Using Statistical Information for Efficient Design and Evaluation of Hybrid XML Storage

Lena Strömbäck
Swedish Meterological and Hydrological Institute
Folkborgsvägen 1, 601 76 Norrköping
lena.stromback@smhi.se

Valentina Ivanova, David Hall
Department of Computer and Information Science
Linköpings Universitet
S-581 83 Linköping, Sweden
valentina.ivanova@liu.se, david@dpg.se

***Abstract* — Modern relational database management systems provide hybrid XML storage, combining relational and native technologies. Hybrid storage offers many design alternatives for XML data. In this paper we explore how to aid the user in effective design of hybrid storage. In particular we investigate how the XML schema and statistical information about the data can support the storage design process. In our previous work, we presented our tool HShreX that uses statistical information about a data set to enable fast evaluation of alternative hybrid design solutions. In this paper, we extend this work by presenting more details about the tool and results of an extended evaluation. In particular, this paper gives a detailed presentation on how the tool aids in the storage design and evaluation process.**

Keywords – XML, Hybrid XML management, indexing, storage design.

I. INTRODUCTION

The rapid increase in web based applications yields an increasing interest in using XML (eXtensible Markup Language) for representation of data. XML is able to represent all kinds of data ranging from marked-up text, through so called semi-structured data to traditional, well-structured datasets. Supporting the flexibility that makes XML appealing is challenging from data management and technical perspectives. Several approaches have been used including native databases and shredding XML documents into relations. In practice, hybrid storage that combines native and relational solutions is of large interest. Hybrid storage is provided by the major relational database vendors (Oracle, IBM DB2 and Microsoft SQL Server). They offer interesting options for storage design where native and relational storage can be used side by side. In our previous work [1], we present our tool HShreX that uses statistical information about a data set to aid in hybrid storage design.

Several studies evaluate different solutions for XML management. As an example, [2] and [3] provide general benchmarks for XML data while [4] and [5] gives a case study of XML data within bioinformatics. For shredding, a number of different strategies are available [6]. It is well known that the choice of translation strategy affects the efficiency [7][8][9] and that the translation can be optimized in many ways. However, comparisons of different storage strategies [10] and hybrid XML storage [11] [12] [13] has, so far, only been studied in a few cases. The above studies discuss a number of features that may have an impact on how to achieve efficient storage; the complexity and regularity of the XML structure; how the data is queried, i.e., the access patterns for different entities in the data set; and the frequency of references to other sources.

In this paper, we further explore these issues by investigating the impact of the application on the performance of the database. The properties we are focusing on are the XML schema structure and statistical properties of the data set. In Section II, we motivate and discuss the goals of our work that extends the discussion from [1]. This is followed by a discussion of properties and measurements relevant for storage design in Section III. We present our tool that enables fast evaluation and exploration of storage solutions in Section IV. Here, we extend the presentation from [1], which give a better understanding on how the tool can be used for a fast analyze of properties for a dataset. Statistical analysis of the data sets used for the tool evaluation is presented in Section V. In Section VI, we further extend the previous evaluation to show the feasibility of the tool. Related work is presented in Section VII. The paper is summarized by presenting our future vision in Section VIII. Our long term goal with the work is to present a method that can suggest a set of plausible hybrid storage models for an application.

II. MOTIVATION AND GOALS

Previous work [5][7][14][15][16] has defined efficient shredding methods for XML data into relational databases that result in fast query times. For hybrid storage, the situation is more complex where an inappropriate choice of storage design can lead to poor performance [17]. In general, automatically shredded relational XML mappings can lead to a rather large and complicated structure of relations. On the other hand, storing entire XML documents natively in XML keeps the structure completely intact to the cost of slow access to the data. For hybrid XML storage, we have the choice to store parts of the XML structure as relations and other parts as XML and can gain from the benefit of a good data model and relatively fast performance. The design of a good hybrid storage model is complex and dependent on the requirements for the specific application [17].

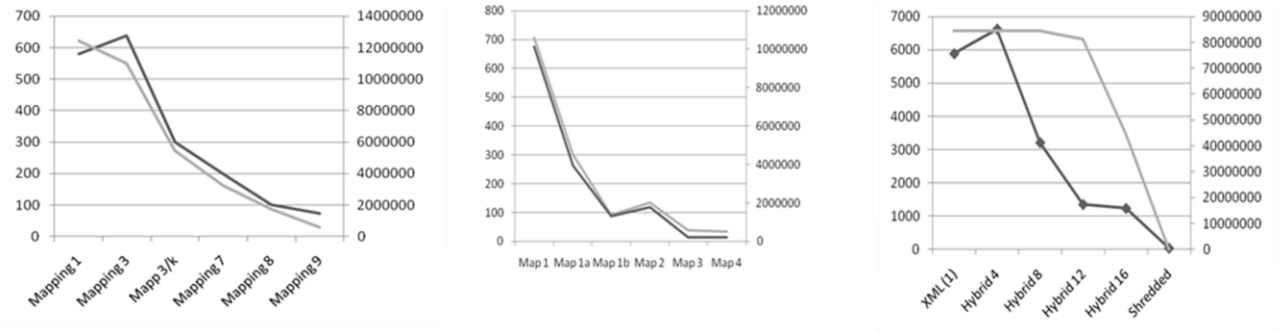

Figure 1. Run times [ms] (black) and data size [bytes] (grey) for PSI-MI (left), UniProt (middle) and Michigan Benchmark (right)

It is important to determine, which properties are relevant for designing efficient hybrid storage. Previous work [12][17] discusses a number of guidelines to take into account during the design process. These guidelines give general advices on how to store data, and we summarize the guidelines from [17] as they provide general goals for this paper.

Guidelines for hybrid XML storage:

1. Keep together what naturally belongs together. Parts of the data that corresponds to a semantic entity is likely to be used together. Therefore it is in many cases a good idea to keep it stored as XML and not shredded into many relations or different representations.
2. Do not shred parts of the XML where the schema allows large variation. As a relational representation is less flexible than XML it is usually preferred to store parts where the schema allows variation as XML.
3. Analyze the data to decide actual variation. The XML schema gives a good intuition of the possible variation of data but it does not give the full picture.
4. Prefer relational representation for elements that are critical for performance. Here, the intuition is to identify the XML elements that are critical for query performance and common queries for the application.
5. Prefer the representation that is required for query results. For the case where the application requires that the result from the query should be returned as XML and not as a relational table shredding is not beneficial.
6. Avoid shredding where new versions of the schema are likely to change.

The above guidelines are easy to use and help the user to design fairly efficient hybrid XML storage for many applications. It should be noted, though, that there are many cases where the different guidelines points in different directions and where the best tradeoff is given by the need of the application. Therefore there is a need for further evaluations and studies.

Exploration and evaluation of alternative solutions is a time consuming task. Methods and tools, to aid the user in design of hybrid storage, and measurements, that could give hints on how to make choices, are of high importance. Based on the guidelines we can conclude that in order to refine the design guidelines we need to explore properties of the XML structure, the XML schema or DTD and the structure of actual data.

