE`, characterizing the database interaction found in database driven applications

ing oneself, while at the same time decoupling interdependencies of application components.

Most of the general-purpose frameworks implement the MVC architectural pattern. The following overview will concentrate on general concepts of MVC and later on cover some state-of-the-art web application frameworks, tackling different application domains each.

2.1 Model-View-Controller

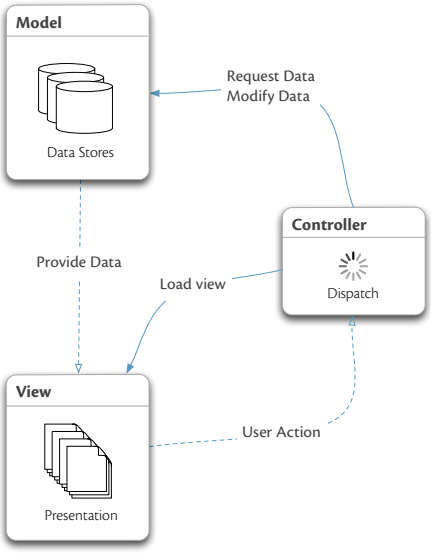

FIGURE 2.1: Model-View-Controller Overview

MVC is a well-known and generic software pattern. It splits code into three distinct, loosely coupled components, which interact with each other using well-defined interfaces. The pattern is used to make program design more flexible and extensible: features can be easily changed and enhanced without endangering the already existing functionality. This idea is based on one basic observation: while user interfaces frequently change and need to be deployed in manifold areas (examples are different platforms and types of applications, web applications, mobile apps & desktop applications), the domain logic of an application is usually much more stable. Using MVC even encourages collaboration in software projects,

as one developer might start implementing views and presentational logic while another one copes with data abstraction and domain logic.

2.1.1 The Model

The model is about all data-related issues and logic. In traditional software development, this often translates directly to underlying tables or views located in a database management system. The model is also accountable for enforcing various constraints on data structures, such as known from relational databases. Some papers [9, 32] also differ between an *active model*, which has a notification mechanism—usually implemented with the Observer pattern—and notifies its views or controller of changes, and the *passive model*, which is completely unaware of the fact that it belongs to a MVC architecture. In general, web applications are stateless and follow a strict request-response cycle, which is why we will talk about passive models in the following unless mentioned otherwise.

2.1.2 The View

The only task of the view component is to request data from the model, and present it to the end-user. Usually, a view is instantiated by the controller, which also passes on the required data. Note that views are not authorized to perform any updates: the controller triggers all changes and modifications in the data structures.

2.1.3 The Controller

Controllers maintain the state and business logic of the application; they act as glue between the models and their views. They process user actions and provide their respective views with data obtained from the model. As controllers have originally been designed for the implementation of graphical user interfaces of desktop applications, they have been the interface responsible for dispatching the event loop of particular views. Controllers receive events from a view (triggered by the user's keyboard or mouse, or timers) and update the state of a model. While, in its pristine definition, a controller was not supposed to act as a mediator between the view and the model, this gradually changed with the emergence of web-based frameworks. The special characteristics of web applications lead to the dissemination and adoption of various controller patterns, most notably that of the:

Front Controller as a single point of entry for processing new HTTP requests,
Page Controller coordinating the logic on single web pages, and
Application Controller defining the business logic of the entire application.

These controllers are often cascaded: the *front controller* accepts incoming requests, passes them on to a page controller's action, which in turn generates a new view, sends it back to the *page controller*, which then sends the finalized response back to the client.

From an architectural point of view, different actors on different application layers are driven by different requirements; this further extends the *separation of concerns* adequately in the context of web applications, as the following illustrates: the HTML format, which represents the view in the eyes of the web server, becomes the model once it is received and rendered by the user's web browser, and the document's object model (DOM) is modified by, e.g., JavaScript, which handles local user interactions.

2.2 State-of-the-Art Implementations

The following paragraphs are to be taken as a guideline and overview on what categories of web application frameworks exist. We will start by covering Ruby on Rails, as its approach might feel natural to most developers coming from traditional *three-tiered* applications.

Ruby on Rails

One of the most popular frameworks for web application development today is, without a doubt, Ruby on Rails. It uses an MVC architecture and includes a lot of tools to facilitate web development. Ruby on Rails comes with a collection of tools, among them WEBrick, a Ruby-written web server application, and lots of prebuilt rake[5] tasks. These tasks aid with low-level maintenance such as creating databases or application modules.

Ruby on Rails set industry standards with its extensive framework support for AJAX in company with graceful degradation, called unobtrusive javascript. This allows AJAX driven websites to still function in browsers without javascript support by falling back on *pure* HTML.

[5] a tool similar to make, written in Ruby

Views are implemented, as in almost any other scripting language based framework, via embedded scripting language code. Developers simply interweave HTML markup and Ruby Code—wrapped in special HTML tags such as in the following snippet:

```
<%puts "Hello World"%>
```

Rails also provides *Scaffolding*, the fully automatic generation of a skeleton application, by introspecting database metadata. It has inspired lots of other frameworks since. Scaffolding allows developers to quickly setup the basic building blocks for an application. From a implementers perspective, the implementation of basic `CRUD` operations demands for lots of developer time and usually is error prone and may even lead to security risks with regards to SQL injections. This is what Ruby on Rails initially wanted to help with: by examining the underlying data, Rails was able to generate the HTML pages and Ruby *glue code* to create, list, edit and delete entries from a database.

The scaffolding feature, in turn, is based on top of a core part of the framework: *ActiveRecord*. ActiveRecord is an object-relational-mapper that encapsulates all database related actions. It allows developers to use their database records as if they were ordinary Ruby objects, containing methods and attributes to perform validations or transformations. This simplifies development a lot and even allows for compound (Ruby) objects that may span several database tables (a sketch of this concept is given in Listing 1).

Listing 1 defines an Object called `Textdocument`. It inherits from ActiveRecord and by *convention* maps to a database table named `textdocument`. It has a property indicating that the field `pages` contains a list (i.e., `has_many`) of `Page` object instances. The related page objects are looked up in the database. By convention, this means that there has to exist a table called `pages`, containing at least one row named `textdocument_id`. These page objects are then fetched, transparently to the developer, and converted to a Ruby object instance as the property is accessed.

```
Class Textdocument < ActiveRecord :: Base
  has_many :pages
end
```

Listing 1: Ruby on Rails Model example.

ActiveRecord comes with the additional benefit of being database independent. Several adapter implementations exist for all major database management systems.

Rails was one of the early popular frameworks that introduced the concept of database migrations, which allowed developers to dynamically adapt their database schemata. Migrations allow developers to create tables programmatically and keep track of different versions.

Google Web Toolkit

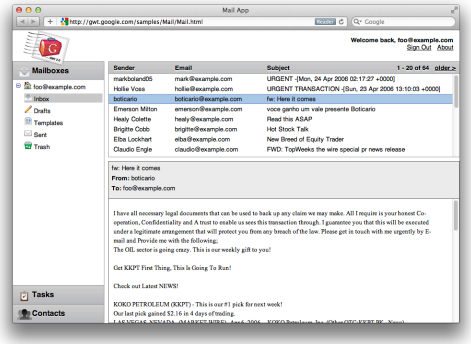

FIGURE 2.2: GWT example: a mail client running inside the browser.

Google's Web Toolkit (GWT)–shifts focus, when compared to Ruby on Rails, or other more traditional development models. Core of GWT's development philosophy is using a single programming language, JAVA, to develop web applications both on client and server. Its specialty is a Java to JavaScript *cross-compiler*, which transparently maps Java objects and method calls to JavaScript that may be executed in the client's browser. This is particularly appealing, mainly for the following reasons:

IDE support as debugging, testing and developing can take place inside a powerful IDE such as Eclipse[6] or NetBeans[7]

RPC Remote Procedure Calls are transparently mapped to AJAX requests where necessary

Objects that are shared between the client (browser) and the server conform to a single specification and do not require manual conversion

GUI frontend interactivity is added by using pre-built widgets. Those widgets may either be programmed, in a way similar to Java's own Swing, or added declaratively in an XML dialect

The technically most interesting part of the framework is, by far, the cross compiler: it does not only bridge the gap between client and server, but also takes care of compatibility issues, which developers usually encounter when targeting multiple browsers.

[6] http://www.eclipse.org/
[7] http://www.netbeans.org/

Yet GWT does not provide any database interaction. Instead this is left completely to the developer and dedicated frameworks such as Hibernate[8].

Google itself uses GWT for the development of Google Mail, Google Maps, or the now discontinued Google Wave.

To recap the above example, redefining the `Textdocument` object in Java is straightforward: Once an instance of `Textdocument` was used on the client-side implementation,

```
public class Textdocument {
  List<Page> pages;
}
```

Listing 2: GWT/Java Model example.

GWT translates this (for the sake of brevity some GWT specific boilerplate code has been stripped) to the following piece of Javascript:

```
this$static.example_client_Textdocuments_pages = // this$static.example contains
                                                 // a reference
  new java_util_ArrayList_ArrayList__V;          // to the class fields
```

Listing 3: GWT/JavaScript Model example, the source code is highly optimized and rather not intended for humans to read

SproutCore

To complete the overview, we will conclude with a description of SproutCore, a framework that tackles only the client. SproutCore is an open source framework written in JavaScript and licensed under the MIT License. SproutCore claims to deliver "desktop caliber applications" to the browser. Contrary to Google Web Toolkit or Ruby on Rails, SproutCore does not involve database interaction, instead it allows data to be pushed and pulled from specific URLs.

[8] http://www.hibernate.org/

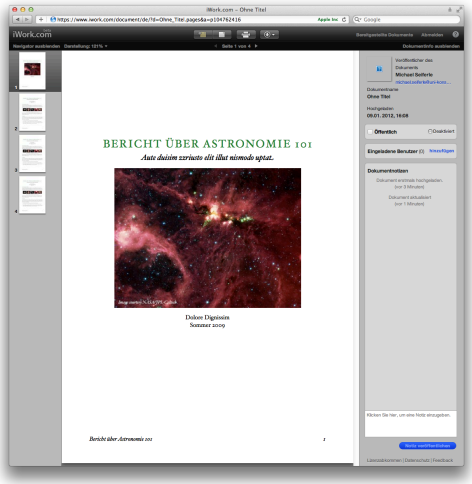

FIGURE 2.3: SproutCore in action on `iWork.com`, showing an Office document.

The developers characterize Sprout as follows:

> "SproutCore applications move business logic to the browser so they can respond to your users' taps and clicks immediately, avoiding an agonizing roundtrip across often intermittent network connections.
>
> As web application users go increasingly mobile, applications can no longer depend on reliable connections to a remote server to do the heavy lifting.
>
> At the same time, web browsers continue to radically improve their ability to quickly process data and deliver polished user interfaces—a perfect opportunity to rethink the architecture of modern web applications." [33]

As SproutCore only runs inside the browser it relies heavily on observing the Document Object Model and allows an event-driven development model that is notified once DOM or data changes. Implementers define data flows and how their applications are supposed to react on events. It is built to make use of the most recent HTML5 features.

```
Textdocuments = SC.Application.create();
Textdocuments.Textdocument = SC.Object.extend({
// initialize the document with an empty list of pages
pages: []
});
```

Listing 4: SproutCore Javascript Model example

SproutCore contains a rich set of UI Widgets, which resemble their desktop application counterparts, and a versatile templating engine. Financially, SproutCore is backed by Apple—who in turn implemented their `icloud.com` browser fronted with SproutCore—and has an ever-growing community.

2.3 Challenges

2.3.1 Real World Data

When storing real world data in database-driven applications we usually face the challenge to fit it into evenly sized, pre-defined rows and columns. This proves not only to be *conceptually difficult*, but also poses computational challenges that tend to scale rather badly.

Thinking of, for example, a document store, one can easily come up with a relational mapping that captures key characteristics, such as every document having a unique id, a name, a creation date and is located at a specific URL. For a (relational) database architect, this immediately translates itself to a schema like the following:

```
DOCUMENT( id, name, crdate, uri, content)
```

Developers may now access each document, its metadata, and its content by executing a simple query as shown in Listing 5:

```
SELECT id, name, crdate, uri, content from DOCUMENT
```

Listing 5: SQL: Retrieving a list of documents

This solution will work just fine, as long as we think of a document in terms of a single stream of data—the content—stored as a large blob. By building this model we lose the means to access a document at a finer level of granularity. A modeling decision made upfront will limit future use-cases, as we will, e.g., never be able to query a single page inside a document.

To allow for even finer granularity, the database architect could have come up with the following decision:

```
DOCUMENT( id, name, crdate, uri)
    PAGE( document_id, page_id, content)
```

The model stores individual pages inside the PAGE table, uniquely identified by a compound key consisting of the foreign key document_id that connects each page with a specific document and a page_id that identifies a page inside a document.

