XomatiQ: Living With Genomes, Proteomes, Relations and a Little Bit of XML

Sourav S Bhowmick
Nanyang Technological University
School of Computer Engineering
Singapore 639798
assourav@ntu.edu.sg

Pedro Cruz   Amey Laud
Helixense Pte Ltd
73, Science Park Drive
02-05 Science Park 1
Singapore 118254
{pcruz, alaud}@helixense.com

Abstract

In this paper, we describe a system called XomatiQ to address the problem of integration, querying and correlation of biological data. XomatiQ is an integral part of the genomics Research Network Architecture *(gRNA). The gRNA provides the development environment in which new applications can be quickly written, and the deployment environment in which they can systematically avail of computing resources and integrate information from distributed biological data sources. Specifically, XomatiQ is build on top of the* Data Hounds *component. The* Data Hounds *transforms data from various sources to XML format and loads them into tuples of relational tables in a standard commercial DBMS. The* XomatiQ *provides capability for querying XML data using the underlying relational engine. XomatiQ has been fully implemented using Java.*

1 Introduction

The problem of integrating and querying heterogenous bioinformatics resources is of immense importance. Most bioinformatics research relies on a combination of a wide set of related public and private databases [15]. These databases contain annotated genomic sequence information [1, 8], or the results of new high-throughput techniques such as microarray experiments [22], or *curated* databases containing carefully scrutinized existing research, systematically compiled by domain experts [35]. It is useful [12] to correlate these databases with those containing medical records [33], information on disease [26], databases on references to literature [7], databases containing information on the properties of chemicals and their molecular structure [46]. Therefore, useful queries do span through a multitude of databases in the categories mentioned above.

However, integrating and correlating biological data from disparate sources is a challenging problem due to the nature of biological data. Most data pertaining to genomics and molecular biology is characterized by being highly evolving and semi-structured [20]. Moreover, genomic data available from public and private repositories is voluminous [36], highly heterogeneous, and not consistently represented. The heterogeneity results from our evolving understanding of complex biological systems; the numerous disparities in modeling biological systems across organisms, across tissue, in different environments and over time; and disparities across the scientific community in their understanding of these systems. Moreover, there continues to be a lack of common standards in the representation of biological data [4]. In fact, the extremely rapid pace of change of the field - in terms of science, technology, and business - discourages the development of stable standards for interoperation and formation of commercial companies that can deliver specialized software of reasonable prices [37]. This results in crippling incompatibilities among software tools and databases. Scientists find it slow, cumbersome, and labor-intensive to establish connections across information resources that fuel scientific research.

Recognizing that integrating and correlating data across disparate biological data sources is essential, researchers and commercial companies have tried various approaches to allow interoperability of formerly incompatible resources. These approaches can be broadly categorized into three types as indicated in [37]. The first approach provides a uniform Web interface to various databases and analysis tools [25, 41, 42, 44]. These systems usually use CGI scripts or Java servlets to execute queries against databases, to call analysis programs, or to search file-based data repositories. However, these systems permit little customization by the user and are not designed to allow "plug and play" of components. The second approach focuses on formulating complex declarative queries that span multiple heterogenous databases [14, 23, 24]. However, these systems require "on-the-fly" mappings from representations in source databases to a global schema, or forces users to express queries in the local schemas of source databases. The third approach focuses on component integration and standardization. The objective is to package heterogenous software tools and databases as components adhering to standard, well-defined interfaces, according to which information can be exchanged [9, 29, 37]. This approach enables interoperation and allows components implementing the same abstract interface to be interchanged. However, its main stumbling block is the standardization of interfaces.

1.1 Overview of our Approach

We take a *warehousing* approach. In this approach we harness data from various sources and transform them to an uniform model and store them locally. Queries for correlating relevant information from multiple databases are then performed on these locally stored data. Apart from the obvious advantages of performance, flexibility, and availability, warehousing biological data has a positive impact on the *privacy* of the relevant information retrieval. Users interested in information from biological sources are typically at the mercy of web-sites and service providers that offer public query interfaces [49] to the data. Moreover, many service providers have the rights to monitor and inspect queries formulated on their data repositories [45]. In the bioinformatics domain, there is a pressing need to prevent service providers from predicting the objective behind the formulation of a set of meaningful queries by a user.

In this paper, we focus on our warehousing approach of integrating and querying biological data from disparate sources. Specifically, Data Hounds and XomatiQ was designed and developed at HeliXense Pte Ltd, Singapore [3] to achieve this goal. XomatiQ is build on top of the *Data Hounds* component. In the *Data Hounds* component, we first generate a relational schema. Second, we transform data from various sources to XML format by creating valid XML documents of the corresponding data. Third, we parse XML documents created from the previous step and load them into tuples of relational tables in a standard commercial DBMS (in our case, Oracle 9i). The *XomatiQ* component supports a visual XML-based query interface. Through the interface, DTD structures of stored XML documents are displayed, and users can formulate queries by clicking the relevant data elements and entering conditions. Such queries are specified in a language similar to XQuery [11] and are transformed into SQL queries over the corresponding relational data. The results are formatted as XML documents (if necessary) and returned back to the user or passed to another application for further processing.

It is worth mentioning that recently there has been considerable research effort made by the database community on storing and querying XML documents using relational database systems. In this context, several techniques proposed for storing XML data in relations and translating XML queries into SQL queries over those relations [21, 34, 40, 48]. This has definitely inspired us to take a "XML-to-relational" approach. To the best of our knowledge, this is the first attempt in the bioinformatics industry to store and query biological data using an "all-XML" system. Observe that XomatiQ creates an illusion of a fully XML-based data management system as the underlying relational system remains hidden from the users.

Note that XomatiQ and Data Hounds are an integral part of the *Genomics Research Network Architecture* (gRNA) [31] which was designed and developed at HeliXense Pte Ltd, Singapore [3] to address the challenges for developing new bioinformatics applications. The gRNA provides a *development environment* and a *deployment framework* in which to maintain distributed warehouses, and to model, query and integrate disparate sources of data. It avails of the flexible data representation provided by XML along with the established reliability and powerful data management capabilities of an RDBMS. It also provides the mechanism to account for disparities in the nomenclature and representation of data, as well as the semi-structured nature of most types of data. In the same context, the architecture strives to prevent the introduction or imposition of new biases. By providing useful domain-specific abstractions and functionalities, and related visual interfaces, the gRNA effectively minimizes the time, cost and degree of human expertise typically required in the bioinformatics application-development process. The reader may refer to [31] for detailed discussion on the architecture and application of the gRNA.