In a preliminary evaluation, we compared the query efficiency with the amount of data stored as XML in the hybrid solution. In our tests, we adopt the shredding principles used in ShreX [14][18] as these principles give a mapping that captures the semantics of a given XML schema for the XML data. To explore hybrid storage we used the extended system HShreX [10][19], which also allows hybrid XML mappings. The general principle behind the mappings of these systems is that complex elements are translated to relational tables. Simple elements and attributes are shredded to a column in their parent table if they occur at maximum once in its parent element. HShreX extends this basic shredding by providing hybrid XML storage, i.e., to allow parts of the structure to be kept as XML in the final database representation. In our study the complexity of the created models varies between one or two relations for the models stored in pure XML to over 100 relations for the fully shredded data models.

The results of these tests are illustrated in Figure 1. The first two graphs show the results for two real data sets from the IntAct [20] and UniProt [21] databases. In this case we can see that the amount of data stored as XML gives a good estimation of the expected query time. For the Michigan Benchmark data [22] the estimation is not as good as for the two other datasets. This means that the amount of data is a good indicator for the performance, but also that further statistics about the data could give us better indicators and aid in effective storage design.

III. AVAILABLE INFORMATION

The general guidelines presented in the previous section show that there are three sources of information that are important to understand storage requirements for a computer application. These are: the general data schema, i.e., the data model (guidelines 1 and 2), samples of data to determine how the data model is used and what parts of the data model are in most common use (guideline 3), and samples of

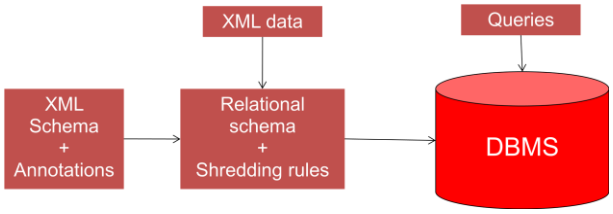

Figure 2. The general architecture of HShreX

queries to determine what kind of queries are often performed for the data (guideline 4 and 5). In this work, we will examine how to use the data model and statistical information for a particular dataset.

As shown in the previous section, the amount of data stored as XML is related to the query performance. However, the prediction we get from simply measuring the amount of data is not enough, we also need to collect more detailed information about the structure of the data. In practice, different parts of the XML schema are populated differently in different data sets. The XML schema carries information about the general structure, but, as for relational databases, the schema does not give a full picture of how this structure is instantiated for a particular dataset. We want to capture this information to create an effective hybrid storage model. In previous work [23], where we worked with generated data, we could see that also the amount of data at various positions in the XML file and the structure of this data had an impact on query performance. We wanted to explore this further and collected the following information:

- Overall statistics for the dataset. With this we mean characterizing the general structure of the dataset. For this purpose we use simple measures, such as, the total number of attributes, elements, and levels in the XML. We also collect the number of elements at each level of the dataset to determine the fan out of the data.
- Diversity of the dataset. To get estimations of diversity we collect the number of elements and attributes for each element or attribute string, at which depths they occur and compare those to the number of overall elements. We also collect information on how many unique search paths occur within the data set and the number of their occurrence.
- Detailed information at each position in the file. This is collected by counting the occurrence of element names at each level in the file. For each combination of parent/child node we count how common the child node is for this parent and collect the minimum, maximum and mean number of times this child occurs for the parent.

Our previous work on generated data has shown that parent/child statistics were of particular interest since this had a large impact on query performance.

IV. A TOOL FOR EVALUATION

To allow easy access to the statistics and aid in evaluating storage alternatives we extended our tool HShreX to include this new information. The new version of the tool can be used to create and evaluate different XML storage models. We start with a description of the general functionality of the system.

The general architecture of HShreX is shown in Figure 2. The system analyses an XML schema and represents it as a tree structure, which facilitates its visual perception. The tree structure helps to easily understand and navigate the schema components as well. The relational schema is likewise created during the schema analyses. Once the database structures are created, large datasets, which corresponds to the currently parsed schema, can be quickly shredded in the database. Each step starting from the XML schema parsing and ending in datasets loading is logged and available for review in a panel under the main work area.

The relational schema is created following the shredding strategy, mentioned above. The actual XML structure is kept by foreign key relations between the created relational tables. These shredding rules are described in [10] and 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 points 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.

The user can alter the data shredding rules using HShreX annotations [10]. In this way, the XML data can be represented in purely native, mixed and shredded storage models. The HShreX annotations provide the opportunity to switch rapidly and flexibly between different storage models, create them in a database and evaluate their performance features.

HShrex's user interface provides three panels, which give more details of the schema elements and their mappings. Figure 4 gives an overview of the information on these panels. In the figure we show details for the element *model*. The first panel (top) lists specific details, such as currently applied HShreX annotations, children elements and attributes and their occurrences, for the *model* element in the XML schema tree. In this case the *model* element has three attributes and no annotations have been applied. The second (middle) shows HShreX mapping of the selected element or attribute in the tree. Following our translation rules, *model* is translated into a relational table, with its three XML

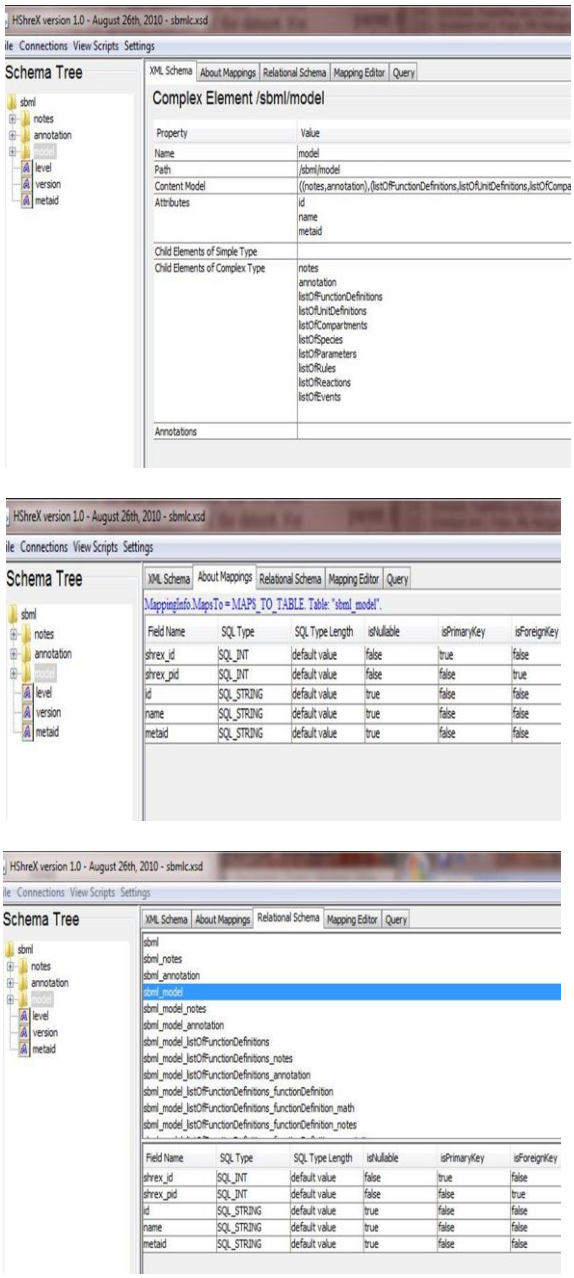

Figure 4. HshreX main panels, XML schema (top), relational mapping (middle) and relational schema (bottom)

attributes translated to attributes in the relational table. Note, in particular, the attributes *shrex_id* and *shrex_pid* used to keep the relational structure. The full relational schema and their relations are available in the third panel (bottom of Figure 4).

