Abstract

Since the born of XML, more and more bioinformatics tools and biological data sources have embraced it as a common way to publish results. Other previous attempts, like ASN.1, have not been so widespread adopted as XML because they were created in the wrong moment, when the computational power was valuable treasure. Now, the volume of available information in molecular biology and bioinformatics in XML format is big enough to think on the use of XML databases. In this paper we will talk about XML technologies, places where XML is being used, the alternatives to mining data sources in XML, and our experiences in that field.

Brief introduction to XML

XML[1] (eXtensible Markup Language) was created in 1998 as an attempt to simplify the complexity of SGML. In fact, XML is a subset of SGML, aimed to lower the hardware requirements of SGML language parsers and processors, meanwhile the information correctness could be checked in a reasonable time. Other reason to create XML was to separate the data representation from the data itself. This is very useful, for instance, when you have published some information and someone else wants to extract, process or show it in a different way. The third one was to have a standard way to store semi-structured information, like the biological one.

Like ASN.1, XML is a flexible way to store information, because you are not stuck to a fixed format. The structure of any XML content can be thought as a tree, and if it is based on a defined structure, it will follow the imposed restrictions. At the beginning, the only way to define an XML format was creating a DTD (Data Type Definition). The used language for these DTDs is a subset of the used for SGML DTDs. Although it is quite expressive, it lacks some key characteristics like a way to define precise restrictions in the stored data or a powerful and extensible data type system. So, XML Schema was created in order to overcome these and other expressiveness problems.

The next important feature in XML is the concept of namespace. When people defined their format, nothing could avoid they had created XML elements with the same name. In principle, that is not a problem, but what does it happen if someone wants to reuse two or more XML formats in a new DTD or XML Schema? Sometimes it was impossible due XML element name collisions. One way to solve the problem is the use of the namespace concept (used by some object-oriented programming languages): if you want to integrate two or more XML formats, each one of them must live under a different namespace, so there is no chance for name collisions.

Due XML success, many other features and technologies have grown under the XML shades: XSL (XML Stylesheets), XPath, XInclude, SOAP, XML-RPC, SVG, WSDL, XQuery, RDF, etc... If we focus on molecular biology and bioinformatics areas, these technologies are more or less integrated and (mis)used, but definitively they are now key tools in the development of new projects.

XML in Biology and Bioinformatics

If we look at the bioinformatics area, the impact of XML technologies has been huge: NCBI Blast tools[2] are able to generate their output in XML, the DAS protocol[3] is based on three XML custom formats, etc... But, even more important, many of the biggest molecular biology data repositories are publicly available in XML: UniProt[4], born from the fusion of SWISSPROT,
There are some efforts about having a central repository for the different XML used formats in biology, and some common standard formats have been developed (BSML, AGAVE, GAME, MIAME/MAGE, IntAct, etc...) in order to ease the information exchange in areas like sequences, interactions, etc... Also, EBI is providing a translation and integration service for EMBL, Genbank and DDBJ databases, so anyone can get nucleotide sequence data in different XML formats, like BSML or AGAVE.

Anyhow, the size of the different XML data sources, both in number of entries and in raw size is increasing at the same rate, or even bigger, than the classical molecular biology data sources. We have to use technologies powerful enough so we can deal with these volumes of information. Until now, most of the ways to mine information from these data sources were related to the use of relational databases, but one of the XML features, semi-structured data representation, sometimes makes it very difficult to realize.

**XML databases: mining XML**

As we have told above, there is already a toolbox full of technologies related to XML. The most interesting ones in the data mining field are the ones related to query XML contents: XPath, XSLT and XQuery. XPath was created to locate XML fragments in a XML tree which followed some given conditions, and both XSLT and XQuery (among others) depend on XPath.

XSLT is the XSL subset which deals with the task of building an XML output tree based on the transformation of XML fragments from a single XML input tree. It is commonly used to translate XML content into HTML or XSL:FO, and the work flow in XSLT can be driven both by XPath expressions and procedural calls. XQuery is the XML query language created to query a set of XML documents, and it is more or less the SQL of XML databases. XML data model is very different from relational model, because the order of the elements in a XML tree does mind, unlike tuples in a relational table. Other difference is that a relational database has a fixed two-level structure (tables, columns inside tables), unlike XML trees. So, XQuery uses XPath as part of its FLWOR (For-Let-Where-Order-Return) expressions, so an XPath expression selects XML fragments a FLWOR expression is going to handle.

Although all these query languages can be used to mine single XML trees, what we want to do is mining a forest. So, what can it be understood as an XML database? Basically, a database which is able to store and organize XML trees, which can be queried using any of the XML query languages, returning XML fragments as the answer of the queries. Even more, a desirable feature in a XML database is the implementation of the XML:DB API, which is a independent way to send queries to a XML database, like JDBC or DBI are for relational databases. Other desirables features are XUpdate, an extension to allow the update of XML fragments belonging to stored XML documents, and support for organizing and managing XML content in collections. The number of available XML databases is increasing each day, and they can be classified in three different approaches: pure XML databases; XML databases using a classical database engine (relational, object oriented) as the underlying technology; and XML extensions for an existing database engines.

