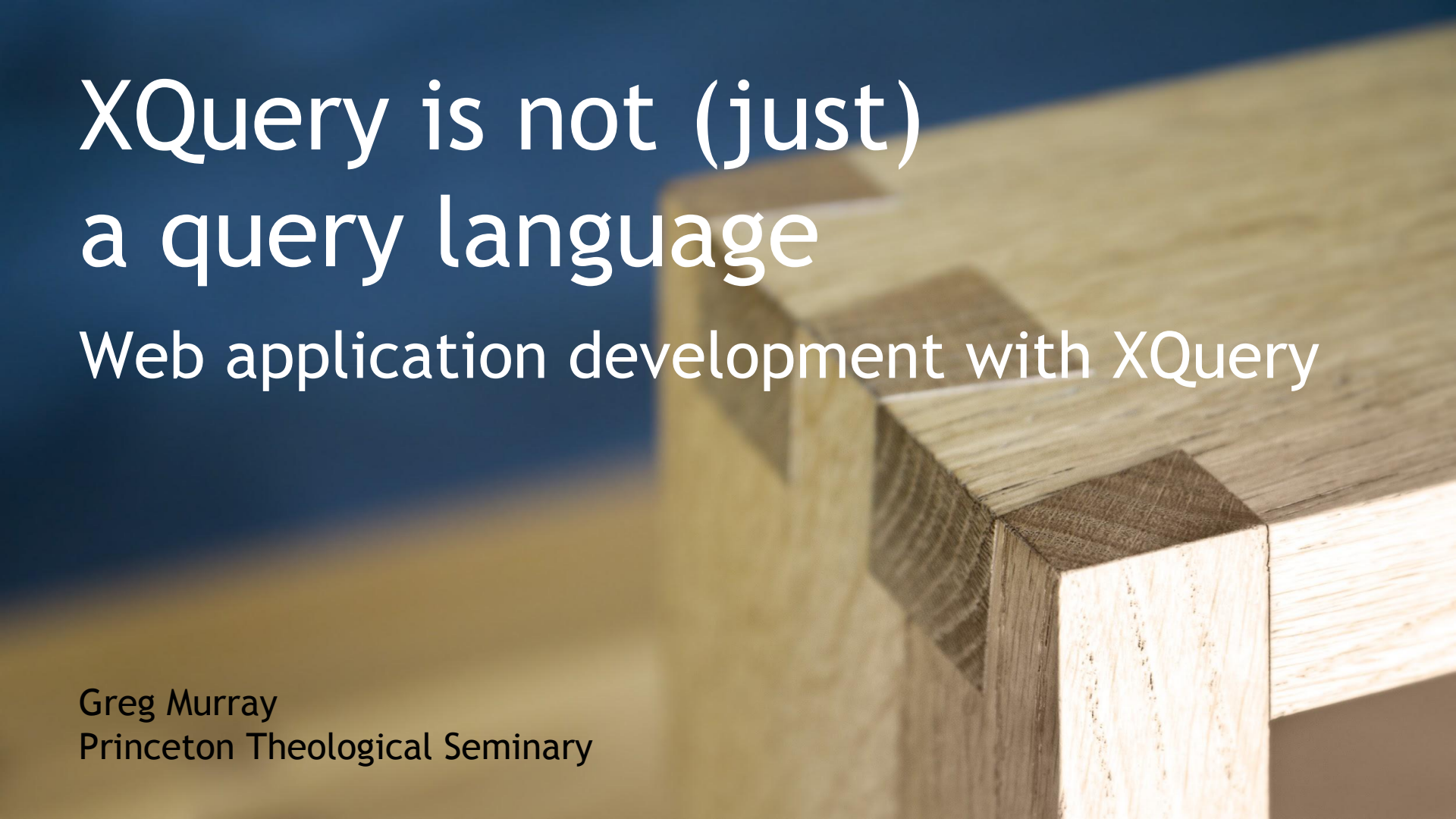
XQuery is not (just) a query language

Web application development with XQuery



Greg Murray
Princeton Theological Seminary

Limitations of the SQL analogy

Limitations of the SQL analogy

XQuery : XML databases :: SQL : relational databases

Limitations of the SQL analogy

XQuery : XML databases :: SQL : relational databases

Widely accepted: “XQuery is a query language for XML”

Limitations of the SQL analogy

XQuery : XML databases :: SQL : relational databases

Widely accepted: “XQuery is a query language for XML”

The truth but not the whole truth

Limitations of the SQL analogy

XQuery : XML databases :: SQL : relational databases

Widely accepted: “XQuery is a query language for XML”

The truth but not the whole truth

XQuery is *both* a query language *and* a programming language

Limitations of the SQL analogy

XQuery : XML databases :: SQL : relational databases

Widely accepted: “XQuery is a query language for XML”

The truth but not the whole truth

XQuery is *both* a query language *and* a programming language

Functional language, all benefits of such languages, e.g. no side effects

Limitations of the SQL analogy

When data is XML, can build entire web applications using XQuery as server-side programming language

Limitations of the SQL analogy

When data is XML, can build entire web applications using XQuery as server-side programming language

Requires implementation-provided extensions

Limitations of the SQL analogy

When data is XML, can build entire web applications using XQuery as server-side programming language

Requires implementation-provided extensions (but multiple widely adopted implementations offer them)

Limitations of the SQL analogy

When data is XML, can build entire web applications using XQuery as server-side programming language

Requires implementation-provided extensions (but multiple widely adopted implementations offer them)

Using HTML5 (not limited to XHTML), CSS, JavaScript

Limitations of the SQL analogy

When data is XML, can build entire web applications using XQuery as server-side programming language

Requires implementation-provided extensions (but multiple widely adopted implementations offer them)

Using HTML5 (not limited to XHTML), CSS, JavaScript

(Not addressing XQuery in the browser here)

Advantages of XQuery

for web delivery of XML content

Advantages of XQuery

XQuery works with XML *natively*

Advantages of XQuery

XQuery works with XML *natively*

Classic technology stack for web development: relational database, accessed via SQL, paired with a given programming language, usually object-oriented

Advantages of XQuery

XQuery works with XML *natively*

Classic technology stack for web development: relational database, accessed via SQL, paired with a given programming language, usually object-oriented

Requires “object-relational mapping”

Advantages of XQuery

XQuery works with XML *natively*

Classic technology stack for web development: relational database, accessed via SQL, paired with a given programming language, usually object-oriented

Requires “object-relational mapping”

Database (relational) and language (OO) have different data models

Advantages of XQuery

XQuery works with XML *natively*

Classic technology stack for web development: relational database, accessed via SQL, paired with a given programming language, usually object-oriented