In this work, the user interface was extended in two directions – to provide more convenient work with HShreX annotations and to visualize more information for a particular dataset.

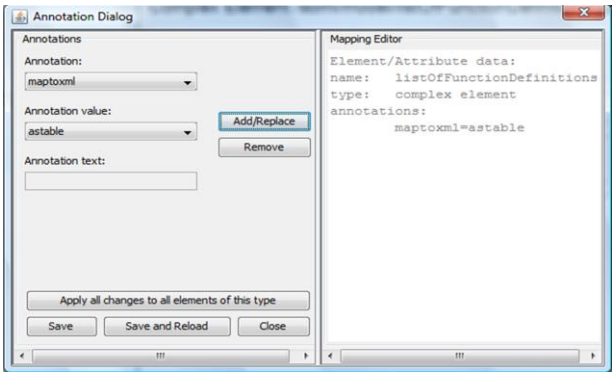

Figure 3. Add/remove annotations dialogue

A. *Annotating the data*

Important for allowing fast evaluations is easy change of the shredded representation of the data. Therefore, HShreX allows the default shredding rules to be influenced via annotations. The supported annotations were originally developed for relational shredding of XML [14] and extended to allow hybrid XML representation [19]. To get a better understanding of the functionality we give an overview of the most 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., it will not be represented in the resulting data model.

outline – used on simple elements (or attributes) where it is desired that they should be stored in a separate table.

withparenttable – used to merge a child with its parent in order to reduce the number of tables in the 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).

These annotations allow a rapid change of shredded hybrid storage model. However, in the original system the user had to open and edit the XML schema textually. This was rather complicated and slowed down the process. Therefore we extended the system with a dialog, allowing the user to alter annotations directly from the schema tree, which is shown in the left pane of HShreX.

Figure 3 shows the dialog that facilitates manipulation of HShreX annotations. While navigating in the schema tree, we can open the dialog for the element or the attribute of interest and process its annotations. The dialog provides functionality for adding annotations, updating, i.e., changing values of available annotations and deleting annotations. Since some combinations of annotations for an element or an attribute are not valid, we validate each annotation regarding

the already available annotations prior to adding. A useful feature is provided through the "Apply all changes to all elements of this type" button, i.e., the currently added/removed annotations will be applied to all elements of this type in the XML schema with a single action. The basic data and the annotations, which apply to the element or the attribute of interest, are listed in the right side of the dialog.

B. Statistical analysis of the dataset.

The second improvement in the user interface is orientated towards the statistical information available for a particular dataset. HShreX obtains this information by analyzing a set of sample XML files representing the dataset. We collect the information described in Section III above. However, for designing the interface it was important to make the statistics easy available for the user at the time when it was needed. Therefore we wanted to integrate statistical information into the HShreX user interface as far as possible.

In HshreX detailed information, for the element or the attribute of interest and its children elements and attributes, is presented in the schema tree when a particular dataset is loaded to the database in use. The resulting interface is shown in Figure 6. When data is loaded into the tool, statistical information is shown in the XML Schema Tree (left part of the figure) and additional information is presented in the XML Schema pane (right part of the figure). To start with the statistics show how common the selected element is (in this case the element *model*). The first three lines in the pane show that there are 251 occurrences of the elements *model*, all of them on the second level in the XML file and in this particular position (path) of the files. As a contrast the same value for the element *speciesReference* is shown in Figure 5. From this statistics we can derive that this elements is very common, all occurrences are on level 6 in the file, but only a bit more than half of them in this particular path.

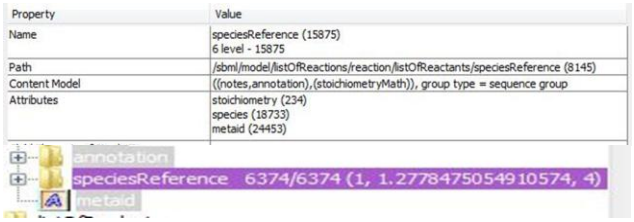

Figure 5. Statistics for the *speciesReference* element.

The remainder of the figures in the XML Schema panel shows for each occurrence of child element or attribute occurring in the selected element the total number of occurrences on the document.

The XML Schema Tree (left in Figure 6) gives more information on the structure of the data. For each child element of *model* it shows how common these are as children to *model*. For instance, *listOfCompartments* occurs in all occurrences of *model* while *listOfRules* only occurs in 129 occurrences of *model*. This information is of particular interest when designing the hybrid model as common elements are often beneficial to shred into relations. Three different colors are used to facilitate user's perception and to show how many times a particular child node appears under its parent element, i.e., different children nodes are colored depending on their frequency of appearance. Thus, the user gets fast and highly useful overview of child nodes and can prioritize his next studies based on this information.

In addition, the figures within parenthesis show how the minimal, mean and maximum number of occurrences for each child elements occurs for this element. For *model*, we can see that each of the child elements occurs exactly once (when they are available). For our second example the

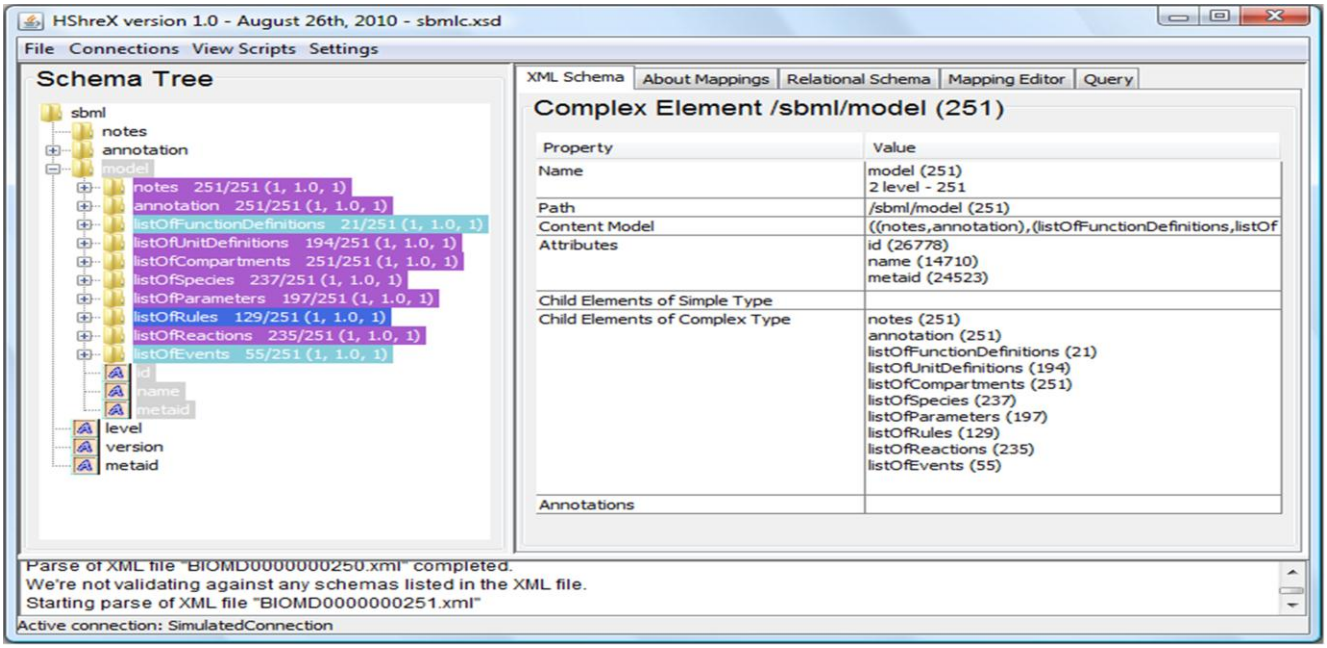

Figure 6. Statistics for the Biomodels dataset as used in the HShreX tool.

