MAX-PLANCK-INSTITUT FÜR WISSENSCHAFTSGESCHICHTE
Max Planck Institute for the History of Science

Max Planck Digital Library (MPDL)

Subproject: Software development: Content based web access to XML documents

Josef Willenborg

Technical Specification: The eSciDoc / eXist REST interface

Version	1.2
Author	Josef Willenborg
Created	September 2010
Last modified	November 2010
Last modified by	Josef Willenborg

Content

1.Introduction 3
2. Implementation variants.....3
3. Architecture.....4
4. REST interface.....5
4.1. CUD operations.....6
4.1.1. Create item.....6
4.1.2. Update item.....6
4.1.3. Delete item.....7
4.1.4. Create container.....7
4.1.5. Update container.....8
4.1.6. Delete container.....8
4.2. XQuery search / execution.....8
4.2.1. Search in items.....9
4.2.2. Execute XQuery scripts on one item.....10
4.2.3. Execute XQuery scripts.....10
5.Literature 11
6.Supplement: some internal implementation stuff 11

1. Introduction

This paper describes how eXist – an efficient XML storing and query system – could be integrated into eSciDoc by specialising and extending the eSciDoc REST interface.

Generally it has the following advantages to integrate eXist into eSciDoc:

- power of eXist directly in eSciDoc: XQuery search and execution for XML documents
- better performance in many cases
  - use of XML based indexes
  - use of eXist/Lucene fulltext indexes
  - fast extraction of XML fragments and nodes
- XSL transformation of XML documents (through XQuery processing)
- XQuery applications could be used
  - MPIWG-MPDL: language technology (e.g. morphological fulltext queries and indexes)
  - MPIWG-MPDL: many XQuery scripts
  - XSL transformation of XML documents (Echo/TEI schema)
  - functx-XQuery library
  - ...
- ...

2. Implementation variants

There are different ways to implement combined eSciDoc and eXist REST services:

a) Implementation in eXist with URL-Rewriting (see also architecture below)

- URL-Rewriting: controller.xql forwards eSciDoc REST-URL's to EsciDocEXistRESTServlet or to eSciDoc REST service
- ESciDocEXistRESTServlet: implements doGet, doPut, doPost, doDelete for all eXist/eSciDoc REST-URL's

b) Implementation as an eXist-REST-Server

- a new static and and big eXist class with the methods doGet, doPut, doPost, doDelete for all eSciDoc Request URL's
- eXist sources itself have be be modified
- some implemention effort

c) Implementation in another XML processing system than eXist

- e.g. BerkeleyDBXML, MonetDB or JavaXML.
- performance is not known
- little knowledge/practise in these systems

d) Implementation in eSciDoc in service layer

- eSciDoc sources have to be modified
- much effort because a MPIWG developer has to get trained in eSciDoc sources
- eSciDoc has to be fully built and compiled, this is problematic when a new version of eSciDoc arrives (merging of the two sources)

e) Implementation in eSciDoc in service layer with JMS calls

- JMS eXist call is done before or after the actual eSciDoc service call: sends a message to eXist for the operation
- no direct communication
- JMS is not part of the actual version 1.2 of eSciDoc, so this implementation is not possible so far (JMS is planned in the next version of eSciDoc)

f) Implementation as an eSciDoc application

- does not implement a real eSciDoc service integration
- is complex and costs much effort