To access a document, developers will have to rebuilt a complete document at runtime as depicted in Listing 6

```
SELECT d.id, d.name, d.crdate,d.uri,
       p.content, p.page_id
FROM   DOCUMENT d,
       PAGE p
WHERE  d.id = p.textdocument_id
ORDER  BY d.id,
          p.page_id
```

Listing 6: SQL: Retrieving a list of *whole* documents by implicitly joining the DOCUMENT and PAGE relations

Yet we have not reached the finest level of granularity possible, thus still reducing the potential of our application in terms of flexibility and extensibility. If, at any given time, a problem demands for individual paragraphs to be extracted from our document base, we (once more) have to split the pages relation[9] to come up with a model like the following:

```
  DOCUMENT( id, name, crdate, uri)
      PAGE( document_id, page_id)[10]
PARAGRAPH( document_id, page_id, paragraph_id, content)
```

In further consequence, a change of the relational model demands for changes at the application level: involved joins will have to be rewritten, possibly in numerous locations. The same holds for making the remaining application aware of the changed data model: as

[9] N.B. we are aware that paragraph splitting could (and in practice most probably will) be handled at the application level, yet this again fosters the creation of non-reusable code *and* only shows the inflexibility of our model

[10] This relation might be completely removed in the course of normalization, if it carries no attributes other than the two listed.

queries are only executed at runtime, developers will have no means to be sure[11] to actually have changed all relevant functions.

Even though Object Relational Mapping (ORM) and dedicated libraries will take most of these tedious tasks off a developers to-do list, the very existence of such mappers shows the relevancy of making modeling decisions carefully. Thus in day-to-day business, changes to the relational model are usually avoided. In consequence the decision *which* and *how* data is stored has to be well thought out. Adding new components or attributes to a model once an application has gone into production, generally tends to be costly and error prone. On the other hand, storing the model at its very finest granularity (e.g., each sentence, or even each word/character, individually in the case of the aforementioned example) wastes resources and adds lots of complexity to query evaluation as, most probably, such a fine level of granularity is not going to be needed for most of the use cases.

2.3.2 Modeling in XML

This is where XML technologies come into play. In the context of this thesis, we stress three main arguments for considering XML as the model of choice:

Data Exchange. XML has been designed with data interchange in mind [8], and succeeded to become the premier format for data interchange on the web. Hence there are numerous programs and resources that process or produce XML natively.

Modeling Freedom. Due to XML's design, architects may model structured content, semi-structured data and unstructured data. In addition standards, such as XML Schema [11] allow developers to regain power, in terms of *type safety*, over their data.

Expressiveness. With XQuery [20] implementers are provided with a fully capable programming language, which is well suited to define business logic *and* data manipulation. While before these two tasks were most often clearly separated by a context switch between scripting languages[12] and database languages[13].

[11] e.g., in terms of static type checking
[12] e.g., PHP, Ruby, Python or Perl
[13] i.e., SQL

Besides that, XML and the web are natural partners by design: When Tim Berners-Lee motivated the "Rule of least Power" he not only insisted on using the tool most suitable for the *job*, but also technology that is most suitable to convey *information*:

> "A different sort of scalability can be found when comparing Turing-complete languages. Although all have equivalent expressive power, functional languages such as Haskell and XSLT facilitate the creation of programs that may be easier to analyze than their imperative equivalents. Particularly when such languages are further subset to eliminate complex features (to eliminate recursion, perhaps, or to focus on template forms in XSLT), the resulting variants may be quite powerful yet easy to analyze. When publishing on the Web, you should usually choose the least powerful or most easily analyzed language variant that's suitable for the purpose."
>
> Tim Berners-Lee in [6]

For that reason, from a software development point of view, **it is all about abstraction**, abstraction at retrieval time. And these abstractions are made notably easy by opting for XQuery as the processing language of choice. Let services decide which data they need and, instead of tailoring data first, we will tailor information on demand.

2.3.3 Programming XML: XQuery & XPath

In a nutshell, XQuery relates to XML the same way SQL relates to relational data: it provides the means necessary to *select* and *manipulate* specific parts of XML documents. But despite data manipulation and retrieval, there is more to XQuery: it is a general-purpose programming language[14], suitable to tackle all kinds of problems.

XQuery is a conglomerate of several `W3C` specifications:

XPath as specified by the W3C in [3] is "an expression language that allows the processing of values conforming to the *XQuery 1.0 and XPath 2.0 Data Model*". The basic building blocks of an XPath expression are so-called steps, made of: an *axis*[15], a *node*

[14] c.f. Kilpeläinen [23] on XQuery for problem solving
[15] such as: `child`, `parent`, `following-sibling`. (for a complete list please refer to the spec)

test[16], and an optional predicate that further restricts the sequence of items that is to be returned by a given step. An example showing how to hierarchically address elements inside an XML document can be found in Listing 17 on page 67.

XQuery and XPath Data Model, the XDM, as specified by the W3C in [4] makes up the information atoms of the language. In addition to the types defined in XML Schema [12] it extends this model most notably by supporting sequences of heterogeneous values (i.e., atomic values such as, strings, but also complex types like nodes and documents). This data model, while tailored for processing XML, may as well be used to handle arbitrary data types, such as JSON or relational data in a uniform manner.

XQuery is a standard defined in [20]. XQuery makes use of XPath expressions in order to select specific parts of a document, and is a strict superset of XPath. It further provides control structures to iterate over sequences of XDM instances. Supplemental to SQL's `SFW`[17], XQuery implements `FLWOR`[18] expressions. XQuery scripts may be organized in modules, each inside its own namespace, which are further broken down in function definitions. This fosters reusability and readability of XQuery code and libraries. Another distinction to XPath is XQuery's ability to explicitly construct new XDM instances programmatically. This enables implementers to transform one XDM instance—e.g., a sequence of numeric values—to, for example, an (X)HTML list.

The example in Listing 7 on the facing page is strongly hypothetical, but is equally relevant in practice: without ever explicitly converting data, we seamlessly switch between simple types, such as integers or strings, and complex elements and preserve their structure. This lays foundation for the expressiveness and power XQuery hands on to developers.

[16] such as: only attributes, only elements, only elements with a specific name
[17] `SELECT, FROM, WHERE`
[18] `FOR, LET, WHERE, ORDER, RETURN`

```
declare function local:even-squares($range as xs:integer+)
  as element(ul){
  <ul>{
    for   $x in $range
    let   $y := $x * $x
    where $x mod 2 = 0
    return <li> { $y } </li>
  }</ul>
};
sum( local:even-squares(1 to 20)/li )
```

Listing 7: An XQuery example showing some of the unique concepts XQuery and the XDM provide: We define a function, `even-squares` that accepts a sequence of integers as its input, and returns an `<ul/>` XML fragment. The FLWOR expression inside the function body iterates through each integer, skipping the odd ones, and constructs a new `<li/>` element containing the current integer's square. This sequence of `<li/>`s is then wrapped inside an `<ul/>` and returned. On this result sequence we apply the XPath expression `/li`, to select each of the constructed `li` elements and compute their `sum`.

3 BaseX Web: XQuery-driven Web Application Framework

3.1 Background

3.1.1 Maturity of Web Applications & Frameworks

Deploying applications for the web has become the preferred mode of operation for both end-user and expert systems. The web is filled with all kinds of hosted software solutions, ready to satisfy almost all information needs that might possibly arise.

There exist numerous machine-readable resources, almost always fostering XML as a lingua franca. Considering that every major news site offers RSS or Atom feeds, there are also REST and SOAP, the de facto standards for program-to-program data exchange and remote procedure calls on the web. In the same way, numerous websites offer applications which can be used with any browser—most famous Google Search, serving as an retrieval tool for literally billions of people each day. Thanks to Google pioneering the web as an application platform, users are more and more willing to accept using their browser for tasks other than just surfing the web. In addition, applications—like Google Mail or Google Reader that feel almost like native applications—most often surpass their desktop counterparts, in terms of features, by tightly integrating independent remote services.

A landmark, the introduction of Ajax [14], showed the way for the years to come: Emerging from a past, where developers thought of JavaScript mostly as a tool to validate forms before sending them, or used it to swap images on mouse-over, Ajax raised the bar: instead of developing applications for a single target operating system, developers were now equipped with a tool chain that allowed them to build applications once and run them on any machine connected to the internet. Likewise all of the heavy lifting could now be off-loaded to dedicated server machines, while clients only had to cope with result representation.

This development led companies to focus on web applications. Even major players, like Apple's iPhone or Palm's webOS, opted for web applications on their platform. In the beginning both even lacked *native* development kits. Clearly, both Palm and Apple wanted their developers to deploy applications directly to the web, without opening their architecture to any native code. From a developers perspective this approach was perceived

ambivalently: there was no need to worry about low-level concepts, such as device types or software versions, yet on the other hand this was part of the problem. Developers had no means to directly access the hardware at all, which was limiting in many cases.

Since then, the frontier between the browser and native applications has been vanishing. The standardization of HTML5 addresses many of the aforementioned issues, among them:

Offline Support gives HTML5 applications a dedicated storage—provided by the runtime environment—to locally cache their data. In addition, developers may subscribe to certain events, check for online connectivity, and perform synchronization. So instead of uploading user data to the server right away, applications may decide to store sensitive information only on a client's machine.

File Access gives developers access to the local file system, such that HTML5 applications may store and retrieve files. This represents a major advantage over the status quo, where developers often switch to proprietary techniques, such as Flash, to access the file system of a client.

Connectivity via WebSockets allows bi-directional communications, as such the remote server is able to notify the client of events.

Graphics enables developers to make use of 3D acceleration hardware with very low effort.

3.1.2 Impedance Mismatch

Object Relational Mapping is a very powerful mechanism, yet it forces developers to live compromises. These compromises are mainly due to a *clash* of paradigms once the object-oriented—data & functionality defined in a procedural manner–and the relational representation—tuples & set-data models—are mapped to one another.

One might argue that building web applications in a functional language like XQuery we would never encounter this problem, which is true to some extent. Nevertheless, with XQuery we are able to retrieve and manipulate persisted data, mostly in the same manner that developers are used to when working with Object Relational Mappings. But contrary to the latter, XQuery has been designed to actually work on these data structures natively, so the efforts of conducting conversions back and forth are no longer required. In XQuery we can safely assume that the data we work on is persisted seamlessly.

The problem of conversion is known as *Object Relational Impedance Mismatch*, and investigates the issues developers face when persisting objects as relations. In [18] the authors identify categories of object relational impedance mismatch, which are grounded in object-oriented paradigms:

Structure. Objects hold both, data—it may even be part of class hierarchy—and functionality. The relational model has no notion of such object-oriented concepts. A tuple in a relational model is only defined by its data, and hierarchies always involve more than one relation. When using XML and XQuery, we do not encounter any persistence issues, XML data never carries functionality and provides *native* support for hierarchies.

Identity. An object has an identity that is not dependent on its internal state (i.e., the data it is holding). Running an object-oriented application twice, the very same object, defined by its internal state, may have a different identity, as it is only a runtime construct. In the relational model, the identity of a tuple is given by its data and primary key. This makes it a trivial task to absolutely identify a tuple, while we have no way of absolutely identifying an object inside an object-oriented program.

Encapsulation. Objects hide their state via methods. Programmers may modify these data in a well-defined way. Rows on the other hand have no such concept, their state is their data and has no such protection. Although database systems provide users with mechanisms to secure their tuples.

Processing model. Relational data processing involves transactions that are sequential, set based applications of functions over tuples. In contrast to this, the object-oriented model, at its core, is the logical grouping of data and functionality. From a relational perspective the definition of two entities belonging to a different category differs only by the chosen column names and data types. From an object-oriented perspective two distinct entities can, and usually will, differ with respect to attached functionality.

3.1.3 Frameworks: Pros and Cons

The former observations proof the ambiguity developers usually face when deciding for or against such assisting frameworks. The benefits are obvious: implementation details and caveats are hidden from the developer. This technique of abstracting implementation details—and concepts—is used with great success and at various levels in software engi-

neering. For example, most programmers will never write a line of assembly code, why should they? It is perceived hard to read, hard to debug and usually hard to collaborate on in teams. Frameworks are, without a doubt, extremely successful and popular in the web world, mainly for the following reasons:

Comfort. The framework handles tedious tasks such as the creation of nicely readable `URLs`, form generation, data validation or access control

Code Reuse. As numerous problems have to be solved over and over again, frameworks assist developers in writing reusable bits of logic once, and using it in many places

Database Access. With the help of Object Relational Mapping a developer is presented with an *unified development stack*, where objects and tables seem smoothly integrated

Inversion of Control. Makes the program's *flow of control* obey to the framework. Thus the framework defines both what is done and when it is done

On the other hand, when hiding away concepts from the developer, frameworks take away power, in terms of programmatic expressiveness, the developer otherwise had. The developer is forced to stay inside the *black box* the framework provides. This black box, actually counter intuitively to all relief it provides, may add another layer of complexity to a project. When moving away from predefined paths, be it due to requirements that are not covered by the framework, or a lack of understanding how to make right use of its functionality, development tends to be even more cumbersome than it was without a framework.

3.2 Related Work

The idea of using server-side XQuery implementations to foster application development is not new and has been around for quite some time in competing open source implementations such as the Sausalito project[1], which claims to bring XQuery to the cloud, or eXist-db[2], one of the early native XML database systems. Both implementations use different approaches and focuses to achieve this goal.

They mainly differ in two aspects, while the latter is more database-centric, the former is about XQuery-powered application logic (exposing data-centric services through a RESTful interface and delegating storage considerations to arbitrary backend systems).

3.2.1 eXist — The XQuery Servlet

eXist-db is one of the oldest open source native XML database management systems, and has always been driven by a growing community. eXist itself runs out of the box and comes packaged with an installer for convenience.