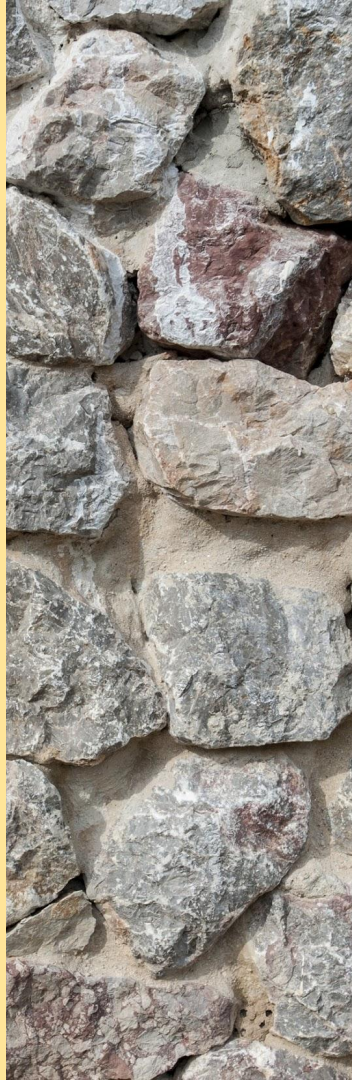
Requires “object-relational mapping”

Database (relational) and language (OO) have different data models

Inherent “impedance mismatch” (Kaufmann; Seiferle) — misalignment/gap between data and language

Advantages of XQuery

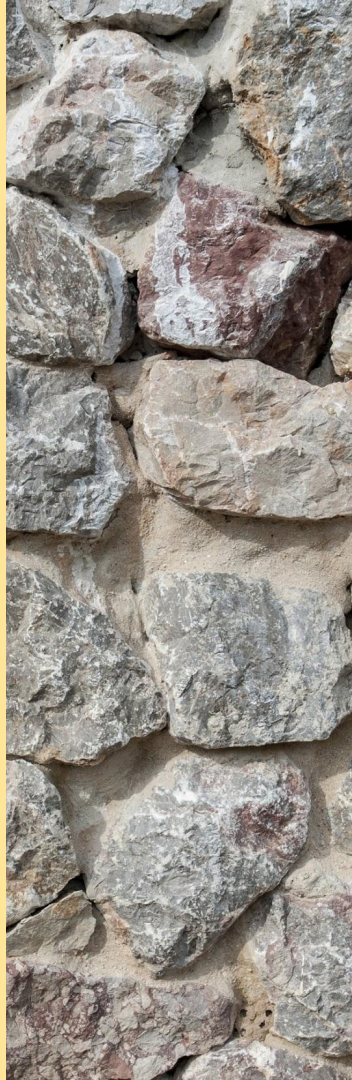
Even with XML data, similar mismatch between XML and OO programming languages



Advantages of XQuery

Even with XML data, similar mismatch between XML and OO programming languages

Object-*document* mapping (not object-relational)

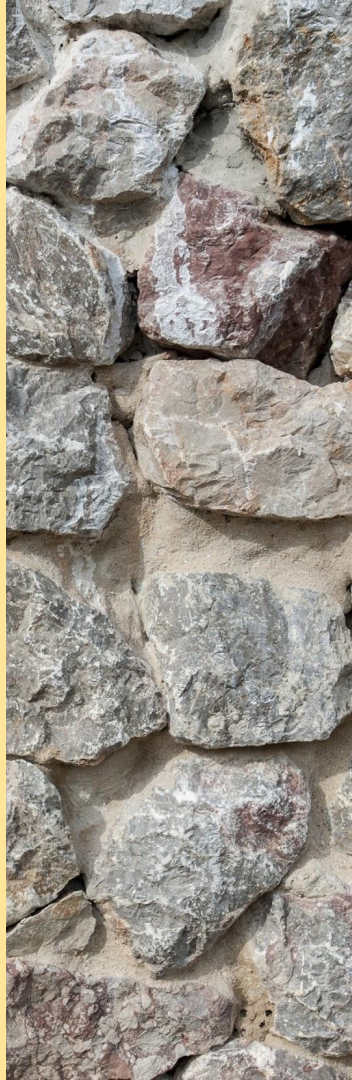


Advantages of XQuery

Even with XML data, similar mismatch between XML and OO programming languages

Object-*document* mapping (not object-relational)

Still a mismatch between data (XML) and language (OO)



Advantages of XQuery

With XML + XQuery, no such mismatch between data and language

Advantages of XQuery

With XML + XQuery, we sidestep entire layers of complexity

Advantages of XQuery

With XML + XQuery, we sidestep entire layers of complexity

Unified, stable technology stack

Advantages of XQuery

With XML + XQuery, we sidestep entire layers of complexity

Unified, stable technology stack

Inherently appealing conceptually, but also has practical benefits

Advantages of XQuery

With XML + XQuery, we sidestep entire layers of complexity

Unified, stable technology stack

Inherently appealing conceptually, but also has practical benefits

No glue code, fewer moving parts, less time on maintenance

Advantages of XQuery

With XML + XQuery, we sidestep entire layers of complexity

Unified, stable technology stack

Inherently appealing conceptually, but also has practical benefits

No glue code, fewer moving parts, less time on maintenance

Thus more time for meaningful coding (developing new or improved functionality, intentional refactoring)

Web application development with XQuery

Limitations of the SQL analogy

Advantages of XQuery for web delivery of XML content

Techniques for web application development with XQuery

Techniques for web application development with XQuery

Utilizing MVC

Techniques: Utilizing MVC

MVC (model-view-controller) wildly popular in web development since advent of Ruby on Rails

Techniques: Utilizing MVC

MVC (model-view-controller) wildly popular in web development since advent of Ruby on Rails

Frameworks for other languages followed suit; nearly universally practiced

Techniques: Utilizing MVC

MVC (model-view-controller) wildly popular in web development since advent of Ruby on Rails

Frameworks for other languages followed suit; nearly universally practiced

Popular because so effective for development/maintenance of complex applications

Techniques: Utilizing MVC

MVC (model-view-controller) wildly popular in web development since advent of Ruby on Rails

Frameworks for other languages followed suit; nearly universally practiced

Popular because so effective for development/maintenance of complex applications

MVC just as indispensable for web development in XQuery as for any other language

Techniques: Utilizing MVC

Problem: confusion surrounding MVC

Techniques: Utilizing MVC

Problem: confusion surrounding MVC

Very general as a concept, can be applied variously in practice

Techniques: Utilizing MVC

Problem: confusion surrounding MVC

Very general as a concept, can be applied variously in practice

Differing methodologies have proliferated, making MVC seem complicated

Techniques: Utilizing MVC

Problem: confusion surrounding MVC

Very general as a concept, can be applied variously in practice

Differing methodologies have proliferated, making MVC seem complicated

MVC often called an “architecture,” but that term is overburdened, unhelpful

Techniques: Utilizing MVC

Much of this confusion is unnecessary

Techniques: Utilizing MVC

Much of this confusion is unnecessary

In essence, MVC is just a technique for organizing code

Techniques: Utilizing MVC

Much of this confusion is unnecessary

In essence, MVC is just a technique for organizing code

Goal is classic “separation of concerns”

