XML Technologies

Web Applications

Christian Grün
Database & Information Systems Group
Universität Konstanz
Web Applications

- more and more applications are purely based on web technologies: JavaScript, HTML5, XML
- advantages: ubiquity of web browsers; accessible on all modern platforms
- SOA: many applications rely on web-based services
- popular server-side technology: PHP, Python, Ruby on Rails, ASP, JSP

Why XQuery?

- HTML is an XML dialect
- more and more textual contents are natively available in HTML5
- less data transformations between layers of a software architecture (e.g. Three-Tier Architecture: presentation, application, storage)
REST: Representational State Transfer

- programming paradigm for web applications
- based on Roy Fielding’s dissertation in 2001 (one author of the HTTP spec)

Four main principles

- each REST service has a unique address (which is the URI)
- REST services are stateless: a request contains all information required to understand and process a message
- a service may return different representations of the addressed resource, (XML, JSON, …), depending on the demands of a client
- services support a defined set of operations (HTTP: GET, POST, PUT, DELETE)
  - surprisingly simple and intuitive (and seemingly obvious)
RESTXQ: RESTful Applications with XQuery

...presented by Adam Retter at the XML Prague 2012 conference.

Motivation

- XML, XQuery, etc. provide no standard web capability
- all HTTP/web extensions are specific to vendors and thus non-portable

Solution

- inspired by Java’s JAX-RS API (🔗 familiar?)
- provides a pre-defined set of annotations to map HTTP requests to XQuery functions
- the HTTP response is generated by the XQuery result
RESTXQ Annotations

- RESTXQ annotations use a fixed namespace: http://exquery.org/ns/restxq
- In BaseX, this namespace is statically bound to the `rest` prefix, and all XQuery functions must be contained in a `library module`
- More details: docs.basex.org/wiki/RESTXQ

Paths

- a RESTXQ resource function has a single `path annotation` with a single string as argument, which contains `path segments` and `templates`
- a `template` contains a variable in curly brackets; its value will be assigned to the corresponding arguments of the XQuery function
- the function will be evaluated if an HTTP request matches the path string
Path examples

- this example contains a path annotation with a single path segment:

```xml
module namespace _ = 'http://dbis.uni-konstanz.de/';
declare %rest:path("/main") function _:_main() {
  <html>
   <h1>Welcome to this start page</h1>
  </html>
};
```

- in the following example, one segment and one template is used:

```xml
(: ...omitting the module namespace... :) 
declare %rest:path("/search/{$name}" ) function _:_search($name) {
   "Specified name: " || $name
};
```
HTTP methods

- equivalent to the *HTTP request methods* (no TRACE and CONNECT): GET, POST, PUT, DELETE, HEAD, OPTIONS
- a resource function may have *zero or more* method annotations:

```xml
declare %rest:DELETE %rest:path("{$id}") function _:delete($id) {
    "Entry with " || $id || " will be deleted."
};
```

- POST and PUT annotations may take an *optional string* as argument:

```xml
declare %rest:PUT("{data}") %rest:path("/upload") function _:put(
    $data as xs:base64Binary) {
    file:write-binary('/tmp/' || current-dateTime(), $data), "Uploaded!"
};
```

- if no method is specified, the function will be invoked for all methods
Content types

- a function can be restricted to a content type; it will only be invoked if the `Content-Type` header of a request matches the specified type:

```perl
declare %rest:consumes("application/xml", "text/xml")
    %rest:path("/consume") function _:consume() {
        "The specified content type header is either application/xml or text/xml."
    }
```

Accept

- a function can be restricted to the type specified in an `Accept` header:

```perl
declare %rest:produces("application/atom+xml")
    %rest:path("/produce") function _:produce() {
        "The specified Accept header is application/atom+xml"
    }
```
Query parameters

- the URI of an HTTP request may contain a *query string*, which is usually resolved to name/value pairs (e.g. `duckduckgo.com/search?q=xml`)
- a query parameter annotation has two or more arguments:
  - the first argument contains the *name* of a query parameter
  - the second argument contains the variable name the *value* should be bound to
  - if the HTTP request contains *no matching* names, additional arguments will be bound to the variable as sequence:

```xml
declare %rest:query-param("q", "{$query}"),
%rest:path("/search") function _:query($query as xs:string*) {
    "The following terms were requested:" || $query
};
```

- note that a variable name may be specified more than once
**HTML form fields**

- form parameters may be passed on when POST is used
- in RESTXQ, their values are extracted from both GET and POST requests
- the syntax is the same as for query parameters:

```xml
declare %rest:form-param("q", "{$query}")
%rest:path("/search") function _:form($query as xs:string*) {
   "Specified query terms:" || $query
};
```

- a HTML snippet that triggers the request may look as follows:

```html
<form action="http://localhost/search" method="post">
   <input type="text" name="q">
   <input type="submit" value="Suche">
</form>
```
HTTP headers

- an HTTP request contains instructive information that can also be bound to variables:

```perl
declare %rest:header-param("User-Agent","{$agent}")
   %rest|:path("/check-agent") function _:headers($agent) {
     "You are using '{ } as browser/crawler." 
    }
```

Cookies

- cookies may be bound to variables, too:

```perl
declare %rest:cookie-param("username","{$user}")
   %rest:path("/check-user") function _:cookies($user) {
     "Value of 'username' cookie: '{ }"
    }
```
Output

- by default, all results will be returned with `application/xml` as content type (synonyms: mime type, media type)
- the content type can be overridden via serialization parameters, bound to the `output` prefix:

```perl
declare %output:media-type("text/plain") %rest:path("text") function _:text() {
    "This is plain text"
}
```

- the type can also be modified by specifying another output method:

```perl
declare %output:method("html") %rest:path("html") function _:html() {
    <html><body>Finally.. Some HTML</body></html>
}
```
some more serialization parameters can be specified to e.g. generate XML and doctype declarations:

```
declare
%output:method("xhtml")
%output:omit-xml-declaration("no")
%output:doctype-public("-//W3C//DTD XHTML 1.0 Transitional//EN")
%output:doctype-system("http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd")
%rest:path("") function _:index() {
  <html xmlns="http://www.w3.org/1999/xhtml">
    <body>You have just created some valid XHTML output!</body>
  </html>
};
```
Redirections

- when using updating queries, it is often helpful to redirect a browser to another page that indicates if the update was successful
- BaseX provides two solutions for that:
  - if a `rest:redirect` element is returned, status code 302 and the new location will be sent to the client
  - if a `rest:forward` element is returned, the server itself will dispatch the request to the new location
- The following (and last) example will create an endless loop:

```
declare %rest:path("/loop") function _:loop() {
  <rest:forward/>
  <loop</rest:forward>
};
```
Applications with BaseX: DBA

- simple RESTXQ application to organize your databases
- bundled with BaseX, helps you to write your own applications