> *eXist […] a native XML database system, which can be easily integrated into applications dealing with XML in a variety of possible scenarios, ranging from web-based applications to documentation systems running from CDROM. The database is completely written in Java and may be deployed in a number of ways, either running as a stand-alone server process, inside a Servlet-engine or directly embedded into an application.*
>
> *Wolfgang Meier (project leader) on the goals of eXist [28]*

Besides traditional APIs, eXist as well offers all of its functionalities via two web services, an XQueryServlet and an REST-style API.

In order to generate a web page, eXist uses the XQueryServlet to generate XHTML. eXist's XQuery processor is contained in the servlet and maps a URL to an XQuery script file inside the file system.

[1] http://www.28msec.com
[2] http://exist.sourceforge.net/

This approach is similar to the *scripting* style of developing web applications and as such, has a low entry-barrier for developers familiar with languages such as PHP or Perl. This very basic toolset already allows developing whole applications, including a database backend, in XQuery. It is also worth mentioning that eXist's complete administration interface is implemented in XQuery, and completed with eXide, an XQuery IDE[3] that allows developers to implement, run and debug XQuery modules directly in the browser.

eXist's REST implementation runs inside a Servlet context as well, but contrary to the XQueryServlet it stores its XQuery modules directly inside the database. The most recent development version of eXist also introduced RestXQ: a JAX-RS[4] inspired API that allows developers to map URLs to XQuery functions using annotations.

eXist also benefits from a big pool of community created extension modules, which cover lots of problem domains. In this regard eXist may definitely be seen as the implementation setting the standards.

3.2.2 Sausalito — XQuery in the cloud

In 2009, Kaufmann and Kossmann were the first to examine the benefits of developing web applications with XQuery. Their conclusion favored the approach:

> […]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.
>
> Kaufmann and Kossmann in [21]

This research found its commercial descendant in the company 28msec and their product Sausalito, "a suite of tools that allow to write, test, and deploy full-fledged web-based applications, entirely written in XQuery" [1].

[3] Available at http://demo.exist-db.org/exist/eXide/index.html
[4] c.f. http://jcp.org/en/jsr/detail?id=311

28msec argues that "XQuery has an extremely powerful support for database queries, scripting, and full-text search. By using a single programming on all tiers, Sausalito is collapsing web servers, application servers, and databases into a single stack." [1]. They consequently use XQuery for: writing application code, defining data- and access-structures and database access.

Its functions can be invoked with any HTTP client, in general however, Sausalito is an application server for RESTful services. Application logic is completely implemented in XQuery, and the Zorba XQuery Processor[5] is used for evaluating the queries.

The general project structure for a Sausalito project, as described in [1], is as follows: All XQuery code is structured in XQuery modules. Each module usually concentrates on one very specific aspect of the application. Sausalito further discriminates between three kinds of modules: *Handler*, *Library* and *External* Modules.

Handler modules "contain XQuery functions (called handler functions) that build the REST-based interface of your application. Each of the functions is directly exposed using REST and can be called by making an HTTP request with a path component that is identified by the module's file name and the name of the function." [1] Each handler's task is to orchestrate HTTP requests and implement business logic. In the context of MVC their role resembles that of a controller.

The Library Modules define general-purpose functionality, which may be used by other libraries or the handler modules. Some Library Modules even come prepackaged with Sausalito and cover a wide range of features from image processing to authentication. These libraries are not exposed via REST directly, but otherwise do not have any special rules to follow.

External modules contain additional XQuery modules, provided by any third party.

In order to deploy a Sausalito application, 28msec hosts a cloud infrastructure on Amazon Web Services. Once deployed, Sausalito can be seen as a solution for building RESTful

[5] http://zorba-xquery.com/

services with XQuery running in the cloud. The framework uses sophisticated distributed commit protocols [7], as it supports several storage backends. The database backends supported cover distributed key-value-stores, e.g., MongoDB or SimpleDB, or JSON stores, as well as the file system. Figure 3.1 illustrates Sausalito's integrated application stack.

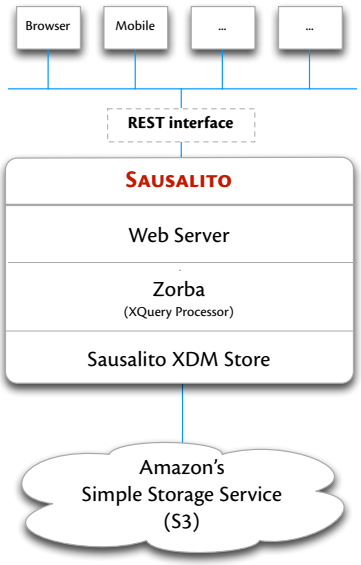

FIGURE 3.1: Sausalito's integrated application stack.

Distinction

So as to benefit from both approaches—eXist's direct style and Sausalito's service oriented architecture—we present BASEX WEB. BASEX WEB aims to be a framework that enables developers to rapidly implement XML-based web projects. A focus has been put on relying exclusively on W3C standards, hence fostering research efforts that influenced these standards.

The main goals we set up for BASEX WEB are versatility and flexibility with respect to different problem demands:

(1) a service-oriented architecture, ready to serve XML or JSON to frontend systems, built on the foundations of a pure X-technology stack

(2) providing a general-purpose MVC architecture, as an infrastructure for own applications covering all application layers, ranging from rendering to storage

In order to separate concerns regarding storage, processing and rendering of data we opted for an MVC architecture. The following chapter will provide an overview of the steps necessary to piggyback a web application framework on top of the BaseX database engine.

3.3 System Overview

From the beginning, BaseX Web was designed to be *just another database client*, as this allowed us to leave BaseX' core unchanged. BaseX Web sends requests expressed in XQuery, and retrieves results (serialized as XML, binary or JSON) from the processing engine.

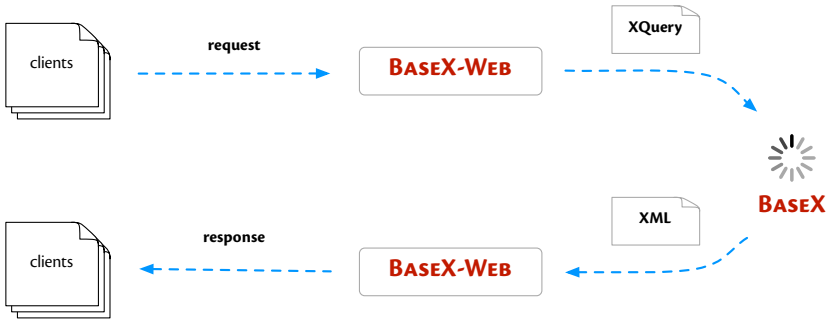

Figure 3.2: System overview: BaseX Web's mode of operation

In BaseX Web, a project holds the "business logic", i.e., the functions we want to perform on the stored data in distinct XQuery modules. Usually this means retrieving data from a database, responding to a client request, and subsequently return a serialized result to the client. As web applications demand for processing options that go beyond XQuery's defined capabilities, such as cookie handling and setting of HTTP headers, we extended BaseX in a non-intrusive manner. The EXPath packaging specification[15][6] is a community driven project that defines standards for developers building XQuery extension libraries. These libraries may then be installed into the database management system and extend the query functionality. Whenever developers encounter cases, which demand for explicitly modifying a HTTP Response, they may do so by calling functions from the prepackaged `web` extension module.

BaseX Web's architecture is grounded on well established building blocks. The use of Java Servlet Technology provides "developers with a simple, consistent mechanism for extending the functionality of a web server"[7]. The framework itself is deployed as a web (appli-

[6] http://expath.org/spec/pkg
[7] http://www.oracle.com/technetwork/java/javaee/servlet/index.html

cation) archive[8], and encapsulates all needed functionalities in a single package. The `war` can then be deployed in a dedicated runtime environment, the Servlet Container. During development and testing the `jetty://` web server[9] was used as a runtime environment, c.f. Figure 3.3: As we make use of BaseX' internal APIs for query processing, BaseX Web directly benefits from all tweaks, features, and optimizations performed by the BaseX query processor.

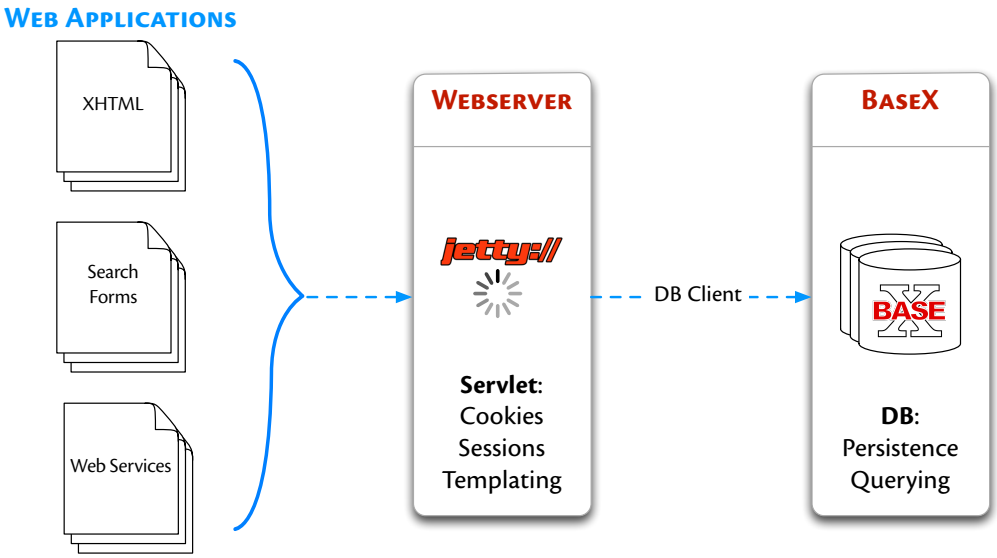

Figure 3.3: System overview: BaseX Web building blocks

Designed with the goals of a unified data modeling language, XML in partnership with XQuery is an ideal match to model, process, and present information resources.

With BaseX Web we remove the need for *glue code* between various interconnected layers of an application, c.f. Figure 3.4. In addition, we can now work without data conversions, which have been needed before in order to match each layers processing paradigms.

Regarding the high-level architecture, we decided to obey the MVC pattern, as its components closely resemble the XML technologies implemented with BaseX Web. Martin

[8] in `war` format
[9] `http://www.eclipse.org/jetty/`

Fowler names three layers in enterprise architectures, which correlate with the Model-View-Controller Architecture:

> **Presentation.** Provision of services, display of information (e.g., in windows or HTML, handle user requests such as mouse clicks, keyboard hits, HTTP requests, command line invocations)
>
> **Domain.** The logic that is the real point of the system
>
> **Data source.** Communicate with databases, messaging systems, transaction managers, and other packages
>
> Fowler [13] on architectural patterns

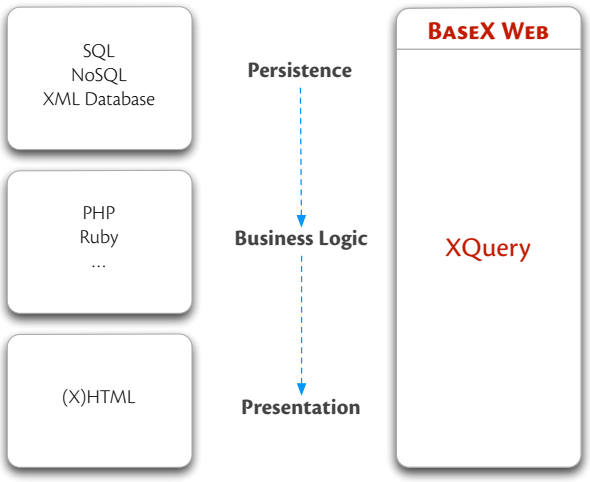

Figure 3.4: Sketch of the architectural model of traditional web applications compared to BaseX Web. The BaseX Web application server provides a complete runtime to host XQuery built web applications, while only talking the *native* languages of the web

Following this high-level overview, we switch to an introduction of the application layout in the following section. Afterwards we will describe each of the involved components and their responsibilities, by following a complete request-response-cycle.

3.3.1 Application Layout

Applications implemented in BaseX Web comply with a fixed project directory layout:

The `models` folder contains XQuery Modules that encapsulate database interaction, and it may contain XML Schema files in order to validate data before persisting it.

The `controllers` folder contains XQuery modules (referred to as controllers) that encapsulate the business logic.

The `layouts` folder contains (X)HTML files with markers that direct BaseX Web to insert evaluated content when sending a response.

The `views` folder contains a folder for each controller available inside the application and has XQuery files—named *actions*—that respond to a unique URL.

```
MyWebApplication
├── models
│   ├── fsml.xq
│   └── (fsml.xsd)
├── controllers
│   └── deepweb.xq
├── layouts
│   ├── ajax.html
│   └── default.html
└── views
    └── deepweb
        ├── search.xq
        ├── ls.xq
        ├── dir.xq
        └── action.xq
```

This procedure asks implementers to explicitly categorize their files, depending on their functionalities, and comes with two benefits:

Facilitate Collaboration. Once learnt, a team of developers may independently work on small subsets of features. Front-end developers, for example, work only on views, while XQuery developers concentrate on a specific set of controllers.

Meaningful URLs. As we are able to map this directory layout to a generic URL, only a small set of rules is required to provide meaningful URLs.