Techniques: Utilizing MVC

Models know how data is represented (modeled), handle database interaction

Techniques: Utilizing MVC

Models know how data is represented (modeled), handle database interaction

Views construct presentation of data

Techniques: Utilizing MVC

Models know how data is represented (modeled), handle database interaction

Views construct presentation of data

Controllers receive input and respond accordingly

Techniques: Utilizing MVC

Models know how data is represented (modeled), handle database interaction

Views construct presentation of data

Controllers receive input and respond accordingly, utilizing models and views

Techniques: Utilizing MVC

Models know how data is represented (modeled), handle database interaction

Views construct presentation of data

Controllers receive input and respond accordingly, utilizing models and views

A more complex application may merit a more complex use of MVC components (multiple controllers and/or multiple models)

Techniques: Utilizing MVC

Models know how data is represented (modeled), handle database interaction

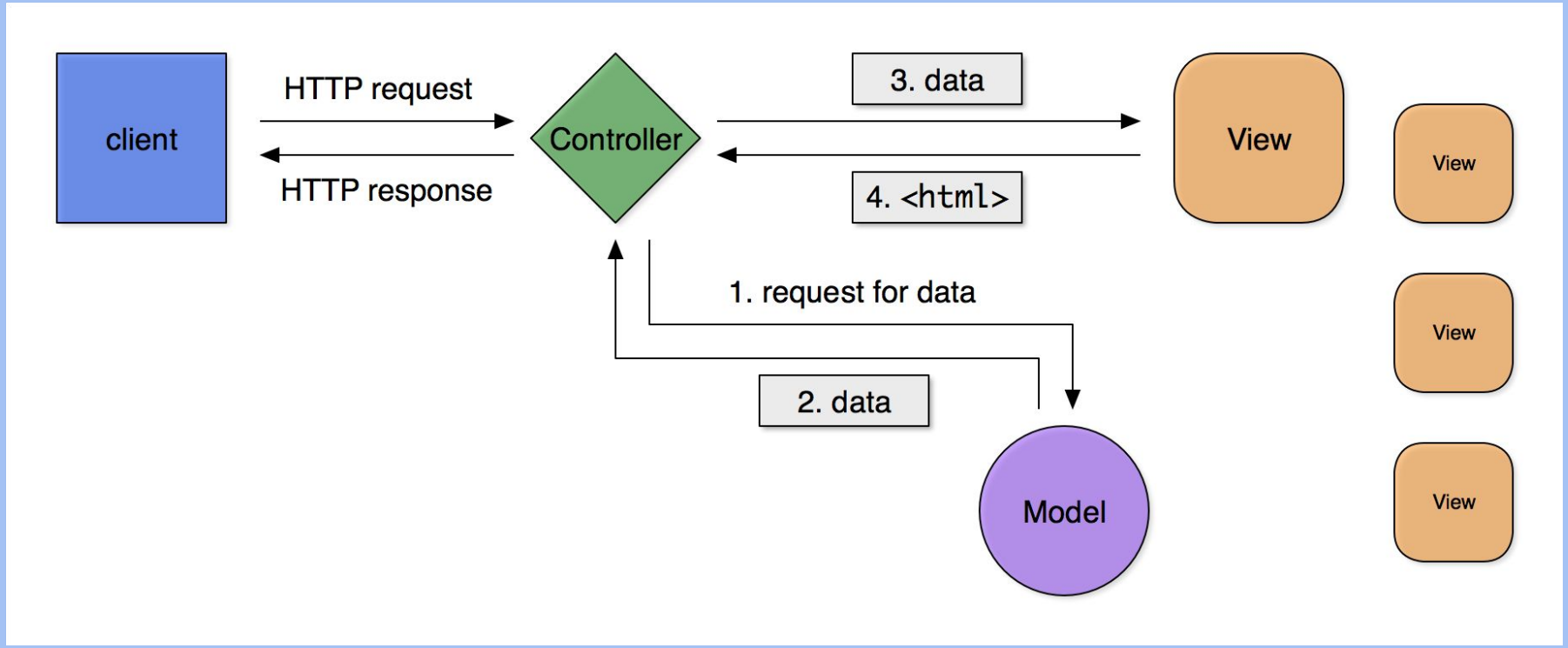
Views construct presentation of data

Controllers receive input and respond accordingly, utilizing models and views

A more complex application may merit a more complex use of MVC components (multiple controllers and/or multiple models)

But nothing wrong with having one controller, one model, a view for each page

Techniques: Utilizing MVC



Techniques: Utilizing MVC

Modular nature of XQuery lends itself easily/directly to MVC

Techniques: Utilizing MVC

Modular nature of XQuery lends itself easily/directly to MVC:

Each model, view, and controller is implemented as an XQuery library module

Techniques for web application development with XQuery

Utilizing MVC

Model

Techniques: MVC: Model

Model handles database interaction

Techniques: MVC: Model

Model handles database interaction

No other MVC component should create, read, update or delete data

Techniques: MVC: Model

Model handles database interaction

No other MVC component should create, read, update or delete data (no CRUD outside the model)

Techniques: MVC: Model

Model handles database interaction

No other MVC component should create, read, update or delete data (no CRUD outside the model)

But with XQuery, we have an additional consideration:

Techniques: MVC: Model

Model handles database interaction

No other MVC component should create, read, update or delete data (no CRUD outside the model)

But with XQuery, we have an additional consideration:

Whether the model should handle all navigation of structure of XML documents

Techniques: MVC: Model

Model handles database interaction

No other MVC component should create, read, update or delete data (no CRUD outside the model)

But with XQuery, we have an additional consideration:

Whether the model should handle all navigation of structure of XML documents

That is, whether only the model should contain XPath expressions

Techniques: MVC: Model

Example: search → search results → “item” view

Techniques: MVC: Model

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <id>a1</id>
  <title>Geschichte des dreissigjährigen Krieges</title>
  <name>Gindely, Anton, 1829-1892</name>
  <date>1869</date>
  <language code="de">German</language>
  <class>History</class>
  <topics>
    <topic>Thirty Years' War, 1618-1648</topic>
  </topics>
</metadata>
```

Techniques: MVC: Model

Example: search → search results → “item” view

In item view, display all bibliographic metadata

Techniques: MVC: Model

In the item view:

```
module namespace item = "http://balisage.net/ns/Bal2016murr0319/views/item";

import module namespace m = "http://balisage.net/ns/Bal2016murr0319/model" at "...";

declare function item:show($id as xs:string) as element(div)
{
  ...
  (: get specified document from database :)
  let $doc := m:get-xml-doc($id)
  (: pull value for book title :)
  let $title := fn:string($doc/metadata/title)
  ...
};
```

Techniques: MVC: Model

In the item view:

```
module namespace item = "http://balisage.net/ns/Bal2016murr0319/views/item";

import module namespace m = "http://balisage.net/ns/Bal2016murr0319/model" at "...";

declare function item:show($id as xs:string) as element(div)
{
  ...
  (: get specified document from database :)
  let $doc := m:get-xml-doc($id)
  (: pull value for book title :)
  let $title := fn:string($doc/metadata/title)
  ...
};
```

Techniques: MVC: Model

In the item view:

```
module namespace item = "http://balisage.net/ns/Bal2016murr0319/views/item";

import module namespace m = "http://balisage.net/ns/Bal2016murr0319/model" at "...";

declare function item:show($id as xs:string) as element(div)
{
  ...
  (: get specified document from database :)
  let $doc := m:get-xml-doc($id)
  (: pull value for book title :)
  let $title := fn:string($doc/metadata/title)
  ...
};
```

Techniques: MVC: Model

In the item view:

```
module namespace item = "http://balisage.net/ns/Bal2016murr0319/views/item";

import module namespace m = "http://balisage.net/ns/Bal2016murr0319/model" at "...";

declare function item:show($id as xs:string) as element(div)
{
   ...
   (: get specified document from database :)
   let $doc := m:get-xml-doc($id)
   (: pull value for book title :)
   let $title := fn:string($doc/metadata/title)
   ...
};
```

Techniques: MVC: Model

In the item view:

```
module namespace item = "http://balisage.net/ns/Bal2016murr0319/views/item";

import module namespace m = "http://balisage.net/ns/Bal2016murr0319/model" at "...";

declare function item:show($id as xs:string) as element(div)
{
  ...
  (: get specified document from database :)
  let $doc := m:get-xml-doc($id)
  (: pull value for book title :)
  let $title := fn:string($doc/metadata/title)
  ...
};
```

Techniques: MVC: Model

Option 1: Get book title directly using XPath

```
module namespace item = "http://balisage.net/ns/Bal2016murr0319/views/item";

import module namespace m = "http://balisage.net/ns/Bal2016murr0319/model" at "...";

declare function item:show($id as xs:string) as element(div)
{
  ...
  (: get specified document from database :)
  let $doc := m:get-xml-doc($id)
  (: pull value for book title :)
  let $title := fn:string($doc/metadata/title)
  ...
};
```

Techniques: MVC: Model

Option 2: Call model to get book title

```
module namespace item = "http://balisage.net/ns/Bal2016murr0319/views/item";

import module namespace m = "http://balisage.net/ns/Bal2016murr0319/model" at "...";

