Efficient XML data management for systems biology: Problems, tools and future vision

Lena Strömbäck, David Hall, Mikael Åsberg
Department of Computer and Information Science
Linköpings Universitet
Linköping, Sweden
Email: lestr, g-davha, g-mikas@ida.liu.se

Stefan Schmidt
Institute of Computer Science
Rostock University
Rostock, Germany
e-mail: mail.stefan.schmidt@googlemail

***Abstract*—Recently, XML has become a very popular representation format for exchange of data within systems biology. This has made large amounts of XML data available on the Internet and there is a need for tools to easily and efficiently manage this data. In this paper we give an overview of existing standards and analyze the situation. We describe two tools that have been developed to provide and experiment with data management for XML standardized data. We evaluate the efficiency for each of the tools, show that they provide more efficient data management and make a proposal for a future combined solution. The paper is an extended version of [1] where we put the work in a larger context of efficient XML data management for systems biology.**

***Keywords*-XML; XQuery; hybrid XML management; graph processing; systems biology**

I. Introduction

During the past few years researchers within bioinformatics and systems biology have started to produce larger and larger quantities of experimental data. The goal in the area is to understand how proteins, genes, and other substances interact with each other within living cells. This is the key to understand the secret of life, and as such it has been set as a major goal for bioinformatics research by the Human Proteome Organization [2] and the US National Human Genome Research Institute [3]. Enhanced understanding in this area is essential for discovering new medical treatments for many diseases.

Within the area the tradition has been to publish results from experiments in databases on the web [4], [5], [6], [7], [8], making it possible for researchers to compare and reuse results from other research groups. The information content, data model and functionality are different between the databases, which makes it hard for a researcher to track the specific information he or she needs. However, most of the databases provide some kind of export facility in one or several XML-based exchange formats for protein interactions, e.g. SBML [9], PSI MI [2], and BioPAX [10].

One important discipline within systems biology where many standards exist and the emphasis of this article are biological pathways and molecular interactions. In this area the data form complex networks and it is important to enable analysis of these networks to detect key molecules for functionality or similarities between different species [11], [12].

One reason for the popularity of XML for exchange of data within bioinformatics and other areas is its flexibility. XML can be used for representing all kinds of data ranging from marked-up text, through so called semi-structured data to well structured datasets. This is a benefit especially within systems biology where datasets often contain well structured parts, such as tables or interaction graphs and unstructured or semi structured annotations or descriptions of, for instance, the experimental setup.

Supporting the flexibility that makes XML appealing is challenging from data management and technical perspectives. Two main approaches have been used, native databases designed specifically for XML and shredding XML documents to relations. More recently, hybrid implementations that combine native and shredding strategies are provided by the major relational database vendors (Oracle www.oracle.com, IBM www.ibm.com/db2 and Microsoft www.microsoft.com/sql/default.mspx). This offers new options for storage design where native and relational storage can be used side by side for different parts of the XML data. Within systems biology the situation is further complicated by the need for graph analysis functionality, which requires complex analysis capacity.

In this paper we will further analyze the situation and present two tools for management of XML data within bioinformatics. The paper starts with a brief overview of availability of standards and data within bioinformatics. Based on this overview we present the goals and motivations for the work. We then present two different tools. The first is a graph analysis extension to XQuery that enables efficient and easy to use graph functionalities. The second is a tool that enables the user to design and compare hybrid XML storage and thus further improve efficiency of the storage model. For each tool we present the main ideas behind them, exemplify the use of the tool and evaluate the performance. At the end of the paper we discuss related work and lay out the direction of a full scale future tool that could be

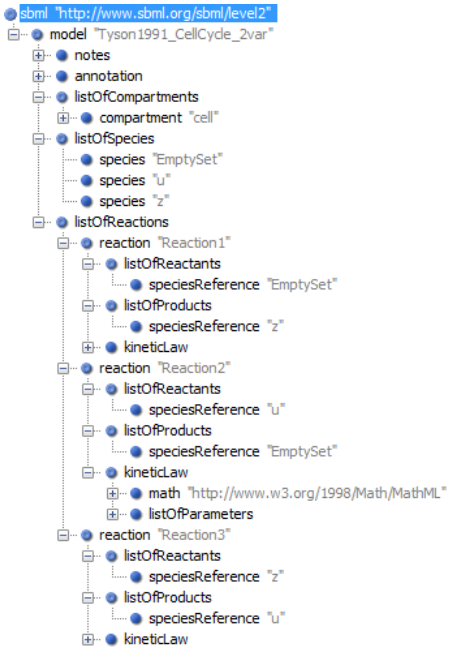

Figure 1. Excerpt of an SBML document.

supportive of data management within bioinformatics.

II. XML STANDARDS FOR BIOINFORMATICS

In a study [13] in 2006 we found 85 XML-based formats in systems biology. These include formats for exchanging information about substances, interactions, pathways, compartments, organisms and experiments.

With the large interest in using XML-based formats for exchange and export of data within systems biology the need for standardization has become obvious. Some formats have become de facto standards or at least widely accepted formats (for example Seq-entry and INSD-Seq [14]), while other are intended as candidates for future standards. Table I is based on the evaluation in [13] and lists examples of commonly used XML-based bioinformatics formats. The version given for each format is the latest version available. However, in many cases, actual use and support in software and databases may be predominant for earlier versions.

Of the formats listed there are formats for representing molecular interactions or pathways, describing structure of substances (DNA, RNA, proteins or other chemical compounds). The formats for interactions and pathways could be either aimed at describing simulation properties (e.g. SBML[9] or CellML[15]) or experimental results (e.g. PSI MI [2]). The formats for structure of substances are often export formats for certain databases.

Figure 1 shows the basic structure of an SBML document. It contains lists of compartments, species and reactions that are part of the simulation model. Internal references are used to connect species to reactions, thereby avoiding redundancy

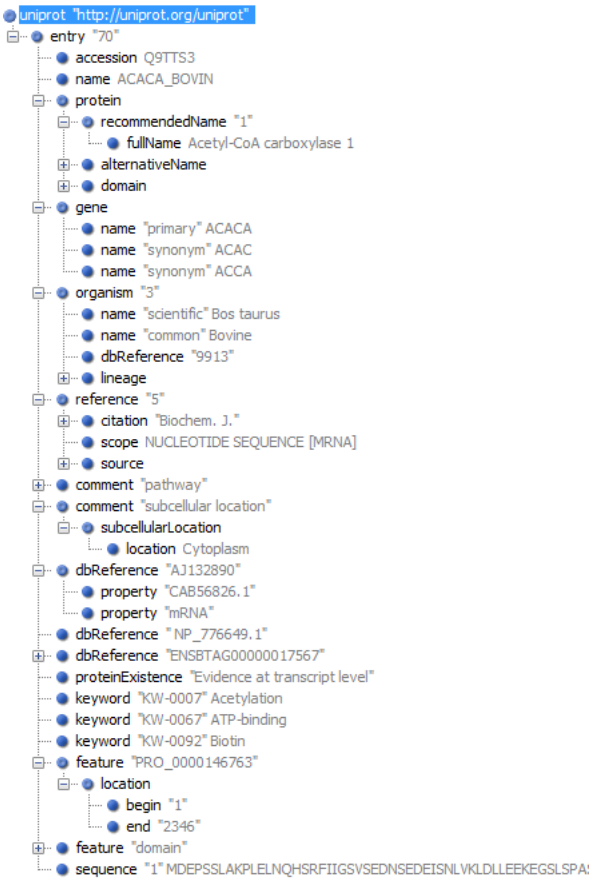

Figure 2. Excerpt of an UniProtKB document.

of species information. Figure 2 shows the basic structure of a UniProtKB document. It contains a list of entries which in turn contains elements with name information for proteins, genes, and organisms, database and literature references, and additional information (annotations). The entries also contain (not depicted in the figure) sequence (for the protein) and keywords (using controlled vocabularies). Here emphasis is on citations, names and taxonomy.

During the latest years efforts to standardize XML-based formats in the bioinformatics area has been intensified. Organizations such as the Proteomics Standards Initiative (PSI) and Institute for Systems Biology (ISB) have developed standards within different fields of bioinformatics. Adoption of standard formats is delayed due to implementation in tools and database APIs/data dumps.