3.3.2 The Servlet Implementation: Request-Response-Loop

This section investigates implementation work and engineering, which have been conducted in order to teach our servlet XQuery. We will describe what is necessary to perform a complete round trip, as depicted in Figure 3.5 on the facing page: a client request → server side parsing of the request URL → building & processing the query → embedding the result in a layout → transferring the result back to the client.

Incoming HTTP requests that refer to an application running in BaseX Web must comply with the following syntax: `/app/$controller/$action`. The servlet will interpret the URL and guarantee that the prerequisites below are fulfilled:

- an XQuery module `controllers/$controller.xq` *may* exist in the current project's directory; if it does, a flag is set
- an XQuery file `views/$controller/$view.xq` *must* exist in the current project's directory

An existing view is read into memory, otherwise an `HTTP/404` error is sent to client's browser.

If the first prerequisite is met, and the controller module exists, it will be imported into the view. This ensures that all functions defined in the controller namespace are available to the view. This convenience function allows the implementer to call controller functions without manually importing each controller that belongs to a view.

After that, the fully assembled view is submitted to the BaseX database server.

Usually HTTP requests contain parameters, other than the actual URL, such as `GET`, `POST`, or `COOKIE`, which need to be processed by the web application framework. In BaseX Web such parameters are bound to an external static variable, typed as a `map`[10], so its values are accessible from inside the framework. The standardization of *maps* is in progress at the time of writing, and currently based on a W3C Working Draft[11]. They have been proposed as an additional data type that allows implementers to use hash maps, hence they were a perfect match to resemble request parameters.

[10] XQuery's notion of Key-Value-Pairs
[11] `http://www.w3.org/TR/xpath-functions-30/`

Aforementioned parameters are then visible in a *view*, as maps with the names: `GET`, `POST`, or `COOKIE`. To access a specific parameter a developer accesses its index: `$POST("firstname")`.

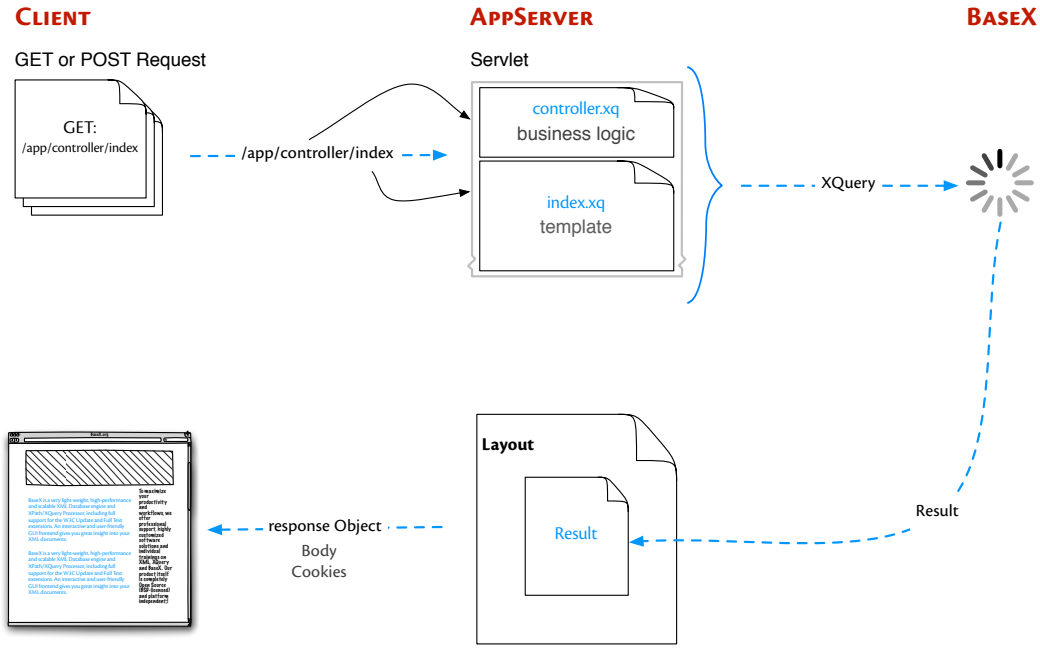

FIGURE 3.5: Full request-response cycle: accessing an URL triggers the construction step to assemble the complete XQuery for submission. This query is then executed, i.e., database elements are fetched and processing is done, and returns its result sequence. This sequence is then embedded into an layout and streamed to the client.

Now that we have assembled the complete query and bound all request parameters to their respective maps, we have a fully working query that is ready to be dispatched to the XQuery engine. To the server this query looks like any other query and is processed without any special arrangements. In case the implementer needs to set parameters related to the `HttpServletResponse`[12] instance, he could do so with the help of the WEB extension module: The invocation of any of these functions sends a notification via BaseX' events mechanism to a watcher running inside the servlet. The notifications implement the command pattern in XML, an example is shown in Listing 8 on the next page. The servlet is notified with a certain event and parses its XML payload to subsequently invoke the specified method with the given arguments.

[12] http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html

Meanwhile, the servlet receives the evaluated query results from BaseX. The servlet embeds this response in `layouts/default.html`, unless the developer set a different template, via XQuery serialization options, and flushes the response to the `ServletOutputStream`. It includes the (optionally modified) headers and the complete body that make up the full web page.

```
<command name="set-cookie">
        <session>21d87371-9e01-4b3c-936a-4c80bad47019</session>
        <arg>Cookiename</arg>
        <arg>This is the cookie's value</arg>
</command>
```

Listing 8: XML Fragment notifying the servlet to add a cookie to the response. An excerpt of the implemented functions may be found in Listing 19 in the Appendix

3.3.3 XQuery Processing & XML Persistence: Database Server

A highly performant and at the same time very flexible storage layer is provided by the BaseX database server. BaseX is a native XML database system and XQuery processor. The XML Store supports updates and, beside the usual name, path, and value indexes, maintains a dedicated full-text index structure. Christian Grün has described it in excellent depth in his dissertation [16].

From inside the web framework, BaseX is used both as a *persistence layer*, addressing storage demands and as a processing layer, implementing business and presentation logic. This approach allows developers to leverage XML's flexible, document-oriented storage model in conjunction with XQuery, a general-purpose programming language with powerful retrieval capabilities.

3.3.4 Application Framework

The coming sections deal with the XQuery Application Framework and its implementation in more detail. Development of the basic architecture was driven by portability demands; the main goal was using as little implementation dependent *glue code* or vendor specific functions as possible. As XML defines a family of languages by design, processing tools such as XQuery enable developers to concisely steer content creation.

The Model: XML, Schema & XQuery

As shown in Section 2.3.2, XML allows developers to overcome many pitfalls when it comes to modeling data: flexible and complex models are implicitly defined by the hierarchical structure of the data. As such, XML developers can rely on a toolchain of modeling related helpers[13]. Inside BaseX Web a model resembles an XQuery module, implementing functions that handle retrieval, validation and updating of underlying databases. This allows developers to:

retrieve well-defined XML fragments off the database for further processing
validate fragments before persisting them inside the database
decouple data related tasks from business logic processing

Schema & Validation. When it comes to XML data modeling, XML Schema [11] immediately springs to mind. XML Schema has been initially developed to "considerably extend[s] the capabilities found in XML document type definitions (DTDs)" [11]. Schema is represented in XML[14] and can in turn be processed by numerous dedicated tools, and even XQuery. Schema allows defining structural constraints, e.g., certain elements have to occur at specific locations, as well as data types. Types are subdivided in simple (for example numerical, textual) types and complex types that define element structures. Listing 18 on page 67 shows a complex type constructed of several simple types.

This allows BaseX Web to make use of Schema when needed by the developer. Developers may decide to run fully-fledged Schema validators, such as Apache Xerces[15] on demand. In addition developers may even implement dedicated modules in XQuery that perform basic validity checks by parsing and evaluating the Schema definitions.

Scaffolding. Based on this observation, we decided to implement very basic *Schema powered* scaffolding that while only supporting a subset of XML Schema's features, serves well to quickly sketch a basic CRUD application layout. Similar to inspecting a relational

[13] e.g., SyncRO Soft's oXygen XML Editor, Microsoft's XML Schema Designer, or Altova's XMLSpy
[14] contrary to Document Type Definitions, which are part of the XML Specification but do not share XML's syntax
[15] http://xerces.apache.org/

database's DDL[16], BaseX Web comes with an XQuery module containing functions that allow users to automatically create HTML form elements, for XML fragments, based on a Schema definition. The process is diagrammed in Figure 3.6.

The View: XHTML

The *view* is the centerpiece of *user interaction*. As such it plays a crucial role inside BaseX Web's architecture: each view corresponds to a unique URL. From the framework's perspective, views are plain XQuery scripts that are used, in their most simple manifestation, to retrieve processed data from a controller, work up HTML, and present it to the user's browser. Hence, the framework is able to leverage XQuery's transformation and serialization properties in order to provide a flexible, self-contained templating language. The results of a view will then be embedded inside a configurable HTML layout template.

[16]Data Definition Language, a language to describe data structures

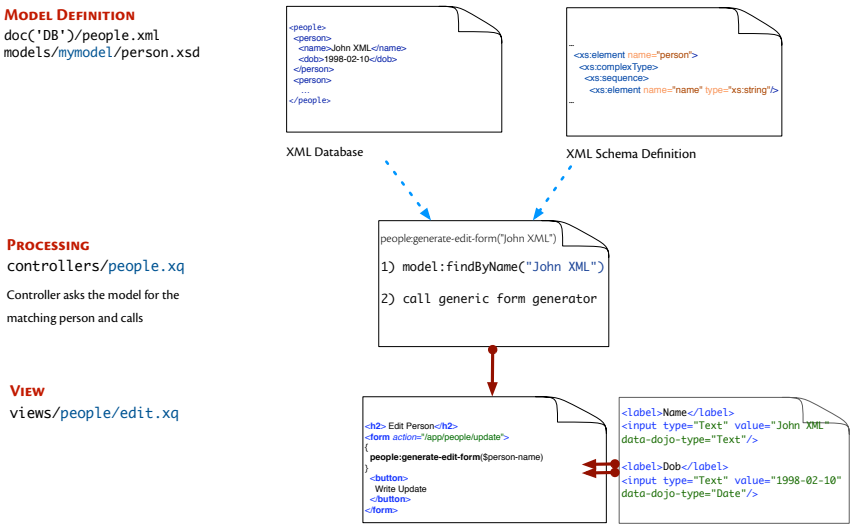

Figure 3.6: The scaffolding process chain: given an arbitrary XML fragment, the generic form generator tries to find a schema file containing possible type information. The fragment is recursively processed in a depth-first traversal. For each element and its attributes, the generator returns HTML form elements, populated with the given values, labels and possibly type information.

Serialization. The XQuery-powered views become even more interesting with the new serialization parameters introduced with XQuery 3.0: output no longer has to be serialized as XML but the original specification is extended to support various formats. In the context of web applications, serialization formats besides (X)HTML, such as JSON as well as raw, i.e., BASE64 encoded binary streams, are suited to serve even more application domains:

JSON as web front ends become more and more JavaScript driven, using JSON to pass around objects gained a lot of momentum. Despite using (X)HTML for AJAX calls, which involves parsing responses inside JavaScript, developers are now able to create native JavaScript Objects and directly work with those on the client.

Binary Data such as media files, can be streamed directly to the browser from inside the framework. As BaseX, since version 7.0, allows users to store binary files[17] in a dedicated backing store, retrieving and serving those files from the database is a viable option compared to plain file-system storage.

From inside a view, setting the appropriate options, as shown in Listing 9, can trigger a serialization other than XML:

```
declare option output:method "jsonml";
<person>
  <name>John XML</name>
  <dob>1998-02-10</dob>
</person>
```

Listing 9: BaseX Web view, serializing its output to JSON. The result will be serialized as `["person", ["name", "John XML"], ["dob", "1998-02-10"]]`

BaseX will handle these serializations automatically, and offers an API for client applications to retrieve the output parameters set. BaseX Web will take these parameters into account when generating the HTTP response that is sent to the client. The appropriate HTTP headers, e.g. `Content-Type:application/json` as in Listing 9, will be set. Depending on the content type the automatic *layout inclusion* is omitted. Binary files for example are always sent to the browser *as is*—there is no need to wrap an HTML layout around an image.

[17] A feature included in the process of turning BaseX in a database file system (see BaseX FS [17] for details)

Correct xhtml. Another argument in favor of a pure XML approach is that correct (X)HTML comes at essentially no additional cost. The correct nesting of elements and syntactic properties of the output are checked at *parse time*. This behavior is highly desirable with current browsers, as each browser has its own handling of erroneous documents. Despite producing only syntactically *correct* fragments, developers are now also enabled to pass around and store native templates directly from inside the language[18]. This approach is also in use in Facebook's `XHP`[19], an implementation in PHP that "[...] has enabled us to build better websites faster" [24].

Controller: Business Logic in XQuery

The controller can be thought of as the negotiator between a view and its model(s). Therefore controllers are XQuery modules that expose functionality specific to working within one or more models. Thinking of applications that allow users to list, filter and display specific pieces of information, a controller function may handle each of these tasks with a dedicated function. As such, the controller requests the data, e.g., specific database entries in alphabetical order, from the model and returns those to the view.