The rest of the paper is organized as follows: We discuss the Data Hounds component of the gRNA in Section 2. Then, in Section 3 we present an XML-based query language for querying the warehoused data (XomatiQ component). We discuss related work in the area of integrating and querying life sciences data in Section 4. Finally, the last section concludes the paper.

2 Data Hounds

Although a large amount of biological data can be downloaded from the Web, integration of them from heterogeneous sources and preprocessing of the integrated data for posing useful queries are challenging tasks. In this section, we discuss how heterogeneous biological data from disparate sources are harnessed and stored locally. Typically, there are two important considerations when working with biological databases:

1. The conversion of a wide variety of different databases into one consistent format.
2. The ability to download and integrate the latest updates

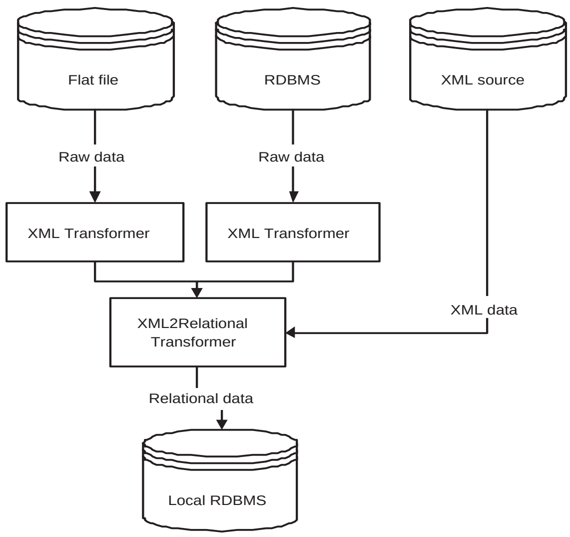


Figure 1. Data Hounds.

to any database without any information being left out or added twice.

DataHounds are built with these two considerations in mind. It enables us to efficiently warehouse data locally. It contains third-party programmable mechanisms to facilitate the transport, wrapping and conversion of remotely located relational tables and flat-files into local warehouses that are conducive to semi-structured data management. Specifically, DataHounds are developed to provide a consistent way of converting existing biological databases into XML format. As XML is easily understood by applications, this conversion will yield benefits in terms of application development and implementation in the gRNA [31]. Figure 1 shows the functional overview of the Data Hounds. We now describe the functional components (*XML-Transformer* and *XML2Relational-Transformer*) of the Data Hounds in turn.

2.1 XML-Transformer

Most of the publicly accessible databases of interest are accessible through internet protocols such as FTP (File Transfer Protocol) and HTTP (Hypertext Transfer Protocol). Typically, updates to these databases are also provided through pre-designated locations through the same protocols. Data Hounds must select one of these designated modes for physically transporting data from its original source.

As biological databases are rarely exactly the same in the structure, converting each one requires a special transformer. The *XML-transformer* module harnesses data (flat files, relational tables, XML data etc.) from disparate biological sources and then converts it into XML form (if required). This involves specifying a set of DTDs for every kind of data in the remote biological sources, and a mapping of the attributes in this data to elements and attributes in the DTDs. The justification of converting remote data to XML format is as follows:

- First, XML data is self-describing (describes the data itself). Hence, it is possible for a program to interpret the data. This means that a program receiving an XML document can interpret it in multiple ways, can filter the document based upon its content, can restructure it to suit the application's need, and so forth.
- Second, increasingly, genomic data is being provided in XML format. New bioinformatics applications also increasingly take XML as input, making it essential that existing non-XML data be easily converted to XML. For instance, the SRS data integration platform [25, 49] now includes an XML parser, allowing XML data to be quickly incorporated into SRS. Several public domain and proprietary XML databanks such as the INTERPRO databank [5] are already in existence.

We now illustrate the XML-transformer module with an example. ENZYME [2] is a repository of information related to the nomenclature of enzymes. It is provided by the ExPASy (Expert Protein Analysis System) Molecular Biology Server of SIB (Swiss Institute of Bioinformatics). It is primarily based on the recommendations of the Nomenclature Committee of IUBMB (the International Union of Biochemistry and Molecular Biology) and it describes each type of characterized enzyme for which an EC (Enzyme Commission) number has been provided. This is a repository that is made available as flat file off its FTP site. It is useful to anyone working with enzymes and helps in the development of computer programs involved with the manipulation of metabolic pathways. The ENZYME database contains the following data for each type of characterized enzyme for which an EC number has been provided: (1) EC number (2) Recommended name (3) Alternative names (if any) (4) Catalytic activity (5) Cofactors (if any) (6) Pointers to the SWISS-PROT entries (or entry) that correspond to the enzyme (if any) and (7) Pointers to the disease (s) associated with a deficiency of the enzyme (if any). Figure 2 shows a sample entry from the ENZYME database. Each entry in the database data file is composed of lines. Different types of lines, each with its own format, are used to record the various types of data which make up the entry. The general structure of a line is given in Figure 3. The line types, along with their respective line codes, are listed in Figure 4. Note that some entries do not contain all of the line types, and some line types occur many times in a single entry. Each entry must begin with an identification line (ID) and end with a terminator line (//).

