ABSTRACT
This paper proposes a solution to a problem with XML storage and retrieval. A real life case study is used to illustrate a scaling problem with XML storage and slow retrieval speeds. The solution can be found in several ways and this research summarizes and evaluates different storing methods and their advantages and disadvantages. This research can assist in finding a storage solution for XML sets according to specific requirements. Using weighting factors on all the characteristics of the different storing methods a Native XML storage solution ended up as the best solution. This resulted in a further research for the best Native solution, which turned out to be Xindici for this specific case study.

Keywords
XML, XML storage, XML retrieval, native XML databases

1. INTRODUCTION
As more and more companies and institutes are using XML for themselves or are confronted with XML by using certain software packages, the need for a solid, scalable and fast XML storage solution becomes more urgent. Storing XML in such a way that information can be retrieved from it in a fast way is a study in itself and many projects and research is done on behalf of this subject.

The real time case study show a problem of a quite general kind and such a problem can be found in any kind of company that handles large amounts of data. The Cleanport (CP) problem, further explained in section 2, is just used to illustrate a scaling issue and to demonstrate how certain requirements can lead to the choice of a native XML database solution [VCM05] instead of a mapping of the documents XML schema into a relational database schema [Sch02] [Tat02] [Sha99] or choosing a file system [VCM05] or Directory Server [Fel01] solution.

2. CASE STUDY DISCRIPTION
The earlier mentioned company, Cleanport, offers Managed E-mail Security (MES), their Platform for Intelligent E-mail Distribution (PIED) analyses millions of messages every day and checks for viruses, spam and bad content.

The general idea behind Cleanport is that the client does not need to install any client side software. The MX records from their domain point directly to the mail servers from Cleanport. Their systems analyze the mail and deliver the non-spam/non-virus e-mails to the email server of the client. All the spam and viruses are put in quarantine, which is accessible online for the end-user to see if no false-positives are filtered out.

A part of this platform is the system that generates all the statistics of the filtered emails on different levels (tower, server, client, domain and user). This system handles a lot of data (for all the users and domains) and has to be fast for the end user and reliable for Cleanport, as their billing system points directly to the mail servers from Cleanport. This field is a plain text field that holds the XML which contains all the statistics for the particular record. This field is a plain text field so it is not even a Blob or something similar which would make it a bit better. On top of this the database grows drastically every year since they have to keep a history of all the statistics.

2.1 Statistics system
The existing system uses the following technique; all the servers in one tower generate a log file which contains a single row of text per scanned email. All these log files are merged into one big file and this file is parsed by a process. This process generates the statistics in an XML-like markup, which then is saved in a data field in the database for every separate record (per domain, user, etc.). To display these statistics on request the XML field is retrieved and is parsed using an XML DOM parser, the XML Document Object Model (XML DOM) defines a standard way for accessing and manipulating XML documents and is defined by the World Wide Web Consortium (W3C) (http://www.w3.org/), which is the main international standards organization for the World Wide Web.

2.2 Problems
There are several problems that arise in this setup which will be highlighted in the following paragraphs.

2.2.1 Data structure
First of all it is absolutely not scalable, all the data ends up in one MySQL database which will contain millions of records spread over just a few tables. Each record is very large because it contains a large text field that holds the XML which contains the actual statistics for the particular record. This field is a plain text field so it is not even a Blob or something similar which would make it a bit better. On top of this the database grows drastically every year since they have to keep a history of all the statistics.

2.2.2 Programs
The previously mentioned problems all have to do with the actual data that has to be saved and the way this is done, but there are also the programs that use the database to insert, update or fetch records. One of these processes is a program that converts all the totals from each day into a monthly total, this process runs at the end of every month. Currently this process is very processor intensive and it takes a lot of time due to the low input/output speeds.

2.2.3 End user
Another problem exists for the end users, when a user logs into his account and he requests his statistics, a query on the database has to be done to fetch the XML. This is fast enough if not too many users do it at the same time, but as it occurs a lot of users make this request in the same time frame which results in low speeds for all of them.