Auto Inclusion. XQuery, in its current state, has no language features that enable dynamic function invocation[20]. Hence dynamically loading modules is impossible in day-to-day practice. So as to avoid including cumbersome XQuery module imports in the prologue of every view the Servlet implementation takes care of checking whether a controller for the current view exists. In case it exists the Servlet prepends its `import`-statement to the view before processing it. This ensures that each view imports at least its belonging controller into its `controller`-namespace, so the developer is able to use its functions. This mechanism might become obsolete in the future, as XQuery 3.0 plans to add support for dynamic function invocation, which could lead to small changes in the overall architecture: a view would no longer actively request controller functionality, but instead be *passively* provided with data generated by a controller's function.

[18] c.f. `http://validator.w3.org/docs/why.html` on why validation and thus syntactically correct markup is essential.
[19] `https://github.com/facebook/xhp`
[20] apart from `util:eval`, which in general should be avoided

Updating Functions. Another speciality, rooted in XQuery's functional nature, is the handling of side effects and updates: *Updating Functions* never return any value, in consequence developers are not able to use imperative style such as: `if(update($record)) then "Update succeeded" else "Update failed"`. Instead, implementers will have to check preconditions, such as the presence and validity of all values that need to be updated inside the model, and subsequently issue the actual updating query if all preconditions are met. In addition, updating and non-updating functions must not be interweaved, thus an updating function is not allowed to call a non-updating function and vice-versa. This forces developers to rely on a pattern known as "Post/Redirect/Get" [27]:

Post. The page receiving the post, processes the data and instead of returning a result page, it sends a redirect header.

Redirect. Converts the `POST`ed data to `GET` data and redirects to the result page.

Get. Data is now available for a result view that either confirms the update was successful, or in case of incomplete data shows the input form again. As the input data is still available in `GET`, the form fields may be pre-populated.

This process is modeled in XQuery as shown in Listing 10.

```
if(model:check($POST))
  then(
    web:redirect( "/app/model/view/?entry={$POST('uuid')}",
      web:get-params($POST),
      "Your Comment has been saved"),
    model:insert($POST)
  )
else
    web:redirect( "/app/model/add/",
    web:get-params($POST),
      "Please fix the following errors.")
```

Listing 10: Post-Redirect-Get pattern. A controller implementing the `PRG` pattern. If all validation criteria are met, the model's insert function is called and a redirect header is sent. In case the check fails, the user is redirected to the referring page.As the input parameters have been stored in `$GET`, a pre-populated form may be displayed to the user again, so he can fix the errors.

Despite suiting XQuery's processing model, the Post-Redirect-Get pattern comes with the advantage of avoiding duplicate form submission in cases when a user reloads the page that processed the `POST` input data.

3.4 Evaluation: Performance & Costs of the Glue Code

The BaseX Web framework is built directly on top of BaseX. As such, it directly benefits from, and depends on, the performance of BaseX'. In order to benchmark BaseX Web, we decided to judge the overall performance by the number of *complete transactions* per second. We chose not to measure the raw performance of the underlying database system, but the actual cost of interfacing the servlet engine with an XQuery processor. The results for the plain execution of an XQuery expression in the local mode of BaseX without concurrency is provided[21] to give an impression of what would technically be possible.

This synthetic benchmark is conducted with two application scenarios in mind: we benchmark a *static* view—i.e., no database interaction is used, and XQuery is only used to generate markup—and a view that involves database interaction in order to retrieve elements and tailor them to produce the desired output markup.

For each scenario, we benchmarked a total of 2.400 requests, increasing the number of concurrent users each time the benchmark was invoked. The number of concurrent users will be {1, 10, 20, 40, 80, 160}.

3.4.1 Benchmark Scenario

The benchmarks are conducted on a machine with the following configuration:

Hardware *iMac11,3*; Intel Core i3, 3,2 GHz; 8 GB Ram
Software *Mac OS 10.7.2*; Java 1.6.0_29; jetty 7.4.5.v20110725; BaseX Web 0.7.1, BaseX 7.1

The HotSpot™ `-server` optimizations are enabled, the maximum amount of useable memory is fixed to `4096MB` for all benchmarks. To warm up the JVM and operating system caches, all benchmarks are run once before measuring the actual time.

[21] the column BaseX depicts the number of requests a standalone instance of BaseX is able to handle

Baseline

Concurrency	# rq / sec
1	1475.00
10	2728.50
20	2414.81
40	2193.10
80	2401.56
160	1328.62

The baseline results are benchmarked against an empty[22] `index.html` document without any XQuery processing involved in order to obtain the maximum possible number of requests per/ second. Even with 160 concurrent requests, jetty handles the load well and serves around 1300 empty result pages per second.

Simple XQuery view

Conc.	# rq / sec	BaseX
1	409.82	5882.35
10	483.05	—
20	492.38	—
40	486.29	—
80	489.18	—
160	480.81	—

The *Simple XQuery* view evaluates the sequence `1 to 10` and embeds it into a layout template before sending it back to the user. The table shows the performance results for query evaluation, embedding of the results into the layout, and transferring the result page to the client. The maximum number of requests served is around 480 per second for 160 concurrent users.

XQuery view with Database Interaction

Conc.	# rq / sec	BaseX
1	334.39	2173.91
10	381.11	—
20	378.81	—
40	373.00	—
80	363.72	—
160	337.43	—

The *XQuery view with database interaction* evaluates a sequence of book elements, stored in the BaseX database, embeds it as `<li>{$bookname}</li>` into a template and sends it back. The table shows the performance for retrieving the sequence, converting it to an Html list, embedding the results into an HTML template, and transferring the result page to the client. The maximum number of requests served is 330 per second for 160 concurrent users.

[22] size zero

3.4.2 Result interpretation

While we have to pay a certain prize for integrating the XQuery processor into a Servlet's processing pipeline, the result of 300 requests per second shows that the overhead is surprisingly low, and could even be optimized if necessary.

There are two major bottlenecks. The first one is an external one: jetty *only* serves around 1.300 requests per second for the `NOOP` baseline results.

The second one, is more interesting with regards to BaseX Web: the actual costs introduced for parsing the URL, checking for the availability of the corresponding view and controller files and sending the complete query to the server. This process has to be repeated for every request and involves I/O on the machine running jetty. This process could be optimized by introducing a caching infrastructure, that holds the fully assembled query for each unique URL and its HTTP parameters. By doing so, we already avoid the I/O related to checking the presence of views and controllers. This approach could eventually even be extended to cache the query results instead of only assembled queries. This way, costly I/O could be avoided at both ends, the Servlet engine and inside the BaseX database management system.

BaseX-Web & Real World Data

In the following chapter, we will show the implementation of an application implemented in BaseX Web. The application will be designed around a corpus, provided in XML, which contains library data found in the University of Konstanz. We start by rapidly bootstrapping an expert retrieval system, which is particularly easy for a real-world scenario when using our framework. We will then evaluate BaseX Web in terms of performance and extensibility.

4 Application: Bootstrapping an Expert Retrieval System with BaseX Web

Under the patronage of *"Open Access"* universities and other public authorities are currently in the process of setting up *institutional repositories (IR)*. IRs collect research articles, journals and other intellectual output of an institution. They can be seen as *the* single source to access any digital asset an institution can provide. Among other things, IRs are responsible for the preservation and archivation of the stored data. The ultimate goal, however, is to provide public access to the material. Therefore, state-of-the-art software for running IRs is online-based to provide open access to a worldwide audience. Obviously a sophisticated retrieval system to tap the full potential of the digitally stored documents is crucial. On the foundation of our architecture kickstarting such a system is not only elegant, but also straightforward.

The Library of the University of Konstanz maintains an institutional repository, called *Konstanz Online Publication System (KOPS)*. In order to support public access it is run as an *Online Public Access Catalog (OPAC)* system. The open source software package *DSpace* provides the tools for management of digital assets and is the system behind KOPS. It is realized as a web application and therefore suited well to be re-implemented with the help of our architecture.

Our goal for this chapter is to demonstrate that a basic OPAC system can easily be implemented by XML database management system and its extensions. BaseX Web is the framework used to develop an alternative, lightweight XML OPAC system. Given the raw data from KOPS and nothing at hand than BaseX, BaseX FS [17] and BaseX Web, we strive to kickstart a performant expert retrieval system with significantly less implementation effort and architectural overhead than existing solutions.

4.1 KOPS - An Online Public Access Catalog

According to Babu and O'Brien [2] web-based online public access catalogues began to appear in the late 1990s. As catalogues, they demonstrate advances on traditional OPACs,

especially in terms of remote access by users and their potential to integrate many document types and sources via a single interface.

Web OPACs provide the following functionality [2]:

- The usual features of traditional OPACs such as,
  - storing bibliographic and sometimes full-text databases
  - providing direct access to a library's bibliographic database
  - providing instructional help
  - display of search results in readily understandable form
  - sometimes remote access from the library's location
  - information about community events
  - providing links to circulation files, reference help etc.
  - providing search through a variety of access points such as author, title, keyword, subject, periodical title, series, class number, ISSN or ISBN, etc.
- The ability to use hypertext links to facilitate navigation through bibliographic records
- A move towards emulation of the appearance and search features similar to those found in search engines
- Linking to full text when available
- Ability to help bring a convergence in searching of all electronic information available through one interface, e.g., catalogues, CD-ROMS, Internet sources etc.

The Library of the University of Konstanz provides such a web-based OPAC called "Konstanz Online-Publikations-System" (KOPS). The information platform is integrated in the Network of Open Access Repositories, an initiative to support "open access to knowledge in the sciences and humanities" [26].

With the help of KOPS:

- Members of the University can publish digital documents and make them available on the internet.
- An online search interface is available that supports simple and advanced keyword search options as well as full-text retrieval.

At the time of writing, KOPS contains 12,312 library entries. 4,163 entries were only library-oriented metadata with no document attached, and 8,149 were available with the complete full-text.

4.1.1 Bootstrap an XML-OPAC system

Our goal is to demonstrate that a modern XML database management system like BaseX is able to implement an expert retrieval system, such as KOPS, without ever leaving the unified X technology stack.

With the help of two extensions to the XML-DBMS, namely BaseX FS, which allows to uniformly express (arbitrary) file system content in XML [17], and BaseX Web, our proposed web application framework, we expect to realize a much more simple and lean system architecture. Conventional architectures (such as the before mentioned DSpace system) are constructed as a combination of different products (i.e., a relational database management system, a separate full-text index engine (such as Apache's Lucene) and many more). Programmers and administrators have to be experts in a multitude of subcomponents. Interdependencies have to be mastered and a lot of glue code, written in different languages, is necessary to finally construct the main system.

In our approach a major advantage is stemming from this fact that programmatic access to all system components (such as full-text indexes) is provided in a consistent and transparent way through a single language, XQuery. A core idea of BaseX Web is to establish an application framework in which no glue code is required. Impedance mismatch can easily be avoided since we operate on a single data model throughout the complete system stack (data is XML, business logic is XQuery, result presentation is XHTML).

Utilizing BaseX FS we are able to provide a low entry barrier for the data. We load information "as is" and have no need to define a data model in advance. Data is uniformly expressed in XML.

In the following, we will provide an overview of the steps required to kickstart a base system. It will later on be configured towards an online retrieval system.

4.1.2 Basic System Setup

Setting up the basic infrastructure is straight-forward. All we need is a server with Java installed and support for the Filesystem in Userspace (FUSE) framework. Next, we setup the BaseX database server and load the BaseX FS extension.

As mentioned the library data set consists of 8,149 publications. In principle, there are two ways to import them into the database system.

The first and simplest approach is to just make use of the generic BaseX FS' transducers described in [17].

The PDF transducer extracts content from the original regular file and aggregates metadata, annotations, full-text, and embedded images into a unified XML representation. The original file is stored as raw data as well. Performing a bulk load of the initial data is a three step procedure:

1. Use BaseX FS and mount an empty database as file system in userspace
2. Copy the original PDFs into the database/file system
3. Construct an XML view of the data by utilizing BaseX FS's transducers

The resulting database will contain a uniform view on the metadata, formerly available only to dedicated programs; it will allow us to handle these data in a standardized and generic way. Throughout the whole process, our mapping does not contain any information specific to our use case. Instead, we extract all *information as is*, leaving alone any assumptions which data we are going to need afterwards. Listing 11 shows how full-text of a document is stored in the database[1].

Librarians have manually processed documents in KOPS and valuable bibliographic metadata is available for each entry. Thanks to the extensible architecture of the BaseX FS transducer facility, we additionally can leverage such non-standard metadata. A specialized transducer can be plugged into the transducer chain in order to obtain the data as `opacinfo` (see Listing 12 on the next page).