Sometimes several standard formats for the same type of information are developed. In the mass spectrometry area standardization attempts led to mzData[20] (PSI) and mzXML[19] (ISB), both of which are supported in different tools. The two organizations has been working on a joint standard, mzML[23], that combines aspects of mzData and mzXML and version 1.0.0 was released in June 2008 [24]. Another release, 1.1.0, was made in 2009 [24] for fixing

Name	Ver.	Year	Defined by	Purpose	Data
SBML [9]	2.4	2008	Systems Biology Workbench development group.	A computer-readable format for representing models of biochemical reaction networks.	Data available from many databases, for instance, KEGG, www.genome.jp/kegg/ and Reactome, www.reactome.org.
PSI MI [2]	2.53	2006	HUPO Proteomics Standards Initiative.	A standard for data representation for protein-protein interaction to facilitate data comparison, exchange and verification.	Datasets available from many sources, for instace IntAct www.ebi.ac.uk/intact/, and DIP http://dip.doe-mbi.ucla.edu/.
BioPAX [10]	L. 3 (0.92)	2008	The BioPAX group.	A collaborative effort to create a data exchange format for biological pathway data.	Datasets available from Reactome www.reactome.org
CellML [15]	1.1	2002	University of Auckland and Physiome Sciences, Inc.	Support the definition of models of cellular and subcellular processes.	CellML Model Repository (240 models) www.cellml.org.
CML [16]	2.2	2003	Peter Murray-Rust, Henry S. Rzepa.	Interchange of chemical information over the Internet and other networks.	BioCYC www.biocyc.org.
EMBL-xml [14]	1.1	2007	European Bioinformatics Institute.	More stability and fine-grained modelling of nucleotide sequence information.	EMBL www.ebi.ac.uk/embl .
UniProt KB [17]	1.28	2009	UniProt Consortium	XML Schema for UniProtKB	Swiss-Prot and TrEMBL www.uniprot.org
INSD-seq [14]	1.5	2009	International Nucleotide Sequence Database Collaboration	The purpose of INSDSeq is to provide a near-uniform representation for sequence records.	EMBL www.ebi.ac.uk/embl and GenBank www.ncbi.nlm.nih.gov/Genbank .
Seqentry	n/a	n/a	National Center for Bio-technology Information.	NCBI uses ASN.1 for the storage and retrieval of data such as nucleotide and protein sequences. Data encoded in ASN.1 can be transferred to XML.	Entrez www.ncbi.nlm.nih.gov/Entrez.
MAGE-ML [18]	1.1	2003	Microarray Gene Expression Data.	To facilitate the exchange of microarray information between different data systems.	ArrayExpress www.ebi.ac.uk/arrayexpress.
Mz XML [19]	2.1	2004	Institute for Systems Biology	The common file format for mass spectrometry data.	PeptideAtlas www.peptideatlas.org, Sashimi sashimi.sourceforge.net, Open Proteomics Database http://apropos.icmb.utexas.edu/OPD.
Mzdata [20]	1.05	2005	HUPO Proteomics Standards Initiative.	To capture peak list information. Its aim is to unite the large number of current formats into one.	
AGML [21]	2.0	2004	Medical University of South Carolina.	To model the concept of annotated gel (AG) for delivery and management of 2D Gel electrophoresis results.	AGML Central http://bioinformatics.musc.edu/agml2/web/pages/
ProtXML [22]	n/a	n/a	Institute for Systems Biology	A format for storage, exchange, and processing of protein identifications created from ms/ms-derived peptide sequence data.	
PepXML [22]	n/a	n/a	Institute for Systems Biology	A format for storage, exchange, and processing of peptide sequences derrived from ms/ms scans.	

Table I
AVAILABLE STANDARDS, CREATORS AND AVAILABILITY.

shortcomings that had hindered the implementation of the standard.

Due to the nature of the field, the community has realized that there will exist a plethora of competing formats and a number of specifications on minimum information required within different fields has been devised, e.g. MIAME [25] (Minimum Information About a Micro-array Experiment) for micro-array data, MIAPE [26] (Minimum Information About a Proteomics Experiment) for proteomics data, MIGS (Minimum Information about a Genomic Sequence) for genomics data and MIRIAM [27] (Minimum Information Requested In the Annotation of bio-chemical Models) for system biology models. They often require use of controlled vocabularies. Other requirements could be literature source references or information about from which organism data was collected.

Given the situation today there will continue to exist a large number of XML-based bioinformatics formats in the future. In addition, several formats for storing the same type of data and different versions of the same formats will be used simultaneously.

III. Goals and Motivation

There are a number of tools available for management and processing of XML data. In addition, there are also a number of dedicated tools available for handling data in the special designed standards for bioinformatics. Examples of such tools are simulation tools and visualization tools, e.g. Cytoscape [28] and GNU MCSim [29], that offers import and export in various predefined XML formats. The focus for this paper are applications where there is a need for complex information retrieval, i.e. where the user needs to combine the data to gain new information. In addition we assume that the user is interested in data from several different databases, exported in several of the standards described above. The most natural way to provide this is to store the data within a database and query it.

However, for bioinformatics this puts hard requirements on the data management solution. On the one hand the data that we want to use is downloaded from the web in one of the many XML standards that are available within the area. This means that we need solutions where it is fast to import the data into the database and where little effort needs to be spent on designing the storage solution. On the other hand many of the tasks that we are interested in, for instance, combining and comparing information from several datasets or graph analysis, requires quite complex queries on the dataset. Previous studies have shown that native XML solutions do not perform well when the query complexity grows [30].

The main goal for this work is to explore ways for more efficient data processing within bioinformatics. Our primary goal is query efficiency, easy import and reuse of data in any of the bioinformatics standardized formats is also an important issue. We will address these issues in two important tools. The first addresses graph processing capabilities, and suggests a standard independent extension to XQuery that provides easy to use and efficient graph processing of XML data. The second tool provides an easy way of exploring more efficient storage models for the data. The motivation for this is that a pure native XML storage yields too inefficient querying for the data while a relational storage provides more efficient querying. The goal for our second tool is to provide easy creation and import of data to a hybrid XML storage model.

For our first tool we address cases where the databases provide data export in one or several XML exchange formats for protein interactions, e.g. SBML [9], PSI MI [2], and CellML [15]. These datasets available in XML provide descriptions of interaction networks or graphs [31]. Therefore, it would be beneficial for the user to enable querying and analysis based on the XML format, i.e. to be able to query the data using XQuery. Our goal is to find a solution that can preserve the full functionality of XQuery and in parallel provide an efficient handle for graph analysis. As many standardized data representation formats exist for the area it is important to find a general solution where all XML-based data formats can be used.

To reach our goal we need a way to enable graph processing directly in the XML environment. One solution would be to implement graph queries directly in XQuery [32]. However, our initial studies of this [30] were disappointing. The queries get complex and inefficient to compute, which make it impractical for biologists that may have limited knowledge in programming. Therefore, we want to provide graph functionality within XQuery by extending the language. As we do not want to change the core functionality of XQuery we want to add graph functionality through addition of built-in functions which make them available directly from XQuery. The first tool we describe presents an extension to XQuery which allows extended analysis on graphs. The main application for the work is biological interactions, but the extension is generic and capable of handling graphs represented as XML also for other applications. In section IV we give a general description of the chosen solution, our implementation and an evaluation of the tool.

For our second tool we will investigate how well hybrid databases as provided by modern relational database managers [33], [34], [35], [36], [37] can match the requirements of bioinformatics. With hybrid solutions the user can choose to use either native or relational storage for his data. It is also possible to combine the solutions and store parts of a document as XML and other parts of it as relations. Consequently the user can work with XQuery for parts of the data and SQL for other parts. He can also choose to retrieve results from queries in the format the data is stored or to convert it to the format he prefers.

We aim at combining the benefit of native XML databases,

which is an easy to use solution, with the efficiency provided by relational databases. The main drawback with this solution is the cost for designing the hybrid storage, i.e. to decide which parts of the XML code that should be stored as relations and which parts should preferably be preserved in its original XML structure. The work by Moro et al. [38] addresses this problem by providing guidelines for when parts of the XML structure should be translated or not. They also provide a tool where the user can design hybrid XML storage.

In our case the problem is a bit different. Our starting point is the already available standards within bioinformatics which provides us with the XML data model and in most cases also the XML schema defining this model. Therefore we want a solution where we can use this information as a basis for the hybrid storage. We have chosen to adapt the solution by Amer-Yahia et al. [39]. They present a system, ShreX, which automatically can map an XML schema to relational tables and import the resulting schema and dataset into a database.

In section V we present a tool HShreX which extends the original system to hybrid databases. We have also extended it with some further functionality to vary the mapping into relations. We present the main architecture behind the system, an illustrating example and an evaluation illustrating the benefit of using the system.

Together the two tools provide a powerful workbench for analyzing bioinformatics data. In practice they can be used as two separate tools. However, at the end of the article we discuss how they can be combined into one single environment.

IV. An XQuery extension for graphs

Our first tool provides efficient and easy to use graph analysis functionality for XQuery and was previously presented in [1]. In particular, we want to find a solution that is applicable to all standards within the area of molecular interactions and pathway data. We also want to find a solution where existing efficient graph algorithms can be reused within the environment. We choose to do this by extending XQuery with specialized functions for graph analysis. The goal for our work is to find a solution that adds new graph functionality that blends well into existing XQuery functionality and does not introduce new features to XQuery itself. At the same time we want the data, algorithms and results to be accessible from XQuery. As the solution should be independent of XML format, graphs should be freely modeled by the XQuery/XPath expressions and changes to the original XML data should not be necessary.

A. Architecture

One of the challenges is to provide a solution that is independent of XML format, as the external functions must know which parts of the XML file constitutes the graph. To deal with this we define a common graph model that the supplied functions are operating on. In addition to this our solution must contain handles for connecting the original XML representation to the general graph model. The selected graph model enables labeled directed graphs. It has been chosen so that it captures the most common properties for biological pathways.