2.2.4 Query’s
All the query’s that are done in the system currently all just select the XML field in the database and then parse that field themselves using an XML DOM parser. This is not very effective as new methods using an XML query language can do this way faster, this we be discussed later.

2.2.5 Database locks
On a daily basis the statistics are updated by the process which parses the log file. This process locks the table as it updates the
new statistics. This results in a problem for the end users as the database becomes unreadable during this lock.

2.3 Problem statement
All the problems mentioned above have to do with the way the data is stored and the way this data can be retrieved. As no XQuery [GT04] or XPath [Olt02] is used to update the XML fields directly in the database or to retrieve just the part we need from the XML field, the system is by far not as fast as it could be. Thereby in its current structure it will keep growing every year and therefore it will slow down drastically every year. A proper solution has to be found to store the XML files that contain the statistics in such a way that it is fast and scalable for many years to come. We note here that just buying more hardware (which becomes cheaper at a constant rate according to Moore’s law) is no permanent solution for the problem, it just postpones the problem.

3. RESEARCH METHOD AND QUESTIONS
3.1 Research questions
The following research questions were formulated after an initial research on the subject.

1. How can proper (relational) storage of XML and an XML query language like XQuery or XPath improve speeds and scalability of a large XML platform?
   1.1. What kind of method for storing XML documents is suitable for specific requirements?
   1.2. Are there any open-source projects that can assist in implementing this suitable storage method?
   1.3. Does this storing solution also offer retrieval methods that are sufficient for our requirements?

3.2 Research method
This research is a combination of a literature review with an action research, but only the first phase of the action research is completed. We try to find a solution to the XML storage problem presented by Cleanport, for which an action research is very suitable, however as the answer to the research questions became clearer it showed that just an action research was not the right way to find a solution to the problem. This was for two particular reasons, the first reason is that there is done a lot of direct study on efficient XML storage; we just had to find the right one, this pushed the research towards a literature review. The second reason was that if we wanted to keep it as just an action research and actually implement the new storage solution we would get into trouble with the time that can be used for this research. For these reasons the research questions changed a bit and became more targeting to finding a solution for the problem, this resulted in a literature study combined with the first phase of an action research which embraced XML storage in the whole and a more narrow study on different open-source (we only did study for open-source solutions as Cleanport preferred this) Native XML stores.

4. STORAGE METHODS
Some distinctly different methods are found when we are talking about popular XML storage solutions. These methods differ in the way data is stored and the possibilities to retrieve just partial data or entire XML documents. Also scalability and flexibility are main issues when storing XML sets. We will discuss the different methods and we will lay out the advantages and disadvantages per method.

4.1 File-system-oriented
File-system-oriented XML storage methods store the XML documents as ASCII files directly on the file system or the XML files are stored in a database management system (DBMS) as binary large objects (Blobs) or character large objects (Clobs). This last method is quite memory efficient as the blob or clob manager distributes the byte streams over multiple disk pages on the basis of various criteria [VCM05]. Nevertheless this method shows a big drawback as we have to parse the entire XML document in order to rearrange or access components of a document. As this method is quite out-dated and new way faster and more efficient methods have emerged we will not discuss this method in further detail, we will just point out the main advantages and disadvantages.

Advantages:
- Easy implementation
- Good for small XML sets
- Memory efficient (Blobs/Clobs)

Disadvantages:
- Scalability (bad with large XML sets)
- Updates are hard

4.2 (Object-)Relational DBMS
Relational databases can be used to map XML documents to relational tables, see an example XML document in Figure 1 and its corresponding (rough) mapping in Figure 2.

![Figure 1: Sample XML file](image)

![Figure 2: Sample relational mapping of XML file](image)
For this mapping a lot of good algorithms are available to convert XML nodes and their relations to database tables with their tuples according to the schema of the document (XML schema, DTD, etc.). After such a mapping is complete we have to convert the original queries over XML documents to the related SQL query over the newly created tables and tuples. After these steps we only have to present the result of the query back to the user in correct XML [Sha99].