[1] A more detailed database excerpt is shown in Appendix Listing 20 on page 69

```
    <folder name="fulltext">
      <folder name="pages">
        <folder name="page" number="1">
          <fact name="text">
Interactive exploration of fuzzy clusters using Neighborgrams
Michael R.Berthold — Bernd Wiswedel — David E.Patterson

Department of Computer and Information Science,University of Konstanz,Box M712,78457 Konstanz,Germany

Data Analysis ResearchLab,Tripos Inc.,USA

Abstract
We describe an interactive method to generate a set of fuzzy clusters for classes of interest of a
given,labeled data set.

The presented method is therefore best suited for applications where the focus of analysis
lies on a model for the minority class or for small  to medium-sized data sets.

The clustering algorithm creates one dimensional models of the neighborhood for a set of patterns
[…]
          </fact>
```

Listing 11: KOPS-FSML.xml: Extracted full-text from online resource.

```
[…]
<file name="1896748.pdf" suffix="pdf" st_size="533883">
  <folder name=".1896748.pdf.deepfs">
    <folder name="opacinfo">
      <fact name="pagecount">17</fact>
      <fact name="author">Berthold, Michael</fact>
      <fact name="author">Wiswedel, Bernd</fact>
      <fact name="author">Patterson, David E.</fact>
      <fact name="title">Interactive exploration of fuzzy clusters using Neighborgrams</fact>
      <fact name="town">Konstanz</fact>
      <fact name="publisher">Bibliothek der Universität Konstanz</fact>
      <fact name="year">2005</fact>
      <fact name="format">Online-Ressource</fact>
      <fact name="note">Article</fact>
      <fact name="signature">|004</fact>
      <fact name="language">Englisch</fact>
      <fact name="category">Informatik</fact>
      <fact name="url">http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-65525</fact>
      <fact name="creation-date">November 17, 2004 21:34:22 (UTC)</fact>
      <fact name="modification-date">October 13, 2008 14:42:40 (UTC +02:00)</fact>
    </folder>
```

Listing 12: KOPS-FSML.xml: Bibliographic metadata about online resource.

At this point, an XML view of the initial data set and the corresponding original files are available in the DBMS. The original KOPS data can now be queried and processed, using a standardized, declarative API written in XQuery. We can now proceed to configure our online retrieval application. Leveraging the BaseX Web application framework developers can use the database to analyze, search, and discover all kinds of information at various granularities since all assets are homogeneously represented inside the database.

4.1.3 Setting up a Project

The starting point for XQuery application development with BaseX Web is installing the application skeleton as depicted in Section 3.3.1 on page 31.

After this, we start by defining the user requests our system will respond to. Second we define the internal representation of retrieval results, such that we can transform them to XHTML afterwards. In contrast to its relational counterpart—as XQuery is a fully capable programming language—developers experience a much higher degree of expressiveness while at the same the need for scripting language glue code has been eliminated.

An example query conducting a search for all works that match a given `key,value` pattern is given in Listing 13. It returns a well-defined type, `element(file)*`, for further processing. The function accepts two arguments, a `key as xs:string` and its desired `value as xs:string`. Now that the search functionality is defined, as an XQuery function, we are ready to add its definition to a controller under the framework's supervision, located in `controllers/opac.xq`.

```
(: Search works matching a given key/value combination: :)
declare function opac:keyValue($key, $value){
  //file[.//fact[ ./@name eq  $key and . eq $value]]
};
```

Listing 13: `opac.xq` — A XQuery function returning all `file` elements matching a specific `key,value` combination.

To further extend the search functions—more elaborate examples will be given in Section 4.2—developers will have to do nothing but add more XQuery functions to the controller. Each defined function can also be called from other contexts. In general the con-

troller defines only functionalities relevant to the search process, not the final result representation.

We afterwards define how the resulting sequence of file elements[2] will be presented to the user in (X)HTML. To do so we provide the framework with a view. This view will not only request and transform the retrieval results, it is as well a unique, machine-accessible resource, which provides an interface to underlying data.

With respect to the definition of `opac:keyValue` we create a *view*, `views/opac/simple-search.xq` that:

- receives the search input via HTTP Parameters,
- invokes `opac:keyValue` to extract the relevant data from the database,
- iterates over the result sequence and transforms the file elements to browser-friendly (X)HTML

By convention our view is now accessible at `http://xmlopac/app/opac/simple-search`.

The resulting view code is depicted in Listing 14. This generic approach allows us to uniformly handle any sequence of file elements, as long as they share common characteristics, no matter what their origin was.

```
for $media in opac:keyValue($field, $value)
return
<div>
<h2>{$media//fact[@name eq "title"]/text()}</h2>
    <p>
    written by {$media//fact[@name eq "author"]/text()}
    on {$media//fact[@name eq "creationdate"]/text()}
    </p>
</div>
```

Listing 14: `simple-search.xq` — The result page view, invoking a controller function.

Figure 4.1 depicts how the components work together to form the basic infrastructure.

[2] i.e., `element(file)*`

FIGURE 4.1: The core components of the web architecture:
Model contains the complete data that has been extracted from KOPS.
View represents an URL and coordinates user requests to parametrized XQuery function calls.
Controller holds the logic necessary to retrieve and return the search results.

The screenshot shows the computed result rendered inside a browser when opening `http://xmlopac/app/opac/simple-search?field=author&value=Berthold,%20Michael`

4.2 Evaluation Setup

The following evaluation has been conducted in the scope of [17] and puts BaseX Web to the test. In the previous section we have shown how to quickly setup a base infrastructure to drive a search and retrieval system. We now want to put our system to the test and examine if it's equally fast when it comes to the evaluation of common search requests.

To conduct our real-world data study, we obtained a full dump of all data available online in KOPS and transformed it into an XML representation. As already shown, the resulting XML database instance contains all entries of the original data, the bibliographic metadata and, whenever available, the full-texts of the actual PDF documents.

Some key characteristics for the input data and the resulting database are displayed in Table 4.1.

Input statistics		**Index statistics**	
Size of input data	17 GB	Size of full-text index	614 MB
# files	8,149	# full-text index entries	1,984,734
# PDF pages	254,299	# XML nodes	3,671,331
# authors	25,793	# <fact/> elements	668,191

Table 4.1: Statistics on the original KOPS library resources and the thereof constructed database `kops-fsml.xml`

All queries were benchmarked against BaseX Version 7.1 with the following settings: `java -server -Xmx4096m -Xms1024m`. To make results more reliable, we restarted BaseX before each test, then every query was run 20 times to warm the caches. Next, the actual measurements were performed by running the query again for 10 times and storing the average response times, which include all evaluation steps (parsing, compilation, evaluation of the query, and serialization of the results).

4.3 Queries and Performance Results

4.3.1 Keyword Search

Due to its simplicity, keyword search has turned out to be one of the most dominant approaches for expressing one's information needs on the internet. Keywords are filled in by users into search fields, then matched against inverted indexes for an underlying text corpus, and all documents are returned that contain the keywords and potentially related terms. Stemming the text corpus or enhancing the full-text with thesauri and language specific features can be used to derive related terms.

As a consequence, high performance in keyword search scenarios is crucial for a system's acceptance, and the full-text extension of XQuery [10] provides a standardized way of formulating such requests for XML. A keyword search in XQuery Full-Text that retrieves relevant document files can, for example, be expressed as shown in Listing 15.

QUERY: Keyword search using XQuery Full-Text

```
let $words := ("problem", "properties")
return
//*[text() contains text {$words} all words ]/ancestor::file
```

Listing 15: A keyword search function for the OPAC XQuery module (`opac.xq`).

So as to benchmark the keyword search performance, we randomly selected 10 keywords of the text corpus and performed a keyword search for all possible combinations of those 10 words. Each query was run 10 times against a document corpus containing 250, 500, 1000, 2000, 4000 and 8000 source documents.

RESULTS. Table 4.2 on the facing page shows the runtime statistics for each of the six database instances. The results can be read as follows: For the largest database containing 8000 documents all $\binom{10}{2} = 45$ keyword search queries could be evaluated in a total time of 706.32ms. The fastest query took 8.92ms and the slowest 36.75ms. On average the evaluation could be performed in 15.70ms. Adding up all matching documents this results in a

total number of 27, 169 hits (the single number of hits for each query can be derived from Table 4.3).

Corpus size	250	500	1000	2000	4000	8000
Total time	12,04	30,55	63,59	164,67	367,61	706,32
Min	0,06	0,14	0,43	1,62	4,59	8,92
Max	0,66	2,28	5,33	9,07	14,09	36,75
AVG	0,27	0,68	1,41	3,66	8,17	15,70
Total # of Hits	709	1570	3135	6813	14037	27169

TABLE 4.2: Runtime statistics for the keyword search queries against six differently sized corpora.

Figure 4.2 is showing two graphs depicting these results again. For each of the six database instances on the abscissa it shows

(a) how many documents have been evaluated as matches and
(b) how much time this evaluation took on average in ms.

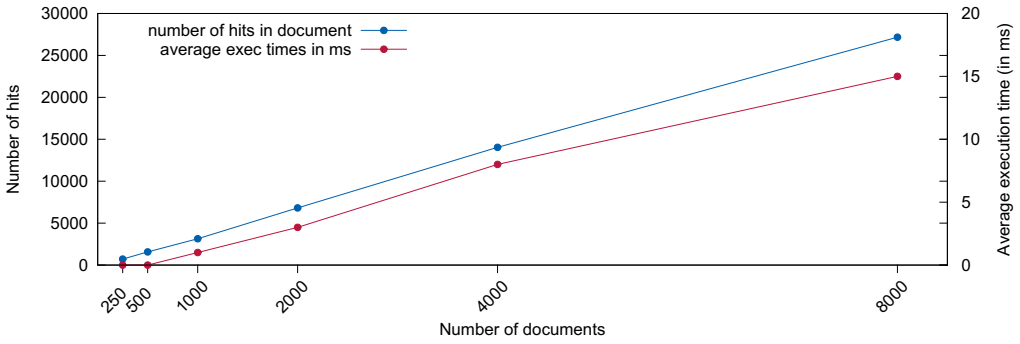

FIGURE 4.2: Average runtime in ms (red line/right y-axis) to evaluate 45 keyword queries on each of the six corpora (x-axis). Blue line/left y-axis shows the accumulated number of matching documents.

Detailed performance results for the 8000 source documents are shown in Table 4.3 on the next page. It depicts all 45 possible combinations of the keywords in question, and shows the absolute number of hits and the time needed for returning the results of particular keyword combinations.

	germany	problem	change	science	formation	situation	space	properties	material	power
germany	—	645 / 15.28	919 / 22.07	1387 / 36.75	976 / 20.06	438 / 12.26	554 / 13.17	1010 / 18.98	777 / 17.58	506 / 13.64
problem	—	—	813 / 21.14	669 / 17.13	417 / 11.59	795 / 21.61	724 / 21.57	625 / 16.98	438 / 12.53	508 / 15.12
change	—	—	—	860 / 21	842 / 20.49	666 / 18.04	628 / 17	863 / 21.41	541 / 14.84	612 / 18.25
science	—	—	—	—	799 / 18.55	338 / 9.72	492 / 13.07	793 / 17.95	393 / 10.73	483 / 13.34
formation	—	—	—	—	—	386 / 10.63	380 / 10.65	912 / 21.65	608 / 15.36	291 / 8.92
situation	—	—	—	—	—	—	338 / 10.24	371 / 10.07	376 / 10.66	321 / 10.53
space	—	—	—	—	—	—	—	628 / 17.54	314 / 9.09	404 / 11.65
properties	—	—	—	—	—	—	—	—	622 / 16.53	407 / 11.45
material	—	—	—	—	—	—	—	—	—	300 / 9.5
power	—	—	—	—	—	—	—	—	—	—

TABLE 4.3: Number of results and time for generating the results for a keyword search against the 8000 file database.

ANALYSIS: Due to the exploitation of the full-text indexes, all query runtimes scale linearly for the tested database instances. Index lookup itself is neglectable and the most limiting factor in terms of performance is the number of the results, as this determines the amount of data to be serialized. Hence, very large corpora may be searched yielding very fast response times. In our specific case, the slowest query, searching for the keywords *germany* and *science*, executes in 36.75ms on the largest corpora yielding 1.387 result documents.

4.3.2 Phrase Search

There are numerous cases when plain keyword searches alone are too limiting. Phrase search is a highly needed functionality for most current retrieval systems. It enables users to search for, e.g., compound names, terms and sentences containing words in a fixed order. Phrase searching removes *noise*, added by documents that contain the keywords but not necessarily in the order a user wanted.

Table 4.4 lists phrases of lengths two to five, which have been manually selected from the corpus. The phrases themselves consist of keywords that were chosen in a way such that their respective number of index entries covered a range from rare to frequent occurrence. The runtime statistics show a much higher variance than in the previous test case: query runtimes do not increase with the number of hits; an explanation will be given in the analysis. An XQuery script, shown in Listing 22 on page 71, has been used to generate the results in Table 4.4.

Query: Phrase search using XQuery Full-Text `//*[text() contains text 'with respect to' phrase]`

Results. Table 4.4 shows the conducted phrase searches and their average execution time for ten runs. For each phrase the number of matching nodes is given, and each phrase's keyword is listed with the number of occurrences in the Full-Text index.

	T(ms)	# matching node	chosen phrase, with number of index entries per keyword
Q 1	0.45	0	minor: 2218; drawback: 450
Q 2	1.25	2	major: 8553; deficiency: 368
Q 3	2.88	79	particularly: 4800; strong: 9900
Q 4	5.33	18	special: 5669; interest: 7380; group: 15147
Q 5	6.10	51	major: 8553; contribution: 4139
Q 6	11.10	593	Related: 17695; Work: 17362
Q 7	30.36	1107	Experimental: 12858; results: 36192
Q 8	42.57	2	Stabilisieren: 203; konnte: 18118; sich: 73862; dieses: 18674; System: 28553
Q 9	81.98	50	We: 53641; conclude: 2958; with: 102476
Q 10	167.86	48	I: 87473; would: 19880; like: 17708; to: 119519; express: 2142
Q 11	222.91	8571	with: 102476; respect: 10168; to: 119519
Q 12	248.23	5	major: 8553; advantage: 3319; of: 148306; our: 26799
Q 13	276.81	2949	As: 96236; shown: 23813; in: 228856
Q 14	367.56	5105	in: 228856; contrast: 12264; to: 119519

Table 4.4: Phrase searches: The average runtime per phrase query is shown. Queries 9–14 clearly stand out in terms of time taken.

