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

Software comparision and selection

Version	1.0
Author	Josef Willenborg
Created	17.09.2008
Last modified	11.11.2008
Last modified by	Josef Willenborg

Content

1. Software comparision

We examine different software systems by means of their main functions and features for our user requirements in the area of web based access to XML-documents. We limit the examination to three systems: Lucene, eXist and Oracle. These three systems promise the best results for our basic needs but if they are proved not to be applicable systems such as Tamino, DB2 and Postgres could be examined.

Functions which are mandadory for our requirements are marked bold. If one system has a minus in one of these functions it could not be selected in our decision process.

1.1.Basic features

Function/Feature	**Lucene**	**Oracle**	**eXist**
Open software	++	-	++
Price	++ (free)	+ (development and production: 654 Euro for 2 years)	++ (free)
Customizable (extensible for specific needs)	++	+	++
Easy maintenance and usage	+	+	+
Powerful in functionality	+	++	++
Scalable	+	++	+
Supported platforms	++	++	++
Further development of the software for our needs (fulltext search for old languages, version management in repository etc.)	+ (with 11 main developers)	+ (with many main developers)	+ (with 5 main developers)

1.2.Repository

Function/Feature	**Lucene**	**Oracle**	**eXist**
Repository of files (hierarchical, with folders)	-	+	+
Automatic index update when documents are created, updated or deleted	-	-	+
Trigger for events in folder implementable	-	+ (implementable in PL/SQL)	+ (implementable in XQuery and Java)
Access by HTTP/FTP (WebDAV)	-	+	+
Access by SQL	-	+	-
Versioning of files	-	- (but restricted functionality	- (only planed)

		in repository is available)	
Repository of relational data (as table content; content could also be a reference to a file or URL)	-	++	-
RDBMS with SQL access (for specific needs)	-	++ (complex)	++ (XQuery access to external SQL databases over JDBC driver)

1.3.Indexing

Function/Feature	**Lucene**	**Oracle**	**eXist**
Fulltext index encoded in UTF-8[1]	+	+	+
XML index encoded in UTF-8	+	+	+
Many document formats (xml, pdf, doc, html, ...)	+ (each format has to be implemented)	++ (over 150 formats)	- (only XML, further types with Lucene extension[2])
Range index	+	+	+
NGram index (encoded in UTF-8, option for Chinese)	-	-	+

1.4.Fulltext querying

Function/Feature	**Lucene**	**Oracle**	**eXist**
Query encoding in UTF-8	+	+	+
Wildcard querying: * (left, middle, **right** in the word)	+ (middle, right)	++ (left, middle, right)	++ (left, middle, right)
Case sensitive querying	+ (with filter)	+ (with filter)	+
Logical operators: and, or, andNot	+	+	+
Near operator	+	+	+
Wildcard querying: ? (one character)	-	+	+
Querying with e specific set of characters: e.g. li[v,f]e	-	-	+
Writing similarity (fuzzy)	+	+	+ (with N-Gram index)

1.5.Querying by attributes

Function/Feature	**Lucene**	**Oracle**	**eXist**
Querying with attributes (e.g.	+	+ (XPath	+ (XPath-

1 characters of the form ſ (longs) are encoded as one UTF-8 character internally
2 remark of Wolfgang Meier: „you could still define a trigger on the binary resource and create your own Lucene-based index along with an XQuery extension function to integrate Lucene results into an XQuery search“

author, title)		operator)	operator)
Date range queries	+	+	+

1.6.XML querying

Function/Feature	Lucene	Oracle	eXist
Support of XPath and XQuery	- (but implementable with Java)	++	++
Retrieve the XML fragment between two milestones (see chapter 5)	- (but implementable with Java)	+ (with own servlet)	++ (with own XQuery Java extension)

