**Technical Report**
**KN–2010–DiSy–01**

# Distributed Systems Group Database & Information Systems Group

# JAX-RX

*Unified REST Access to XML Resources*

**Sebastian Graf, Lukas Lewandowski and Christian Grün**

Distributed Systems Laboratory and
Database & Information Systems Group

Department of Computer and Information Science

University of Konstanz – Germany

REST nowadays represents, besides SOAP, one common way to access distributed resources in a web-affine manner. While SOAP can be easily utilized by high-level programming languages like *Java* (e.g. *JAX-WS* as one common standardized way), REST catches up regarding straight usages (e.g *JAX-RS* regarding *Java*).

With the clean and direct usage of *JAX-RS*, common layers for standardised access on heterogeneous XML-resources can be defined. This is what the project *JAX-RX* stands for: Based on XML as modern resource in the *WWW*, we defined a common application programming interface to access *Java* enabled XML resource easily in a common way. Using a common architecture for the uniform resource locator on the one hand and defining suitable interfaces on the other hand, every XML generating resources can be "RESTified" with nearly no effort. This technical report describes the motivation, the architecture and the usage of our XML-enabling *API* called *JAX-RX*.

# 1 Introduction

REST [1] has emerged to a de facto standard access interface to distributed resources, especially in the realm of web applications. Based on the observation that states are very difficult to handle in distributed environments, REST represents a paradigm where no states are handled at all: Each request is encapsulated and valid by itself and each resource is directly addressed. Many popular and widely used environments apply REST to read and write remote and distributed data sources.

To make direct use of this paradigm within Java environments, JAX-RS [2] was introduced. JAX-RS simplifies and unifies the task of parsing the URL, in which the request to a specific resource is encapsulated, and offers an elegant way to access resources without checking substrings and handling requests. This is realized by implementing a JAX-RS interface and annotating methods and parameters. Thereby, the annotations are directly related to the URL of the REST request.

Since XML is a core technology for web-based applications, we want to motivate a common interface layer for XML-based resources accessible over REST. The clean architecture of JAX-RS allows us to build an interface for uniform access to distributed resources. This approach, which we call JAX-RX (Java API for RESTful XML resources), enables third party applications to be accessible over a platform independent, up-to-date interface layer. To demonstrate the ease of use of our thin layer, we are offering implementations for existing XML databases.

Furthermore, JAX-RX allows for the retrieval of WADL [3] descriptions, based on the JAX-RS reference implementation Jersey [2]. WADL is a complete XML-based description of all RESTful Web Services that we offer within our REST layer. Two alternatives exist to retrieve the WADL description:

1. A GET request can be sent to a standardized URL:

   ```
   http://ADDRESS/application.wadl
   ```

   This will return a description of all available URL resources with their corresponding offered services.
2. an OPTIONS request can be sent to the relevant URL resource, which will return the WADL description for the requested URL.

Our WADL description contains information on the offered resources, such as the URL, HTTP request type (GET, PUT, POST, and DELETE), path parameters, URL query parameters, request and response media type (e.g. text/xml), all based on the underlying implementation. More to that, clients can generate appropriate stubs out of this WADL description, similar to WSDL [4] in SOAP-based Web Services scenarios [5], (example generator tool: wadl-cmdline [3]) to speed up the implementation of the client/server interaction.

## 2 JAX-RX

We present JAX-RX, an interface project which provides a unified access to XML resources within Java-based REST applications. Inspired by the URL syntax of eXist [6], a well-known XML database which already supports REST, we developed an extensible framework to provide access to XML on the one hand without restricting functionality of the resource on the other hand.

### 2.1 URL architecture

The common architecture of the URL is one main aspect of JAX-RX. All URLs which rely on JAX-RX-enabled resources have the following layout:

```
http://SERVER/IMPLEMENTATION/jax-rx/RESOURCE(/NODE)?PARAM1=VALUE1&...
```

The extensions for the different REST methods are defined as follows:

1. The GET and POST methods allow users to directly specify parameters in the target URL. A GET or POST request comprises the following list of optional parameters:
   - **query** Query expression (XPath [7]/XQuery[8])
   - **xsl** Transformation via XSLT stylesheet
   - **output** XSLT/XQuery serialization options [9] (separated by commas)
   - **count** Number of items to be returned (default: all)
   - **start** Index position of the return values (default: 1)
   - **wrap** Resulting wrapping by additional elements (default: yes)
   - **revision** Revision of the data
   - **command** Implementation-dependent database command

   If a GET request does not refer to a specific XML resource, JAX-RX returns a standardized output. The output lists descendant access paths and parameters which are supported by the underlying system.
2. A DELETE request is removing a resource. The resource itself can be a database, a collection or resource within a database or, if supported, a directly accessible node. No additional parameters are allowed in the DELETE method: Only those resources can be removed which can be referenced by the path. More complex delete operations on a database are performed via XQuery Update or the update language extensions of a specific database.
3. A PUT request adds new content to the addressed resource. Similar to the DELETE method, PUT is parameter free. Again, XQuery Update can be utilized to perform more complex update operations.