This method seems to be an alternative to the fact that we are denying the last 20 years of work in developing relational databases and fast querying techniques used with them when we are just looking at new techniques like Native XML stores and the use of new XML query languages. Using this technique we can just insert the XML documents into relational databases and execute queries over them, all this at reasonably good speeds. So one big advantage of mapping XML to relational databases is the exploitation of several good DBMS features such as scalability, concurrency control and recovery services. Nevertheless this method also factorizes XML content into several structures which can make processing of the content less efficient. Therefore it does not guarantee great flexibility and good space utilization [VCM05]. Another aspect is the fact that you need XML schemas or a DTD to efficiently convert the XML document into relational tables. It happens in a lot of cases especially now companies have XML flying around all over the place because it is “the new standard of internet data exchange” that there is no schema available whatsoever, which counts the use of a RDMSN out. What also discourages the use of a RDMS is the factorization of an XML document after it has been inserted into the database. What happens is that the result isn’t equal to the originally inserted XML file because of the mapping when it was inserted and the mapping back as it was requested, this is especially the case with complicated large XML files that have deep trees (like Microsoft Office files).

What we did not discuss so far are the Object RDMS’s which offer the opportunity to store XML files as objects as well as tables, this offers support for abstract data types.

Overall these are the advantages and disadvantages.

Advantages:
- Scalability
- Reliability (besides XML factorization)
- Easy implementation
- Concurrency control
- Recovery services
- Abstract data type support (ORDBMS)

Disadvantages:
- XML document factorization
- DTD or other schema necessary
- Non-flexible
- Updating is hard

4.3 Native XML database
After the introduction of XML there were problems in storing and maintaining large XML sets. At first this was solved by using RDBMS’s for storing and updating XML files. This method is still often used (as described above), but it is outdated and researchers came with solutions to the overlap generated by the use of RDMS solutions; Native XML storage. Native XML storage solutions have been officially defined by the XML:DB Initiative as databases that [VCM05]:
- use the XML document as the fundamental unit of logical storage
- use a logical model for the XML document itself
- require a particular underlying physical storage model

A typical structure of such a solution is shown in Figure 3 which shows the structure of TIMBER [Jag02].

![Figure 3: TIMBER Native XML structure](image-url)
TIMBER is an open-source (not for commercial use and therefore not in our overview) Native XML storage engine developed at the University of Michigan. Here we see the structure of a typical Native XML store which uses managers for the metadata, data storage, indexes and the incoming data.

Lately a lot of Native XML databases emerged which use different methods in storing and retrieving data. For example XPath [Olt02], XQuery [GT04] and XUpdate are implemented on top of a lot of these systems for efficient retrieval and manipulation of stored XML files.

Thereby we see on top of the structure the query parser, optimizer and evaluator, which are used to implement XQuery and XPath (and XUpdate in some cases).

Native XML storage solutions define a logical model to guide the storage of XML documents, similar to the way a table guides storage in a relational DBMS, and thus free the data store from DBMS restrictions [VCM05].

Another important aspect of Native XML solutions is the fact that no XML schema or DTD is required while storing XML files. As the relational mapping of XML files to databases is dependent on these schemas we see that Native solutions offer a great flexibility on this part, without the loss of efficient storage and retrieval.

Advantages:
- Scalability
- Data-access speed
- Reliability
- Improved access performance
- Efficient space utilization
- Semi-structured and configuration free data support

Disadvantages:
- Immature

### 4.4 Directory Servers

Directory servers use the Lightweight Directory Access Protocol for data and representation. LDAP allows access to directories which are organized in a tree hierarchy and therefore they are suitable for storing XML Documents [Fel01]. As this seems to be a solid way of storing the XML it is however not a good method for updating the XML stored in the directories. The update performance of XML files stored in the directories is very low which automatically results in a no-go for the CP system as we need updates every 5 minutes. Nevertheless the use of a directory-server approach can be very good for systems that only want to retrieve and query data in the database as LDAP is optimized for this kind of use.