	Reactome	BioModels
Files	1	251
Levels	6	8
Total elements	31502	93673
Total Attributes	38062	124756
Elements on each level	1 1 3 9144 8358 13995	250 254 1814 20008 32375 31020 7951 1
Total depth	6	8
Mean depth	5,15	5,2
Unique elements	13	35
Unique paths	14	70

Table 1: Statistical overview of two our selected datasets

Figure 7. Statistical data for the *reaction* element. Reactome (top) and BioModels (bottom).

statistics in Figure 5 shows that each occurrence of *listOfReactants* has one to four child elements (*speciesReference*), the mean number of occurrences is 1.3. The amount of statistical data visualized in the schema tree is small, however, our experience have shown that it is the most useful part of the information available for the dataset. The schema tree representation of statistical information aids the user decision on what annotations are appropriate to be used for a particular dataset and helps to construct proper queries with higher efficiency. Further, the statistics can help the user to create indexes and optimize queries. The other part of the statistical data described in the previous section can be found in "Open Main Statistics" and "Open All Statistics" dialogs under the "File" menu. In addition HShreX can give a summary of all statistics. This summary

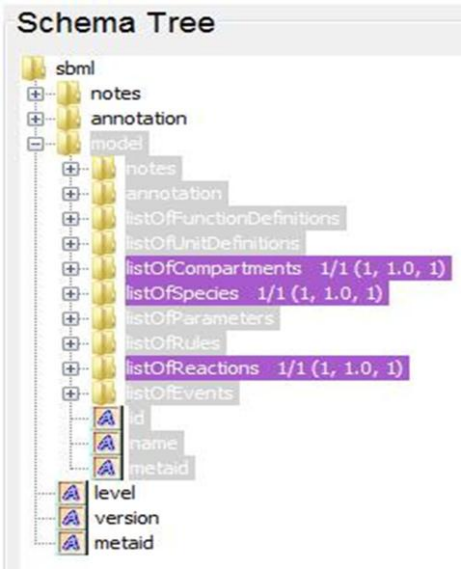

Figure 8. Analysis for the Homo Sapiens dataset.

also contains some general facts about the file collection, such as, total number of elements attributes and characters, the maximum and mean depth of the XML data, the total number of unique elements and paths in the data. This data gives a very quick overview of the dataset before designing the storage solution.

V. STATISTICAL ANALYSIS OF DATA

In this section, we show how the statistics can be used to explore two selected datasets. For the study we have selected two datasets represented in the SBML 2.1 (Systems Biology markup Language version 2.1) XML schema [24] XML standard. To explore the benefit of our tool and the statistical information, we used it for designing hybrid shredding and evaluate its performance on the Homo Sapiens dataset from the Reactome database [25] and on the BioModels dataset. Reactome dataset contains an export of data from the Reactome dataset and while the BioModels dataset contains simulation models for pathways.

It turns out that the overall structure of the two datasets is very different. A quick overview of the statistical information provided by HShreX is given in Table 1. From the table we can see that the BioModels data is about three times as large as the Reactome data in terms of number of attributes and elements. It is also clear that the Reactome data have less depth and higher fan out than the BioModels data. This means that the data in the first dataset is spread in depth (the data is stored on many levels) and the data in the second dataset is spread in width (the data is populated almost equally within the dataset).

More interesting is, however, that the number of unique elements and especially unique paths is much larger in the BioModels data, this hints that there is much more variety in the BioModels data than in the Reactome data. We can use the user interface to further investigate the differences.

The statistics available directly in the HShreX schema tree for the two datasets are available in Figures 6 and 8. This pane gives detailed information for the occurrence of

the nodes and their parents and presents a clear view of data

```
Shredded:
SELECT a."id", a."name"
FROM sbml_model_listOfReactions_reaction a,
        sbml_model_listOfReactions_reaction_listOfReactants b,
        sbml_model_listOfReactions_reaction_listOfReactants_speciesReference c
WHERE a."shrex_id" = b."shrex_pid"
      AND b."shrex_id" = c."shrex_pid"
       AND c."species" = 'REACT_5251_1_Oxygen';

Native:
SELECT reaction.query(   'for $i in /reaction/listOfReactants/speciesReference
          where  $i/@species = "REACT_5251_1_Oxygen"
          return <Details> {$i/../../@id} {$i/../../@name} </Details>') "data"
FROM sbml_model_listOfReactions_reaction
WHERE reaction.exist('/reaction/listOfReactants/speciesReference
                      [@species="REACT_5251_1_Oxygen"]') = 1;
```

Listing 1. Sample query for SBML – Query 1

distribution in the particular dataset. For Reactome dataset all data are collected in the *listOfCompartments*, *listOfSpecies* and *listOfReactants* elements. For the BioModels dataset, the data is much more spread over different parts of the XML schema. Figure 7 shows a further analysis of some part of the data, in this case the *reaction*, and shows that the same relation holds, BioModels data is more diversified than Reactome data.This analysis shows us that a hybrid model for Reactome data can be very simplified as only parts of the XML structure needs to be represented. It also shows that for both datasets *reaction* is one element critical for performance and thus important for further studies.

VI. Evaluating the Approach

Examining the mentioned datasets, using the HShreX interface, we noticed that some of the elements and their parents occur more often than others, thus our research will be more productive if we concentrate on them. In this section, we will discuss how we work with HShreX in two different application domains.