Writing the XML-transformer module for the ENZYME database involves specifying a DTD for the data in the flat-

```
ID   1.14.17.3
DE   Peptidylglycine monooxygenase.
AN   Peptidyl alpha-amidating enzyme.
AN   Peptidylglycine 2-hydroxylase.
CA   Peptidylglycine + ascorbate + O(2) = peptidyl(2-hydroxyglycine) +
CA   dehydroascorbate + H(2)O.
CF   Copper.
CC   -!- Peptidylglycines with a neutral amino acid residue in the
CC   Penultimate position are the best substrates for the enzyme.
CC   -!- The enzyme also catalyzes the dismutatation of the product to
CC   glyoxylate and the corresponding desglycine peptide amide.
PR   PROSITE; PDOC00080;
DR   P10731, AMD_BOVIN ; P19021, AMD_HUMAN ; P14925, AMD_RAT ;
DR   P08478, AMD1_XENLA; P12890, AMD2_XENLA;
//
```

Figure 2. A Sample Entry in the ENZYME Database.

Characters	Content
1 to 2	Two character line code. Indicates the type of information contained in the line
3 to 5	Blank
6 up to 78	Data

Figure 3. Structure of a line.

Code	Type	Description
ID	Identification	Begins each entry, 1 per entry
DE	Description	>=1 per entry
AN	Alternate name(s)	>=0 per entry
CA	Catalytic activity	>=0 per entry
CF	Cofactor(s)	>=0 per entry
CC	Comments	>=0 per entry
DI	Diseases	>=0 per entry
PR	Cross-references to PROSITE	>=0 per entry
DR	Cross-references to SWISS-PROT	>=0 per entry
//	Termination line	Ends each entry

Figure 4. Line types and their codes.

file and a mapping of attributes from the flat-file to elements and attributes in the DTD. Figure 5 shows the DTD of the ENZYME database generated based on the entries in the database. Note that the root element hlx_enzyme consists of a db_entry corresponding to each entry of the ENZYME data (from ID to //). Each db_entry consists of:

- $enzyme\_id$ (mapped from ID.)
- $enzyme\_description$ (mapped from DE; at least one occurrence.)
- $alternate\_name\_list$ (mapped from AN; consists of zero or more alternate_name's.)
- $catalytic\_activity$ (mapped from CA; optional element; can be more than one.)
- $cofactor\_list$ (mapped from CF; consists of zero or more cofactor's.)
- $comment\_list$ (mapped from CC; consists of zero or more comment's.)
- $prosite\_reference$ (mapped from PR; zero or more; each $prosite\_reference$ will have $prosite\_accession\_number$ as an attribute.)
- $swissprot\_reference\_list$ (mapped from DR; consists or zero or more reference's; each reference will have two attributes - $name$ and $swissprot\_accession\_number$.)
- $disease\_list$ (mapped from DI; consists of zero or more disease's. Each disease will have mim_id (MIM catalogue number of the disease) as an attribute.)

Once the DTD is generated the XML-Transformer reads the input flat file and produce a set of XML documents. The rules for XML formation is defined by the DTD we have developed and the line codes in the input flat file. The algorithm looks for ID, DE, AN, CA, CF, CC, DI, PR, DR respectively in the lines (keeping in mind that they may or may not be present) and output to the XML file as per the rules specified in the DTD. Since in our DTD, we have defined hlx_enzyme to consist of only one element db_entry, our algorithm produces one XML file per entry in the sample data. Figure 6 shows the XML version of the sample entry in Figure 2.

2.2 XML2Relational-Transformer

The second component in Figure 1, i.e., the *XML2Relational-transformer* parses XML data generated from the previous step and loads them into tuples of relational tables in a standard commercial DBMS (in our

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT hlx_enzyme (db_entry)>
<!ELEMENT db_entry (enzyme_id,enzyme_description+, alternate_name_list,
catalytic_activity*, cofactor_list, comment_list, prosite_reference*,
swissprot_reference_list, disease_list)>
<!ELEMENT enzyme_id (#PCDATA)>
<!ELEMENT enzyme_description (#PCDATA)>
<!ELEMENT alternate_name_list (alternate_name*)>
<!ELEMENT alternate_name(#PCDATA)>
<!ELEMENT catalytic_activity (#PCDATA)>
<!ELEMENT cofactor_list (cofactor*)>
<!ELEMENT cofactor (#PCDATA)>
<!ELEMENT comment_list (comment*)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT prosite_reference (#PCDATA)>
<!ATTLIST prosite_reference
        prosite_accession_number NMTOKEN #REQUIRED>
<!ELEMENT swissprot_reference_list (reference*)>
<!ELEMENT reference (#PCDATA)>
<!ATTLIST reference name CDATA #REQUIRED
        swissprot_accession_number NMTOKEN #REQUIRED
>
<!ELEMENT disease_list (disease*)>
<!ELEMENT disease (#PCDATA)>
<!ATTLIST disease
        mim_id CDATA #REQUIRED
>
```

Figure 5. DTD of the ENZYME Database.

case, Oracle 9i). For example, the XML data generated from the ENZYME database is transformed to relational tuples and stored in the RDBMS. Our database is designed based on consideration of the following issues:

- *Generic schema:* The XML documents are modeled in our system by a generic relational schema, which is independent of any particular instance of XML data.
- *Preservation of document order:* As XML data is ordered and commercial RDBMS are optimized for unordered relational model, we have implemented a mechanism so that document order is captured in the RDBMS. We achieve this by treating order as a data value. Note that this is particularly important for reconstruction of the XML documents from the tuples as well as for evaluation of order-based functionalities of XQuery (such as `BEFORE` and `AFTER` operators, *range* predicates, and numeric literals).
- *Sequence and non-sequence data:* Researchers typically look into sequences to find patterns or motifs within that may be of biological significance. Non-sequence data is usually annotations that may result from computational analysis of sequences, expert annotation describing the function and location, comments, authors, known and predicted interactions with other components etc. Types of queries posed on DNA or protein sequences are generally different from those posed on non-sequence data. Hence, we differentiate between the sequence and non-sequence data in our database.
- *String and numeric data:* It is also necessary to distinguish between string and numeric data in XML data. Note that all these data appear as strings in the biological sources. For example, several databases store annotations that are of numeric type such as the length of a sequence, it location in a chromosome, homology scores, coordinates of an atom, etc.. Common queries often require to compare these numeric types across large datasets in order to establish the relationships of the biological entities.
- *Keyword-based search:* Finally, our design supports efficient keyword-based searches in the relational database system.

Note that details of the relational schema are proprietary and therefore beyond the scope of this paper.

Since an XML document is an example of semistructured data, an obvious question is – why not use semistructured database to store this data in lieu of relational DBMS. While this approach will clearly work, we transform the XML data to relational tuples for the following reasons:

- First, though there has been increasing research in XML data management, development of a full-fledged commercial XML data management system for managing very large volumes of XML data is still in its