Definition A *graph* is defined as a quadruple $G := (V, E, FV, FE)$ with:

- V, the set of vertices.
- E, the set of edges. An edge describes the relation between its two endpoints - the two connected vertices $((v, w) \in E; v, w \in V)$. Furthermore, parallel edges are not allowed, so no two distinct edges may have the same endpoints. Edges with identical endpoints, so called loops, are not allowed.
- I, is a set of identifiers used to denote properties, e.g. name or weight of edges.
- L, is a set of labels, i.e. the values of the properties, usually a substance name or the weight of an edge.
- $FV : V \times I \rightarrow L$, is a set of mappings associating labels for each vertex and a given identifier.
- $FE : E \times I \rightarrow L$, a set of mappings associating labels for each edge and a given identifier.

Hence, labels can be attached to vertices and edges to provide additional information, for instance, enabling graph algorithms to incorporate weights. Graphs may be directed or undirected. The focus for this work have been to investigate connectivity. Therefore we made the restriction to not allow parallel and looping edges since they not give extended information to the graph queries of our interest. The resulting model can capture all information inherent in the protein interaction and pathway standard descriptions presented in the previous sections. Hyperedges, i.e. edges connecting several nodes, can be represented by a set of edges in our model while identifiers and labels can be used to represent information not directly captured by the vertices and edges.

The final step needed for our solution is a way to map the data between the original XML format and our graph model. To achieve the required functionality we need handles to load, get and execute graph analysis on our graph model. The load functionality constitute mapping from the original XML data to the graph model. The mapping between the original XML format and graph model is done by specifying XPath expressions. These define which parts in the original format that corresponds to an edge, vertex or a label. Executing these expressions will result in pointers to XML items that are used to build the desired graph. The remaining functionality is used to import graph data back to the XQuery environment. This can be done either by fetching all or part of the graph (*get*) or by retrieving a graph as part of

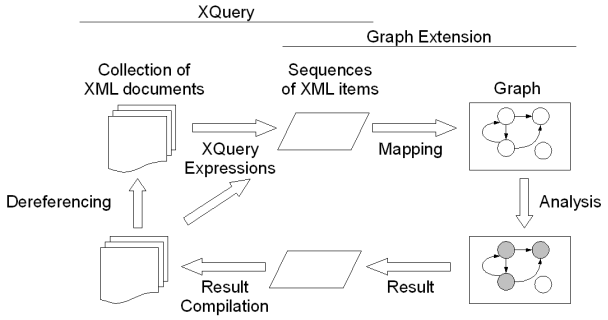

Figure 3. Architecture for the extension

executing an supplied algorithm on the graph (*execute*). As the data returned from the graph package normally constitute only part of the data (the graph information or in most cases a subgraph) from the original format we decided to use GraphML for its representation instead of the original graph format. This gives a clear distinction between returned results and the original data file.

The resulting architecture for our extension is depicted in Figure 3. As desired XQuery is used to address the data subject to analysis; the graph extension uses the graph model to process the analysis. The user can create graphs that are represented internally in the graph extension. Other XQuery expressions allow the user to execute graph queries by utilizing functionality available in the external graph package. The result from these queries is received by the user as an XML representation of the graph. If the user wants he or she can then link these results to the original XML file by a referencing mechanism.

B. Implementation

A prototype implementation in Java was built on the native XML database eXist, its XQuery processors and the JUNG graph framework in order to investigate usability, performance and overall strength and weaknesses in practice. We chose eXist version 1.0.2 (*exist.sourceforge.net*) since it is an open source native XML database with an extensible XQuery implementation in Java. The JUNG graph framework version 1.7.6 (*jung.sourceforge.net*) has been chosen to implement the graph model. JUNG is like eXist written in Java and supports directed and undirected graphs, hypergraphs, bipartite graphs and labels for vertices and edges; therefore it easily satisfies the proposed graph model.

To enable an environment where it was easy to experiment with different functionality and several graphs in parallel we introduced a set of functions. First we added two functions to create and delete graphs explicitly (*createGraphs* and *releaseGraphs*). Secondly the load functionality is implemented by a set of easy to use functions to define the properties like vertex, edge and their labels (*loadVertices*, *loadLabeledVertices*, *loadEdges* and *loadLabeledEdges*). Finally we implemented two functions for retrieveing the graph data or results of an algorithm (*getGraphs* and *execute*). In this implementation, especially the load functions rely on related sequences. Therefore, the document order, i.e. the order in which XML nodes appear in the XML serialization of a document, is the default order if no ordering is defined.

The required reference mechanism that are used to link from graph data back into the original XML document are implemented according to Chamberlin et al. [40] by two functions, *fn:ref* and *fn:deref*. Obviously, the functions can work correctly only if the node IDs are stable, regardless of changes to the document if updates are allowed. Updates to XML documents are not considered in the graph extension.

C. Example

We illustrate how the extension works by showing an example using the SBML [9] data. An example data model in SBML is given in figure 4. The example in figure 5 illustrates the usage of the functions in the implementation.

- The root element of the XML data is bound to `$doc`. (Expression 2)
- One directed graph is created and bound to `$graph`. (Expression 3)
- Two variables bound the IDs of interesting molecules. (Expression 4 and 5)
- Then all IDs of the species element are selected by an XPath expression, loaded as vertices into the referenced graph. (Expression 6)
- A FOR-expression is used to access and load each reaction into the graph. The URI of each reaction serves as edge ID and is retrieved with `xqueryp:ref($reaction)`. The expression `$reaction/s:listOfReactants/s:speciesReference/@species` relates to the vertices defined in the previous step. (Expression 7)
- After defining the graph's properties the shortest path is calculated, the returned XML node is in GraphML format, the edge ID holds the reference to the original SBML data. The edge IDs are selected by `//edge/@id` and then resolved by `xqueryp:deref` which are the reactions in SBML representing the shortest path between the specified substances. (Expression 8)
- Finally, the graph is deleted. (Expression 9)

D. Evaluation

To evaluate our approach we have performed a series of experiments. We were in particular interested in three properties; the overall performance for graph analysis for biological pathways data; comparing this with using plain XQuery; and finally an analysis of the performance of loading graphs. All experiments were done on a notebook

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model id="Tyson1991CellModel_6" name="Tyson1991_CellCycle_6var">
    + <annotation>
    <listOfSpecies>
      + <species id="C2" name="cdc2k" compartment="cell">
      + <species id="M" name="p-cyclin_cdc2" compartment="cell">
      + <species id="YP" name="p-cyclin" compartment="cell">
      ... more species
    </listOfSpecies>
    <listOfReactions>
      <reaction id="Reaction1" name="cyclin_cdc2k dissociation">
        <annotation>
           <rdf:li rdf:resource="http://www.reactome.org/#REACT_6308"/>
           <rdf:li rdf:resource="http://www.geneontology.org/#GO:0000079"/>
        </annotation>
        <listOfReactants> <speciesReference species="M"/> </listOfReactants>
        <listOfProducts>  <speciesReference species="C2"/>
                          <speciesReference species="YP"/>  </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply> <times/> <ci> k6 </ci> <ci> M </ci> </apply></math>
          <listOfParameters> <parameter id="k6" value="1"/> </listOfParameters>
        </kineticLaw>
      </reaction>
      + <reaction id="Reaction2" name="cdc2k phosphorylation">
      ... more reactions
    </listOfReactions>
  </model>
</sbml>
```

Figure 4. SBML representation of the Tyson Cell model as it is represented in the Biomodels (www.biomodels.net) database. The example has been abbreviated and simplified to improve readability.