A. Bioinformatics data

Our discussion in the previous section showed that *reaction* and *model* are important elements in our SBML datasets. Therefore in our examples we have applied the HShreX annotation **maptoxml** to the *reaction* and to the

```
Shredded:
SELECT d."species", b."shrex_pid", e."species
FROM sbml_model_listOfReactions_reaction_listOfReactants b,
        sbml_model_listOfReactions_reaction_listOfProducts c,
        sbml_model_listOfReactions_reaction_listOfReactants_speciesReference d,
        sbml_model_listOfReactions_reaction_listOfProducts_speciesReference e
WHERE c."shrex_pid" = b."shrex_pid"
      AND b."shrex_id" = d."shrex_pid"
      AND c."shrex_id" = e."shrex_pid"
       AND d."species" = 'REACT_5251_1_Oxygen';

Native:
SELECT reaction.query(   'for $react in //reaction,
          $rtant in $react/listOfReactants/speciesReference,
          $prod in  $react/listOfProducts/speciesReference
          return <path> {data($rtant/@species)} {data($react/@id)}
                        {data($prod/@species)} </path>') "test"
FROM sbml_model_listOfReactions_reaction
WHERE reaction.exist('//reaction/listOfReactants/speciesReference
                      [@species="REACT_5251_1_Oxygen"]') = 1;
```

Listing 2. Sample query for SBML – Query 2

model elements in the XML schema. This particular annotation/value combination has been selected in order to force the HShreX application to store these parts of the data as pure XML in the corresponding database. If we do not apply any HShreX annotations, the data in the datasets is represented in a shredded storage model. HShreX has been forced to represent the data in a hybrid and in a pure native storage models applying the **maptoxml** annotation to the *reaction* and *model* elements respectively.

We have chosen two of the major database servers available on the market and set up their options related to the XML data representation in various configurations. Using the database servers XML storage capabilities we are able to store the XML data with or without associating it with corresponding XML schema. The database servers run on HP Proliant DL380 G6 Server with two Intel Xeon E5530 Quad Core HT Enabled processors running at 2.4 GHz (in total 16 logical processors) and 30 GB RAM.

We have created different SQL queries (exemplified in Listing 1 and Listing 2) and executed them against the three storage models and different database configurations. In Query 1, the simpler among both, we retrieve details for a reaction where one of its participants is specified. In the second query, we join details for reactions and reactions to extract participants and products for all reactions. First we executed the two queries using only the homo sapiens

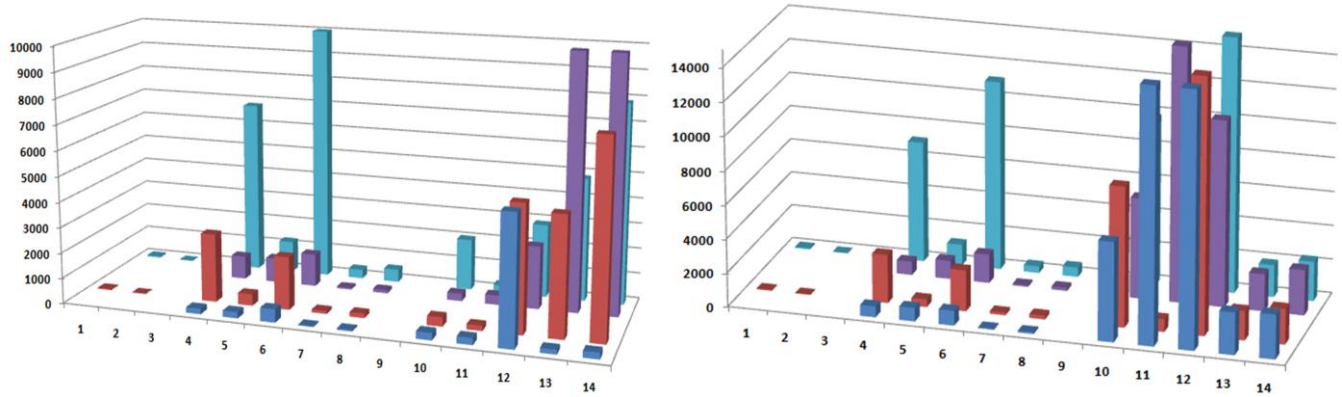

Figure 9. Performance [ms] for Query 1 (left) and Query 2 (right) where: ■ homo sapiens dataset with index, ■ homo sapiens dataset without index, ■ homo sapiens and biomodels datasets with index and ■ homo sapiens and biomodels datasets without index

dataset. After that we loaded both datasets at the same time and evaluated how the response time changes when the size of the data stored in the database increases. The measured performance can be influenced by other processes running on the server. To reduce this influence, the queries from the figures were executed ten times per condition set, and the averages of the results are presented.

First runs were made without any additional optimization. Based on the statistics, proper XML indices, for each variation of database storage options, were created and the same queries were executed again. Thus, we benefit from the statistical information available for a particular dataset in three ways: we can use the statistics to choose the best place for the HShreX annotations regarding our interests and in this way switch flexibly and rapidly between different storage models. We are as well able to create proper, for each storage model, indices based on the view of the data distribution in the particular dataset. A final advantage is that we can optimize our SQL queries not only creating indices but rewriting them based on the data distribution and complexity.

The results from the two different query executions are shown in Figure 9. The equivalent positions on the 'X' coordinate in both of the charts correspond to equivalent condition sets of database storage options. The results from positions 1 and 2 correspond to a fully shredded storage, positions 4 – 8 correspond to a hybrid storage and positions 10 – 14 correspond to a pure native XML storage. Positions 4 – 8 use the same conditions sets of database storage options as positions 10 – 14, however the HShreX annotation is applied to different elements. As we expected, there is a clear relation between the storage model and the query performance, i.e., the execution times are fastest in the shredded storage and slowest in the pure native storage.

Examining the positions 4 – 14 in both result sets we can clearly see that the query performance varies with a different amount for the different database storage options when the size of the data in the database increases. The performance is usually improved when the XML indices are created. It is worth noting that this is not true for position 11 in Query 2 where the performance drops considerably when the index is used. While positions 4 – 8 in the two results sets are comparable, positions 10 – 14 have a lot of differences. Positions 13 and 14 in the first results set have the worst performance among the results for pure native storage while in the second results set they have the best performance. Analyzing positions 13 and 14 in the first result set shows that indices have excellent performance when the size of the data is relatively small and their performance decrease when the data size increases. It is worth noting as well the differences between positions 7, 8 and respectively 13, 14 in the results for Query 1. Positions 7, 13 and 8, 14 respectively have the same database storage options – positions 7 and 8 give the best results while positions 13 and 14 give the worst.

Analyzing the two result sets we can conclude that indices provide better results when used with the hybrid storage than with the pure XML storage. The indices efficiency increases when the size of the data in the hybrid storage increases. During results analysis, we need to consider that the results are also affected from the database servers XML storage capabilities and created indices. The benchmark results are influenced from the data distribution in the datasets as well as the SQL queries construction. The statistical data available in HShreX facilitates and aids our decision where to put HShreX annotations and SQL indices and thus HShreX assists us in fast storage construction.