An XML schema defines the structure for the POST request. If the wrap parameter was specified in a query, the results are embraced by additional elements. Systems may implement additional information for the returning nodes, such as unique node or result identifiers.

Summarizing, the proposed syntax offers a convenient way to communicate with XML databases and facilitates a system-independent access to remote XML resources. Apart from the uniform access of external resources, however, a major objective of JAX-RX is to allow for easy, server-side REST implementations.

### 2.2 Interface project

Unified access is not limited to the URL syntax. Based on JAX-RS [2], we provide a set of interfaces for XML databases and query processors. To offer a JAX-RX implementation, an implementor has to comply with the following requirements:

1. JAX-RX interface, which holds methods for each REST method, has to be implemented. The interface is located in *org.jaxrx.JaxRx*
2. The implementing classes must be kept in a single folder, which is referenced by a system property. The key of this property is denoted by *jaxrx.implementation*.

Matching these requirements, JAX-RX can be used with the standard Jersey-Servlet [2] by denoting the JAX-RX resource package. Upcoming HTTP requests are evaluated with the help of two resource classes, which are both JAX-RS enabled:

- **org.jaxrx.resource.JaxRxResource** This resource is processing system-independent requests, such as HTTP operations based on the host or the system itself. All parts of the URL, which are not accessing an XML resource, are evaluated with the help of this class.
- **org.jaxrx.resource.XMLResource** The other resource is related to the XML documents. Since such resources need to call the implementing system, suitable classes which are implementing the necessary interface are searched in the package which is referenced over the system variable *org.jaxrx.implementation*. The name of the system must equal the one encapsulated in the HTTP request. The classes are loaded once in a runtime of the servlet.

Both resources act as the central place for incoming requests and outgoing responses. They are validating the request and guarantee a standardized output of the response. The validation takes place on different levels. First, the request is validated against the JAX-RX specification. Additionally, the XML input of POST requests is validated against the XSD schema. After a successful check, the input is checked against the specifications of the implementing system. Using the defined parameters, an implementing system can inform the client side on the concrete subset of available parameters. If this check was successful as well, the request is forwarded to the implementation of the related interface. After evaluating the request, a response is returned to JAX-RX, processed to match the JAX-RX return value specification if necessary and returned to the client.

This results in a clean interface structure with the following benefits:

- With *org.jaxrx.resource*, one central package is handling all requests. In this package, all classes are JAX-RS enabled. All requests are parsed and responses are returned. That means that only one resource package is set in the Jersey servlet.
- By default, JAX-RX comes with a Java DOM [10] implementation. This implementation serves as an example and allows for a quick start, since no additional implementation is needed to try out JAX-RX.

– Any external resources can be accessed as long as the resource is implementing the requested interface and the implementing classes are located in a package, which is referenced by the system property *org.jaxrx.implementation*.
– No resource needs to be aware of web services. With JAX-RX, all servlet technology is encapsulated in JAX-RX itself. JAX-RX is released as a Maven [11] project to speed up the integration in existing XML databases.

## 3 Example Workflows

Below, we give five examples for the usage of JAX-RX. The examples are based on BaseX [12] and Treetank [13], two native XML database systems which are being developed at the University of Konstanz:

**GET:**
**Return all city elements with the name Panama from the factbook resource, stored in TreeTank**
GET Request on

```
http://localhost/treetank/jax-rx/factbook?query=//city[name='Panama']
```

Response:
HTTP/1.1 200 OK Content-Type: text/xml Server: Jetty(6.1.21)

**Listing 1.** GET Response of resource "city='Panama'"

```
<jax-rx:result xmlns:jax-rx="http://jax-rx.sourceforge.net">
  <city id="f0_1936"  country="f0_908" province="f0_19646"
  longitude="-79.55"  latitude="8.96667">
    <name>Panama</name>
    <name>Panama City</name>
    <population year="90">594800</population>
    <located_at type="sea" water="f0_38596"/>
  </city>
</jax-rx:result>
```

**PUT: Insert the specified XML document into the books resource, stored in BaseX**
PUT Request on

```
http://localhost/basex/jax-rx/books
```

**Listing 2.** PUT Request of resource "books"

```
<books>
  <book>
    <name>Effective  Java</name>
    <author>Joshua Bloch</author>
  </book>
</books>
```