```xml
<?xml version="1.0" encoding="UTF-8"?>
<hlx_enzyme>
   <db_entry>
      <enzyme_id>1.14.17.3</enzyme_id>
      <enzyme_description>Peptidylglycine monooxygenase.</enzyme_description>
      <alternate_name_list>
         <alternate_name>Peptidyl alpha-amidating enzyme</alternate_name>
         <alternate_name>Peptidylglycine 2-hydroxylase</alternate_name>
      </alternate_name_list>
      <catalytic_activity>
         Peptidylglycine + ascorbate + O(2) = peptidyl(2-hydroxyglycine) +
      </catalytic_activity>
      <catalytic_activity> dehydroascorbate + H(2)O </catalytic_activity>
      <cofactor_list>
         <cofactor>Copper</cofactor>
      </cofactor_list>
      <comment_list>
         <comment>
            Peptidylglycines with a neutral amino acid residue in the
            Penultimate position are the best substrates for the enzyme.
         </comment>
         <comment>
            The enzyme also catalyzes the dismutatation of the product to
            glyoxylate and the corresponding desglycine peptide amide.
         </comment>
      </comment_list>
      <prosite_reference prosite_accession_number="PDOC00080"/>
      <swissprot_reference_list>
         <reference name="AMD_BOVIN" swissprot_accession_number="P10731"/>
         <reference name="AMD_HUMAN" swissprot_accession_number="P19021"/>
         <reference name="AMD_RAT" swissprot_accession_number="P14925"/>
         <reference name="AMD1_XENLA" swissprot_accession_number="P08478"/>
         <reference name="AMD2_XENLA" swissprot_accession_number="P12890"/>
      </swissprot_reference_list>
      <disease_list/>
   </db_entry>
</hlx_enzyme>
```


Figure 6. XML Data of Figure 2.

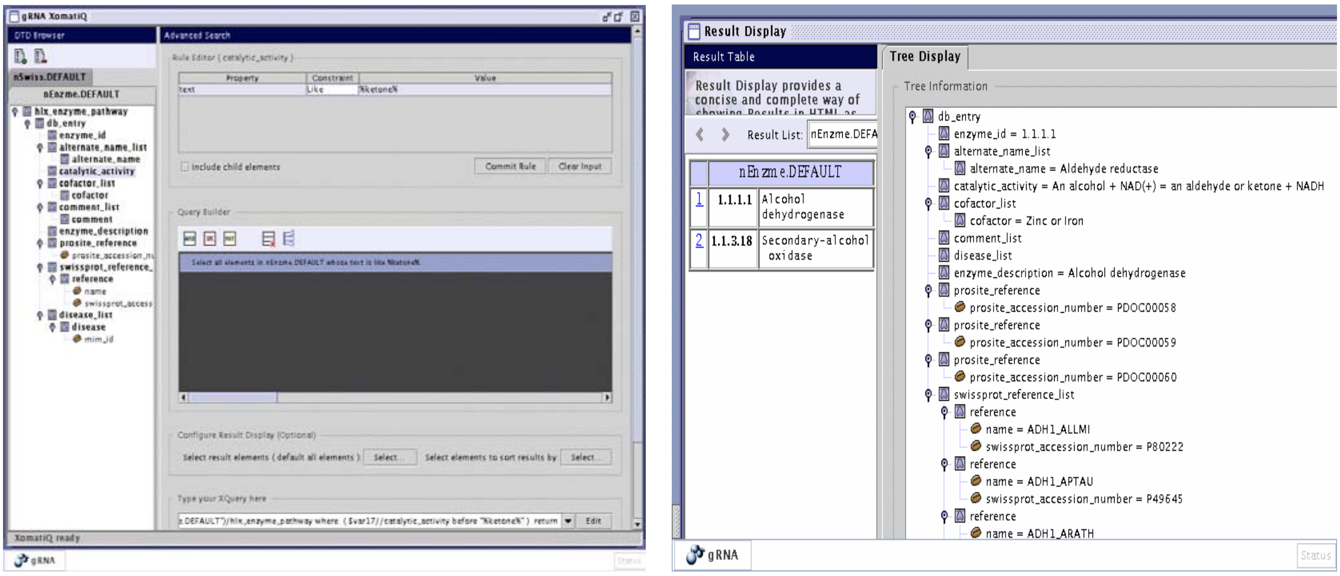
infancy. On the other hand, twenty years of work invested in relational database technology has made it commercially very successful and ensured simplicity, stability and expressiveness. As relational databases are prevalent in most commercial companies, no additional costs are incurred. Furthermore, RDBMSs are capable of storing and processing large volumes of data (up to terabytes) efficiently.

- Second, state-of-the-art query optimization and query processing algorithms still rely on relational model. However, algorithms for equivalent efficiency and ease-of-use, but designed for XML, are more difficult to find [34]. In fact, special-purpose XML query processors are not mature enough to process large volumes of data [34].
- Third, recent research [21, 32, 34, 40, 48] demonstrates that it is indeed possible to use standard commercial relational database systems to store, index and evaluate powerful queries over XML documents. It is possible to take XML queries, data sets, and schemas and process them in relational systems without any manual intervention. This means that all of the power of relational database systems can be brought to bear upon solving the XML-query problem.
- Finally, by using a standard commercial relational database systems, we can exploit the concurrency access and crash recovery features of an RDBMS.

Data Hounds also have the capability to update the data in the warehouse based on the changes to the remote sources. Once the changes have been committed to the local warehouse, the Data Hounds sends out triggers to related applications, indicating changes to the warehouse.

3 Querying Biological Data

Since data from remote biological sources are first transformed into XML documents, it should be possible to query the contents of these documents. One should be able to issue queries over a set of XML documents to extract, synthe-



(a) XomatiQ GUI. (b) Query results.

Figure 7. Querying ENZYME database.

size, correlate, and analyze their contents. However, when XML data is stored in relational systems, it makes it difficult to formulate query by looking at the relational schema as the original structure of the XML documents are not well reflected by the relational schema. In fact, the XML structure of the underlying data makes it conducive to using an XML query language. Furthermore, the possibility of using a standard XML query langauge has a commercial advantage. Using an XML-based query language at the interface level in our system allows us to hide the proprietary relational schema that we do not want to disclose.

The XomatiQ component provides an XML query language to facilitate the querying of one or more distributed or local warehouses managed within the gRNA. The syntax for the query language is based on the W3C XQuery specification [11]. The XQuery language is still work in progress, and our query language is valid with respect to the syntax and semantics defined as of June 2001. Advances in standardization process may slightly change the semantics of the language. Specifically, we use the FLWR expressions (for-let-where-return) of XQuery in XomatiQ. The *for-let* clause makes variables iterate over the result of an expression or binds variables to arbitrary expressions, the *where* clause allows specifying conditions on the variables, and the *return* clause can construct new XML element as output of the query.

Due to the pressing need in bioinformatics, the XomatiQ extends the syntax of the XQuery specification by making provisions for simple keyword-based queries, similar to those found in web-based search engines. The extension simply allows the user to specify keywords that are implicitly meant to be located close to one another in the same XML document. We have implemented a subset of the features of XQuery which we believe is sufficient to provide all the functionalities and components required by biologists to analyze their data in a meaningful way [38]. We now discuss the main steps of a query scenario in XomatiQ.

3.1 Query Formulation

There are two ways of posing a query in XomatiQ. Expert users that are familiar with the syntax and semantics of the query language can formulate the query in text form, while novice users can use an user-friendly visual query interface which can guide the user to formulate queries. We believe the GUI-based query formulation technique has greater importance in bioinformatics as we do not expect the biologists to be well-versed with the syntax and semantics of the query language.

Figure 7(a) depicts the screen dump of the visual interface of XomatiQ. It consists of two panels. The left panel displays the DTD structure of the XML documents to be queried. The right panel is for users to specify the target attributes to be retrieved and the conditions to be satisfied by the retrieved data elements. User can click on the elements in the DTD portion to select them and enter the conditions at the right hand panel. Once the user completes the formulation of the query, it can be transformed to the text form by clicking on the "Translate Query" button.