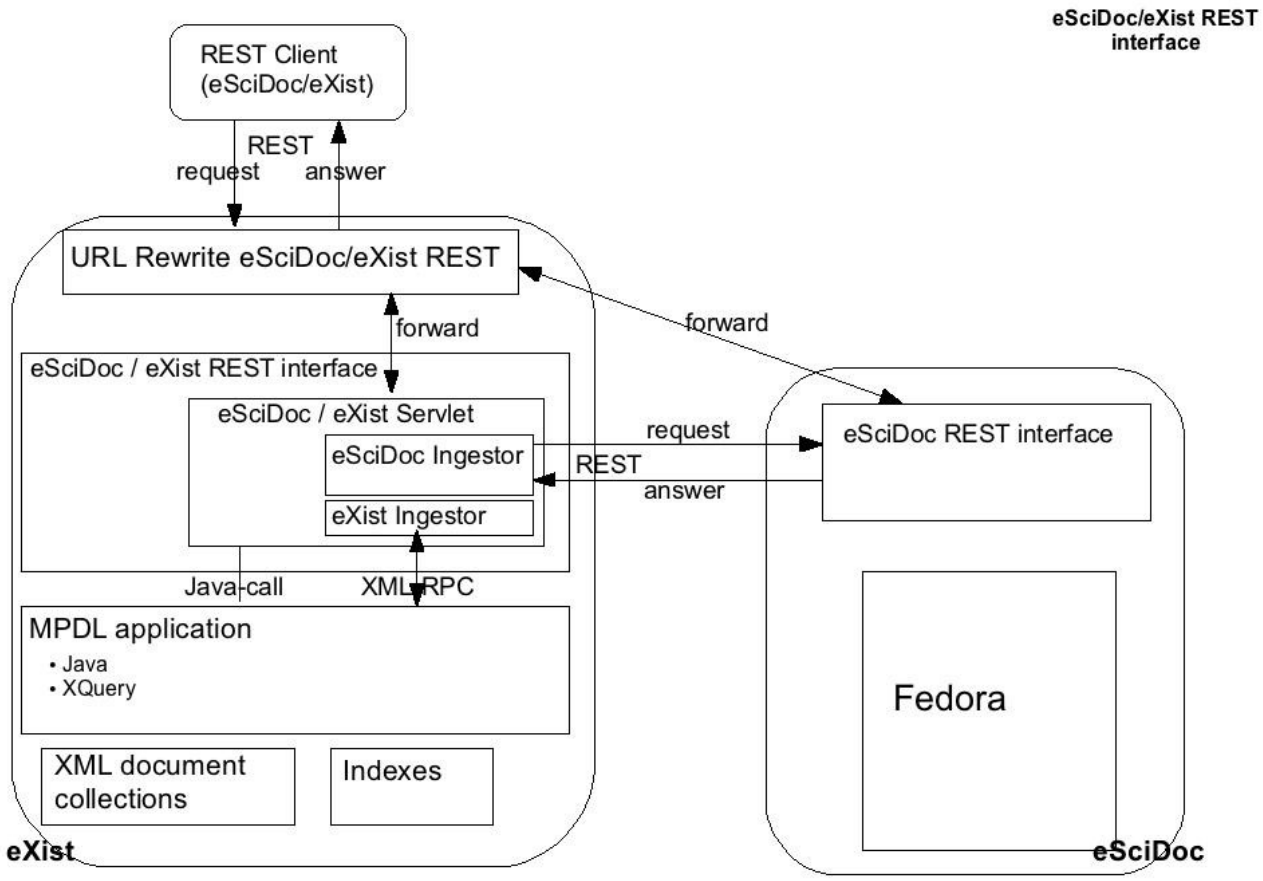
The variant a) seems to be the best way to implement the combined eSciDoc and eXist REST service approach. The advantages are as follows:

- relative easy URL-Rewriting of the incoming REST URL's by a controller.xql: is already tested
- relative easy integration of the EsciDocEXistRESTServlet into the available MPDL-eXist prototype: is already tested
- some of the eSciDoc REST calls are already implemented in the available MPDL eXist prototype and could be called directly by the EsciDocEXistRESTServlet
- MPIWG has an experience of 2 years with eXist
- the main developer of eXist (Wolfgang Meier) is directly connected to MPIWG and could help fast if there are problems with eXist
- the implementation could be set into eSciDoc with relative small effort when a JMS service is available in eSciDoc. So an important demand is that eSciDoc implements a JMS service in its next version.
- the overall implementation effort is smaller than in the other variants

3. Architecture

In eSciDoc fulltext documents are stored as components in an item in a container whereas in eXist they are stored as XML-files in a collection. So an eSciDoc container corresponds to an eXist collection and an eSciDoc item corresponds to an eXist XML file. To integrate eXist functionality in the eSciDoc REST interface some of the eSciDoc REST services are specialized (CUD operations on XML documents) and some are build new (XQuery search and execution).

The architecture of the system could be seen here:



All eSciDoc REST requests are forwarded by the eXist URL Rewrite mechanism. When a „normal“ eSciDoc-URL is requested it is directly forwarded to the eSciDoc REST interface. When a eXist-URL is requested (eXist CUD operation, XQuery search/execution) it is forwarded to the new implemented eSciDoc/eXist REST interface (eSciDoc/eXist-Servlet).