Response:
HTTP/1.1 201 CREATED Content-Length: 0 Server: Jetty(6.1.21)

**DELETE: Delete a node with id 469 from the factbook resource, stored in TreeTank**
DELETE-Request on

`http://localhost/treetank/jax-rx/factbook/469`

Response:
HTTP/1.1 200 OK Content-Length: 0 Server: Jetty(6.1.21)

**POST: Perform an XQuery Update expression on the books resource, stored in BaseX**
POST-Request on

`http://localhost/basex/jax-rx/books`

**Listing 3.** POST Request on resource "books"

```
<query xmlns="http://jax-rx.sourceforge.net">
   <text>
     <![CDATA[insert node
       <book>
         <name>RESTful Web Services</name>
         <author>Leonard Richardson</author>
         <author>Sam Ruby</author>
       </book> as last into /books]]>
    </text>
</query>
```

Response:
HTTP/1.1 200 OK Content-Length: 0 Server: Jetty(6.1.21)

**POST: Return all books with Sam Ruby as author from the books resource, stored in BaseX**
POST-Request on

`http://localhost/basex/jax-rx/books`

**Listing 4.** POST Request to get from the "books"resource

```
<query xmlns="http://jax-rx.sourceforge.net">
  <text>
    for $x in /books/book where  $x/author='Sam Ruby' return $x
  </text>
</query>
```

Response:

**Listing 5.** Post Response from resource "books"

```
<jax-rx:result xmlns:jax-rx="http://jax-rx.sourceforge.net">
  <book>
    <name>RESTful Web Services</name>
    <author>Leonard Richardson</author>
    <author>Sam  Ruby</author>
  </book>
</jax-rx:result>
```

## 4 Conclusion and Future Work

JAX-RX has been released as an early beta on Sourceforge [14]. We want to encourage users to give feedback in our interface layer at the earliest moment possible. Apart from implementations for the XML databases BaseX and TreeTank, we ship JAX-RX with a reference implementation for Java DOM [10], which allows implementors to quickly get in touch with JAX-RX. In near future, we plan to add an eXist implementation in our interface framework.

JAX-RX offers numerous incentives for future work. We are investigating the option of building a persistence layer on top of JAX-RX for XML resources, similar to Hibernate [15]. This would allow us to further push XML as a core web technology, not only when we talk about data exchange. Secondly, we will use JAX-RX as a base for a distribution framework, which will allow a distributed and secure handling of various XML resources in different systems. JAX-RX will soon offer authentication and authorization methods for user management operations as well as encryption support.

We believe that JAX-RX is a mandatory step to uniformly access remote and distributed XML resources. Current API layers for XML resources are often based on the implementation framework itself. Focusing a major Java use case the development and implementation of web applications we feel the need for a unified interface layer which offers a generic and light-weight access to distributed XML resources. We definitely believe that we can fulfill these requirements with JAX-RX.

## References

[1] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000, chair-Taylor, Richard N.

[2] M. Hadley and P. Sandoz, "JAX-RS: Java API for RESTful Web Services," *Java Specification Request (JSR)*, vol. 311, 2009.

[3] T. Takase, S. Makino, S. Kawanaka, K. Ueno, C. Ferris, and A. Ryman, "Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0," *IBM Reasearch*, 2008.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL) 1.1," 2001.

[5] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. Nielsen, A. Karmarkar, and Y. Lafon, "SOAP version 1.2 part 1: Messaging framework," 2003.

[6] W. Meier, "eXist: An open source native XML database," *Web, Web-Services, and Database Systems*, pp. 169–183, 2002.

[7] J. Clark, S. DeRose *et al.*, "XML path language (XPath) version 1.0," 1999.

[8] S. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu, "XQuery 1.0: An XML query language," *W3C working draft*, vol. 15, 2002.

[9] J. Clark *et al.*, "XSL transformations (XSLT) version 1.0," *W3C Recommendation*, vol. 16, no. 11, 1999.

[10] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne, "Document object model (DOM) level 3 core specification," *W3C Recommendation*, 2004.

[11] A. Maven, "Project Homepage."

[12] C. Grün, A. Holupirek, and M. Scholl, "Visually Exploring and Querying XML with BaseX," in *Proc. of the 12th GI-conference on Database Systems in Business, Technology and Web (BTW), Aachen, Germany*, 2007.

[13] S. Graf, "TreeTank, a native XML storage," 2009.

[14] L. Lewandowski, S. Graf, and C. Grün. [Online]. Available: http://jax-rx.sourceforge.net/

[15] C. Bauer and G. King, *Java Persistance with Hibernate*. Dreamtech Press, 2006.