```
FOR     $a IN   document("hlx_embl.inv")/hlx_n_sequence,
        $b IN   document("hlx_sprot.all")/hlx_n_sequence
WHERE           contains ($a, "cdc6", any)
        AND     contains ($b, "cdc6", any)
RETURN  {
                $b//sprot_accession_number,
                $a//embl_accession_number
        }
```

Figure 8. Keyword-based XomatiQ Query.

```
FOR     $a IN   document("hlx_enzyme.DEFAULT")/hlx_enzyme
WHERE           contains($a//catalytic_activity, "ketone")
RETURN  {
                $a//enzyme_id,
                $a//enzyme_description
        }
```

Figure 9. Sub-tree Query.

Specifically, XomatiQ provides the user the following three modes to formulate queries visually:

- *Keyword-based search mode*: This is used to perform simple keyword searches in the selected database(s). This kind of search can be used to help users to "browse" the relevant databases, or as a filter for formulating more structured queries. For example, we may specify a query to search for the cell division cycle protein "cdc6" through all entries in the EMBL and Swiss-Prot databases and return the accession numbers of the relevant documents in these databases. The corresponding text format of this query is shown in Figure 8.
- *Sub-tree search mode*: This is used to perform simple text searches in the selected sub-tree(s) in the database(s). Note that this mode is useful if the user wants to limit the search to selected sub-trees within the data source(s) instead of the entire data source(s). For example, Figure 7(a) specifies a search for the keyword "ketone" contained in the catalytic activity element of the ENZYME database. We wish to return the enzyme id and the description of those enzymes which match this query. The corresponding text format of the query is shown in Figure 9. The results of the query executed against a partially warehoused ENZYME database is shown in Figure 7(b). Clicking on each enzyme id (left panel) displays the corresponding XML document in the right panel. Note that XomatiQ allows the user to specify complex conjunctive and disjunctive constraints in the where clause using logical operators.
- *Join query mode*: This mode allows us to correlate or join multiple databases. Let us illustrate this with an example. Consider the warehoused ENZYME database as introduced in Section 2. Suppose we have also warehoused data from EMBL. Suppose the user wants to specify a query that finds all the EMBL entries from the division invertebrates that have a direct link to enzymes characterized in the ENZYME database. In effect the query performs a join operation between the database references. The query checks if the attribute $qualifier\_type$ has the value "EC number" and if so compares the value of the element $qualifier$ with the $enzyme\_id$ from the ENZYME database. The query results must return the EMBL accession numbers and descriptions that satisfies the query. The screen shot of the query is depicted in Figure 10. Observe that the left and rightmost panels display the DTDs of the EMBL and ENZYME databases. The join condition is specified in the middle panel by specifying the joining elements. The corresponding text format of the query is shown in Figure 11.

3.2 XQ2SQL-Transformer

Once a user formulates a query using XomatiQ, it is fed to the *XQ2SQL-transformer* module which rewrites the query to corresponding one or more SQL queries over the relational generic schema. These queries are evaluated against the database where the data is stored in relational tables. Note that the design and implementation of this step is inspired by the recent research done in [32, 34, 40, 48]. As the relational schema of our system is proprietary, discussion related to the transformation to SQL queries is beyond the scope of this paper.

We have created a set of indexes by meticulous analysis of the query plans generated by the Oracle's query optimizer. Our experience shows that majority of XomatiQ queries which are important in bioinformatics domain can be evaluated efficiently over relational database systems.

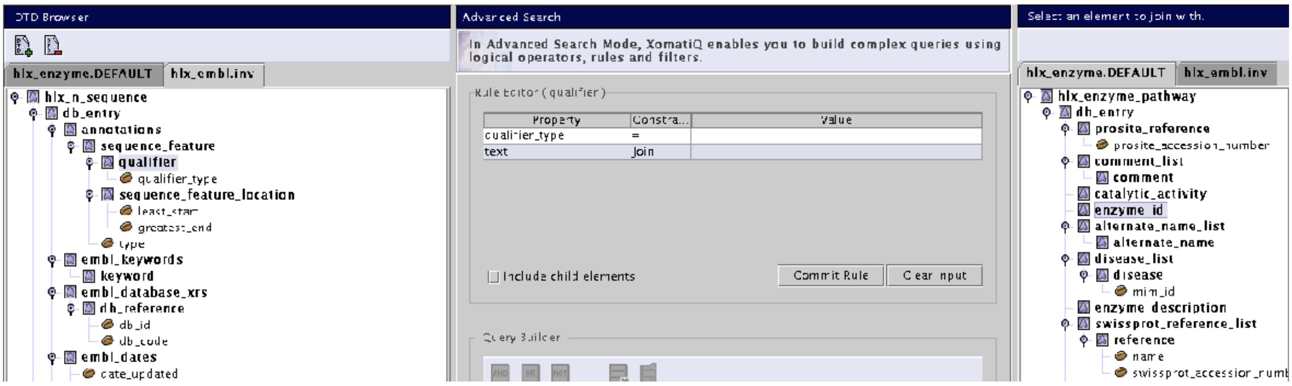


Figure 10. Screen shot of the join query.