```
1: declare namespace s = "http://www.sbml.org/sbml/level2";
2: declare variable $doc {doc("reactome/homo_sapiens.xml")};
3: declare variable $graph {graph:createGraphs("org.exist.xquery.modules.graph.JUNGGraphImpl",true())};
4: declare variable $source {"R_111584_xanthosine_5_monophosphate"};
5: declare variable $target {"R_29398_Pyruvate"};
6: graph:loadVertices($doc//s:listOfSpecies/s:species/@id, $graph),
7: for $reaction in $doc//s:listOfReactions/s:reaction
     return   graph:loadHyperEdge(xqueryp:ref($reaction),
                        $reaction/s:listOfReactants/s:speciesReference/@species,
                        $reaction/s:listOfProducts/s:speciesReference/@species,$graph),
8: xqueryp:deref(graph:execute("dijkstraShortestPath",($source, $target), false(),
                                                  true(), $graph)//edge/@id),
9: graph:releaseGraphs($graph)
```

Figure 5. Example on how to use extended graph functionality in XQuery.

with Windows XP Professional, a 1.6GHz Pentium Mobile and 1GB main memory.

The first experiments exemplify how well the graph extension scales for graphs with a few thousand vertices. In the experiments sample test series were successfully and efficiently executed on real application data from the Reactome [6] and KEGG [7] databases. The Reactome data set is stored in one SBML document of 1.2MB comprising 3054 substances and 1917 reactions which were resolved into 4832 edges. The KEGG data set is stored in 92 SBML files with a total of 1,2MB comprising 1652 substances and 1122 reactions which were resolved into 1296 edges.

Figure 6 shows the results of 100 passes of the Dijkstra shortest path on pairs of substances from the Reactome data set where the path length was 3, 5, 10 and 25, i.e. the query presented in section 4 with selected start and end nodes. The reason for running each query 100 times is to reduce the impact of other processes, such as Java garbage collection

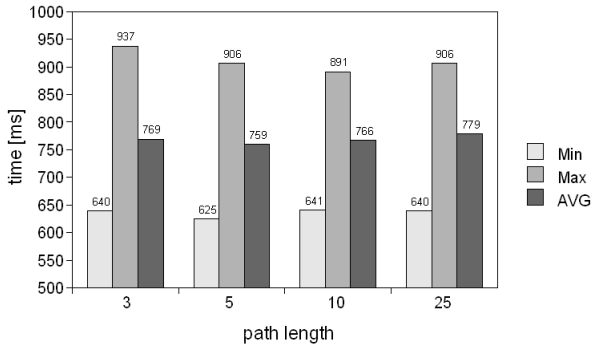

Figure 6. Performance on the Reactome dataset.

that may affect the result. Analogously, figure 7 shows the results of 100 passes of the Dijkstra shortest path on pairs of substances from the KEGG human data subset where the path length was 3, 5, 10 and 14 on the same query.

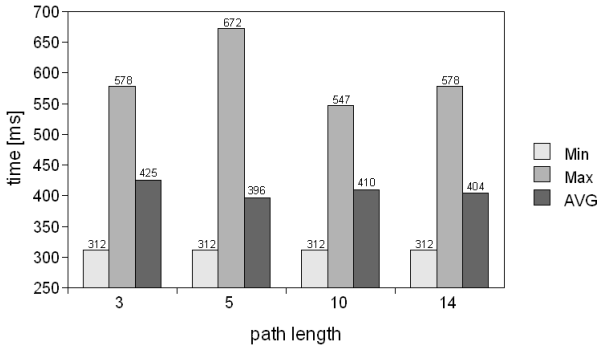

Figure 7. Performance on the KEGG dataset.

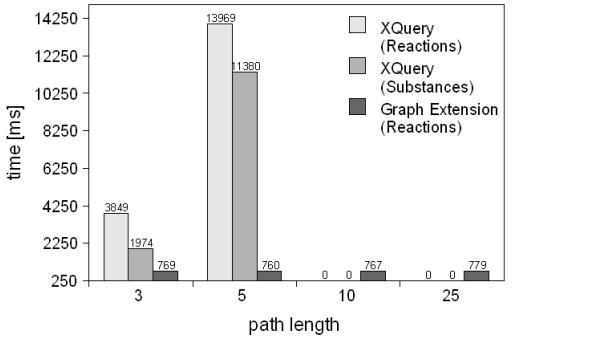

Figure 8. Performance comparison of our extension with plain XQuery.

The query times include creating, populating and deleting the graph, the execution of the algorithm and the XML representation of the paths based on the original SBML data. For these tests caching of shortest path results within JUNG was deactivated. This shows that our graph extension works well for the tested data.

Secondly, we wanted to compare our results with using plain XQuery. An adapted depth-first search algorithm in XQuery was implemented for comparison. All shortest paths are searched within the given depth. The implementation shown in figure 9 uses two functions. The *local:findPath* creates the spanning search tree recursively, found paths are marked with a `<found/>` element. The function expects three parameters, the cut-off depth, the start and the end vertex of the path. The second function, *local:getParent* is used to traverse the found path up and collect the parents. This implementation demonstrates how complicated and inefficient it is to build an algorithm based on sequences of items and temporary XML fragments.

Figure 8 illustrates the results together with the query times on the same data set with the graph extension. The query for XQuery (Reactions) takes longer because it presents the path with all reaction elements, whereas the query for XQuery (Substances) presents a condensed version as a list of elements with the reaction IDs and only the substance IDs found by the path search as attributes. It must be noted, that the comparison with the graph extension is not completely fair. The Dijkstra's shortest path algorithms used within the graph extension only returns the single shortest path whereas the XQuery implementation completely explores all shortest path within the specified depth. The query times for a path length of 5 are still acceptable if the data volume is disregarded, but the query does not finish on the same data set within an hour with 10 as cut-off depth. One reason is certainly that with every step the search tree grows tremendously by the fact that the query does not sufficiently detect cycles.

Finally, we wanted to analyze the performance of loading graphs into the graph module to understand how much of the total execution time that were spent on creating the graphs. For these experiments we used the Reactome dataset. As for the total execution time we compared our loading performance with an XQuery expression retrieving the same information from the data file. From this experiment we can conclude that loading the data is very fast. In fact, most time is spent on retrieving the data from the XML file. The execution of Dijkstra's shortest path is even faster and because of this the differences between different path lengths are marginal. The divergence between different path lengths is roughly between 5ms and 20ms on average. In comparison, the difference between minimum and maximum performance are significant, but still under half a second.

A final remark is that the presented results refer to small amounts of data in particular in regards to data volumes databases are built for and for our tests in memory processing could be used. Query times increase dramatically if the whole KEGG data set is utilized including different species (132MB, 12122 files), because data is stored highly redundantly. In that case most time is spent on the XQuery expressions to retrieve the sequences of items to map onto vertices and edges. The data volume to process the analyze is reduced because of the integrated duplicate elimination. This behaviour is beneficiary for scenarios, where we can expect that the user loads the data into the database and then runs a series of analysis on the dataset.

E. Discussion

The general architecture for our extension proposes that XPath expressions are used to declare the XML data and the graph model. In our implementation we choose to implement this as a set of load functions which makes use of side effects. This is controversial since XQuery is a side-effect free query language. The main problem with introducing side effects is that query optimization is hindered. The order of execution of the graph functions matters putting restrictions on optimization. However, the evaluation for all other XQuery expression can still be optimized without further limitations. Our view is supported by Chamberlin et al. [40] who state that global optimization is difficult in a mixed language environment.

```
declare function local:findPath($start as xs:string,
                                $end as xs:string, $n as xs:integer) {
  for $species in
    $doc//s:reaction[s:listOfReactants/s:speciesReference/@species = $start]
                               /s:listOfProducts/s:speciesReference/@species
  return<item reaction="{$species/../../../@id}"species="{$species}">  {
    if($species = $end) then <found/> else
      if($species = $start or $n = 1) then () (: loop or max :)
       else local:findPath($species, $end, $n - 1)} </item>};

declare function local:getParent($itema as node()?) {
  if($itema[@species]) then (local:getParent($itema/..), <node>
     {$itema/@species} {$itema/@reaction} </node>)         else ()};

<paths>{for $found in local:findPath($source, $target, $maxLength)//found/..
   return <path> <node species="{$source}"/{local:getParent($found)} </path>}
</paths>
```

Figure 9. XQuery version of findpaths used for comparison with our extension.

To avoid side effects one solution would be to implement the graph as an extended index to the database. This is possible in for instance eXist 1.1. With this solution the functionality for creating the graph would be analogous to creating an index and performed when new files are loaded into the database. We did not choose this solution, as it would give us less freedom to experiment with different graph realizations of a dataset which was a goal for this version of the tool. An alternate solution would be to make use of further developments of XQuery like XQueryP [40], [41] and XQuery! [42]. XQuery! proposes to extend XQuery with a set of side-effecting operations, especially handy for XML updates [42]. Therefore it introduces a new operator that allows applying a sequential mode to an XQuery fragment. XQueryP introduces even more features to extend XQuery for application logic [40], [41]. Another approach to separate the concerns of assembling the graph and querying it using XQuery could be to annotate the XML schema of the source format defining the desired structure and elements of the graph.

V. User designed hybrid storage

Our second tool [43] investigates how well hybrid databases as provided by modern relational database managers [33], [34], [35], [36], [37] can match the requirements of bioinformatics. With hybrid solutions it is also possible to combine the solutions and store parts of a document as XML and other parts of it as relations. Our aim is to combine benefit of native XML databases, which is an easy to use solution, with the efficiency provided by relational databases and minimizing the cost for designing the hybrid storage. Our starting point is the already available XML schema defining the model for the chosen standard. Our tool allows the user to take benefit from and experiment with hybrid XML storage.

A. Architecture

HShreX [43] is a tool that automatically can, from an XML Schema, create a native, relational, or hybrid data model. HShreX builds upon a previous tool ShreX [39] developed for shredding XML data into pure relational storage.

The starting point for HShreX is the XML Schema. When the user loads a schema in HShreX, it first creates an internal schema model, which is a tree-like structure specifying the details of the schema. Once the schema model has been created it is traversed in order to determine mapping information (e.g., the simple XML element *name* should be mapped to a field called *name* in the table *xyz*), from which a relational model (that can be pure relational, native, or hybrid) is created. The exact characteristics of the resulting model depend on a default set of shredding rules which can be influenced by using annotations in the XML schema. The user can now inspect the relational model and redesign it using schema annotations until a desired one has been created. When a satisfactory model has been created, it can be loaded onto a live database. This is done by a relation generator which generates scripts adapted to the chosen relational database manager. After this step, data can be loaded by opening XML files. A data converter looks up mapping information and generates a script with tuple insertion statements and runs it when all the data has been read. The architecture is visualized in figure 10. The default shredding rules include the following behavior:

- Complex elements are shredded into tables. All tables will get a primary key field named *shrex_id*. If the complex element is not a root element it will also get a foreign key field named *shrex_pid* that point to its parent. This preserves the tree structure in the original XML data. If the complex element can have simple content (i.e., text content), a special field is created in the table to hold any such content.
- Simple elements are shredded into columns in their parent table if they can occur at most once under their parent. If a simple element can occur more than once under its parent it will be outlined to a separate table.
- Attributes are shredded into columns in their parent table.

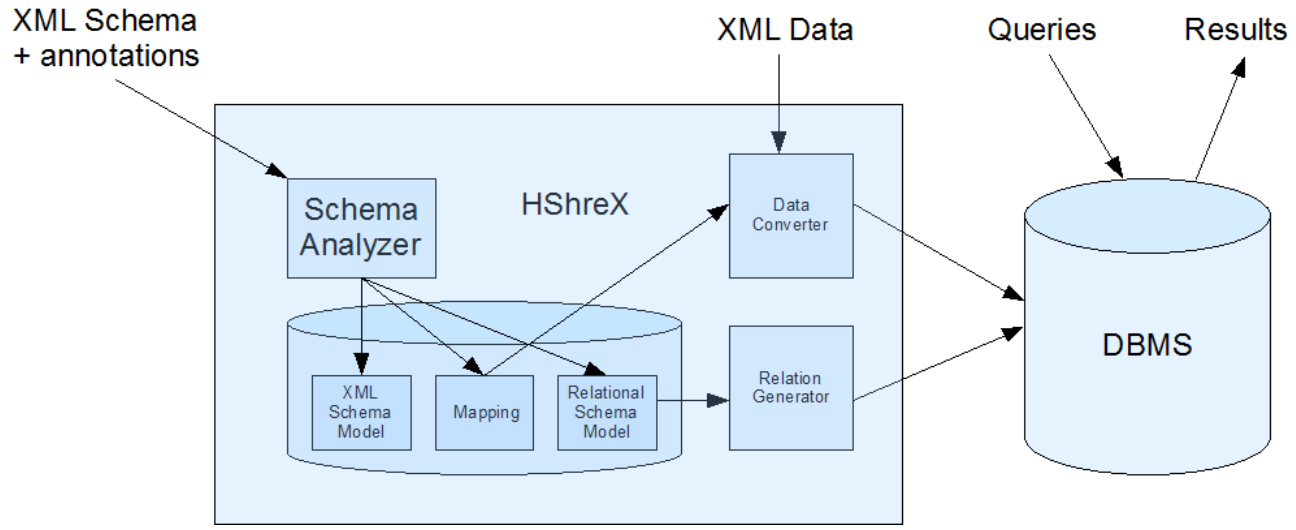

Figure 10. HShreX architecture