Advantages:
- Optimized for queries
- Effective data retrieval
- Tree-based

Disadvantages:
- Bad in updating XML

### 4.5 Weighting factors

In our choice for a storage method there are several important factors that make us choose one specific method above the other popular methods. All the factors that can be of importance while searching an XML storage solution are set out against the different methods in Table 1. We have to keep in mind that for every project certain factors weigh heavy while others are of latter importance.

In the case described in paragraphs 2 and 3 we point out several problems, however what we did not mention are the requirements the new system has to meet. Some of the mentioned weighting factors are very important for the statistics system while others can be ignored. We also have to take into account what new features the system might want to use like a real-time system which enables end-users to view their filtering statistics in real-time. For making the right choice we take some factors into account and we give these factors a weight of importance, then we put these factors in a table against the possible methods and we give every factor a certain score per method, as shown in Table 1. This gives us Table 2 were we can see the overall scores per method combined with the weight for each factor.

This concludes in way the best score for a Native XML storage solution, which was quite predictable as native XML data storage tools are a more recent and evolving choice, because they've emerged to resolve and address all the overlap problems which popped up when first using RDMS solutions or just simple file system solutions. It also is the most logic choice as flexibility and update/retrieval speeds are very important for the new CP system which also excludes the use of Directory Servers, as they do not support fast updating of the XML.

However after choosing for a native XML solution the next problem arises at the horizon; which open-source native XML

---

### Table 2: XML storage methods and their score for the CP project

<table>
<thead>
<tr>
<th>Storage method</th>
<th>Ease of implementation</th>
<th>Scalability</th>
<th>Memory efficiency</th>
<th>Use of XML Schema’s</th>
<th>Storage efficiency</th>
<th>XML document factorization</th>
<th>Retrieval</th>
<th>Updating</th>
<th>Flexibility</th>
<th>Weighted total score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cleanport weight</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>-3</td>
</tr>
<tr>
<td>File-system-oriented</td>
<td>2 / 2</td>
<td>-2 / -6</td>
<td>1 / 2</td>
<td>-1 / -1</td>
<td>0 / 0</td>
<td>2 / 2</td>
<td>1 / -2</td>
<td>0 / 0</td>
<td>0 / 0</td>
<td>-3</td>
</tr>
<tr>
<td>(Object-)Relational DBMS</td>
<td>2 / 2</td>
<td>2 / 6</td>
<td>0 / 0</td>
<td>2 / 2</td>
<td>1 / 2</td>
<td>-2 / -2</td>
<td>1 / 2</td>
<td>1 / 2</td>
<td>-1 / -3</td>
<td>11</td>
</tr>
<tr>
<td>Native XML</td>
<td>-1 / -1</td>
<td>2 / 6</td>
<td>0 / 0</td>
<td>1 / 1</td>
<td>1 / 2</td>
<td>2 / 2</td>
<td>1 / 2</td>
<td>2 / 4</td>
<td>2 / 6</td>
<td>22</td>
</tr>
<tr>
<td>Directory servers</td>
<td>-1 / -1</td>
<td>2 / 6</td>
<td>1 / 2</td>
<td>0 / 0</td>
<td>1 / 2</td>
<td>1 / 1</td>
<td>2 / 4</td>
<td>2 / 4</td>
<td>-2 / -4</td>
<td>1 / 3 13</td>
</tr>
</tbody>
</table>

The score is built up from: ‘base score’ / ‘score multiplied by weighting factor’
storage solution, which is allowed to be used commercially, suits our (or your) problem best?

5. Native XML choice
Choosing a Native XML store gives us a new challenge. The choice for a particular store is not based on all of the previously shown factors; new factors emerge as different stores offer different storing, retrieval and manipulation methods. Here we have to find out exactly what queries and methods are currently used for manipulating and storing XML. This will give us exact criteria which have to be met by the new solution.

5.1 Cleanport requirements
After analyzing the current CP system and finding out what future wishes for the new system are present we can choose a well over thought Native XML solution by finding out the important criteria. The most important criteria are now explained into further detail; thereby the importance of each criterion for the CP project is also taken into account.