```
FOR      $a IN   document("hlx_embl.inv")/hlx_n_sequence/db_entry,
         $b IN   document("hlx_enzyme.DEFAULT")/hlx_enzyme/db_entry
WHERE            $a//qualifier[@qualifier_type = "EC number"] =$b/enzyme_id
RETURN   {
                 $Accession_Number = $a//embl_accession_number,
                 $Accession_Description = $a//description
         }
```

Figure 11. Text version of the join query.

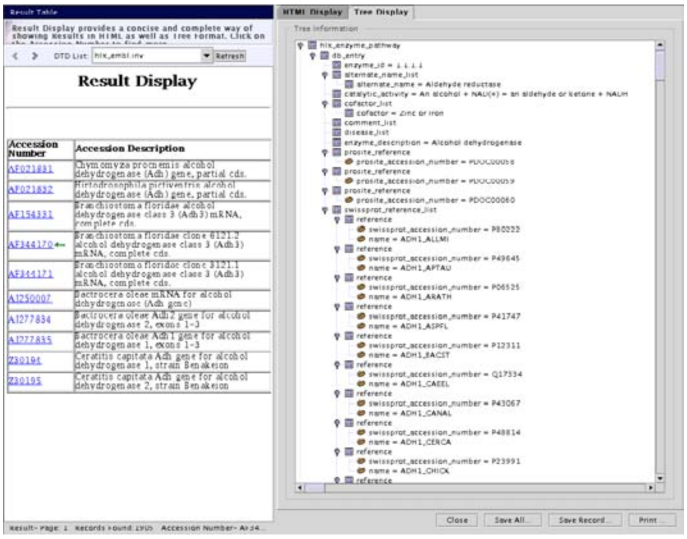


Figure 12. Results of the join query.

3.3 Relation2XML-Transformer

Upon successful execution of the SQL queries, instead of producing as result an XML document they produce the equivalent instance of the generic relational schema. The resultant tuples are either displayed in a simple table format or treated by a tagger module, that structure them into the desired XML format of the result and presented to the user. Note that the tagger module is inspired by the recent work done in [39]. Observe that we provide an option to display the results in XML format or a simple table format because in bioinformatics the user may not always wish to view the results in an XML format. Note that reconstruction of entire large XML document from the tuples is expensive compared to the query processing time in the RDBMS.

Figures 7(b) and 12 depict the screen dumps of the visual interface of XomatiQ query results. It consists of two panels. The left panel displays the results in a table or XML structure format. The right panel displays the tree structure view of the documents satisfying the query. User can click on the elements in the left panel to view the corresponding document in the right panel.

Results returned by XomatiQ can be fed into a variety of applications designed to further process the data with predictive techniques or they can be used to construct contextual reports with several levels of information that can, for example give an integrated view of the annotations to a genome stored in distinct databases in a graphical interface. Applications under the gRNA framework [31] leverage on the ability of XomatiQ to fetch data dispersed in several databases (different formats, different types of biological information).

4 Related Work

SRS: SRS (Sequence Retrieval System) [25, 49] is a fast, popular and effective data retrieval system for indexed flat-file text data sources that also provides simple filtering and linking capability. It was designed to retrieve data directly from formatted text files, a widely used format in biological databases such as EMBL [1] and Swiss-Prot [10]. In the SRS approach, formatted text files are indexed to define classes using its own scripting language called *Icarus* which are then queried through a web based interface. Its data model is structured text, i.e., tag/value pairs, and therefore its query language is limited to index lookups and following predefined links between data sources. Recently, and recognizing the advantages of the self-descriptive XML format, the SRS indexes XML files into the same meta-representation as the one used to index the flat files. However, SRS was developed almost ten years ago, when the overall requirements were considerably less. Moreover, it does not manipulate and query data using the power of a database management system. Compared to an XML query language (such as XQuery [11]), *Icarus* is less expressive in querying XML data. Searches are only permitted on pre-defined indexed attributes whereas XomatiQ permits searches on attributes at any level, and joins may be performed as needed between two or more data sources.Finally, unlike the Data Hounds, SRS does not automatically update the local data with respect to the source regularly, that can be particularly painful with large and frequently updated data sources such as Genbank.

Kleisli: The Kleisli [16, 19, 23] system transforms and integrates heterogeneous data sources using a complex object data model and CPL, a powerful query language inspired by work in functional programming languages. Kleisli provides connectivity to various data sources in bioinformatics and genomics. In Kleisli databases are queried by constructing functions that access the databases in their native format; in gRNA, component databases are warehoused, transformed to a consistent format, and queried using an XML-based query language. Furthermore, the CPL is a functional language that is relatively uncommon and difficult to learn as far as either biologists or programmers are concerned.

Discovery Link: IBM's Discovery Link [27, 28] is an SQL-based heterogeneous database integration system based on the Garlic research prototype [17] and the DB2/UDB DataJoiner [6] federated database management system for relational databases. Our approach is different in the sense that we take a warehousing approach. Furthermore, we use the power of XML query language to pose complex meaningful queries. Note that such queries often requires more sophisticated conditions than the SQL query langauge can express, for example, regular expression pattern matching.