declare function item:show($id as xs:string) as element(div)
{
  ...
  (: get specified document from database :)
  let $doc := m:get-xml-doc($id)
  (: pull value for book title :)
  let $title := m:get-book-title($doc)
  ...
};
```

Techniques: MVC: Model

Example: search → search results → “item” view

In item view, display book title

Trivial example, but in real-world applications XPath expressions will be:

Techniques: MVC: Model

Example: search → search results → “item” view

In item view, display book title

Trivial example, but in real-world applications XPath expressions will be:

- More elaborate

Techniques: MVC: Model

Example: search → search results → “item” view

In item view, display book title

Trivial example, but in real-world applications XPath expressions will be:

- More elaborate
- Distributed throughout presentation code across multiple XQuery modules

Techniques: MVC: Model

Example: search → search results → “item” view

In item view, display book title

Trivial example, but in real-world applications XPath expressions will be:

- More elaborate
- Distributed throughout presentation code across multiple XQuery modules
- Same XPath expressions needed more than once

Techniques: MVC: Model

When knowledge of XML document structure is confined to model, if markup changes, only model code has to be refactored

Techniques: MVC: Model

When knowledge of XML document structure is confined to model, if markup changes, only model code has to be refactored

Cleaner separation of concerns

Techniques: MVC: Model

When knowledge of XML document structure is confined to model, if markup changes, only model code has to be refactored

Cleaner separation of concerns

Separation → practical benefits: flexibility, less maintenance

Techniques: MVC: Model

Use cases for multiple models

Techniques: MVC: Model

Use cases for multiple models:

When application works with XML content in more than one markup language, could have a model for each

Techniques: MVC: Model

Use cases for multiple models:

When application works with XML content in more than one markup language, could have a model for each

When application includes full-text searching, could group search-related functions into a separate model

Techniques for web application development with XQuery

Utilizing MVC

View

Techniques: MVC: View

View provides the presentation

Techniques: MVC: View

View provides the presentation

Example: search → search results → item page

Techniques: MVC: View

View provides the presentation

Example: search → search results → item page: each is a view, each is a separate XQuery module

Techniques: MVC: View

View provides the presentation

Example: search → search results → item page: each is a view, each is a separate XQuery module

A view can be for human use (HTML)

Techniques: MVC: View

View provides the presentation

Example: search → search results → item page: each is a view, each is a separate XQuery module

A view can be for human use (HTML), or machine use (XML or JSON)

Techniques for web application development with XQuery

Utilizing MVC

Controller

Techniques: MVC: Controller

Controller receives input, responds accordingly

Techniques: MVC: Controller

Controller receives input, responds accordingly

In a web application, controller accesses input from HTTP request (URL parameters, form data, request headers, cookies)

Techniques: MVC: Controller

Controller receives input, responds accordingly

In a web application, controller accesses input from HTTP request (URL parameters, form data, request headers, cookies), then calls the appropriate view

Techniques: MVC: Controller

Controller receives input, responds accordingly

In a web application, controller accesses input from HTTP request (URL parameters, form data, request headers, cookies), then calls the appropriate view

XQuery doesn’t provide built-in functions for such HTTP-specific matters

Techniques: MVC: Controller

Controller receives input, responds accordingly

In a web application, controller accesses input from HTTP request (URL parameters, form data, request headers, cookies), then calls the appropriate view

XQuery doesn’t provide built-in functions for such HTTP-specific matters, so implementation has to supply them

Techniques: MVC: Controller

Controller receives input, responds accordingly

In a web application, controller accesses input from HTTP request (URL parameters, form data, request headers, cookies), then calls the appropriate view

XQuery doesn’t provide built-in functions for such HTTP-specific matters, so implementation has to supply them (but multiple widely adopted implementations do)

Web application development with XQuery

Limitations of the SQL analogy

Advantages of XQuery for web delivery of XML content

Techniques for web application development with XQuery

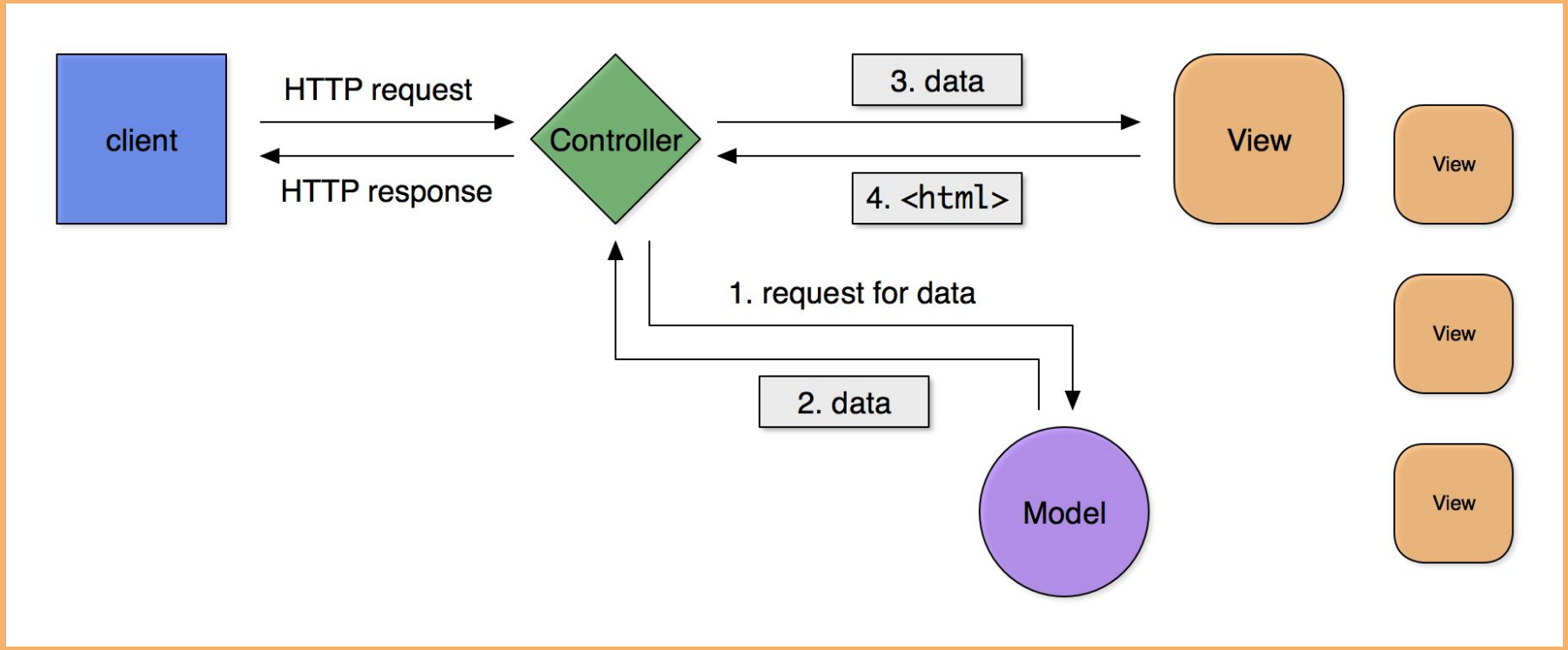
- Utilizing MVC

Keeping functions testable

Keeping functions testable

When responding to an HTTP request using MVC, processing forms a loop

Keeping functions testable



Keeping functions testable

When responding to an HTTP request using MVC, processing forms a loop

Throughout this loop, there will be decision points

Keeping functions testable

When responding to an HTTP request using MVC, processing forms a loop

Throughout this loop, there will be decision points — a given function in a given module must have access to the input values (URL parameters, etc.)

Keeping functions testable

When responding to an HTTP request using MVC, processing forms a loop

Throughout this loop, there will be decision points — a given function in a given module must have access to the input values (URL parameters, etc.)

Example: “search results” view — how to sort results, how many results to display at a time, etc.

Keeping functions testable

When responding to an HTTP request using MVC, processing forms a loop

Throughout this loop, there will be decision points — a given function in a given module must have access to the input values (URL parameters, etc.)

Example: “search results” view — how to sort results, how many results to display at a time, etc. — indicated by input from user on HTTP request

Keeping functions testable

When responding to an HTTP request using MVC, processing forms a loop

Throughout this loop, there will be decision points — a given function in a given module must have access to the input values (URL parameters, etc.)

Example: “search results” view — how to sort results, how many results to display at a time, etc. — indicated by input from user on HTTP request

XQuery implementations provide functions to get input values ...

Keeping functions testable

```
(: MarkLogic example :)
let $sort := xdmp:get-request-field("sort")
```

Keeping functions testable