1.7.Language specific querying

Function/Feature	Lucene	Oracle	eXist
Lemmatizing in different languages, stem operator	+ (11 spoken languages not including italian, greek and classic latin)	++ (all major spoken languages including italian and greek, classic latin)	+ (with Lucene module)
Lemmatizing extensible by programming language	++ (implementable in Java)	++ (Oracle PL/SQL procedure which could call Java methods)	++ (with Lucene module implementable in Java)
Lemmatizing extensible with language specific dictionary	+ (implementable in Java)	++ (build in)	+ (with Lucene module)
Enrichment of the original query with word forms out of an external dictionary	+ (implementable in Java)	++ (not necessary becaus already contained in stemming)	+ (with Lucene module, also implementable in XQuery)
Querying with a thesaurus for documents	- (but implementable in Java)	++	- (with Lucene module implementable in Java)
Thesaurus maintenance and querying in thesauruses	+ (Synonyms with WordNet could be activated)	++	+ (with Lucene module)

1.8.Query results

Function/Feature	Lucene	Oracle	eXist
Encoded in UTF-8	+	+	+
Customizable	++ (Java)	++ (Java, SQL, XPath, XQuery, result tables)	++ (XPath, XQuery)
Sortable alphabtically by fields (author, publication year)	+	+	+
Sortable by relevance	+	+	+(with Lucene module)
Occurences of the query terms could be presented in result document (enrichment of the original XML document with KWIC)	+	+	+

1.9. Web development

Function/Feature	Lucene	Oracle	eXist
Web interfaces	+ (JSP)	+ (JSP, SOAP, XPath/XQuery, WebDAV, dynamic SQL)	++ (JSP, SOAP, REST, XMLRPC, Atom Publishing Protocol, XPath/XQuery, WebDAV)
Development of web applications	+	-	++
XQuery servlet for generating web pages (similar to JSP)	-	-	++
XQuery extensible with own Java modules	-	-	++
Web server with Servlet-Container	+ (Apache Tomcat)	++ (built in proprietary web container)	++ (built in Jetty, also Tomcat possible)
Example web application for querying documents (with KWIC etc.)	+	-	++

1.10.Performance

See chapter 3 for more precise information.

Function/Feature	Lucene	Oracle	eXist
Query response time	++	++	++
Indexing time	++	++	++
Document size	++ (> 2 GB, efficiency is no problem)	++ (18 MB is tested, should be much more)	++ (but system hangs if it is too big (> 100 MB))
Number of documents	++ (> 1.000.000)	++ (many, highly scalable)	++ (up to $2^{31)}$

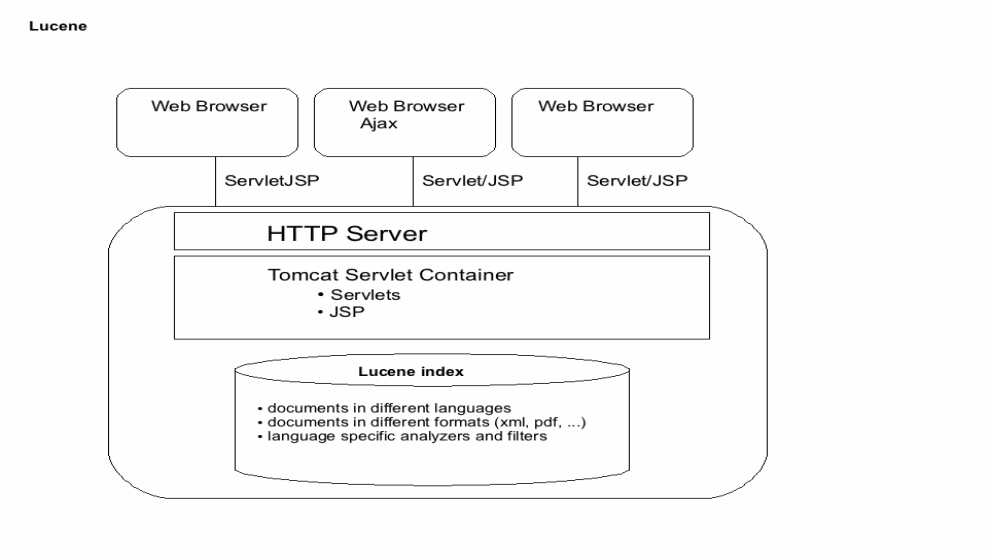
1.11. Bugs[3]