5.1.1 Use of XML Schema’s (or DTD)
The use of Schema driven XML documents is optional in most of the Native XML stores, but in some of them there are some advantages when using a schema for inserting the documents. One of these advantages is found in query validation while translating relative paths into absolute paths. XML schema’s also improve indexing by using index techniques optimized for the documents schema. These indexes can speed up query times.

Another advantage is reduced space consumption as element tags can be replaced by schema nodes, which take less space then full length tag names.

A last but not least advantage is the speed up of queries that require identification of parent/child relations. This helps to exploit the hierarchical structure of XML documents while executing queries.

The CP system does not use XML schema’s what so ever and is not very likely to do so in the future. The chance that unformatted new statistical data will be entered into the store in the near future is quite big and therefore the need of XML schema advantages is not there right now. However, it is possible to create schema’s for the current data if this shows to maximize the storage efficiency.

5.1.2 Storage efficiency
Some of the systems use a very efficient storage method, especially the ones mentioned above that use XML schema’s to maximize the storage efficiency.

For CP the retrieval and update speeds are way more important then the amount of space used to store a million XML files. If the overhead of the stored data is 30% then this extra 30% of storage capacity is not a problem if it enhances the update and retrieval process. Therefore storage efficiently is not very important for the CP statistics system.

5.1.3 XML document factorization
Document factorization can be of great importance when it has to be possible to retrieve a full XML document from the store instead of just querying some nodes of a document. As some systems store the documents in an underlying relational database or in a tree form it can be quite exhaustive to deliver the full document. Thereby the possibility is present that the document will not be fully equal to the original document because of the mapping.

The CP system is not relying on document factorization as it only needs certain nodes from certain documents when the end user requests these. This makes XML document factorization of a latter importance in making the right choice.

5.1.4 XPath
XPath (XML Path Language) is an expression language for addressing parts of XML documents. The language is based on a tree representation of the XML document and provides the ability, by using parent-child-ancestor-descendant type of relations to navigate trough the tree and find the nodes requested [Olt02]. Thereby it offers a number of functions to do summation, concatenation, type matching and more basic expression language features. More information can be found on the site of the W3C (http://www.w3.org/TR/xpath).

For the CP project such a query language is needed to perform the tasks the system has to do. XPath seems to be sufficient for the currently supported features however in the near feature a wider functionality of the statistics system might require a more extensive query language like XQuery, explained in the next part.

5.1.5 XQuery
XQuery is a query language which can query collections of XML using an SQL-like syntax. XQuery is developed by the XML Query working group of the previously discussed W3C. The earlier mentioned XPath 2.0 actually is a subset of XQuery 1.0 and therefore offers fewer options. XQuery provides the means to extract and manipulate data from documents that can be viewed as XML, such as relational databases or plain XML documents. Nevertheless XQuery 1.0 does not offer features to update XML documents, which has to be done by using XUpdate. For more information on XQuery you can visit the site on W3C (http://www.w3.org/TR/xquery/).

For the CP project it is good enough to have the capability of having either XPath or XQuery, but as mentioned before the preference goes to the support of XQuery, in combination with XUpdate, which will be discussed in the next part.

5.1.6 XUpdate
XUpdate is a project from the XML:DB initiative (http://xmldb-org.sourceforge.net/), who did not want to wait for XQuery 2.0 which is intended to support similar update capabilities. XUpdate has not become an official standard, but nevertheless it is used by a wide group of adaptors who also wanted descent update capabilities without waiting for XQuery 2.0.

For the CP project it will be very useful to be able to update XML documents directly, as we need to update user/server/etc. data on a daily basis per document.

5.1.7 Flexibility
Flexibility of a data store is a very interesting topic which also looks back at the use of XML schemas and the conventional mapping of documents to relational databases. As some stores offer great flexibility in accepting any kind of XML document, given it has a correct syntax, or even images.

For CP it is important to have a flexible solution as some feature projects might also do statistics using this data store, not knowing what type of XML will be entered into the system then, and whether is schema-less or not.