```
<xs:element name="minisbml">
  <xs:complexType>
     <xs:sequence>
        <xs:element name="author" type="PersonType"/>
        <xs:element name="molecule" type="Moleculetype"
                    minOccurs="1" maxOccurs="unbounded"/>
        <xs:element name="reaction" type="Reactiontype"
                    minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="Moleculetype">
  <xs:attribute name="name" type="xs:string"
                use="required"/>
</xs:complexType>

<xs:complexType name="Reactiontype">
  <xs:sequence>
     <xs:element name="reactant" type="Moleculetype"
                 minOccurs="0" maxOccurs="unbounded"/>
     <xs:element name="product" type="Moleculetype"
                 minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string"
                use="required"/>
</xs:complexType>

<xs:complexType name="PersonType">
  <xs:sequence>
     <xs:element name="name" type="xs:string"/>
     <xs:element name="affiliation" type="xs:string"
                 minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Figure 11. Sample XML schema

```
<minisbml>
  <author>
     <name>Lena Strömbäck</name>
     <affiliation>IDA</affiliation>
  </author>
  <molecule name=M1/>
  <molecule name=M2/>
  <molecule name=M3/>
  <reaction name=R1>
     <reactant name=M1/>
     <reactant name=M2/>
     <product name=M3/>
  </reaction>
  <reaction name=R1>
     <reactant name=M3/>
     <product name=M2/>
  </reaction>
</minisbml>
```

Figure 12. Sample XML document

In figure 11 and 12 a sample XML schema is shown with an accompanying XML document, respectively. The schema lacks annotations so it will be processed by HShreX using the default shredding rules, yielding the relational model found in figure 13.

Shredding a schema using just the default rules will in most cases create a pure relational model. The only exception is elements that have the type *anyType*, i.e. elements that have no XML structure definition in the schema, which are mapped to XML. In many cases this will cause a large number of tables to be created, which can be a problem because it makes the model hard to understand and overview. Another problem with models that suffer from an explosion of tables is that semantically related data run a risk of being separated into different tables. Combined this can make the task of writing queries complex and performance can suffer. Therefore, HShreX allows the default shredding rules to be influenced via annotations. A number of annotations are supported and they are used on the schema to change the default shredding rules. A document describing all annotations supported by HShreX can be found on http://hshrex.sourceforge.net/. Here follows a few of the more important annotations:

- *maptoxml* – makes this part of the XML tree to be stored natively. The annotation can be used on both complex and simple elements.
- *ignore* – this part of the XML tree will be ignored, i.e.

minisbml:

shrex_id
1

minisbml_molecule:

shrex_id	shrex_pid	name
1	1	M1
2	1	M2
3	1	M3

minisbml_reaction:

shrex_id	shrex_pid	name
1	1	R1
2	1	R2

minisbml_reaction_reactant:

shrex_id	shrex_pid	name
1	1	M1
2	1	M2
3	2	M3

minisbml_reaction_product:

shrex_id	shrex_pid	name
1	1	M3
2	2	M2

minisbml_author:

shrex_id	shrex_pid	name
1	1	Lena Strömbäck

minisbml_affiliation:

shrex_id	shrex_pid	affiliation
1	1	IDA

Figure 13. Generated relational tables.

it will not be represented in the resulting data model.

- *outline* – used on simple elements (or attributes) where it is desired that they be stored in a separate table.
- *withparenttable* – used to merge a child with its parent in order to reduce the number of tables in model. This annotation can be used only for children with a single occurrence in the parent.
- *tablename* – can be used to simply rename a table but a more powerful use is to merge two tables that do not have a parent/child relationship (in those cases the annotation described above, *withparenttable*, is used).

Maptoxml, *ignore* and *withparenttable* are new annotations for HShreX whereas the other annotations work as in the previous ShreX tool. In addition the system allows varying the underlying basic shredding principle. This will not be further discussed here.

B. Implementation

HShreX is developed in Eclipse and written in Java version 1.6. The main development platform is Windows Vista, but HShreX also runs on Windows XP and Linux. A large part of what HShreX does is processing XML and for that Xerces2-J version 2.9.1 is used. HShreX knows how to communicate with IBM DB2 9.5 fixpack 1 or later and Microsoft SQL Server 2008 but in order to do that HShreX needs drivers supplied by the vendors. For Microsoft SQL Server sqljdbc4.jar is used and for IBM DB2 the dependency is db2jcc4.jar. A large set of unit tests is part of the HShreX sourcebase and to run them one needs JUnit version 4.3 or later. HShrex together with documentation can be obtained in binary and source form at http://hshrex.sourceforge.net/.

```
<xs:element name="minisbml">
  <xs:complexType>
     <xs:sequence>
        <xs:element name="author" type="PersonType"
                    shrex:maptoxml="true"/>
        <xs:element name="molecule" type="Moleculetype"
                    minOccurs="1" maxOccurs="unbounded"
                    shrex:ignore="true"/>
        <xs:element name="reaction" type="Reactiontype"
                    minOccurs="0" maxOccurs="unbounded"
                    shrex:maptoxml="true"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>
```

minisbml:

shrex_id	author
1	<author> <name>Lena Strömbäck</name> <affiliation>IDA</affiliation> </author>

minisbml_reaction:

shrex_id	shrex_pid	xml
1	2	<reaction name="R1"> <reactant name="M1"/> <reactant name="M2"/> <product name="M3"/> </reaction> <reaction name="R1"> <reactant name="M3"/> <product name="M2"/> </reaction>
2	1	<reaction name="R1"> <reactant name="M3"/> <product name="M2"/> </reaction>

Figure 14. Hybrid mappings with maptoxml and ignore.

C. Example

To illustrate how HShreX can be used we give two examples of using the annotations to design the shredding. The first example in figure 14 illustrates how the hybrid mapping can be used. In this example the aim is to map the information about authors and reactions to XML and remove information about molecules (assuming these are not interesting for the current information need). This kind of mapping is common in bioinformatics since most of the bioinformatics standards are very rich and define a large amount of elements for representing various portions of information. In many real cases parts of this information are not interesting for the end user or many of those elements is not even used by the source exporting the data.

To achieve this shredding we have added *maptoxml* anno-

```
<xs:element name="minisbml">
  <xs:complexType>
     <xs:sequence>
        <xs:element name="author" type="PersonType"
                    shrex:withparenttable="true"/>
        ...rest of definition in figure 2...
     </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="Reactiontype">
  <xs:sequence>
     <xs:element name="reactant" type="Moleculetype"
                 minOccurs="0" maxOccurs="unbounded"
                 shrex:tablename="participant"/>
     <xs:element name="product" type="Moleculetype"
                 minOccurs="0" maxOccurs="unbounded"
                 shrex:tablename="participant"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
```

minisbml:

shrex_id	name
1	Lena Strömbäck

minisbml_molecule:

shrex_id	shrex_pid	name
Content as in Figure 4		

minisbml_reaction:

shrex_id	shrex_pid	name
Content as in Figure 4		

participant:

shrex_id	shrex_pid	name
1	1	M1
2	1	M2
3	1	M3
4	2	M4
5	2	M5

minisbml_author_affiliation:

shrex_id	shrex_pid	name
Content as in Figure 4		

Figure 15. Example of withparenttable and tablename.

tations to the *author* and *reaction* elements in the definition of *minisbml*. As shown in the bottom of the figure this results in adding *author* as an attribute in the *minisbml* table. The *minisbml_reaction* will still be generated, but with all its content as XML in the XML column of the table. To remove the molecule information we have added the annotation *ignore* in the XML schema.