Shredded:

```
SELECT WDDDSO.processor, WDP.shrex_pid
FROM workflow_dataflow_processors WDP,
        workflow_dataflow_datalinks WDD,
        workflow_dataflow_datalinks_datalink WDDD,
     workflow_dataflow_datalinks_datalink_sink WDDDSI,
     workflow_dataflow_datalinks_datalink_source WDDDSO,
     workflow_dataflow_processors_processor  WDPP
WHERE WDP.shrex_pid = WDD.shrex_pid
     AND WDPP.shrex_pid = WDP.shrex_id
     AND WDDD.shrex_pid = WDD.shrex_id
      AND WDDDSI.shrex_pid = WDDD.shrex_id
     AND WDDDSO.shrex_pid = WDDD.shrex_id
     AND WDDDSI.processor = WDPP.name
     AND WDPP.name = module_name;
```

Native:

```
SELECT dataflow.COLUMN_VALUE
FROM workflow_dataflow,
XMLTable('for $i in //dataflow,
                $p in $i/processors/processor/name,
                $d in $i/datalinks/datalink
              where  $p = $d/sink/processor
                and $p = $name
              return if(exists($d/source/processor))
                   then < Details >{$d/source/processor}{$i}</ Details >
                   else< Details ><processor>null</processor></ Details >'
              PASSING module_name AS "name", "dataflow") dataflow;
```

Listing 3. Input query

B. *Provenance data*

Scientists in the natural sciences use workflow management systems to facilitate their work in development, management and execution of data and computation intensive experiments. These experiments can be described as a sequence of connected activities, where the output of one activity is an input to the following. The experiments are run multiple times with different configurations of parameters where results are produced by each execution. The results obtained from different configurations as well as the configurations itself are highly important for the scientists. They are used for further analysis of the results, as well as sharing and reusing experimental data. The scientific workflow management systems offer tools for describing experiments (workflows), keep track at each step of their evolution and execution and store the resulting data products in an easily reproducible format. Efficient methods for searching and retrieving large amounts of data are essential for the scientists in their everyday work, in this context.

Each workflow system stores the relevant information in its own internal format; however most of them can export the workflows and execution data as XML. Hence usually the workflows are shared in the community in the XML format corresponding to a particular vendor XML schema. Using our tool HShreX and the particular schema the user can obtain a fast overview of the data and to create a storage corresponding to its requirements.

Shredded:

```
UPSTREAM_QUERY (module_name)
SELECT WDDDSO.processor AS preceeding_name, WDP.shrex_pid
FROM workflow_dataflow_processors WDP,
        workflow_dataflow_datalinks WDD,
        workflow_dataflow_datalinks_datalink WDDD,
        workflow_dataflow_datalinks_datalink_sink WDDDSI,
        workflow_dataflow_datalinks_datalink_source WDDDSO,
        workflow_dataflow_processors_processor WDPP
WHERE WDP.shrex_pid = WDD.shrex_pid
          AND WDPP.shrex_pid = WDP.shrex_id
          AND WDDD.shrex_pid = WDD.shrex_id
          AND WDDDSI.shrex_pid = WDDD.shrex_id
          AND WDDDSO.shrex_pid = WDDD.shrex_id
          AND WDDDSI.processor = WDPP.name
          AND WDPP.name = module_name;
      RETURN UPSTREAM_QUERY (preceeding_name)
```

Native:

```
UPSTREAM_QUERY (module_name)
SELECT dataflow.COLUMN_VALUE AS preceeding_name
FROM workflow_dataflow,
XMLTable('for $i in //dataflow,
                $p in $i/processors/processor/name,
                $d in $i/datalinks/datalink
             where  $p = $d/sink/processor
               and $p = $name
             return if(exists($d/source/processor))
                  then <Details>{$d/source/processor}{$i}</Details>
                  else <Details><processor>null</processor></Details>'
             PASSING module_name AS "name", "dataflow") dataflow;
    RETURN UPSTREAM_QUERY (preceeding_name)
```

Listing 4. Upstream query

There is a set of specific queries that are highly important for scientists in this domain. These are the input and output queries [26] (discover the activities immediately before and after a particular activity), the upstream and downstream queries [26] (discover the activities before and after a particular activity in the whole workflow), activity details query [26] (shows all parameters for an activity), different version queries [27] (show permanent and temporary changes in the workflows structures). Since the input and upstream queries are foundations for more complex queries in this area, they were chosen to show the capabilities and benefits from our tool.

In this experiment, we use a set with approximately 600 files generated by the Taverna [28] workflow management system. Each file contains at least one workflow. So the total dataset contains around 1100 workflows, since an activity can be a workflow on its own. Studying the corresponding schema and the dataset (guideline 3), and taking into account

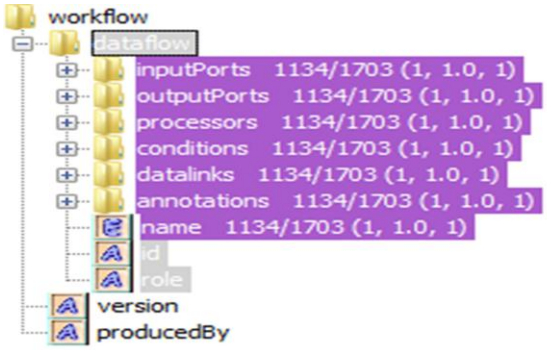

Figure 10. Statistical data for the *dataflow* element

an additional knowledge for the selected queries (guideline 5), we selected the *dataflow* and the *processors* elements to apply the HShreX annotation **maptoxml**. The structure of the dataset is shown in Figure 10. The *dataflow* element represents the whole experiment (workflow) and the *processors* element represents the activities in it. The *datalinks* element is another important element – it shows how the activities are connected and it has a significant place in the domain specific queries. As described in the previous section, the **maptoxml** annotation will force our tool to store the corresponding parts of the XML as pure XML. When the annotation is applied to the elements the data is represented in pure native and respectively in hybrid storage models.

We have implemented the input and upstream queries (Listing 3 and 4) as SQL functions and executed them against the three storage models. The input query retrieves the activities that immediately precede a given activity. First each workflow is checked for presence of the activity (identified by its name) and when the activity is available the *datalink* elements are explored in order to find the immediately preceding activities. In the upstream query, all activities that precede a given activity in a workflow are retrieved. In order to find all preceding activities in the workflow the input query is executed for every previously selected activity until the beginning of the workflow is reached. Since the upstream query is highly dependent on the structure of the workflow we select and evaluate the query performance for two different activities, which are at

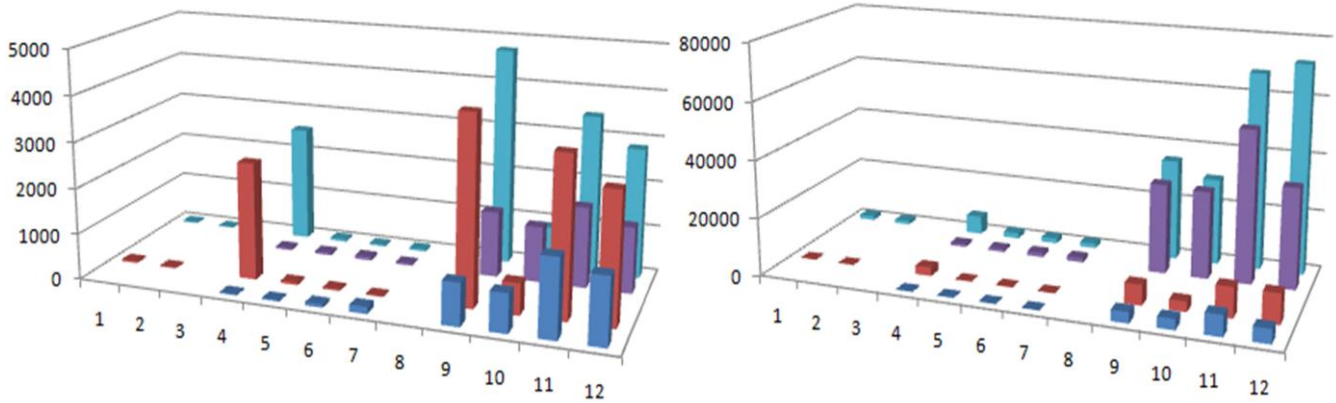

Figure 11. Performance [ms] for the input query (left) and the upstream query (right) where: ■ short path (to the activity regarding the workflow beginning) with index, ■ short path without index, ■ long path (to the activity regarding the workflow beginning) with index and ■ long path without index

different distance from the first activity in the workflow. The two queries were executed on the same database servers and with similar XML storage options as the queries discussed above. Initially, they were executed without any optimizations and then using the statistics in our tool proper indices was created.