5.1.8 Reliability
Reliability of a data store is mostly concerned with the capability of restoring an XML document from a relational storage solution to its original format. However when using a Native XML Solution we do not have to worry about this as most of the stores don’t chop the XML documents into pieces before storing it, this prevents the possible loss of integrity after restoring a document. Native XML stores use a tree representation of the XML documents or they store the XML files in collections, which makes it very easy to factorize the original XML document without the possible loss of content or structure. Another point of reliability is of course the stability of the system as it has to be able to work at high performance rates 24/7 which makes the choice within the CP project a bit harder
as we have to be sure that the system will not break down twice a week or so. Therefore reliability will also concern the possibility of running mirrors of the data store to prevent the store from crashing and doing so losing data and not being able to process requests of the end-user or the data stream from the internal system.

5.1.9 Large XML files
Large, complex XML files like Microsoft Office XML files can be structured in a very complex way and can therefore get quite large. Not every store is good in processing and handling these very large files, while others are built to handle just them.

For CP the current statistics system only uses very easy data-centric shallow XML files that mostly contain numbers as values and merely no attributes or document-like content. Therefore we have to look for a store that handles a lot of small files good instead of large files.

5.1.10 Collection-based / Tree-based
Some data stores use a tree-based representation of the XML documents which mean that the document is stored in a internal tree-representation based on the documents hierarchical structure. This makes it easier to access specific parts of a document instead of fetching an entire document. Using collections is another approach of storing the documents in which related XML documents are stored into collections of one of the following types [VC05]:

- Typed collections which contain a schema (based on the XML schema or DTD) and all documents in the collection must conform to that schema.
- Untyped collection contain ant number of XML documents, regardless of their schemas and how they relate.
- Hierarchical collections, which are organized as file system directories. These collections can be nested, and, because each contains a root collection, they’re considered part of an XPath query string.

Collections are especially efficient for applications that involve querying and indexing data sets whereas tree-based stores are efficient for navigation oriented applications because the tree is easier to parse than linked tables.

Within the CP project we will be looking for a tree-based store as we want to request specific nodes within the document, and we also want to update specific nodes within the document.

5.1.11 Document-centric / Data-centric
Document-centric XML is most of the time not very structured and pre-set following a strict DTD, which makes the use of a DTD capable store less efficient. When we look at data-centric XML there are some stores that use an underlying relational database which can be very effective in storing data-centric XML.

For CP we will be looking mostly for a solution that support data-centric as well as document-centric XML since we will be inserting both, however most of the documents will be data-centric and therefore our preference goes to data-centric document support.

5.1.12 Real-time handling
Real-time handling of incoming XML streams is a feature not all stores support. The current statistics system runs in batches per day, but a dream come true would be to update statistics every 5 minutes or so. This would mean that the system will need to update XML files all the time, and not batched. Some systems also offer input stream capabilities which will have to be explored into further detail to see what this can do for the new statics system.

5.1.13 Summation queries
The current system runs a script every end of the month to make a monthly total of all the days of the month. This total is then stored in the XML file itself. A new and way better way would be to just query the total out of the XML file by summing up all the daily totals on the fly using XPath or XQuery. This would mean we don’t even need to store this data if the query is fast.
enough for displaying it to the end user on the moment he requests it.

### 5.1.14 Scalability

An important problem and one of the main reasons of this research is that the old system was not scalable. The size of the entire filtering system has grown way larger and thus the statistics system can support. This problem has to be prevented in the new system. The growing capacity of the company is fairly large and we don’t want to implement an entire new system after only 2 years. This means that the new store has to be able to handle a lot of XML files while keeping the queries fast. Some of the stores show great performance while handling millions of files, as an example eXist can handle 2^{11} files. The system has to be able to grow with the rest of the company in the near future without many time consuming modifications.

### 5.1.15 Connectivity

All the stores have their own interfaces for communicating with the applications that have to use the data store as XML storage solution. Some stores have stand-alone interfaces for browsing and searching the store for specific content, others only offer an XML-RPC server to connect with from your own interface and insert and manipulate data in the store.