Merck & Co. Approach: Merc & Co. [47] have created a set of applications using Perl and Java in combination with XML technology to install biological sequence databases into an Oracle RDBMS. An user-friendly interface using Java has been created for database query. Unlike XomatiQ, they do not exploit the power of an XML query language. Queries are based on simple conditions and does not support pattern matching, joins etc..

TINet: Target Informatics Net (TINet) [24] is a data integration system developed at GlaxoSmithKline, based on the Object-Protocol Model (OPM) [18] multidatabase middleware system of Gene Logic Inc. TINet follows primarily the federated model supplemented by limited use of the warehousing approach. In this approach data sources are not transformed or loaded into a single storage format, but are accessed in their native formats as required. It uses a Multidatabase Query System (MQS) to query and explore multiple heterogeneous data sources that have OPM views. Queries against the MQS are expressed in the OPM multidatabase query language (OPM-MQL) and the query results are returned as OPM data structures. TINet permits the output of a query to be returned in XML format. However, it does not take the advantage of an XML-based query language. Hence, it is much more difficult to impose complex queries using TINet. Furthermore, it does not have a user-friendly GUI to pose complex queries which makes query formulation a difficult task for biologists.

TAMBIS: The Tambis system (Transparent Access to Multiple Biological Information Sources) provides a view of heterogeneous biological data sources by means of the TaO (TAMBIS Ontology), an ontology of biological terminology based on a description logic. Semantic knowledge-representation is the main focus of TAMBIS, rather than the syntactic integration which is the primary focus of the XomatiQ and the other systems surveyed.

5 Conclusions

We have demonstrated how the Data Hounds and XomatiQ provides an efficient and systematic mechanism for warehousing and querying biological data. They also represents a successful strategy for consistently using XML for representing and querying biological data. Specifically, the Data Hounds component transforms data from various sources to XML format and load them into tuples of relational tables in a standard commercial DBMS. The XomatiQ component provides the capability for querying XML data using the underlying relational engine. Our system has been fully implemented using Java. We use Oracle 9i as the underlying RDBMS.

By providing practical ways of dealing with the heterogeneity and semistructured nature of biological data, the gRNA provides a practical backbone and an engineering approach to guide the development of genomics centric tools. We see the gRNA (supported by Data Hounds and XomatiQ) as an efficient way to integrate heterogeneous, semistructured, distributed biological data.

Acknowledgment We would like to thank other members of the team at HeliXense for making the Data Hounds and XomatiQ a reality. This includes Dr. D T Singh, George Rajesh, Joshy George, Joyce John, Pathik Gupta, Sandeep Prakash, Vivek Vedagiri, Naresh Agarwal, Reina Angelica, and Yusdi Santoso.

References

[1] European Molecular Biology Lab Nucleotide Database. `http://www.ebi.ac.uk/embl/`.

[2] ENZYME Database. `http://www.expasy.org/enzyme/`.

[3] Helixense Pte Ltd. `www.helixense.com`.

[4] Interoperable Informatics Infrastructure Consortium (I3C). `www.i3c.org`.

[5] INTERPRO Databank. `www.ebi.ac.uk/interpro/`.

[6] DataJoiner. `www.ibm.com/software/data/datajoiner/`.

[7] Medline. `http://www4.ncbi.nlm.nih.gov/PubMed/`.

[8] National Center for Biotechnology Information's Genbank. `http://www.ncbi.nlm.nih.gov/Genbank/`.

[9] NetGenics Inc.. `http://www.netgenics.com`.

[10] The SWISS-PROT Protein Knowledge Base. `http://www.expasy.org/sprot/`.

[11] XQuery 1.0: An XML Query Language. `www.w3.org/TR/2001/WD-xquery-20011220`.

[12] R. B. ALTMAN, T. KLEIN. Challenges for Biomedical Informatics and Pharmacogenomics. *Annual Rev Pharmacol Toxicol* 42:113-133, 2002.

[13] D. E. BASSETT, M. B. EISEN, M. S. BOGUSKI. Gene Expression Informatics – It's All in Your Mine. *Nature Genetics*, Vol 21, pp. 51–55, 1999.

[14] P. G. BAKER, A. BRASS, S. BECHHOFER ET AL. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *In Proceedings of 6th International Conference on Intelligent Systems for Molecular Biology*, pp. 25–34, 1998.

[15] A. D. BAXEVANIS. The Molecular Biology Database Collection: 2002 Updates. *Nucleic Acids Res.*, Vol 30, pp. 1–12, 2002.

[16] P. BUNEMAN, S. B. DAVIDSON, K. HART, G. OVERTON, L. WONG. A Data Transformation System for Biological Data Sources. *Proceedings of the 21st International Conference on Very Large Databases (VLDB 1995)*, 1995.

[17] M. J. CAREY, L. HASS, P. M. SCHWARZ ET AL. Towards Heterogeneous Multimedia Information Systems: the Garlic Approach. *Proceedings of the International Workshop on Research Issues in Data Engineering (RIDE '95)*, Taipei, Taiwan, 1995.

[18] I. A. CHEN, V. M. MARKOWITZ. An Overview of the Object-Protocol Model and OPM Data Management Tools. *Information Systems*.

[19] S. Y. CHUNG, L. WONG. Kleisli: A New Tool for Data Integration in Biology. *Trends Biotechnol.*, 17(9), pp. 351–355, 1999.