The results from the executions of the input and the upstream queries are shown on the left and respectively on the right side in Figure 11. The results on the first two rows (dark blue and red color) on both figures are obtained using an activity close to the beginning of the workflow, while the results on the other two rows are obtained using a distanced (from the beginning) activity. Analogically to the presentation of the bioinformatics data results, the equivalent positions on the 'X' coordinate in both of the charts correspond to equivalent condition sets of database storage options. The results from positions 1 and 2 correspond to a fully shredded storage, positions 4 – 7 correspond to a hybrid storage and positions 9 – 12 correspond to a pure native XML storage. Positions 4 – 7 use the same conditions sets of database storage options as positions 9 – 12, however the HShreX annotation is applied to different elements.

Here, as well as in the bioinformatics data results, the query execution times are fastest in the shredded storage model. The queries performance for the hybrid storage (positions 4 – 7 in both result sets) is very good, sometimes even comparable with the performance in the fully shredded storage. The structure of the queries, where the joins are mainly between shredded relations, has a particular influence on these results. It should be noted that the indices lead to significant improvement in the input query execution time for position 4. Nevertheless, some positions (for instance position 7 on the left figure) in the hybrid storage show a small loss of performance, when the indices are created. A careful examination shows that the query execution times for these positions, obtained in the first run after creating the indices, are very slow. Each query was run ten times per condition set (the average time is shown here) in order to reduce the influence of other processes running on the server at the same time. Although the other execution times for the mentioned positions are very fast, these extreme values influence the average of the results. The query performance, in the pure native XML storage, is usually improved when the indices are created, except the position 10 in both result sets, where the indices lead to worse performance. As expected, due to the upstream query definition, the execution times are dependent on the distance to the selected activity regarding the beginning of the workflow (first two rows on the right figure against the next two rows).

Since each scientific workflows management system has different internal representation of the data, the sharing and reusing of already existing workflows is limited. Thus our current work in the domain of the scientific workflows is orientated towards development of common data representation, optimized for domain specific queries (some of them were mentioned earlier). Since the scientific workflows are best described as directed acyclic graphs, our data model is naturally based on a graph model. Most of the domain specific queries are related to graph traversal and other graph operations as well. In this context we have implemented the input and upstream queries in our specially designed model. A comparison between their performance in the graph model and the shredded storage obtained with our tool HShreX is presented in Table 2. Note that although the HShreX shredded storage is not optimized according to the requirements in the scientific workflows domain, it has comparable performance with the storage specially designed for the domain requirements.

Query	HShreX storage	Special storage
Input (short path)	43	50
Input (long path)	24	17
Upstream (short path)	183	81
Upstream (long path)	1463	330

Table 2: Comparison between query execution times [ms] in HshreX and in our specially designed storage

VII. Related Work

The work presented in this article combines ideas from several different 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 [5][7][14][18]. The intention with these approaches is to create efficient storage for the XML data. The resulting data model is often hard to understand and is usually hidden from the user via an interface providing automatic query translation of XQuery into the model.

The other related area is hybrid XML storage for relational databases. The vendors offer different underlying representation for the XML type, in some cases it is a byte representation of the XML, in other cases it is some kind of shredding of the XML data [8][16][29][30]. In addition, 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 become time consuming.

Interesting work [31] has addressed the question of properties of XML data and generating statistical and comparative measurements of XML datasets. However, this work concentrates on overall measures of properties of the dataset and does not consider the more detailed statistical measurements that we have found most useful in our work.

Other related work is found within database optimization [32][33]. Query optimization can rely on statistics of data and query use for fine tuning their performance [9][34]. However, these statistics are often dependent on the internal database representation instead of based on the original dataset as is necessary for our work. It would be interesting to include these measurements in our work to see whether they could give added value to our indicators.

VIII. CONCLUSION AND FUTURE DEVELOPMENT

The extended HShreX tool is very promising and our tests confirms that our tool is very useful for aiding in storage design. Using the tools and statistics improves the evaluation process and makes it possible to compare a high number of alternative hybrid database designs. The statistical analysis gives powerful insight in the structure of data and aids not only in how to shred the data but also in how to construct indices. The added details and experiments, which extend this paper from [1], verify these results.

In particular, we want to compare our set of measurements with the more advanced statistical methods used in [34]. The final goals would be to use the measure to provide suggestions of beneficial hybrid data models for the end user, to further automate the process of storage design. To reach this goal it is crucial to have access to series of data with specific properties to fine tune the indicators and tests. Also for this issue we have made a first solution for generating data with desired properties [23], which can be integrated into our tool.

One bottleneck with our method is that hybrid data models are very complex to query due to the mix of query languages. We are currently using SQL/XML, however, if we consider a user that want to work on the data as if it was XML, this is not feasible. Options are automatic query translations from XQuery to the defined model or to provide a higher level query language for the user.

Another very interesting question is hybrid storage solutions with several DB architectures as a backend, for instance pure native XML databases or specialized databases for graphs or RDF storage. This becomes particularly important for applications where parts of the data contain RDF code or represent graphs as is the case for many system biology standards. We have previously evaluated different combinations [10][13] and would like to include also these options in the HShreX Framework.

ACKNOWLEDGMENT

We acknowledge the financial support from the Center for Industrial Information Technology and the Swedish Research Council. We are also grateful to Juliana Freire for support and fruitful discussions regarding this work and for Mikael Åsberg for implementation work on the HShreX tool.

REFERENCES

[1] L. Strömbäck, V. Ivanova, and D. Hall, Exploring Statistical Information for Applications-Specific Design and Evaluation of Hybrid XML storage., *Proceedings of the International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2011)*, Jan. 2011, pp. 108-113.

[2] AR. Schmidt, F. Waas, M. Kersten, MJ. Carey, I. Manolescu, and R. Busse, XMark: A Benchmark for XML Data Management, *Proceedings of the International Conference on Very Large Databases (VLDB 2002)*, Aug. 2002, pp. 974–985.

[3] BB. Yao, MT. Özsu, and N. Khandelwal, XBench Benchmark and Performance Testing of XML DBMSs, *Proceedings of the IEEE International Conference on Data Engineering (ICDE 2004)*, Mar. 2004, pp. 621-633.

[4] L. Strömbäck, Possibilities and Challenges Using XML Technology for Storage and Integration of Molecular Interactions, *Proceedings of the International Workshop on Database and Expert Systems Applications*, Aug. 2005, pp. 575-579, doi:10.1109/DEXA.2005.154.