Analysis. As shown in Table 4.4 more than half of the selected phrases evaluate in less than 50ms due to exploitation of the full-text index.

While most phrases are evaluated in interactive time, we were interested in the limits of the presented architecture, thus we considered the phrases (Q9-Q14) that took much longer than the other queries, c.f. Figure 4.3, for more thorough analysis.

As the results indicate, there is no direct relationship between the query times and the number of results. Instead, queries with large result sets may be evaluated fast while other, slower evaluated queries yield much smaller result sets.

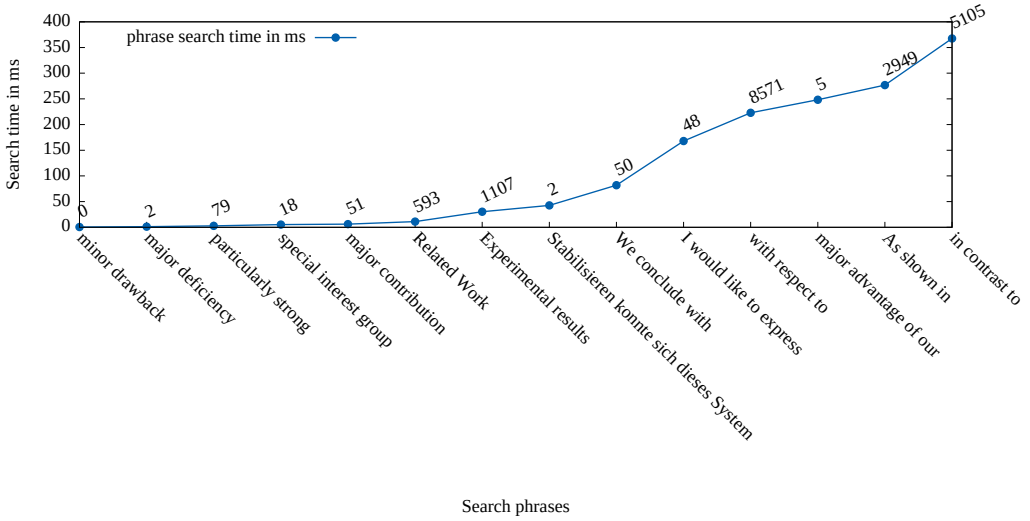

FIGURE 4.3: Phrase search: Result graph showing the average runtime needed to search for each phrase and the total number of matching nodes.

One general observation, which can be derived from the resulting times is that phrases containing a frequent word (i.e., *stopwords* that happen to occur frequently in natural language corpora or proper names) tend to be evaluated an order of a magnitude slower than phrases made up only of rare words.

Increasing the total number of keywords will pose an additional penalty on the runtime; as each keyword adds an intermediate result list to the workload.

This is mainly due to BASEX' evaluation strategy:

- all index hits for each keyword are evaluated as intermediate result lists containing `node ids` and
- subsequently merged in order to produce a complete result set.

Thus, in the worst case, lots of large intermediate result lists per keyword have to be sorted and merged, to often produce only very small final results.

Possible solutions to overcome the problems with large intermediate result sets could involve the following suggestions:

- The pipelining concept could be pushed down to the index access operation: instead of materializing all resulting node references in one run, they could be returned block-wise or one by one. This way, only those nodes will be requested that are actually required by a query, and retrieval can be skipped if it is clear that the remaining references will not be part of the final result.
- The pipelined retrieval could also be utilized to skip node retrieval whenever a query only requests parts of the result. As an example, a limitation to the first n results means that retrieval can be skipped as soon as those results have been evaluated.

4.3.3 Boolean Search

Boolean search is another basic technique supported by many retrieval systems, and considered in the context of this evaluation. It introduces the operators `AND`, `OR` plus `NOT`, which allow users to exclude or include terms, or combine them in an arbitrary fashion. These operators are commonly used to cut down result sizes and filter unwanted hits from result listings.

Query. Boolean search using XQuery Full-Text

```
//*[text() contains text "germany" ftand ftnot "problem"]/ancestor::file
```

Results Query results for the fully-sized OPAC corpus are depicted in Table 4.3.3 on the next page.

Analysis. When compared to the keyword search shown before, we see a *linear* degradation in performance. This again is explained by the fact that possibly large intermediate results will have to be merged by BaseX in order to produce the result set. Once more the queries perform fast enough regarding interactivity constraints. An upper bound in our example is set by query `change/\¬(problem)` yielding 3,486 result nodes in `175.13`ms.

w1∧¬(w2)	germany	problem	change	science	formation	situation	space	properties	material	power
germany	–	4574	4483	4168	4382	4731	4585	4202	4528	4673
		165.79 ms	154.87 ms	140.47 ms	141.89 ms	156.94 ms	139.52 ms	169.72 ms	140.8 ms	136.14 ms
problem	3251	–	3252	3306	3325	3321	3281	3286	3336	3348
	173.73 ms		175.13 ms	167.65 ms	170.35 ms	166.29 ms	152.81 ms	165.41 ms	163.97 ms	155.28 ms
change	3439	3486	–	3458	3414	3555	3504	3439	3528	3528
	159.77 ms	171.13 ms		156.73 ms	156.08 ms	162.56 ms	148.84 ms	153 ms	154 ms	144.54 ms
science	3786	4106	4023	–	4023	4204	4137	4014	4185	4176
	143.55 ms	162.19 ms	154.2 ms		143.71 ms	162.9 ms	138.74 ms	144.41 ms	145.08 ms	136.19 ms
formation	2734	2862	2781	2737	–	2904	2876	2745	2850	2894
	136.45 ms	153.72 ms	151.56 ms	133.34 ms		148.74 ms	126.66 ms	127.8 ms	128.21 ms	125.16 ms
situation	2768	2747	2714	2837	2790	–	2843	2796	2850	2859
	154.08 ms	156.02 ms	154.75 ms	152.14 ms	153.56 ms		138.8 ms	148.51 ms	142.76 ms	135.29 ms
space	2155	2176	2162	2176	2193	2262	–	2176	2221	2222
	125.63 ms	130.55 ms	128.62 ms	122.56 ms	119.82 ms	125.89 ms		115.86 ms	116.93 ms	108.28 ms
properties	2808	2974	2917	2895	2811	3064	2960	–	2956	3023
	142.32 ms	153.58 ms	146.36 ms	138.09 ms	130.64 ms	154.07 ms	126.4 ms		131.86 ms	128.93 ms
material	2375	2554	2503	2516	2457	2576	2581	2427	–	2576
	133.97 ms	148.37 ms	145.77 ms	133.83 ms	126.44 ms	136.36 ms	122.7 ms	126.19 ms		120.46 ms
power	2066	2085	2040	2063	2127	2155	2089	2088	2131	–
	121.55 ms	129.79 ms	122.01 ms	117.08 ms	117.49 ms	120.91 ms	105.56 ms	115.57 ms	112.28 ms	

TABLE 4.5: Boolean search performance results and hits. Each combination of two keywords has been executed against the database.

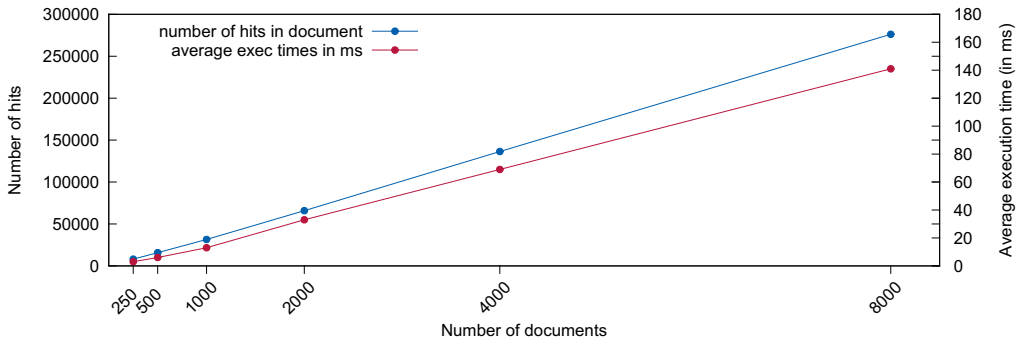

FIGURE 4.4: Average runtime for the boolean full-text queries, ran against six different sized corpora.

4.4 Summary

The previous observations have shown that the X-technology stack is ready to cope with state-of-the-art requirements and able to deliver retrieval infrastructure needed to build even complex systems. XML DBMS may be considered mature enough to drive production-ready retrieval system.

Yet XQuery with its various extensions is capable of delivering more than just state-of-the-art: due to the hierarchical nature of XML and its ability to contain structured as well as

unstructured data, users are able to exploit these characteristics in order to improve the relevance of their search results. The expressiveness of the language can be applied to numerous problems. To give an idea of what kind of questions may be answered, consider a search for documents, which contain the words "substrate" & "transformation" in a maximum distance of at most 4 words, followed by another page containing the word "compound".

```
(: Exploiting structural and textual proximity. :)
let $words:= ("substrate", "transformation"),
    $following := "compounds"
return
//*[ text() contains text {$words} distance at most 4 words
      and
      following-sibling::*/text() contains text {$following}
]/ancestor::file
```

Listing 16: XQuery example exploiting structure and textual proximity.

Skilled experts can steer the system from within a single language. XQuery gives transparent access to underlying system components such as the full-text engine (hardly to achieve in a traditional general-purpose system) and allows implementers to work directly on the underlying data.

All of this can be done in a single domain specific technology stack reducing the complexity of both, system components involved and technologies to be mastered by developers. BaseX Web sprung from the desire to develop graphical user interfaces and user applications that do not depend on the internal APIs of a database, such as, e.g., the native visualizations of BaseX.

5 Conclusion & Future Work

Implementing and designing BaseX Web was an interesting experience, first and foremost for actually working with XML instead of only using it in parts of the application.

Conclusion

The examinations and implementation ultimately led to an infrastructure that allows developers to build, deploy and run data-driven applications in a pure XML-technology stack. Development is conducted in a high-level, functional language, and provides fully capable database support. Compared to the status-quo, complex modeling decisions will not necessarily lead to a bloated system architecture, but instead still allow developers to concisely express what kind of data they are interested in.

BaseX Web became a powerful web application framework, ready to run expert search and retrieval systems on a uniform technology stack and eliminates the need for a multitude of different languages, paradigms and glue components. Besides, implementing web services, providing data for other applications, becomes a breeze: it only involves creating new views, which work with the very same XQuery controllers but serialize their results in XML or JSON instead of (X)HTML as before.

Kaufmann and Kossmann concluded their work "Developing an Enterprise Web Application in XQuery" with the words:

> "Today, the biggest concern in adopting this approach [*using the uniform W3C technology stack (author's note)*] is that there are no mature application servers available, but we believe that the situation will change soon in this regard. […] In the future, more experience with other applications [*others, than the evaluated demo application in the paper (author's note)*] is needed."
>
> Kaufmann and Kossmann in [21]

BaseX Web is our contribution regarding these concerns. Even though the framework still lacks a little polishing and some portions of convenience code, such as fully capable scaffolding, extension libraries to handle authentication and email, we are confident that implementing such features in XQuery modules will even broaden BaseX Web's application scenarios. BaseX Web—as a lightweight and highly extensible framework—has already proven its capabilities, in production and teaching.

Future Work

BaseX Web is also available as a public open source project on GitHub and contributors are welcome to join.

The further development will have to take place mainly in two areas: our primary goal is establishing more libraries to make day-to-day developer tasks easier. Sure enough setting up an application inside BaseX Web is easy, yet it requires profound knowledge of XQuery and XML technologies in order to get started right away. All of these libraries are to be developed as EXPath modules, and as such their use is not limited to BaseX Web. Such extensions will as well be beneficial for the overall feature richness of BaseX. The lack of extensions modules, for a rather young language like XQuery, is also very present in the community and actively addressed with efforts such as EXPath or EXQuery. As such open source implementers and commercial XQuery processors are highly interested in defining standard toolkits, ready to use even in different applications. For now, we are confident that eventually this problem will be solved.

The second area we are going to concentrate on, is a tighter integration of BaseX Web with BaseX' core APIs. Although momentarily the clear distinction of BaseX Web being only yet another database client has its conceptual merits, we do believe that an additional, more tight integration will be beneficial regarding performance. This might as well foster more synergies with BaseX' new RESTful XQuery API, proposed by Retter in [30], in order to build service-oriented applications. Joining this path of development one might even consider making our view completely passive and only inject data for a given view via annotations. Another area of interest is caching: as depicted in Section 2 on page 4 even in very dynamic scenarios we often face parts of a page that do not change. With this observa-

tions in mind we already ran first experiments, using memcached[1] and Project Voldemort[2] to provide server side caching. The results have already been promising and showed big performance gains for the scenarios benchmarked.