For eXist CUD operations the type of the item/container has to be determined. For this we have different possibilities:
  a) item/container with ContentModel "exist:xml" (/cmm/content-model/escidoc:exist-xml)
  b) item/container with Context "exist:xml" (/ir/context/escidoc:38600)
  c) item/container with a metadata field "<dc:format>text/xml</dc:format>"
Variant a) seems to be the best solution and will be selected.

The namespace „exist:“ is added to be compatible with later parallel extensions of eSciDoc by other XML processing systems such as BerkeleyDBXml or MonetDB with their own namespaces.

4. REST interface

To integrate eXist/XQuery processing into eSciDoc some of the available eSciDoc REST services are specialized (CUD operations) and some services are built new (XQuery search / execution).

4.1. CUD operations

4.1.1. Create item

Create a new item of type „exist-xml“ and add it to the member list of the container with the provided id.

HttpRequest	POST /ir/container/<container-id>/create-item
Input from Uri	<*container-id*> The id of the Container.
Input from Body	The XML representation of the item to be created corresponding to XML- schema "item.xsd". The eXist identifier has to be set in „/item/md-records/md-record/metadata/exist-identifier“
Output	The XML representation of the created item corresponding to XML-schema "item.xsd".
Possible errors	...

The request header contains the eSciDoc cookieId. The item of type "exist:xml" is set as RequestEntity onto the PostMethod.
During execution of the method first the item of type "exist-xml" is created in the given eSciDoc container (by an eSciDoc REST call). Second this item is created in the respective eXist collection (by an eXist XML-RPC call).
Result (responseBody) of the operation is an item of type"exist:xml" with the new eSciDoc itemId.

For example an item of type exist:xml could be created by the following Java HttpClient calls:

```
File itemXmlFile = new File("/your/path/yourItemFile.xml");
String urlStr =
  "http://mpdl-proto/mpdl/escidoc/ir/container/escidoc:4711/create-item";
PostMethod method = new PostMethod(urlStr);
method.setRequestHeader("Cookie", "escidocCookie=" + cookieId);
FileInputStream requestFileInputStream = new FileInputStream(itemXmlFile);
InputStreamRequestEntity inputStreamRequestEntity =
  new InputStreamRequestEntity(requestFileInputStream);
method.setRequestEntity(inputStreamRequestEntity);
httpClient.executeMethod(method);
String resultItemXmlStr = method.getResponseBodyAsString();
```

4.1.2. Update item

Update an item of type „exist-xml“ .

HttpRequest	PUT /ir/item/<item-id>
Input from Uri	<*item-id*> The id of the Item to be updated.
Input from Body	The XML representation of the Item to be created corresponding to XML- schema "item.xsd". The eXist identifier has to be set in „/item/md-records/md-record/metadata/exist-identifier“
Output	The XML representation of the updated Item corresponding to XML-schema "item.xsd".
Possible errors	...

The request header contains the eSciDoc cookieId. The item of type "exist:xml" is set as RequestEntity onto the PutMethod.
During execution of the method first the item of type "exist-xml" is updated in eSciDoc (by eSciDoc REST call). Second this item is updated in eXist (by eXist XML-RPC call).
Result (responseBody) of the operation is an item of type"exist:xml" with the new eSciDoc modification date.

For example an item could be updated by the following Java HttpClient calls:

```
File itemXmlFile = new File("/your/path/yourItemFile.xml");
String urlStr = "http://mpdl-proto/mpdl/escidoc/ir/item/escidoc:4711";
PutMethod method = new PutMethod(urlStr);
method.setRequestHeader("Cookie", "escidocCookie=" + cookieId);
FileInputStream requestFileInputStream = new FileInputStream(itemXmlFile);
InputStreamRequestEntity inputStreamRequestEntity =
  new InputStreamRequestEntity(requestFileInputStream);
method.setRequestEntity(inputStreamRequestEntity);
httpClient.executeMethod(method);
String resultItemXmlStr = method.getResponseBodyAsString();
```

4.1.3. Delete item

Deletes an item of type „exist-xml“ .