```
(: MarkLogic example :)
let $sort := xdmp:get-request-field("sort")
```

```
(: BaseX equivalent of preceding example :)
import module namespace request = "http://exquery.org/ns/request";
...
let $sort := request:parameter("sort")
```

Keeping functions testable

Have to access these input values, but doing so inherently creates a problem:

Keeping functions testable

Have to access these input values, but doing so inherently creates a problem:

Functions that directly access HTTP parameters can’t be independently tested

Keeping functions testable

Have to access these input values, but doing so inherently creates a problem:

Functions that directly access HTTP parameters can’t be independently tested

Building robust, maintainable web applications requires *unit testing*

Keeping functions testable

Have to access these input values, but doing so inherently creates a problem:

Functions that directly access HTTP parameters can't be independently tested

Building robust, maintainable web applications requires *unit testing*

whereby a single unit of code (in XQuery, a function) is tested

Keeping functions testable

Have to access these input values, but doing so inherently creates a problem:

Functions that directly access HTTP parameters can't be independently tested

Building robust, maintainable web applications requires *unit testing*

whereby a single unit of code (in XQuery, a function) is tested

in isolation from the rest of the application

Keeping functions testable

Have to access these input values, but doing so inherently creates a problem:

Functions that directly access HTTP parameters can't be independently tested

Building robust, maintainable web applications requires *unit testing*

whereby a single unit of code (in XQuery, a function) is tested

in isolation from the rest of the application

in a deliberately controlled environment

Keeping functions testable

Have to access these input values, but doing so inherently creates a problem:

Functions that directly access HTTP parameters can't be independently tested

Building robust, maintainable web applications requires *unit testing*

whereby a single unit of code (in XQuery, a function) is tested

in isolation from the rest of the application

in a deliberately controlled environment

using sample data

Keeping functions testable

If a function accesses a value taken from the HTTP request

Keeping functions testable

If a function accesses a value taken from the HTTP request,

when that function gets called for testing purposes (outside the context of any HTTP request)

Keeping functions testable

If a function accesses a value taken from the HTTP request,

when that function gets called for testing purposes (outside the context of any HTTP request),

that value will always be empty

Keeping functions testable

If a function accesses a value taken from the HTTP request,

when that function gets called for testing purposes (outside the context of any HTTP request),

that value will always be empty

No HTTP request, no request parameter

Keeping functions testable

If a function accesses a value taken from the HTTP request,

when that function gets called for testing purposes (outside the context of any HTTP request),

that value will always be empty

No HTTP request, no request parameter

So how do we independently verify the behavior of the function?

Keeping functions testable

Solution: provide a function in controller (receives input)

Keeping functions testable

Solution: provide a function in controller (receives input)

to access URL parameters (or other input values)

Keeping functions testable

Solution: provide a function in controller (receives input)

to access URL parameters (or other input values)

and put them in an XQuery map (or other structure)

Keeping functions testable

Solution: provide a function in controller (receives input)

to access URL parameters (or other input values)

and put them in an XQuery map (or other structure)

Then controller can pass that map to any given view

Keeping functions testable

Solution: provide a function in controller (receives input)

to access URL parameters (or other input values)

and put them in an XQuery map (or other structure)

Then controller can pass that map to any given view

and view can pass map to internal functions as needed

Keeping functions testable

Solution: provide a function in controller (receives input)

to access URL parameters (or other input values)

and put them in an XQuery map (or other structure)

Then controller can pass that map to any given view

and view can pass map to internal functions as needed

Thus all functions have access to input values needed to make decisions

Keeping functions testable

Now we can test any given function in a controlled environment

Keeping functions testable

Now we can test any given function in a controlled environment (outside any HTTP request)

Keeping functions testable

Now we can test any given function in a controlled environment (outside any HTTP request)

1. Construct a map containing sample input values

Keeping functions testable

Now we can test any given function in a controlled environment (outside any HTTP request)

1. Construct a map containing sample input values
2. Pass map to function being tested

Keeping functions testable

Now we can test any given function in a controlled environment (outside any HTTP request)

1. Construct a map containing sample input values
2. Pass map to function being tested
3. Verify expected return value

Web application development with XQuery

- Limitations of the SQL analogy
- Advantages of XQuery for web delivery of XML content
- Techniques for web application development with XQuery
  - Utilizing MVC
  - Keeping functions testable

Improving code portability and sharing

Improving code portability and sharing

Web applications in XQuery require implementation-provided functions

Improving code portability and sharing

Web applications in XQuery require implementation-provided functions

Easy for codebase to become peppered with calls to those functions

Improving code portability and sharing

Web applications in XQuery require implementation-provided functions

Easy for codebase to become peppered with calls to those functions

Results in code that requires extensive refactoring to port

Improving code portability and sharing

Web applications in XQuery require implementation-provided functions

Easy for codebase to become peppered with calls to those functions

Results in code that requires extensive refactoring to port

And inhibits sharing code

Improving code portability and sharing

Web applications in XQuery require implementation-provided functions

Easy for codebase to become peppered with calls to those functions

Results in code that requires extensive refactoring to port

And inhibits sharing code

Contributes to fragmenting XQuery web development community

Improving code portability and sharing

Using available standardizations

Code portability/sharing: Using standardizations

XQuery web development remains a niche area, but there are options for standardizing practices

Code portability/sharing: Using standardizations

XQuery web development remains a niche area, but there are options for standardizing practices

EXQuery group (exquery.org) has proposed several specifications for standardizing functionality across XQuery implementations

Code portability/sharing: Using standardizations

XQuery web development remains a niche area, but there are options for standardizing practices

EXQuery group (exquery.org) has proposed several specifications for standardizing functionality across XQuery implementations

some of these are specific to XQuery web development

Code portability/sharing: Using standardizations

XQuery web development remains a niche area, but there are options for standardizing practices

EXQuery group (exquery.org) has proposed several specifications for standardizing functionality across XQuery implementations

- some of these are specific to XQuery web development
- of these, the most widely implemented is RESTXQ

Code portability/sharing: Standardizations: RESTXQ

RESTXQ (proposed by Adam Retter in 2012) utilizes XQuery 3.0 annotations to associate a given URL with a corresponding XQuery function

Code portability/sharing: Standardizations: RESTXQ

RESTXQ (proposed by Adam Retter in 2012) utilizes XQuery 3.0 annotations to associate a given URL with a corresponding XQuery function

Replaces URL rewriting

Code portability/sharing: Standardizations: RESTXQ

RESTXQ (proposed by Adam Retter in 2012) utilizes XQuery 3.0 annotations to associate a given URL with a corresponding XQuery function

Replaces URL rewriting

Instead of rewriting URLs (from user-friendly to code-friendly), RESTXQ maps URLs to functions directly

Code portability/sharing: Standardizations: RESTXQ

Example: search → search results → item view

Code portability/sharing: Standardizations: RESTXQ

Example: search → search results → item view

Say item view is requested by URL like `/item/a1`

Code portability/sharing: Standardizations: RESTXQ

Example: search → search results → item view

Say item view is requested by URL like `/item/a1`

With URL rewriting, translate `/item/a1` to `/?id=a1`

Code portability/sharing: Standardizations: RESTXQ

Example: search → search results → item view

Say item view is requested by URL like `/item/a1`

With URL rewriting, translate `/item/a1` to `/?id=a1`

Then web server passes that URL to XQuery processor

Code portability/sharing: Standardizations: RESTXQ

```
(: MarkLogic example :)
let $url:= xdmp:get-request-url()
let $regex-id := "/item/(.+)/?"
return
  if (fn:matches($url, $regex-id)) then
    (: extract identifier :)
    let $id := fn:replace($url, fn:concat("^.*", $regex-id, ".*$"), "$1")
    (: replace "/item/whatever" with "/" (implying "/default.xqy") :)
    let $start := fn:replace($url, $regex-id, "/")
    (: determine which separator to use :)
    let $separator := if (fn:contains($url, "?")) then "&amp;" else "?"
    (: append "id=whatever" as a URL parameter :)
    return fn:concat($start, $separator, "id=", $id)
  (: else if ... :)
  (: else if ... :)
  else $url
```

Code portability/sharing: Standardizations: RESTXQ

```
(: MarkLogic example :)
let $url:= xdmp:get-request-url()
let $regex-id := "/item/(.+)/?"
return
  if (fn:matches($url, $regex-id)) then
    (: extract identifier :)
    let $id := fn:replace($url, fn:concat("^.*", $regex-id, ".*$"), "$1")
    (: replace "/item/whatever" with "/" (implying "/default.xqy") :)
    let $start := fn:replace($url, $regex-id, "/")
    (: determine which separator to use :)
    let $separator := if (fn:contains($url, "?")) then "&amp;" else "?"
    (: append "id=whatever" as a URL parameter :)
    return fn:concat($start, $separator, "id=", $id)
  (: else if ... :)
  (: else if ... :)
  else $url
```

Code portability/sharing: Standardizations: RESTXQ