A pure XML database is defined as the one which uses its own custom storage format and query processor. A XML database based on a classical database engine stores digested XML content in a database, and it translates input queries to a set of queries to the underlying database engine. XML extensions in a database engine are composed by special data types and a set of procedures and functions which deal with these special data types, and they can be used and embedded in a native query to the database. From our point of view, some XML extensions cannot be though as XML databases in some cases, because they don't usually provide the same integration levels as the other approaches.
Our experiences with XML databases

There are some very powerful commercial XML databases and XML extensions, like the ones from Oracle and Software AG. After looking for open-source XML databases in the web, we found two promising products, still in development: eXist[9] and dbXML[10]. Both of them are coded in Java, which has its advantages and its drawbacks. On one hand, most advanced XML-related libraries are available in Java. On the other hand, Java programs are slower and use more memory than the equivalent ones coded in other languages like C or C++, and XML libraries in Java tend to be much less efficient than the corresponding C and C++ equivalents.

Both products are native XML databases which implement XML:DB API and XUpdate, and they also provides additional methods to query and access the stored information: HTTP GET, XML-RPC, servlets, monolithic standalone access, etc... They have some interesting extensions, like full text indexing (FTI) and full text search (FTS), but we have not compared these capabilities with the ones from other specialized products like glimpse or OpenFTS project. dbXML can be queried using XPath and XSLT, meanwhile eXist uses both XPath and XQuery. As any database management system, these XML database engines need some hints to improve query response times, and eXist and dbXML diverge in this point, because eXist build indexes over almost any imaginable XML component (elements, attributes, attribute values, text values) by default, meanwhile dbXML only builds them under explicit notification.

Our tests on these databases were done using IntAct as a mid-size XML data source (25~30MB), and UniProt as a big XML data source (5GB). Also, we have used XSLT and Xalan C++[11] as a measurement of the advantages we were getting using XML databases over processing raw XML content. The machine we have used for the tests is a PC with 1GB of memory, a Athlon 1GHz processor and a 540 GB array for the storage. We have also used IBM, Sun, and BEA Java Virtual Machines for our tests, and we have found Sun JVM had better performance than the others because these products are I/O intensive both in query and store operations. The programs we have built to launch queries to the XML databases have been written in Perl, and we have used both XML-RPC and HTTP-GET interfaces for that task. These programs issued both easy and complicate queries, related to returning XML fragments from the content and the number of them, generating new XML fragments for the output and doing correlated queries.

We found eXist configuration more intuitive than dbXML because we put it working spending less time than in dbXML, but it depends on the users expertise. Also, we had some problems using the dbXML XML-RPC interface because some information. The time used for the storage of a mid-size XML data source was a few minutes with both products, meanwhile the storage process of UniProt took 4 days in eXist, and it didn't finish in dbXML. At the query level, Xalan C++ needs to read each XML file it is going to query and it uses 4 times the size of the input XML in memory terms. Both eXist and dbXML memory requirements can be fixed to an upper level, but the current implementations have some problems dealing with big indexes, mainly created for big XML data sources like UniProt. These products outperform standalone tools like Xalan C++ both in terms of memory and speed, and query times were going for IntAct from 5~10 minutes in Xalan to only a few seconds in eXist and dbXML.

Queries over UniProt are beyond Xalan limits, because it has to spend too much time and memory before issuing the query itself. Also, we couldn't test UniProt queries on dbXML because we were unable to store it in this database engine. The time responses we obtained over UniProt using eXist were very disparate, from a few seconds in simple XPath expressions to unaffordable simple correlated queries. Our conclusion is that these XML databases are not ready for these information volumes yet.

In both cases, a limiting factor we have found is the number of returned results. If we are able to deal with the whole output at once, the fetch process is very fast. But, if we have to fetch the results one by one the recovery loop slows down too much, half due to the XML database and half due the Perl libraries overhead we had to use. Other limiting factor we have seen while fetching lots of results is XML output creation from the query, because it is much faster returning an existing
XML fragment from a stored XML document than creating an output based on some extracted values from the query.

**Conclusions**

XML databases are needed to properly mine our ever growing data sources. Available query languages are expressive enough, but the existing query processing engine implementations have to be improved to tackle more complicated queries and volumes. Yet they are now usable in production for mid-size volumes, many products are not still mature enough to be used over huge data repositories, like UniProt or XEMBL.

**References**

[1] W3C Consortium ([http://www.w3.org](http://www.w3.org))


[4] UniProt ([http://www.ebi.ac.uk/UniProt](http://www.ebi.ac.uk/UniProt))

[5] IntAct ([http://www.ebi.ac.uk/IntAct](http://www.ebi.ac.uk/IntAct))


[7] XEMBL. ([http://www.ebi.ac.uk/xembl](http://www.ebi.ac.uk/xembl))