Function/Feature	Lucene	Oracle	eXist
little server crashes	++	++	++
little memory leaks	++	++	+ (Java heap space problem with special XQueries in sandbox)

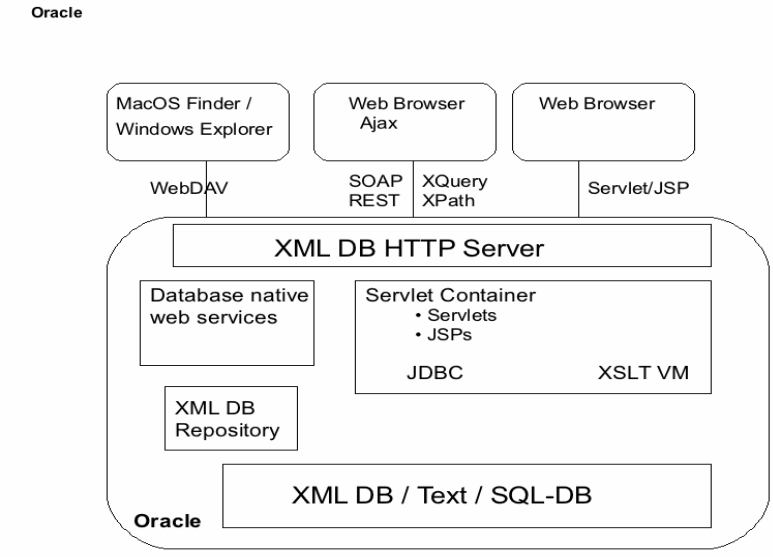
3 limited functional test (no mass test)