```
(: MarkLogic example :)
let $url:= xdmp:get-request-url()
let $regex-id := "/item/(.+)/?"
return
  if (fn:matches($url, $regex-id)) then
    (: extract identifier :)
    let $id := fn:replace($url, fn:concat("^.*", $regex-id, ".*$"), "$1")
    (: replace "/item/whatever" with "/" (implying "/default.xqy") :)
    let $start := fn:replace($url, $regex-id, "/")
    (: determine which separator to use :)
    let $separator := if (fn:contains($url, "?")) then "&amp;" else "?"
    (: append "id=whatever" as a URL parameter :)
    return fn:concat($start, $separator, "id=", $id)
  (: else if ... :)
  (: else if ... :)
  else $url
```

Code portability/sharing: Standardizations: RESTXQ

```
(: MarkLogic example :)
let $url:= xdmp:get-request-url()
let $regex-id := "/item/(.+)/?"
return
  if (fn:matches($url, $regex-id)) then
    (: extract identifier :)
    let $id := fn:replace($url, fn:concat("^.*", $regex-id, ".*$"), "$1")
    (: replace "/item/whatever" with "/" (implying "/default.xqy") :)
    let $start := fn:replace($url, $regex-id, "/")
    (: determine which separator to use :)
    let $separator := if (fn:contains($url, "?")) then "&amp;" else "?"
    (: append "id=whatever" as a URL parameter :)
    return fn:concat($start, $separator, "id=", $id)
  (: else if ... :)
  (: else if ... :)
  else $url
```

Code portability/sharing: Standardizations: RESTXQ

```
(: MarkLogic example :)
let $url:= xdmp:get-request-url()
let $regex-id := "/item/(.+)/?"
return
  if (fn:matches($url, $regex-id)) then
    (: extract identifier :)
    let $id := fn:replace($url, fn:concat("^.*", $regex-id, ".*$"), "$1")
    (: replace "/item/whatever" with "/" (implying "/default.xqy") :)
    let $start := fn:replace($url, $regex-id, "/")
    (: determine which separator to use :)
    let $separator := if (fn:contains($url, "?")) then "&amp;" else "?"
    (: append "id=whatever" as a URL parameter :)
    return fn:concat($start, $separator, "id=", $id)
  (: else if ... :)
  (: else if ... :)
  else $url
```

Code portability/sharing: Standardizations: RESTXQ

```
(: MarkLogic example :)
let $url:= xdmp:get-request-url()
let $regex-id := "/item/(.+)/?"
return
  if (fn:matches($url, $regex-id)) then
    (: extract identifier :)
    let $id := fn:replace($url, fn:concat("^.*", $regex-id, ".*$"), "$1")
    (: replace "/item/whatever" with "/" (implying "/default.xqy") :)
    let $start := fn:replace($url, $regex-id, "/")
    (: determine which separator to use :)
    let $separator := if (fn:contains($url, "?")) then "&amp;" else "?"
    (: append "id=whatever" as a URL parameter :)
    return fn:concat($start, $separator, "id=", $id)
  (: else if ... :)
  (: else if ... :)
  else $url
```

Code portability/sharing: Standardizations: RESTXQ

URL rewriting code tends to be idiosyncratic and cumbersome to maintain

Code portability/sharing: Standardizations: RESTXQ

URL rewriting code tends to be idiosyncratic and cumbersome to maintain

“creating a spaghetti of if/else statements” (Retter)

Code portability/sharing: Standardizations: RESTXQ

URL rewriting code tends to be idiosyncratic and cumbersome to maintain

“creating a spaghetti of if/else statements” (Retter)

But with RESTXQ, we can directly map `/item/a1` to an XQuery function

Code portability/sharing: Standardizations: RESTXQ

```
(: BaseX example :)
declare
%rest:path("/item/{$id}")
function c:get-item-view($id as xs:string) as element(div)
{
  (: variable $id has the value specified on the URL :)
  ...
};
```

Code portability/sharing: Standardizations: RESTXQ

```
(: BaseX example :)
declare
%rest:path("/item/{$id}")
function c:get-item-view($id as xs:string) as element(div)
{
  (: variable $id has the value specified on the URL :)
  ...
};
```

Code portability/sharing: Standardizations: RESTXQ

```
(: BaseX example :)
declare
%rest:path("/item/{$id}")
function c:get-item-view($id as xs:string) as element(div)
{
  (: variable $id has the value specified on the URL :)
  ...
};
```

Code portability/sharing: Standardizations: RESTXQ

```
(: BaseX example :)
declare
%rest:path("/item/{$id}")
function c:get-item-view($id as xs:string) as element(div)
{
  (: variable $id has the value specified on the URL :)
  ...
};
```

Code portability/sharing: Standardizations: RESTXQ

Locates the URL mapping code at the function declaration itself

Code portability/sharing: Standardizations: RESTXQ

Locates the URL mapping code at the function declaration itself (not in a separate “rewrite engine” module)

Code portability/sharing: Standardizations: RESTXQ

Locates the URL mapping code at the function declaration itself (not in a separate “rewrite engine” module)

Annotation is concise, declarative, standardized, easy to read and maintain

Code portability/sharing: Standardizations: RESTXQ

Locates the URL mapping code at the function declaration itself (not in a separate “rewrite engine” module)

Annotation is concise, declarative, standardized, easy to read and maintain

RESTXQ supported in BaseX, eXist, and MarkLogic — thus effectively a de facto standard

Code portability/sharing: Standardizations: RESTXQ

Locates the URL mapping code at the function declaration itself (not in a separate “rewrite engine” module)

Annotation is concise, declarative, standardized, easy to read and maintain

RESTXQ supported in BaseX, eXist, and MarkLogic — thus effectively a de facto standard

RESTXQ easy to incorporate into MVC

Code portability/sharing: Standardizations: RESTXQ

Locates the URL mapping code at the function declaration itself (not in a separate “rewrite engine” module)

Annotation is concise, declarative, standardized, easy to read and maintain

RESTXQ supported in BaseX, eXist, and MarkLogic — thus effectively a de facto standard

RESTXQ easy to incorporate into MVC

Add RESTXQ annotations to functions in the controller

Web application development with XQuery

- Limitations of the SQL analogy
- Advantages of XQuery for web delivery of XML content
- Techniques for web application development with XQuery
  - Utilizing MVC
  - Keeping functions testable
  - Improving code portability and sharing
    - Using available standardizations

Improving code portability and sharing

Isolating implementation-specific functions

Isolating implementation-specific functions

Implementation-provided functions pose main barrier to code portability/sharing

Isolating implementation-specific functions

Implementation-provided functions pose main barrier to code portability/sharing

Technique: a library module containing generically named/namespaced functions ...

Isolating implementation-specific functions

Implementation-provided functions pose main barrier to code portability/sharing

Technique: a library module containing generically named/namespaced functions ...

Within generic function, call corresponding implementation-provided function

Isolating implementation-specific functions

Implementation-provided functions pose main barrier to code portability/sharing

Technique: a library module containing generically named/namespaced functions ...

Within generic function, call corresponding implementation-provided function

Throughout codebase, whenever needed, call applicable generic function

Isolating implementation-specific functions

Implementation-provided functions pose main barrier to code portability/sharing

Technique: a library module containing generically named/namespaced functions ...

Within generic function, call corresponding implementation-provided function

Throughout codebase, whenever needed, call applicable generic function

Creates a layer of abstraction, isolates implementation-dependent code from all other application code

Isolating implementation-specific functions

Example: Set up “implementation” module

Isolating implementation-specific functions

Example: Set up “implementation” module, add generically named function for getting value of a given HTTP request parameter

Isolating implementation-specific functions

Example: Set up “implementation” module, add generically named function for getting value of a given HTTP request parameter