[1] http://memcached.org/
[2] a pure Java distributed key-value store http://project-voldemort.com/

```
<documents>
  <document>
    <title>Hello World</title>
    <paragraph>This  is the first paragraph</paragraph>
    <paragraph>This is the second paragraph</paragraph>
  </document>
  <document>
    <title>Hello Universe</title>
    <paragraph>This  is the first paragraph</paragraph>
    <paragraph>This is the second paragraph</paragraph>
  </document>
</documents>/child::document[child::title = "Hello World"]
```

Listing 17: XPath example, showing the hierarchical navigation capabalities. No shorthand notation has been used to more closely resemble the explanations given in Section 2.3.3 on page 15. `/child::document[child::title = "Hello World"]`

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="dob" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 18: XML Schema defining a simple model for a complex type person consisting of a name and a date of birth. Person is a complex type that aggregates the simple types `dob` and `name`.

```
module namespace web="http://basex.org/lib/web";
(:~
 : Sets the content type
 : @param $type the content type
:)
declare function web:content-type($type as xs:string){};
(:~
 : Disable caching for the current request.
:)
declare function web:no-cache(){};
(:~
 : Redirects the user to a given location,
 :        saves a message that is retrievable via web:flash()
 : @param $location URI to redirect to
 : @param $message system flash message
:)
declare function web:redirect($location as xs:string,(:…:)){};
(:~
 : Retrieves the message saved in the current
 : session flash cookie ($COOKIE('flash')) and deletes this cookie afterwards.
:)
declare function web:flash() { (:…:)};
(:~
 : Sets a cookie with the specified parameters.
 : @param $name the cookie name
 : @param $value the cookie value
 : @param $expires expires in seconds
 : @param $path the cookie path
:)
declare function web:set-cookie($name (:…:))  {};
(:~
 : Returns the cookie with name $name.
 : Wrapper for: $COOKIE($name)
 : @param $name name of the cookie
:)
declare function web:get-cookie($name) { (:…:)};
```

Listing 19: The XQuery web module

```
[…]
<file name="1896748.pdf" suffix="pdf" st_size="533883">
  <folder name=".1896748.pdf.deepfs">
    <folder name="opacinfo">
      <fact name="pagecount">17</fact>
      <fact name="author">Berthold, Michael</fact>
      <fact name="author">Wiswedel, Bernd</fact>
      <fact name="author">Patterson, David E.</fact>
      <fact name="title">Interactive exploration of fuzzy clusters using Neighborgrams</fact>
      <fact name="town">Konstanz</fact>
      <fact name="publisher">Bibliothek der Universität Konstanz</fact>
      <fact name="year">2005</fact>
      <fact name="format">Online-Ressource</fact>
      <fact name="note">Article</fact>
      <fact name="signature">|004</fact>
      <fact name="language">Englisch</fact>
      <fact name="category">Informatik</fact>
      <fact name="url">http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-65525</fact>
      <fact name="creation-date">November 17, 2004 21:34:22 (UTC)</fact>
      <fact name="modification-date">October 13, 2008 14:42:40 (UTC +02:00)</fact>
    </folder>
    <folder name="fulltext">
      <folder name="pages">
        <folder name="page" number="1">
          <fact name="text">
Interactive exploration of fuzzy clusters using Neighborgrams
Michael R.Berthold — Bernd Wiswedel — David E.Patterson

Department of Computer and Information Science,University of Konstanz,Box M712,78457 Konstanz,Germany

Data Analysis ResearchLab,Tripos Inc.,USA

Abstract
We describe an interactive method to generate a set of fuzzy clusters for classes of interest of a
given,labeled data set.

The presented method is therefore best suited for applications where the focus of analysis
lies on a model for the minority class or for small  to medium-sized data sets.

The clustering algorithm creates one dimensional models of the neighborhood for a set of patterns
[…]
          </fact>
        </folder>
        <folder name="page" number="2">
          <fact name="text">[…]</fact>
        </folder>
        […]
      </folder>
    </folder>
  </folder>
</file>
```

Listing 20: A single document inside the KOPS database.

```
<phrase>
  <search>minor drawback</search>
  <ms>0.45</ms>
  <hits>0</hits>
  <index>
    <ftcount name="minor">2218</ftcount>
    <ftcount name="drawback">450</ftcount>
  </index>
</phrase>
<phrase>
  <search>major deficiency</search>
  <ms>1.25</ms>
  <hits>2</hits>
  <index>
    <ftcount name="major">8553</ftcount>
    <ftcount name="deficiency">368</ftcount>
  </index>
</phrase>
<phrase>
  <search>Stabilisieren konnte sich dieses System</search>
  <ms>42.57</ms>
  <hits>2</hits>
  <index>
    <ftcount name="Stabilisieren">203</ftcount>
    <ftcount name="konnte">18118</ftcount>
    <ftcount name="sich">73862</ftcount>
    <ftcount name="dieses">18674</ftcount>
    <ftcount name="System">28553</ftcount>
  </index>
</phrase>
<phrase>
  <search>major advantage of our</search>
  <ms>248.23</ms>
  <hits>5</hits>
  <index>
    <ftcount name="major">8553</ftcount>
    <ftcount name="advantage">3319</ftcount>
    <ftcount name="of">148306</ftcount>
    <ftcount name="our">26799</ftcount>
```

Listing 21: XML fragment depicting the results of the benchmarks in Section 4.3.2 on page 56.

```
declare function local:find($p as xs:string){
  //*[text() contains text {$p} phrase ]
}
let $phrases:= ("minor drawback",
  "major deficiency",
  "major contribution",
  "particularly strong",
  "special interest group",
  "Related Work",
  "Experimental results",
  "Stabilisieren konnte sich dieses System",
  "We conclude with",
  "I would like to express",
  "major advantage of our",
  "with respect to",
  "As shown in",
  "in contrast to"
)

for $phrase in $phrases
  let
    $hits  := count(local:find($phrase)),
    $ms    := util:ms(local:find($phrase))
  order by $hits
return <phrase>{
  <search>{$phrase}</search>,
  <ms>{$ms}</ms>,
  <hits>{$hits}</hits>,
  <index>{
    for $w in tokenize($phrase," ")
    return
      <ftcount name="{$w}">{
        count(db:fulltext(., $w))
      }</ftcount>
  }</index>
}</phrase>
```

Listing 22: Functions to benchmark the Phrase Search performance.

List of listings

List of Figures

Bibliography

[1] **28msec**, *Sausalito: XQuery in the Cloud*, 2012. [Online]. Available: http://www.28msec.com/documentation/overview (visited on 01/20/2012).

[2] **R. Babu and A. O'Brien**, "Web OPAC interfaces: an overview," in *The Electronic Library*, vol. 18, 2010, pp. 316–330. DOI: 10.1108/02640470010354572.

[3] **A. Berglund, S. Boag, D. Chamberlin, M. Fernández, M. Kay, J. Robie, and J. Siméon**, *XML Path Language (XPath) 2.0*, 2010. (visited on 01/13/2012).

[4] **A. Berglund, M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh**, *XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition)*, 2002. (visited on 01/13/2012).

[5] **T. Berners-Lee and R. Cailliau**, "Worldwideweb: proposal for a hypertext project," *European Particle Physics Laboratory (CERN)*, 1990. [Online]. Available: http://www.w3.org/Proposal.html.

[6] **T. Berners-Lee and N. Mendelsohn**, "The rule of least power," *World Wide Web Consortium, TAG Finding*, 2006. [Online]. Available: http://www.w3.org/2001/tag/doc/leastPower-2006-02-23.

[7] **M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska**, "Building a database on S3," in *SIGMOD Conference*, **J. T.-L. Wang**, Ed., ACM, 2008, pp. 251–264, ISBN: 978-1-60558-102-6.

[8] **T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau**, *Extensible markup language (XML) 1.0*, 2008. [Online]. Available: http://www.w3.org/TR/REC-xml/ (visited on 01/11/2012).

[9] **S. Burbeck**, "Applications programming in smalltalk-80 (tm): How to use model-view-controller (mvc), 1987," 1987. [Online]. Available: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

[10] **P. Case, M. Dyck, M. Holstege, S. Amer-Yahia, C. Botev, S. Buxton, J. Doerre, J. Melton, M. Rys, and J. Shanmugasundaram**, *XQuery and XPath Full Text 1.0*, 2011. [Online]. Available: http://www.w3.org/TR/xpath-full-text-10/ (visited on 01/11/2012).

[11] **D. C. Fallside and P. W. S. Edition)**, *XML Schema Part 0: Primer Second Edition*, 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-0/ (visited on 01/11/2012).

[12] **D. C. Fallside and P. Walmsley**, *XML Schema Part 0: Primer Second Edition*, 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-0/ (visited on 11/28/2011).

[13] **M. Fowler**, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002, ISBN: 0321127420.

[14] **Garrett, J.J. and others**, *Ajax: A New Approach to Web Applications*, 2005. [Online]. Available: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications (visited on 2005).

[15] **F. Georges**, *EXPath - Standards for Portable XPath Extensions*, 2011. [Online]. Available: http://expath.org (visited on 10/04/2011).

[16] **C. Grün**, "Storing and Querying Large XML Instances," PhD thesis, Universität Konstanz, 2010.

[17] **A. Holupirek**, "Declarative access to filesystem data, New application domains for XML database management systems," Ph.D. Thesis, University of Konstanz, Germany, 2012.

[18] **C. Ireland, D. Bowers, M. Newton, and K. Waugh**, "A classification of object-relational impedance mismatch," in *Proceedings of the 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications*, Washington, DC, USA: IEEE Computer Society, 2009, pp. 36–43, ISBN: 978-0-7695-3550-0. DOI: 10.1109/DBKDA.2009.11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1545012.1545492.

[19] **M. Jazayeri**, "Some trends in web application development," *2007 Future of Software Engineering*, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1254719.

[20] **M. D. J. S. Jonathan Robie Don Chamberlin**, *Xquery 3.0: an xml query language*, 2011. [Online]. Available: http://www.w3.org/TR/2011/WD-xquery-30-20111213/ (visited on 01/11/2012).

[21] **M. Kaufmann and D. Kossmann**, "Developing an Enterprise Web Application in XQuery," in *ICWE*, **M. Gaedke, M. Grossniklaus, and O. Díaz**, Eds., ser. Lecture Notes in Computer Science, vol. 5648, Springer, 2009, pp. 465–468, ISBN: 978-3-642-02817-5.

[22] **M. Keith, M. Schnicariol, M. Keith, and M. Schnicariol**, "Chapter: introduction," in *Pro JPA 2*, Apress, 2010, pp. 1–16, ISBN: 978-1-4302-1957-6. DOI: 10.1007/978-1-4302-1957-6_1.

[23] **P Kilpeläinen**, "Using XQuery for problem solving," *SOFTWARE—PRACTICE AND EXPERIENCE*, vol. Manuscript to appear in Software - Practice and Experience, Apr. 2011. [Online]. Available: http://www.cs.uku.fi/~kilpelai/RDK11/exercises/Ex8Files/xqueryProblems.pdf.

[24] **M. Laverdet**, *XHP: A New Way to Write PHP*, 2010. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/xhp-a-new-way-to-write-php/294003943919 (visited on 10/02/2010).

[25] **A. Leff and J. T. Rayfield**, *Web-Application Development Using the Model/View/Controller Design Pattern*. IEEE Computer Society, Sep. 2001, ISBN: 0-7695-1345-X. [Online]. Available: http://dl.acm.org/citation.cfm?id=645344.650161.

[26] **N. Lossau**, "Der Begriff »Open Access«," *Open Access. Chancen und Herausforderungen—ein Handbuch—, Bonn*, pp. 18–22, 2007. [Online]. Available: http://www.unesco.de/fileadmin/medien/Dokumente/Kommunikation/Handbuch_Open_Access.pdf (visited on 02/11/2012).

[27] **J. McCarthy and S. Krishnamurthi**, "Interaction-safe state for the web," *Scheme and Functional Programming*, 2006.

[28] **W. Meier**, "eXist: An Open Source Native XML Database," in *Web, Web-Services, and Database Systems*, **A. Chaudhri, M. Jeckle, E. Rahm, and R. Unland**, Eds., vol. 2593, ser. Lecture Notes in Computer Science, 10.1007/3-540-36560-5_13, Springer Berlin / Heidelberg, 2003, pp. 169–183, ISBN: 978-3-540-00745-6. [Online]. Available: http://dx.doi.org/10.1007/3-540-36560-5_13.

[29] **T. Reenskaug**, "Models - views - controllers," Xerox PARC, Tech. Rep., 1979. (visited on 08/11/2011).

[30] **A. Retter**, "RESTful XQuery," *XML Prague 2012*, p. 91, 2012. [Online]. Available: http://www.xmlprague.cz/2012/files/xmlprague-2012-proceedings.pdf.

[31] **T. C. Shan and W. W. Hua**, "Taxonomy of Java Web Application Frameworks," vol. 0, Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 378–385, ISBN: 0-7695-2645-4. DOI: http://doi.ieeecomputersociety.org/10.1109/ICEBE.2006.98.

[32] **R. Singh and H. Sarjoughian**, "Software Architecture for Object-Oriented Simulation Modeling and Simulation Environments: Case Study and Approach," TR-009, Computer Science & Engineering Dept., Arizona State University, Tempe, AZ, Tech. Rep., 2003.

[33] **Strobe Inc.**, *Sproutcore - about*, 2011. [Online]. Available: http://sproutcore.com/about/ (visited on 12/26/2011).