2. Architecture

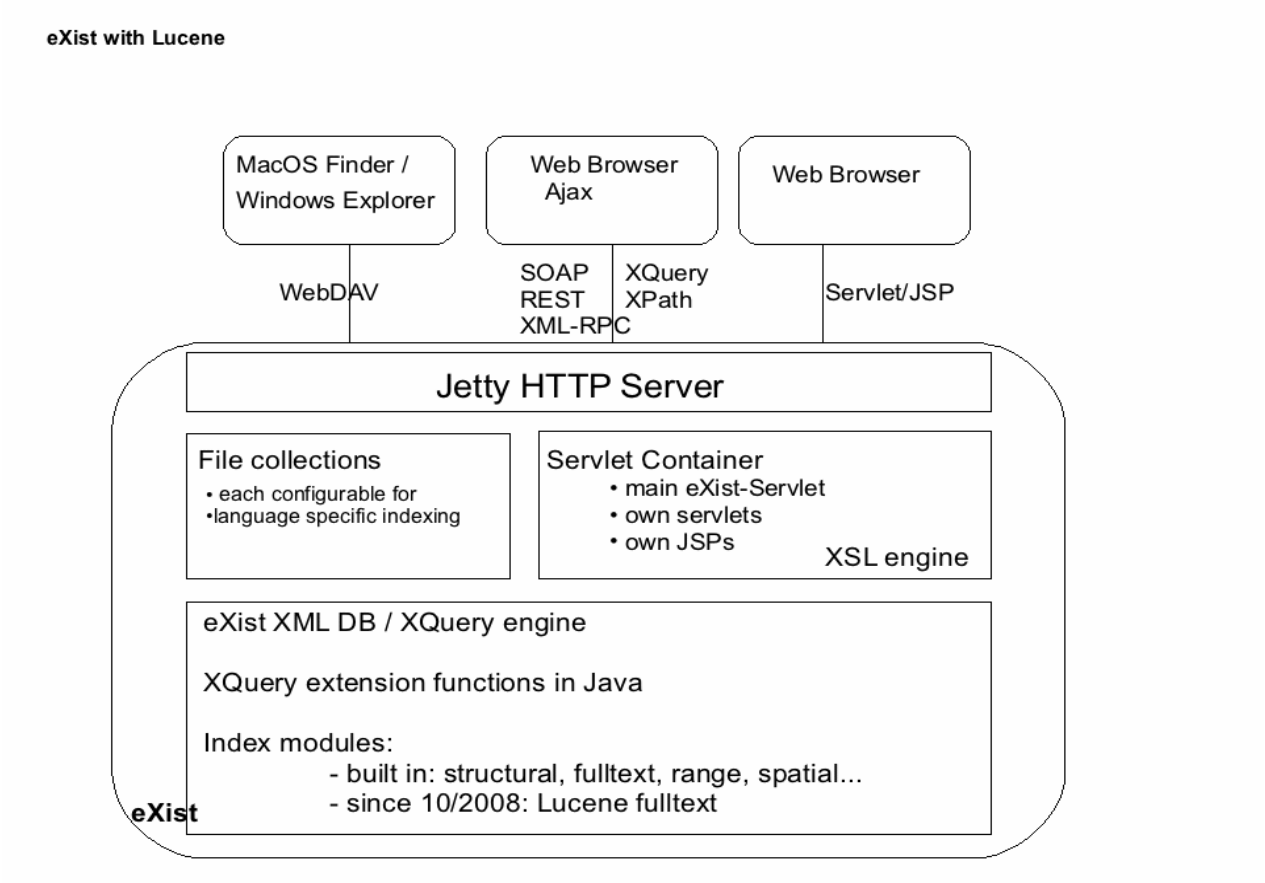
2.1.Lucene



2.2. Oracle



2.3. eXist with Lucene



3. Performance test

3.1.Lucene

Hardware	Document base	Indexing time	Query time	Remarks
Mac Pro with 2 x 2,66 Ghz Dual Core Intel Xeon Processor with 2 GB RAM	500 big XML documents (each with 300 pages)	?? minutes 8 hours (incl. OCR-index)	Fulltext: < 0,3 sec	OCR index included

3.2.eXist

Hardware	Document base	Indexing time	Query time	Remarks
Suse Linux Server AMD Opteron Processor, 6 GB RAM	11000 small XML documents (< 100 KB) and a few big XML documents (15 MB)	1 hour	Fulltext: < 0,3 sec XML: < 1 sec	BBAW test
Mac Pro with 2 x 2,66 Ghz Dual Core Intel Xeon Processor with 2 GB RAM	111 XML-documents (all 130 MB, between 4 KB - 18 MB) with XML index and Lucene fulltext index	11 minutes	Fulltext: < 0,3 sec XML: < 1 sec	Path operator „/archimedes//s“ which delivers 300.000 result sentences needs 1 sec.; select one specific page needs 1 sec, select 100 specific pages needs 100 sec; some specific queries need longer

3.3.Oracle

Hardware	Document base	Indexing time	Query time	Remarks
Windows XP	111 XML-	7,5	Fulltext:	index: one xml

Workstation with 2 x 2,66 Ghz Dual Core Intel Xeon Processor with 2 GB RAM	documents (all 130 MB, between 4 KB - 18 MB) with XML index and fulltext index	minutes	< 0,3 sec XML: < 1 sec.	type index and one text index on the same table column; some specific queries need longer

Oracle delivers methods to increase the performance:

- schema definition for the document column produce automatically object relational data with better indexes so that XML queries are faster (XPath rewrite).
- function based indexes for specific needs
- scaling of the hardware

4. Datastore

A special requirement in our project is the persistent publication of documents by authors or institutions. Documents of different versions and formats (XML, PDF, etc.) should be archived with their metadata and in different document collections in a persistent way. For this an open datastore system should be integrated into our overall indexing and querying system. Special operations such as create, update and delete on versions of documents and document collections have to trigger the same operation to the indexing system.

Candidates for such a datastore system are Subversion (with own extensions), Zope and Fedora.

5. Retrieve the part between two milestones

A special requirement is to retrieve and show the XML fragment between two milestones (in TEI this are the elements „milestone“, „pb“, „cb“, „lb“) in a web browser. For example in the document „caver_metod_020_it_1891.xml“ we want to search for the XML fragment between page break n=464 and n=465. The problem is that page breaks could occur at any place of the XML document (e.g. just in the middle of an italic markup in a paragraph or in a table etc.).

With Oracle the operators in XQuery for querying such a fragment are supported (following::*) but they are not performant (response times over 5 seconds) and also deliver only in simple cases the desired results.
With eXist these operators in XPath and XQuery are not supported yet (following::* is not possible).