```
xquery version "3.1";
(: BaseX example :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";
import module namespace request = "http://exquery.org/ns/request";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  request:parameter($name)
};
```

Isolating implementation-specific functions

Example: Set up “implementation” module, add generically named function for getting value of a given HTTP request parameter

```
xquery version "3.1";
(: BaseX example :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";
import module namespace request = "http://exquery.org/ns/request";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  request:parameter($name)
};
```

Isolating implementation-specific functions

Example: Set up “implementation” module, add generically named function for getting value of a given HTTP request parameter

```
xquery version "3.1";
(: BaseX example :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";
import module namespace request = "http://exquery.org/ns/request";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  request:parameter($name)
};
```

Isolating implementation-specific functions

Example: Set up “implementation” module, add generically named function for getting value of a given HTTP request parameter

```
xquery version "3.1";
(: BaseX example :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";
import module namespace request = "http://exquery.org/ns/request";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  request:parameter($name)
};
```

Isolating implementation-specific functions

Example: Set up “implementation” module, add generically named function for getting value of a given HTTP request parameter

```
xquery version "3.1";
(: BaseX example :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";
import module namespace request = "http://exquery.org/ns/request";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  request:parameter($name)
};
```

Isolating implementation-specific functions

Allows creating an “implementation” module for any given XQuery implementation that supports web application development

Isolating implementation-specific functions

Allows creating an “implementation” module for any given XQuery implementation that supports web application development

```
xquery version "1.0-ml";
(: MarkLogic equivalent :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  xdmp:get-request-field($name)
};
```

Isolating implementation-specific functions

Allows creating an “implementation” module for any given XQuery implementation that supports web application development

```
xquery version "1.0-ml";
(: MarkLogic equivalent :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  xdmp:get-request-field($name)
};
```

Isolating implementation-specific functions

Allows creating an “implementation” module for any given XQuery implementation that supports web application development

```
xquery version "1.0-ml";
(: MarkLogic equivalent :)
module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation";

declare function imp:http-request-parameter($name as xs:string) as xs:string?
{
  xdmp:get-request-field($name)
};
```

Isolating implementation-specific functions

Import “implementation” module for target implementation, call generic function

Isolating implementation-specific functions

Import “implementation” module for target implementation, call generic function

```
(: BaseX example :)
module namespace c = "http://balisage.net/ns/Bal2016murr0319/controller";

import module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation"
  at "implementation-basex.xqm";

...
  (: when needed, get value of "sort" parameter so we can pass it to a view :)
  let $sort := imp:http-request-parameter("sort")
```

Isolating implementation-specific functions

Import “implementation” module for target implementation, call generic function

```
(: BaseX example :)
module namespace c = "http://balisage.net/ns/Bal2016murr0319/controller";

import module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation"
  at "implementation-basex.xqm";

...
  (: when needed, get value of "sort" parameter so we can pass it to a view :)
  let $sort := imp:http-request-parameter("sort")
```

Isolating implementation-specific functions

Import “implementation” module for target implementation, call generic function

```
(: BaseX example :)
module namespace c = "http://balisage.net/ns/Bal2016murr0319/controller";

import module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation"
  at "implementation-basex.xqm";

...
  (: when needed, get value of "sort" parameter so we can pass it to a view :)
  let $sort := imp:http-request-parameter("sort")
```

Isolating implementation-specific functions

Import “implementation” module for target implementation, call generic function

```
(: BaseX example :)
module namespace c = "http://balisage.net/ns/Bal2016murr0319/controller";

import module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation"
  at "implementation-basex.xqm";

...
  (: when needed, get value of "sort" parameter so we can pass it to a view :)
  let $sort := imp:http-request-parameter("sort")
```

Isolating implementation-specific functions

To target MarkLogic, controller code is identical except filename

Isolating implementation-specific functions

To target MarkLogic, controller code is identical except filename

```
(: MarkLogic example :)
module namespace c = "http://balisage.net/ns/Bal2016murr0319/controller";

import module namespace imp = "http://balisage.net/ns/Bal2016murr0319/implementation"
  at "implementation-ml.xqy";

...
  (: when needed, get value of "sort" parameter so we can pass it to a view :)
  let $sort := imp:http-request-parameter("sort")
```

Isolating implementation-specific functions

This technique greatly facilitates:

Sharing code with individuals/projects

Isolating implementation-specific functions

This technique greatly facilitates:

- Sharing code with individuals/projects
- Porting entire application?

Isolating implementation-specific functions

This technique greatly facilitates:

- Sharing code with individuals/projects
- Porting entire application?

By simply swapping out one “implementation” module for another

Isolating implementation-specific functions

This technique greatly facilitates:

- Sharing code with individuals/projects
- Porting entire application?

By simply swapping out one “implementation” module for another

As more code uses such “implementation” modules, more code becomes implementation-independent

Isolating implementation-specific functions

This technique greatly facilitates:

- Sharing code with individuals/projects
- Porting entire application?

By simply swapping out one “implementation” module for another

As more code uses such “implementation” modules, more code becomes implementation-independent — across one’s own projects, perhaps also across XQuery web development community?

Bibliography

Kaufmann, Martin and Donald Kossmann. “Developing an enterprise web application in XQuery” (2009).

Retter, Adam. “RESTful XQuery: Standardised XQuery 3.0 annotations for REST” (2012).

Seiferle, Michael. “Implementing web applications using XQuery: XML from front to back” (2012).

Greg Murray
Princeton Theological Seminary