[5] Strömbäck, D. Hall, M. Åsberg, and S. Schmidt, Efficient XML data management for systems biology: Problems, tools and future vision, *International Journal on Advances in Software, vol. 2(2-3)*, 2009, pp. 217-233, Invited contribution.

[6] D. Floresco and D. Kossmann, Storing and Querying XML Data using an RDMBS, *IEEE Data Engineering Bulletin, vol. 22(3)*, 1999, pp. 27-34.

[7] B. Bohannon, J. Freire, P. Roy, and J. Siméon, From XML Schema to Relations: A Cost-Based Approach to XML Storage, *Proceedings of the IEEE International Conference on Data Engineering (ICDE 2002)*, Feb.-Mar. 2002, pp. 64-75, doi:10.1109/ICDE.2002.994698.

[8] H. Georgiadis and V. Vassalos, XPath on steroids: Exploiting relational engines for XPath performance, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2007)*, Jun. 2007, pp. 317-328, doi:10.1145/1247480.1247517.

[9] T. Grust, J. Rittinger, and J. Teubner, Why Off-the-Shelf RDMBMSs are Better at Xpath Than You Might Expect, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2007)*, Jun. 2007, pp. 949-958, doi:10.1145/1247480/1247591.

[10] L. Strömbäck and D. Hall, An evaluation of the Use of XML for Representation, Querying, and Analysis of Molecular Interactions, *In: T. Grust et. al. (Eds) Current Trends in Database Technology – International Conference on Extending Database Technology 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web*, Mar. 2006, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4254, 2006, pp. 220-233, doi:10.1007/11896548_20.

[11] I. Mlynkova, Standing on the Shoulders of Ants: Towards More Efficient XML-to-Relational Mapping Strategies, Proceedings of the *International Workshop on Database and Expert Systems Applications*, Sep. 2008, pp. 279-283, doi:10.1109/DEXA.2008.16.

[12] MM. Moro, L. Lim, and Y-C. Chang, Schema Advisor for Hybrid Relational-XML DBMS, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2007)*, Jun. 2007, pp. 959-970, doi:10.1145/1247480-1247592.

[13] L. Strömbäck and S. Schmidt, An Extension of XQuery for Graph Analysis of Biological Pathways, *Proceedings of the International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2009)*, Mar. 2009, pp. 22-27, doi:10.1109/DBKDA.2009.16.

[14] S. Amer-Yahia, F. Du, and J. Freire, A Comprehensive Solution to the XML-to-Relational Mapping Problem, *Proceedings of the ACM International Workshop on Web Information and Data Management*, Nov. 2004, pp. 31-38, doi:10.1145/1031453.1031461.

[15] D. Barbosa, J. Freire, and AO. Mendelzon, Designing Information-Preserving Mapping Schemes for XML, *Proceedings of the International Conference on Very Large Databases (VLDB 2005)*, Aug.-Sep. 2005, pp. 109-120.

[16] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton, Relational databases for querying XML documents: Limitations and opportunities, *Proceedings of the International Conference on Very Large Databases (VLDB 1999)*, Sep. 1999, pp. 302-314.

[17] L. Strömbäck and J. Freire, XML Management for Bioinformatics Applications, *Computing in Science and Enineering, vol.13*, Sep./Oct., 2011, pp. 12-23, http://doi.ieeecomputersociety.org/10.1109/MCSE.2010.100.

[18] F. Du, S. Amer-Yahia, and J. Freire, ShreX: Managing XML Documents in Relational Databases, *Proceedings of the International Conference on Very Large Databases (VLDB 2004)*, Aug.-Sep. 2004, pp. 1297-1300.

[19] L. Strömbäck, M. Åsberg, and D. Hall, HShreX – A Tool for Design and Evaluation of Hybrid XML storage, *Proceedings of the*

International Workshop on Database and Expert Systems Applications, Aug.-Sep. 2009, pp. 417-421, doi:10.1109/DEXA.2009.33.

[20] B. Aranda et al., The IntAct molecular interaction database in 2010, *Nucleic Acids Research*, Oct. 2009, pp. 1-7, doi:10.1093/nar/gkp878.

[21] The UniProt Consortium The Universal Protein Resource (UniProt), *Nucleic Acids Research, vol. 36(1)*, 2008, pp. D190-D195, doi:10.1093/nar/gkm895.

[22] L. Runapongsa, JM. Patel, HV. Jagadish, Y. Chen, and S. Al-Khalifa, The Michigan Benchmark: Towards XML Query Performance Diagnostics, *Information Systems, vol. 31(2)*, Apr. 2006, pp. 73-97, doi:10.1016/j.is.2004.09.004.

[23] D. Hall and L. Strömbäck, Generation of Synthetic XML for Evaluation of Hybrid XML Systems, *In: M. Yoshikawa et al. (Eds) Database Systems for Advanced Applications 15th International Conference, International Workshops: GDM, BenchmarX, MCIS, SNSMW, DIEW, UDM, Apr. 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6193*, 2010, pp. 191-202, doi:10.1007/978-3-642-14589-6_20.

[24] M. Hucka et al., The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, *Bioinformatics, vol. 19(4)*, 2003, pp. 524-531, doi:10.1093/bioinformatics/btg015.

[25] Reactome – a curated knowledgebase of biological pathways http://reactome.org 25.09.2010.

[26] L. Moreau et al., Special issue: the first provenance challenge, *Concurrency and Computation: Practice and Experience, vol. 20(5)*, 2007, pp. 409–418.

[27] C. Scheidegger, D. Koop, H. Vo, J. Freire, and C. Silva, Querying and creating visualizations by analogy, *IEEE Transactions on Visualization and Computer Graphics 13(6)*, 2007, pp. 1560–1567.

[28] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, Taverna: a tool for building and running workflows of services, *Nucleic Acids Research*, 2006, Volume 34, Issue Web Server issue, pp. 729-732.

[29] K. Beyer, F. Özcan, S. Saiprasad, and B. Van der Linden, DB2/XML:Designing for Evolution, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2005)*, Jun. 2005, pp. 948-952, doi:10.1145/1066157.1066299.

[30] M. Rys, XML and relational Management Systems: Inside Microsoft SQL Server 2005, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2005)*, Jun. 2005, pp. 958-962, doi:10.1145/1066157.1066301.

[31] I. Sanz, M. Mesiti, G. Gurrini, and RB. Llavori, An entropy based characterization of the heterogeneity of XML collections, *Proceedings of the International Workshop on Database and Expert Systems Applications*, Sep. 2008, pp. 238-242, doi:10.1109/DEXA.2008.55.

[32] G. Gottlob, C. Koch, and R. Pichler, Efficient Algorithms for processing Xpath Queries, *ACM Transactions on Database Systems*, vol. 30, No 2, Jun. 2005, pp. 444-491, doi:10.1145/1071610.1071614.

[33] J. McHugh and J. Widom, Query optimization for XML, *Proceedings of the International Conference on Very Large Databases (VLDB 1999)*, Sep. 1999, pp. 315-326.

[34] J. Freire, JR. Haritsa, M. Ramanath, P.Roy, and J. Siméon, StatiX: making XML count, *Proceedings of the ACM SIGMOD International conference on Management of data (SIGMOD 2002)*, Jun. 2002, pp. 181-191, doi:10.1145/564691.564713.