A solution could be to make a „presentational copy“ of the original XML document where the surrounding elements of milestones are resolved to „presentational elements“ which have an explicit begin and end mark:
The preceding elements of a milestone m wich do not have an end mark before m are explicitely end marked just before m and the following elements of a milestone m which do not have a begin mark after m are explicitely begin marked just after m.

For example the original XML fragment with a „pb“ milestone:

```
<pb n=“464“/>
...
<p n=“4711“><s=“4811“><it>Dies ist
<pb n=“465“/>
ein Satz.</it></s></p>
...
<pb n=“466“/>
```

would be resolved presentational as:

```
<pb n="464">
...
<p n=“4711“><s n=“4811“><it>Dies ist</it></s></p>
</pb>
<pb n=“465“>
<p n=“4711“><s n=“4811“><it>ein Satz</it></s></p>
...
</pb>
<pb n=“466“>
...
</pb>
```

One solution could be to resolve the milestones online at query time with a programming language. For example in Java this is simple and performant. It needs only 0,2 seconds to find such a fragment at the end of a 10 MB file if the file is directly acessible in the file system. Then the surrounding elements have to be resolved which does not need much time.
In eXist one could extend XQuery with own Java modules. Also the access to the XML file could be done directly (by the XML DB API) so that the performance of that operation should be good.
In Oracle a special servlet would have to be implemented and deployed to the Oracle servlet container which does access the database for the XML document. This could cost

some performance.

Another solution for resolving the milestone elements is to preprocess all original XML documents with a programming language. There are two ways in doing this:

1. make a „presentational copy document file“ of each XML document file
2. make a „presentational copy document file“ of each page of each XML document file

These presentational files are also indexed in eXist with the structural index.

Disadvantage of this solution is the double index space for the presentational copies. But this solution is preferred because the query system would be much faster for the very frequent operation of getting a page in a document.

6. PDF documents

For integrating the document format pdf into the query and indexing system there are two ways:

1. in eXist a trigger has to be defined on the binary resource for the pdf documents to update the separate Lucene index and also an XQuery extension function has to be defined to integrate Lucene results into an XQuery search.
2. pdf documents are converted to XML documents with CDATA sections for each page and indexed just the same as the XML documents

7. Decision

7.1.Indexing and query system

Lucene could be the system only for fulltext querying and also for a special integration of own implemented language technology in fulltext searching. Other document formats such as pdf could be integrated with some additional programming effort. A structural index for XML queryies is not supported. Lucene is open software.

eXist (together with the Lucene module) could be the system for both fulltext querying and structural querying and has a built in web engine/container and a basic datastore system. Web applications could be implemented relative easy with the XQuery module which also could be extended in Java for own functionality. Language technology could be integrated with own Java implemented analyzers for Lucene. Further document formats such as pdf could be integrated with relative little effort. eXist is open software.

Oracle could be the system for both fulltext querying and structurural querying and has a built in web engine/container and a basic datastore system. For our language specific needs it has many built in functionality so that less own implementation is necessary. But for extending the language technology vendor specific PL/SQL procedures have to be implemented. Other document formats such as pdf could be integrated for fulltext querying with no implementation effort. Oracle is no open software.

We decide to use eXist with Lucene because:

1. requirements: all mandatory requirements and also many other requirements are satisfied
2. architecture: XML and fulltext query system in one system (with the new Lucene module), many web interfaces.
3. development of own web applications: is easy, clear design is supported.
4. performance/stability: first function and performance test was good.
5. extensibility: could be extended for our needs (language technology, other document formats, repository trigger, etc.). It could also be extended with Java modules, for example to support performant file search methods etc.
6. language technology: Oracle has the most promising built in functionality but the upsetting and extension of that technology is complicated. In eXist with Lucene there is less built in technology but it is more easy to extend. If we find out later that Oracle is easier and better for our language technology than we could trace back to this solution for our fulltext query system
7. repository: a basic repository system already exists. The versioning of documents could be available in one of the next releases.
8. open software: yes.

7.2. Datastore system

The descision for the datastore/repository system will be discussed and reasoned in an externalized document.