HttpRequest	DELETE /ir/item/<item-id>
Input from Uri	<*item-id*> The id of the Item to be deleted.
Input from Body	No input values
Output	No return value
Possible errors	...

The request header contains the eSciDoc cookieId.
During execution of the method first the item of type "exist-xml" is deleted in eSciDoc (by an eSciDoc REST call). Second this item is deleted in eXist (by an eXist XML-RPC call).
Result (responseBody) of the operation is the deleted item.

For example an item could be deleted by the following Java HttpClient calls:

```
String urlStr = "http://mpdl-proto/mpdl/escidoc/ir/item/escidoc:4711";
DeleteMethod method = new DeleteMethod(urlStr);
method.setRequestHeader("Cookie", "escidocCookie=" + cookieId);
httpClient.executeMethod(method);
String resultItemXmlStr = method.getResponseBodyAsString();
```

4.1.4. Create container

Creates a new container of type „exist-xml“ and add it to the member list of the container with the provided id.

HttpRequest	POST /ir/container/<container-id>/create-container
Input from Uri	<*container-id*> The id of the Container.
Input from Body	The XML representation of the Container to be created corresponding

	to XML- schema "container.xsd". The eXist identifier has to be set in „/container/md-records/md-record/metadata/exist-identifier“
Output	The XML representation of the created container corresponding to XML- schema "container.xsd".
Possible errors	...

The request header contains the eSciDoc cookieId. The container of type "exist:xml" is set as RequestEntity onto the PostMethod.
During execution of the method first the container of type "exist-xml" is created in the given eSciDoc container (by eSciDoc REST call). Second the eXist collection named in the container in „exist-identifier“ is created (by an eXist XML-RPC call).
Result (responseBody) of the operation is a container of type"exist:xml" with the new eSciDoc containerId.

4.1.5. Update container

An eSciDoc container is not the same as an eXist collection. For example in eXist a collection could not be renamed directly. So an update of a container of type „exist-xml“ is possible but it has no effect on eXist and only the eSciDoc update of the container is done.

If an eXist collection has to be renamed then this operation could be done in two steps. First the container of type „exist:xml“ is deleted (see below). Second a new container with the new exist identifier is created (see above).

4.1.6. Delete container

Deletes a container of type „exist-xml“ .

HttpRequest	DELETE /ir/container/<container-id>
Input from Uri	<container-*id*> The id of the container to be deleted.
Input from Body	No input values
Output	No return value
Possible errors	...

The request header contains the eSciDoc cookieId.
During execution of the method first the container of type "exist-xml" is deleted in eSciDoc (by eSciDoc REST call). Second the eXist collection named in the container in „exist-identifier“ is deleted (by an eXist XML-RPC call).
Result (responseBody) of the operation is the deleted container.

4.2. XQuery search / execution

eSciDoc search ("Search and Browse" and Filter-Query) is extended by XQuery search and execution. For searching or executing XQueries, the server is called with a specific url for these operations (see below) and additional request parameters:

Request	

parameters	
query	Contains a query expressed in XQuery to be processed by the server (see http://de.wikipedia.org/wiki/XQuery and http://exist.sourceforge.net/)
queryPath	Contains a path to the xquery script on the XQuery server e.g. „/yourApplication/query.xql“
parameters	Contains parameters for the query e.g.: <params> <param name=“queryType“>fulltextMorph</param> <param name=“document“>/echo/la/Benedetti_1585.xml</param> <param name=“mode“>text<param> <param name=“query>multiplicare</param> <param name=“queryResultPageSize“>20</param> </params>
maximumRecords	The number of records requested to be returned. The value must be 0 or greater. Default value if not supplied is determined by the server. The server may return less than this number of records, for example if there are fewer matching records than requested.
startRecord	The position within the sequence of matched records of the first record to be returned. The first position in the sequence is 1. The value supplied must be greater than 0. Default value if not supplied is 1.

4.2.1. Search in items

Search in items of type „exist:xml“. XQueries could be given as XQuery strings or as external scripts as a path with parameters

HttpRequest	GET /exist:xquery/execute?query=<"your xquery"> &maximumRecords=100&startRecord=1 GET /exist:xquery/execute?queryPath=<"/your/path/yourXQuery.xql"> &parameters=<"yourParameters">
Input from Uri	All parameters mentioned in table „Request parameters“ above
Output	xml document in searchRetrieveResponseType-schema that contains the search records, information about the search request, number of hits etc. (see http://www.escidoc.de/schemas/rest/search-result/0.7)
Possible errors	...