[20] K. DECKER, S. KHAN, C. SCHMIDT, D. MICHAUD. Extending a Multi-Agent System for Genomic Annotation. *Cooperative Information Agents* pp 106-117, 2001.

[21] A. DEUTSCH, M. FERNANDEZ, D. SUCIU. Storing Semistructured Data with STORED. *Proceedings of SIGMOD Conference*, 1999.

[22] S. S. DWIGHT, M. A. HARRIS, K. DOLINSKI, ET AL. *Saccharomyces* Genome Database (SGD) provides Secondary Gene Annotation Using the Gene Ontology (GO). *Nucleic Acids Res.*, Vol 30, pp. 69–72, 2002.

[23] S. B. DAVIDSON, P. BUNEMAN, G. OVERTON, L. WONG ET AL. BioKleisli: Integrating Biomedical Data and Analysis Package. *Bioinformatics: Databases and Systems*, Kluwer Academic Publishers, pp. 201–211, 1999.

[24] B. ECKMAN, A. KOSKY, L. A. LARCOCO, JR. Extending Traditional Query-based Integration Approaches for Functional Characterization of Post-genomic Data. *Bioinformatics*, 17(7)587:601, 2001.

[25] T. ETZOLD, A. ULYANOV, P. ARGOS. SRS: Information Retrieval System for Molecular Biology Data Banks. *Methods in Enzymology.*, 266, 114–128, 1996.

[26] A. HAMOSH, A. F. SCOTT, J. AMBERGER ET AL. Online Mendelian Inheritance in Man (OMIM), a Knowledgebase of Human Genes and Genetic Disorders. *Nucleic Acids Res.*, Vol 30 (1), 2002.

[27] L. HASS, P. KODALI, J. RICE ET AL. Integrating Life Science Data - with a Little Garlic. *Proceedings of IEEE Symposium on Bio-Informatics and Biomedical Engineering* (BIBE 2000), Arlington, Virginia, 2000.

[28] L. HASS, P. M. SCHWARTZ, P. KODALI, J. RICE ET AL. Discovery Link: A System for Integrating Life Sciences Data. *Proceedings of IEEE Symposium on Bio-Informatics and Biomedical Engineering* (BIBE 2000), Arlington, Virginia, 2000.

[29] K. JUNGFER, G. CAMERON, T. FLORES. EBI: CORBA and the EBI Databases. *Bioinformatics: Databases and Systems*, Kluwer Academic Publishers, pp. 245–254, 1999.

[30] S. KELLEY. Getting Started with AceDB. *Briefings in BioInformatics* 1:2 pp 131-137, 2000.

[31] A. LAUD, S. S. BHOWMICK, P. CRUZ ET AL. The gRNA: A Highly Programmable Infrastructure for Prototyping, Developing and Deploying Genomics-Centric Applications. *Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002)*, Hong Kong, China, 2002.

[32] Q. LI, B. MOON. Indexing and Querying XML Data for Regular Path Expressions. *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pp. 361-370, Roma, Italy, 2001.

[33] A. MARSHALL. Laying the Foundations for Personalized Medicines. *Nature Biotech.*, 16 Sup 6–8, 1998.

[34] I. MANOLESCU, D. FLORESCU, D. KOSSMANN ET AL. Agora: Living with XML and Relational *Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000)*, Cairo, Egypt, 2000.

[35] C. A. OUZOUNIS, P. D. KARP. The Past, Present and Future of Genome-wide Re-annotation. *Genome Biology*, 3(2), 2002.

[36] T. REICHHARDT. It's Sink or Swim as a Tidal Wave of Data Approaches. *Nature*, 399(6736):517–520, 1999.

[37] A. C. SIEPEL, A. N. TOLOPKO, A. D. FARMER ET AL. An Integration Platform for Heterogeneous Bioinformatics Software Components. *IBM Systems Journal*, 40(2), pp. 570–591, 2001.

[38] R. STEVENS, C. GOBLE, P. BAKER, A. BRASS. A Classification of Tasks in Bioinformatics. *Bioinformatics*, 17(2):180-8, 2001.

[39] J. SHANMUGASUNDARAM, E. SHAKITA, R. BARR ET AL. Efficiently Publishing Relational Data as XML Documents. *Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000)*, Cairo, Egypt, 2000.

[40] J. SHANMUGASUNDARAM, K. TUFTE, C. ZHANG, ET AL. Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceedings of the 25th International Conference on Very Large Databases (VLDB 1999)*, pp. 302-314, Edinburgh, Scotland, 1999.

[41] K. SIROTKIN. NCBI: Integrated Data for Molecular Biology Research. *Bioinformatics: Databases and Systems*, Kluwer Academic Publishers, pp. 11–19, 1999.

[42] R. F. SMITH, B. A. WIESE, M. K. WOJZYNSKI ET AL. BCM Search Launcher– An Integrated Interface to Molecular Biology Data Base Search and Analysis Services. *Genome Research 6*, No. 5, pp. 454–462, 1996.

[43] I. TATARINOV, S. D. VIGLAS, K. BEYER, J. SHANMUGASUNDARAM ET AL. Storing and Querying Ordered XML Using a Relational Database System. *Proceedings of the SIGMOD 2002*, Madison, Wisconsin, 2002.

[44] R. UNWIN, J. FENTON, M. WHITSITT ET AL. Biology Workbench: A Computing and Analysis Environment for the Biological Sciences. *Bioinformatics: Databases and Systems*, Kluwer Academic Publishers, pp. 233–244, 1999.

[45] M. E. VAN EIJK, L. F. KRIST, J. AVORN ET AL. Do the Research Goal and the Databases Match? A Checklist for a Systematic Approach. *Health Policy* , 58(3):263–274, 2001.

[46] Y. WANG, J. B. ANDERSON, J. CHEN ET AL. MMDB: Entrez's 3D-structure Database. *Nucleic Acids Res.*, Vol 30 (1), pp. 249–252, 2002.

[47] G. XIE, R. DEMARCO, R. BLEVINS, Y. WANG. Storing Biological Sequence Databases in Relational Form. *Bioinformatics*, 16(3):288-289, 2000.

[48] C. ZHANG, J. F. NAUGHTON, D. J. DEWITT ET AL. On Supporting Containment Queries in Relational Database Management Systems. *Proceedings of SIGMOD* , Santa Barbara, USA, 2001.

[49] E. M. ZDOBNOV, R. LOPEZ, R. APWEILER, T. ETZOLD. The EBI SRS server-recent Developments. *Bioinformatics*, 18(2), pp. 368–373, 2002.