For CP we will be mostly looking for a data store that supports XML-RPC or any other widely supported communication interface so we can talk with it from every imaginable part of the system. In that way we will be capable of inserting statistics from the entire platform.

### 5.2 Choosing the best solution

The given scores in Table 3 are based on average functionality of the system based on fact sheets and whitepapers and not on suitability for the CP system. This means we have to combine the scores of the tables with our wishes that are discussed in 6.1.1 to 6.1.14 and make a choice for a store that suits our problem best. This is again done by using weighting factors in the table. Please keep in mind that every problem has its own specific requirements, only use the table as a guide into choosing an open-source store. Some open-source Native XML stores are left out of the table as they are no real option. Under these are XDBM, XDB and DBDOM as they offer a small feature set and show no real potential regarding development and support.

From Table 3 we can derive that Berkeley DB XML offers good support for the features we need, however the Berkeley DB is a stand alone database which can be used inside an application, if we want to use the database for multiple purposes we cannot connect to it directly through an XML-RPC interface or something alike. A solution would be to build an own interface on top of the database, but this could be time consuming and unsupported.

Because we want a collection based storage method (and because they also lack XUpdate) 4Suite, myXMLDB, Ozone and XpSQL are not taken into further consideration.

The best solution, taken into account that we do not need XML Schema support, seems to be eXist, it offers full support of all the important features and it is a widely developed and supported store. However, if we take a closer look at Xindice it can also be a good choice, despite the lack of XQuery support.

Xindice is build on the code of dbXML which was given to the Apache XML Project group (http://xml.apache.org/xindice/) for further development. Xindice actually presents itself as a Native XML database which is good in storing large XML sets which consist of small XML documents; this is exactly what we need. If we furthermore take into account that Xindice is developed under the hood of Apache it is a wise choice for the CP project to choose it.

### 6. DISCUSSION

Although this research ends up in choosing a Native XML store the other options should not be fully denied when searching for an XML storage solution. Directory servers show great performance when used for storing data that only has to be queried and retrieved. When the XML collections are not very large and updated/queryed a lot then even a simple File System solution using Blobs or Clobs in a database can be sufficient and time saving. But as mentioned before, every project has its own requirements and these should be taken into consideration before choosing a specific XML storage solution. This research only tries to help in choosing the right solution by considering all possibilities and systems available in open-source. A further study can go deeper into Native XML solutions as there are also a lot of commercial solutions available that might be better and faster, which are fully denied in this research.

### 7. CONCLUSIONS

When searching for an XML storage solution, specific requirements can lead to the right choice. As shown in this research, using the CP example, a Native XML store shows great perspective in storing different sorts of XML. Although the use of a relational database has proven to be effective as it builds on years of study, Native XML stores are exploring new and better methods for processing and storing large XML collections. When a storage solution has to be found for storing large XML collections that consist of small XML documents that are updated frequently Xindice shows the right features and good support and therefore it is chosen as XML storage solution for the CP project.

### 8. FUTURE RESEARCH

Future research can continue the action research that was initiated by this research by actually implementing the proposed solution (act) and then observe how it functions in the existing system. After these observations reflections can be made which can introduce another cycle of the action research.

Future research can also look into the commercially available native XML solutions and do a quality comparison with the open-source solutions discussed in this research.

### GLOSSARY

<table>
<thead>
<tr>
<th>CP</th>
<th>Cleanport</th>
</tr>
</thead>
<tbody>
<tr>
<td>MES</td>
<td>Managed E-mail Security</td>
</tr>
<tr>
<td>PIED</td>
<td>Platform for Intelligent E-mail Distribution</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>(O)(R)DBMS</td>
<td>(Object) (Relational) Database Management System</td>
</tr>
</tbody>
</table>

### ACKNOWLEDGMENTS

I like to thank Cleanport for the possibility to examine their system and for assisting me in my research. I would also like to thank Dr. Moody for assisting me during the research and in choosing this interesting topic.

### REFERENCES

[Tat02] Igor Tatrinov et al, "Storing and Querying Ordered XML Using a Relational Database System", *SIGMOD Conference*, 2002