For example the MPIWG-MPDL morphological search application could be requested by the following Java HttpClient calls:

```
String xQueryParams =
  "<params>" +
  "<param name="queryType">fulltextMorph</param>" +
  "<param name="docbase">archimedes</param>“ +
  "<param name="docbase">echo</param>" +
```

```
  "<param name="ftMorphQuery">quantitas</param>" +
  "<param name="language">la</param>" +
  "</params>";
String urlStr =
  "http://mpdl-proto/mpdl/escidoc/exist:xquery/execute?" +
    "queryPath=/mpdl/interface/query.xql" + "&parameters=" + xQueryParams;
GetMethod method = new GetMethod(urlStr);
httpClient.executeMethod(method);
String resultStr = method.getResponseBodyAsString();
```

4.2.2. Execute XQuery scripts on one item

Exceute XQuery scripts on one item of type „exist:xml“. XQueries could be given as XQuery strings. The eXist document identifier is fetched in the item in path „/item/md-records/md-record/metadata/exist-identifier“ (e.g. /echo/la/Benedetti_1585.xml).

HttpRequest	GET /ir/item/<item-id>/exist:xquery/execute ?query=<"your xquery">&maximumRecords=100&startRecord=1
Input from Uri	All parameters mentioned in table „Request parameters“ above
Output	Result (responseBody) of this operation is the result of the execution of the XQuery in eXist (any output format).
Possible errors	...

For example all sentences in item „/ir/item/escidoc:4711“ could be requested by the following Java HttpClient calls:

```
String xQuery = "//echo:s";
String urlStr =
   "http://mpdl-proto/mpdl/escidoc/ir/item/4711/exist:xquery/execute?" +
    "query=" + "xQuery + &maximumRecords=100&startRecord=1";
GetMethod method = new GetMethod(urlStr);
httpClient.executeMethod(method);
String resultStr = method.getResponseBodyAsString();
```

4.2.3. Execute XQuery scripts

Execute any XQuery scripts. XQueries could be given as XQuery strings or as external scripts as a path with parameters

HttpRequest	GET /exist:xquery/execute ?query=<"your xquery">&maximumRecords=100&startRecord=1 GET /exist:xquery/execute ?queryPath=<"/your/path/yourXQuery.xql"> &parameters=<"yourParameters">
Input from Uri	All parameters mentioned in table „Request parameters“ above
Output	Result (responseBody) of this operation is the result of the execution of the XQuery in eXist (any output format).
Possible errors	...

For example the MPIWG-MPDL morphological search application for document „/echo/la/Benedetti_1585.xml“ could be requested by the following Java HttpClient calls:

```
String xQueryParams =
  "<params>" +
  "<param name="queryType">fulltextMorph</param>" +
  "<param name=\"document\">/echo/la/Benedetti_1585.xml</param>"
  "<param name=\"mode\">text</param>“ +
  "<param name="query">multiplicare</param>" +
  "<param name=\"queryResultPageSize\">20</param>" +
  "</params>";
String urlStr =
  "http://mpdl-proto/mpdl/escidoc/exist:xquery/execute?" +
    "queryPath=/mpdl/interface/doc-query.xql" + "&parameters=" + xQueryParams;
GetMethod method = new GetMethod(urlStr);
httpClient.executeMethod(method);
String resultStr = method.getResponseBodyAsString();
```

5. Literature

MPIWG-MPDL project: see https://it-dev.mpiwg-berlin.mpg.de/tracs/mpdl-project-software

eSciDoc core services: see https://www.escidoc.org/JSPWiki/en/CoreServices

eXist: see http://exist.sourceforge.net

MPDL project: see http://mpdl.mpg.de

6. Supplement: some internal implementation stuff

escidoc.cmmId=/cmm/content-model/escidoc:persistent4 == "ct"
escidoc.contextId=/ir/context/escidoc:38600
escidoc.archimedesContainerId=/ir/container/escidoc:42507
escidoc.echoContainerId=/ir/container/escidoc:38602