Figure 15 demonstrates an alternative way to make the relational mapping easier to understand and use. In this case we do not want to use hybrid storage. Instead the goal is to remove unnecessary relations in the generated shreddings; in this case we can move the *author* to the *minisbml* table since only one author per table is allowed, and force all participants of reactions to be shredded into on single relation, thus decrease the number of relations generated by HShreX.

Removing the *author* relation is achieved by using the annotation *withparenttable*. To shred several substructures into the same table the annotation *tablename* can be used as renaming substructures into the same *tablename* forces the corresponding data to be shredded into the same table.

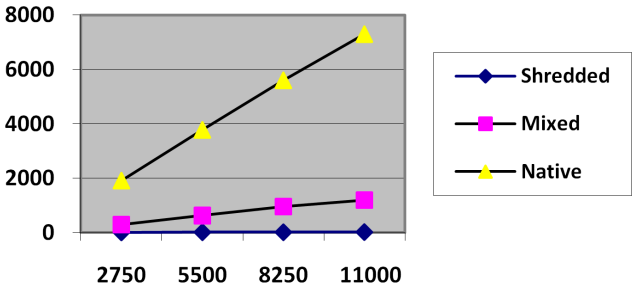

Figure 16. Query performance [ms] with growing datasets (number of UniProt entries on the y axis).

```
SELECT accession
FROM entry, accession, comment, subcellularLocation, location
 WHERE entry.shrex_id = accession.shrex_pid
 AND entry.shrex_id = comment.shrex_pid
 AND comment.shrex_id = subcellularLocation.shrex_pid
 AND subcellularLocation.shrex_id = location.shrex_pid
 AND location.nodeValue='Cytoplasm';
```

Figure 17. UniProt query (for mapping 1)

D. Evaluation

In this section we will evaluate the benefit of working with HShreX. There are two issues, performance of queries and the complexity of data models. We have chosen to work on data available for two commonly used bioinformatics standards SBML 2.1 [9] and UniProt [17]. All tests are done on an AMD Athlon Dual Core 2.9 GHz and 4 GiB RAM.

For our first test we have designed three different data models. The first one is a pure native representation where the XML data files are stored as XML in an XML attribute in one main relation. The second one is a mixed representation, where we have translated parts of the XML into relations and kept other parts as XML. The intuition for creating the mixed representation is to create a hybrid data model reflecting the semantics of the original SBML standard. The third data model is the purely shredded representation produced without any annotations.

There is a clear relation between the choice of model and the query performance as illustrated in figure 16. The query (as it is formulated in SQL for the purely shredded mapping) is listed in 17. The example illustrates the benefit of using the mixed representation in a case where we are joining many tuples. In this case we want to combine data from UniProt (www.uniprot.org). Here, the native representation results in poor performance, while the shredded version is very fast. However, the mixed representation gives a considerable improvement over the purely native representation. This shows that shredding parts of the XML data could have a considerable improvement of the performance.

To illustrate the complexity of the created models we

	Data model	Native	Mixed	Shredded
SBML	*Nr of annotations*	1	21	0
	Nr of relations	3	8	121
UniProt	*Nr of annotations*	1	24	0
	Nr of relations	2	32	121

Table II
INFORMATION ABOUT THE DIFFERENT DATA MODELS

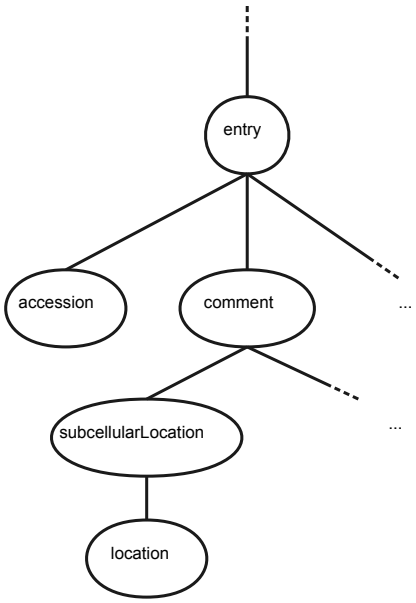

Figure 18. Part of the UniProtKB XML tree. The *comment* node can be seen in figure 2.

present more details of selected models for SBML and UniProt. Table II presents the number of annotations needed and the number of produced relational tables for these mappings. The purely native and the mixed representation produce data models with a limited amount of tables, while the purely shredded model generates many relational tables. This explosion of tables causes data that semantically belong together to be shredded into many places in the data model. The mixed version of our data models creates a data model that provides relational storage for entities that we assume will be commonly accessed in queries and native XML representation used for other parts. We do not use the ignore annotation for this example to make the three models comparable in information content. The examples in the previous section illustrate the intuition on how to build this mapping, basically we add *maptoxml* annotations to the parts to be stored as native XML and *withparenttable* annotations to levels in the XML-tree that we want to omit. As shown in table 2 this is easily done and we only need around 20 annotations for the given schemas.

To further illustrate the impact of shreddings we have also evaluated query performance for all possible hybrid representations relevant for the query in figure 17. There

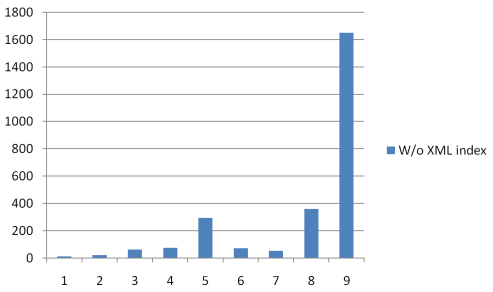

Figure 19. Query performance [ms] for shredded mapping (1), different hybrid mappings (2–8) and native mapping (9).

are seven possible hybrid mappings where a varying degree of the XML subtree affected by the query is stored as a XML value instead of being shredded to relations. Since the sub-tree has two branches (see figure 18) we can design eight different hybrid mappings; *accession* (2 in figure 19), one of *location* (3), *subcellularLocation* (4) and *comment* (5) or *accession* together with one of *location* (6), *subcellularLocation* (7) or *comment* (8) can be mapped as XML. The mappings where *comment* is shredded but *accession* and/or *subcellularLocation* or *location* is mapped as XML all run in under 75 ms on the test system with a dataset of 2750 entries and using no XML indexes. When *comment* is mapped as XML the run time rises to 300–360 ms (depending on whether *accession* is shredded or not). The native mapping is much slower (1650 ms). This demonstrates how choosing a preferable shredding gives acceptable performance and a more comprehensible table structure than the purely shredded mapping. Data stored as XML values do not need serializing back into XML which is a time benefit for certain types of data. Which mapping results in the best query performance while keeping a comprehensible structure is non-intuitive. How efficient a mapping is in terms of performance depends on the query, the structure of the schema and distribution of data within the structure.

VI. RELATED WORK

Regarding related work there is a lot of work on extended functionality for XQuery. Here, the Mark Logic Corporation provides for its XQuery implementation several function libraries to ease application development [44]. In addition, the eXist community has added a number of new functions as function modules to the XQuery implementation, for example a mail, math, SQL and spatial module. Our XQuery extension combines the ideas above to realize graph processing based on the additionally introduced graph model. We also looked at relational database systems and found similar tendencies. Besides, for spatial data applications relational database vendors recognized the need of graph support in areas like biology. One example is the Life Science Platform

by Oracle [45]; another is the Systems Biology Graph Extender (SBGE) for IBM's DB2 database system [46].

The SBGE is of particular interest since it resembles our graph extension tool. It introduces a data model which introduces graphs as a first-class SQL data type. This means that graphs can be manipulated the same way as other data types. In addition it defines operators that can convert data between the graph representation and relational tables containing the corresponding information. The workflow of this extension is very similar to the one of our graph extension. Analogue to the load-functions data stored as SQL tables can be converted to the graph representation. Then operations can be efficiently performed on the graph representation of the data as can be done with our execute-function. Finally, the results can be stored as plain SQL tables or SQL tables containing graphs, similar to access the graph data through the get-function or the XML node returned by the execute function. Similar to our current implementation, the SBGE implementation requires that each graph fit into main memory. SBGE functions can be seamlessly composed in a single SQL query with user defined functions (UDF) written in Java.

The HShreX tool, on the other hand, combines ideas from two related areas for XML storage. The first is the work on automatic shredding of XML documents into relational databases by capturing the XML structure or based on the DTD or XML schema for the XML data [39], [47], [48]. The intention with these approaches is to create an efficient storage for the XML data. The resulting data model is often not easy to understand and is usually hidden from the user via an interface providing automatic query translation of XQuery into the model. Several authors also explore the efficincy of strategies for shredding XML to relational engines [49], [50], [51].

The other related area is work on hybrid XML storage, as provided by the major relational database vendors. The underlying representation for the XML type differs, in some cases it is a byte representation of the XML whereas in other cases it is some kind of shredding of the XML data [33], [34], [37]. These database vendors provide a number of tools to import XML natively or shred the data into the system. These tools are intended for design of one database solution, thus generation and evaluation of alternative solutions becomes time consuming.

Other interesting work regarding design of hybrid storage is the work by Moro et al. [38]. They address the problem by a database design tool based on a conceptual design language and provide guidelines for when parts of the XML structure should be translated or not. In our case the problem is a bit different. The work has similar goals to HShreX but in our case we want to use the already existing XML schema as a starting point.

VII. Towards a future system

We currently use HShreX for creation of hybrid storage models that allows us to compare and evaluate different storage alternatives. Our experience so far is that the system allows fast creation of alternate storage models and that it is easy to create the models that we want to test. However, our experiments so far have highlighted extended functionalities that would be of interest for future versions of the system.

One such is enhanced annotation functionality, for instance to change data typing and add indexes to the created data model. For the moment the system contains a rudimentary implementation for data typing while indexes must be created by hand after loading the model into the database.

The bottleneck of the system is querying for the different data models. This is due to the complexity of the generated data model and the many alternatives provided by SQL/XML. We are investigating ways of automating this process as well, the idea is to use an automatic query translator that suggest a SQL/XML query based on a XQuery query where the user can reformulate the translated query if desired. Currently, we have a solution for using XPath query capabilities within HShreX. This would allow the user to issue XPath expressions inside HShreX that correspond to the original XML data. HShreX will then consult its internal shredding information and query the database for the right data.

Our long term goal is to get a better understanding of how to shred XML into good hybrid data models that is easy to work with and provide an efficient storage model. The final goal is to make HShreX smarter about its shredding rules, i.e., to make HShreX have a more dynamic set of rules and also enable the user to inform HShreX about usage scenarios which would influence these rules. To reach this goal we would like to develop a system which by analyzing data and XML structure could propose different hybrid data models for the user to choose from.

This would also involve combining the two systems i.e. to enable graph functionality directly within HShreX. This could be achieved either by specialized annotations for nodes and edges or possibly also in this case by automatic analysis of the XML data and queries to allow HShreX to automatically detect substructures that should be imported to the graph engine. This would yield a system where the user can choose to store parts of data as graphs, relational or native XML and take advantage of all the possibilities depending on his needs.

For the future it would be interesting to introduce more advanced graph functionality demanded for many biological applications. There is currently a lot of research in specialized and efficient graph management for biological pathways, such as aligning pathways [11] and identifying target molecules for creation of drugs [12]. To extend our solution with this functionality we need to extend the

graph functionality provided by the graph package. The main contribution of this paper, i.e. how to integrate the functionality with XQuery, would, however, be unaffected.

VIII. Conclusion

In this paper we present two tools allowing easy and efficient access and analysis to the large amount of graph related XML data available within systems biology. The first tool is specialised on providing analysis for graph data. A graph model for handling directed and undirected labeled graphs was introduced. Access to the graph model is realized through the XQuery environment. The user can define vertices and edges, execute algorithms and access the graph data as XML for further processing. This results in an efficient framework for processing graph views of XML data with a prototype implementation in eXist and JUNG. The second tools support the user in exploring more efficient storage and querying for XML data. The tool enables hybrid XML storage by adding annotations to the XML schema. We evaluate the tools and show that they provide efficient processing. In the end of the paper we discuss our results and discuss steps towards a future tool that combine the features of the current tools.

Acknowledgment

We acknowledge the financial support of the Center for Industrial Information Technology, the Swedish Research Council and the German Academic Exchange Service. We thank Nahid Shahmehri, Andreas Heuer, Adelinde Uhrmacher and Dr. Holger Meyer for supporting the thesis work which provides the foundation for this paper. We are also grateful to Juliana Freire for input and discussions regarding the HShreX tool.

References

[1] L. Strömbäck and S. Schmidt, "An Extension of XQuery for Graph Analysis of Biological Pathways." in *The First International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA*, 2009.

[2] H. Hermjakob, L. Montecchi-Palazzi, G. Bader, J. Wojcik, L. Salwinski, A. Ceol, S. Moore, S. Orchard, U. Sarkans, C. von Mering, B. Roechert, S. Poux, E. Jung, H. Mersch, P. Kersey, M. Lappe, Y. Li, R. Zeng, D. Rana, M. Nikolski, H. Husi, C. Brun, K. Shanker, S. Grant, C. Sander, P. Boork, W. Zhu, P. Akhilesh, A. Brazma, B. Jacq, M. Vidal, D. Sherman, P. Legrain, G. Cesareni, I. Xenarios, D. Eisenberg, B. Steipe, C. Hogue, and R. Apweiler, "The HUPO PSI's Molecular Interaction format - a community standard for the representation of protein interaction data," *Nature Biotechnology*, vol. 22, no. 2, pp. 177–183, 2004.

[3] F. Collins, E. Green, A. Guttmacher, and M. Guyer, "A vision for the future of genomics research," *Nature*, vol. 422, pp. 835–847, April 2003. [Online]. Available: http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=2003Natur.422..835C

[4] G. Bader, I. Donaldson, C. Wolting, B. Oulette *et al.*, "BIND - The Biomolecular Network Database," *Nucleic Acids Research*, vol. 29, no. 1, pp. 242–245, 2001.

[5] H. Hermjakob, L. Montecchi-Palazzi, C. Lewington, S. Mudali, S. Kerrien, S. Orchard, M. Vingron, B. Roechert, P. Roepstorff, A. Valencia, H. Margalit, J. Armstrong, A. Bairoch, G. Cesareni, D. Sherman, and R. Apweiler, "IntAct - an open source molecular interaction database," *Nucleic Acids Research*, vol. 32, pp. D452–D455, 2004.

[6] G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G. Gopinath, G. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein, "Reactome: a knowledgebase of biological pathways," *Nucleic Acids Research*, vol. 33, no. D428-D432, 2005.

[7] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori, "The KEGG resources for deciphering the genome," *Nucleic Acids Research*, vol. 32, pp. D277–D280, 2004.

[8] P. Karp, M. Arnaud, J. Collado-Vides, J. Ingraham, I. Paulsen, and M. J. Saier, "The E. coli EcoCyc Database: No Longer Just a Metabolic Pathway Database," *ASM News*, vol. 70, no. 1, pp. 25–30, 2004.

[9] M. Hucka, A. Finney, H. Sauro, H. Bolouri, J. Doyle, H. Kitano, A. Arkin, B. Bornstein, D. Bray, A. Cornish-Bowden, A. Cuellar, S. Dronov, E. Gilles, M. Ginkel, V. Gor, I. Goryanin, W. Hedley, T. Hodgman, J.-H. Hofmeyr, P. Hunter, N. Juty, J. Kasberger, A. Kremling, U. Kummer, N. L. Novère, L. Loew, D. Lucio, P. Mendes, E. Minch, E. Mjolsness, Y. Nakayama, M. Nelson, P. Nielsen, T. Sakurada, J. Schaff, B. Shapiro, T. Shimizu, H. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[10] G. D. Bader and M. P. Cary, *BioPAX - Biological Pathways Exchange Language Level 2, Version 1.0 Documentation*, BioPAX workgroup, December 2005.

[11] F. Ay, T. Kahveci, and V. Crecy-Lagard, "Consistent alignment of metabolic pathways without any abstraction modeling." in *International Conference on Computational Systems Biology (CSB)*, 2008.

[12] P. Sridhar, B. Song, T. Kahveci, and S. Ranka, "Mining methabolic networks for optimal drug targets." in *Pacific Symposium on Biocomputing (PSB)*, 2008, pp. 291–302.

[13] L. Strömbäck, D. Hall, and P. Lambrix, "A review of standards for data exchange within systems biology." *Proteomics*, vol. 7, no. 6, pp. 857–867, March 2007. [Online]. Available: http://dx.doi.org/10.1002/pmic.200600438

[14] G. Cochrane, P. Aldebert, N. Althorpe, M. Andersson, W. Baker, A. Baldwin, K. Bates, S. Bhattacharyya, P. Browne, A. van den Broek, M. Castro, K. Duggan, R. Eberhardt, N. Faruque, J. Gamble, C. Kanz, T. Kulikova, C. Lee, R. Leinonen, Q. Lin, V. Lombard, R. Lopez, M. McHale, H. McWilliam, G. Mukherjee, F. Nardone, M. P. Pastor, S. Sobhany, P. Stoehr, K. Tzouvara, R. Vaughan, D. Wu, W. Zhu, and R. Apweiler, "EMBL Nucleotide

Sequence Database: developments in 2005." *Nucleic Acids Res*, vol. 34, no. Database issue, January 2006. [Online]. Available: http://dx.doi.org/10.1093/nar/gkj130

[15] A. Garny, D. Nickerson, J. Cooper, R. W. dos Santos, A. Miller, S. McKeever, P. Nielsen, and P. Hunter, "CellML and associated tools and techniques," *Philosophical Transactions of the Royal Society A*, vol. 366(1878), pp. 3017–3043, 2008.

[16] P. Murray-Rust and H. S. Rzepa, "Chemical Markup, XML, and the World Wide Web. 4. CML Schema," *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 3, pp. 757–772, May 2003. [Online]. Available: http://dx.doi.org/10.1021/ci0256541

[17] The UniProt Consortium, "The Universal Protein Resource (UniProt)," *Nucl. Acids Res.*, p. gkm895, 2007. [Online]. Available: http://nar.oxfordjournals.org/cgi/content/abstract/gkm895v1

[18] P. T. Spellman, M. Miller, J. Stewart, C. Troup, U. Sarkans, S. Chervitz, D. Bernhart, G. Sherlock, C. Ball, M. Lepage, M. Swiatek, W. L. Marks, J. Goncalves, S. Markel, D. Iordan, M. Shojatalab, A. Pizarro, J. White, R. Hubley, E. Deutsch, M. Senger, B. J. Aronow, A. Robinson, D. Bassett, C. J. Stoeckert, and A. Brazma, "Design and implementation of microarray gene expression markup language (mage-ml)." *Genome Biol*, vol. 3, no. 9, August 2002. [Online]. Available: http://dx.doi.org/10.1186/gb-2002-3-9-research0046

[19] P. G. A. Pedrioli, J. K. Eng, R. Hubley, M. Vogelzang, E. W. Deutsch, B. Raught, B. Pratt, E. Nilsson, R. H. Angeletti, R. Apweiler, K. Cheung, C. E. Costello, H. Hermjakob, S. Huang, R. K. Julian, E. Kapp, M. E. Mccomb, S. G. Oliver, G. Omenn, N. W. Paton, R. Simpson, R. Smith, C. F. Taylor, W. Zhu, and R. Aebersold, "A common open representation of mass spectrometry data and its application to proteomics research," *Nature Biotechnology*, vol. 22, no. 11, pp. 1459–1466, November 2004. [Online]. Available: http://dx.doi.org/10.1038/nbt1031

[20] S. Orchard, C. F. Taylor, H. Hermjakob, Weimin-Zhu, R. K. Julian, and R. Apweiler, "Advances in the development of common interchange standards for proteomic data." *Proteomics*, vol. 4, no. 8, pp. 2363–2365, Aug 2004.

[21] R. Stanislaus, L. H. Jiang, M. Swartz, J. Arthur, and J. S. Almeida, "An XML standard for the dissemination of annotated 2D gel electrophoresis data complemented with mass spectrometry results." *BMC Bioinformatics*, vol. 5, no. 1, January 2004. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-5-9

[22] A. Keller, J. Eng, N. Zhang, X.-J. Li, and R. Aebersold, "A uniform proteomics MS/MS analysis platform utilizing open XML file formats," *Molecular Systems Biology*, vol. 1, no. 1, pp. msb4 100 024–E1–msb4 100 024–E8, August 2005. [Online]. Available: http://dx.doi.org/10.1038/msb4100024

[23] E. Deutsch, "mzML: A single, unifying data format for mass spectrometer output," *Proteomics*, vol. 8, no. 14, pp. 2776–2777, 2008. [Online]. Available: http://dx.doi.org/10.1002/pmic.200890049

[24] "mzML 1.1.0 Specification," 2009. [Online]. Available: http://www.psidev.info/index.php?q=node/257

[25] A. Brazma, P. Hingamp, J. Quackenbush, G. Sherlock, P. Spellman, C. Stoeckert, J. Aach, W. Ansorge, C. A. Ball, H. C. Causton, T. Gaasterland, P. Glenisson, F. C. Holstege, I. F. Kim, V. Markowitz, J. C. Matese, H. Parkinson, A. Robinson, U. Sarkans, S. Schulze-Kremer, J. Stewart, R. Taylor, J. Vilo, and M. Vingron, "Minimum information about a microarray experiment (MIAME)-toward standards for microarray data." *Nat Genet*, vol. 29, no. 4, pp. 365–371, December 2001. [Online]. Available: http://dx.doi.org/10.1038/ng1201-365

[26] S. Orchard, H. Hermjakob, R. K. Julian, K. Runte, D. Sherman, J. Wojcik, W. Zhu, and R. Apweiler, "Common interchange standards for proteomics data: Public availability of tools and schema." *Proteomics*, vol. 4, no. 2, pp. 490–491, February 2004. [Online]. Available: http://dx.doi.org/10.1002/pmic.200300694

[27] N. Le Novère, A. Finney, M. Hucka, U. S. Bhalla, F. Campagne, J. Collado-Vides, E. J. Crampin, M. Halstead, E. Klipp, P. Mendes, P. Nielsen, H. Sauro, B. Shapiro, J. L. Snoep, H. D. Spence, and B. L. Wanner, "Minimum information requested in the annotation of biochemical models (miriam)," *Nature Biotechnology*, vol. 23, no. 12, pp. 1509–1515, December 2005. [Online]. Available: http://dx.doi.org/10.1038/nbt1156

[28] T. cytoscape consortium, 2009. [Online]. Available: www.cytoscape.org

[29] F. Bois, "GNU MCSim: Bayesian statistical inference for SBML-coded system biology models," *Bioinformatics*, vol. 25, no. 11, pp. 1453–1454, 2009.

[30] L. Strömbäck and D. Hall, "An Evaluation of the Use of XML for Representation, Querying, and Analysis of Molecular pathways." in *EDBT Workshops.*, 2006.

[31] L. Strömbäck and P. Lambrix, "Representation of molecular pathways: an evaluation of SBML, PSI MI and BioPAX," *Bioinformatics*, vol. 21, no. 24, pp. 4401–4407, October 2005.

[32] W3C, "XQuery 1.0: An XML Query Language." W3C, 2007. [Online]. Available: www.w3.org/TR/2007/REC-xquery-20070123/.

[33] K. Beyer, F. Özcan, S. Saiprasad *et al.*, "DB2/XML:Designing for Evolution." in *SIGMOD 2005*, 2005, pp. 31–38.

[34] M. Rys, "XML and relational Management Systems; Inside Microsoft SQL Server 2005." in *SIGMOD 2005*, 2005.

[35] R. Murthy, Z. Hua Liu, M. Krishnaprasad, S. Chandrasekar, A.-T. Tran, E. Sedlar, D. Flurescu, S. Kotsovos, N. Agarwal, V. Arora, and V. Krishnamurthy, "Towards an enterprise XML architecture," in *SIGMOD 2005*, 2005.

[36] M. Krishnaprasad, Z. Hua Liu, A. Manikutty, J. Warner, and V. Arora, "Native XQuery processing in Oracle XMLDB," in *SIGMOD 2005*, 2005.

[37] Z. Hua Liu, M. Krishnaprasad, and V. Arora, "Native XQuery Processing in XMLDB," in *SIGMOD 2005*, 2005.

[38] M. Moro, L. Lim, and Y.-C. Chang, "Schema Advisor for Hybrid Relational-XML DBMS." in *SIGMOD 2007*, 2007.

[39] S. Amer-Yahia, F. Du, and J. Freire, "A Comprehensive Solution to the XML-to-Relational Mapping Problem." in *ACM International Workshop on Web Information and Data Management (WIDM)*, 2004, pp. 31–38.

[40] D. Chamberlin, M. Carey, M. Fernandez, D. Florescu, G. Ghelli, D. Kossmann, J. Robie, and J. Simeon, "XQueryP: An XML Application Development Language." in *XML 2006*, 2006.

[41] D. Chamberlin, M. Carey, D. Florescu, D. Kossman, and J. Robie, "XQueryP: Programming with XQuery." in *Third International Workshop on XQuery Implementation, Experience, and Perspectives.*, 2006.

[42] G. Ghelli, C. R, and S. J., "XQuery!: An XML query language with side effects," in *Second International Workshop on Database Technologies for Handling XML Information on the Web (DataX 2006)*, 2006.

[43] L. Strömbäck, M. Åsberg, and D. Hall, "HShreX: a Tool for Design and Evaluation of Hybrid XML Storage," in *FLexDBIST 2009*, 2009.

[44] Mark Logic Corporation, "Mark Logic Server, XQuery API Documentation," Mark Logic Corporation, 2007. [Online]. Available: http://xqzone.marklogic.com/pubs/3.0/apidocs/Extension.html

[45] Oracle, "Oracle life science platform," Oracle, 2007. [Online]. Available: www.oracle.com/technology/industries/life_sciences/

[46] B. Eckman and P. Brown, "An overview of data models for the analysis of biochemical pathways." *Systems Biology*, vol. 50, no. 1, pp. 246–259, 2006.

[47] B. Bohannon, J. Freire, P. Roy *et al.*, "From XML Schema to Relations: A Cost-Based Approach to XML Storage." in *IEEE International Conference on Data Engineering*, 2002, pp. 64–75.

[48] D. Floresco and D. Kossman, "Storing and Querying XML data using RDBMS." *IEEE Data Eng. Bull.*, vol. 22, no. 3, pp. 27–34, 1999.

[49] H. Georgiadis and V. Vassalos, "XPath on stereoids: Exploiting relational engines for XPath performance." in *SIGMOD'07*, 2007.

[50] I. Mlynkova, "Standing on the Shoulders of Ants: Towards More Efficient XML-to-Relational Mapping Strategies." in *19th International Workshop on Databaseand Expert Systems Applications,*, 2009.

[51] T. Grust, J. Rittinger, and J. Teubner, "Why Off-the-shelf RDBMSs are better at XPath Than you Might Expect." in *SIGMOD'07*, 2007.