A new approach to protein structure and function analysis using semi-structured databases

William M. Shui, Raymond K. Wong
Stephen C. Graham, Lawrence K. Lee and W. Bret Church

School of Computer Science and Engineering
University of New South Wales
NSW 2052, Australia
School of Molecular and Microbial Biosciences
University of Sydney
NSW 2006, Australia
{wshui, wong}@cse.unsw.edu.au
{stepheng, llee0905}@mail.usyd.edu.au, b.church@biotech.usyd.edu.au

Abstract

The development of high-throughput genome sequencing and protein structure determination techniques have provided researchers with a wealth of biological data. Integrated analysis of such data is difficult due to the disparate nature of the repositories used to store this biological data and of the software used for its analysis. This paper presents a framework based upon the use of semi-structured database management systems that would provide an integrated interface for the collection, storage and retrieval of biological data from existing repositories and of biological information generated by existing analysis programs. A simple implementation that integrates information from databases and analytical programs is presented as a proof of concept.

1 Introduction

The rapid evolution of molecular biology and biochemistry over the last 30 years has fundamentally changed the way in which living systems are studied. The advent of recombinant DNA technology in the 1970s and 80s made the investigation of individual genes and their gene products (proteins) accessible to standard laboratories [43]. Rapid DNA sequencing techniques developed in the 1990s allowed for the investigation of gene sequences at the whole-organism level [29, 28], the drafts of the human genome released last year providing perhaps the best example of how far the technology has progressed [50, 17].

The investigation of biological function using only genomic sequence information has, however, proved difficult. It has long been known that function of proteins *in vivo* is intimately related to their 3-D structure. X-ray crystallography is a method by which the 3-D structures of proteins may be determined at atomic resolution and is the main method by which the 3-D structures of proteins have been determined to date (14698 of 17813 structures in the Protein Data Bank (PDB [9]) were solved by X-ray crystallography as of April 9, 2002). In recent years several initiatives have been started with the aim of determining the structures of a large number of proteins by X-ray crystallography. These structural genomics initiatives aim to address biological questions by combining this newly acquired structural information with other forms of knowledge derived from traditional biochemical techniques, from genetic sequence analysis techniques, and from rising techniques such as DNA microarray technology [45].

While many applications are available for the analysis of data from one or a small number of different biological techniques, a framework for the storage, interpretation and analysis of data from a diverse range of biological investigation techniques has been lacking. Support for investigators themselves integrating new data or data analysis algorithms with an analysis framework as the need arises has also been lacking.

This paper presents a framework based upon semi-structured databases that would allow integrated storage, processing and retrieval of biological data. The framework also supplies an automated cascading mechanism for the updating of data between mapped databases. This paper is organized as follows: related work is discussed in section 2; section 3 contains a description of how we map biological data obtained from existing repositories into more elaborate meta-data in the form of a semi-structured database and a generic framework for mapping from simple to extended semi-structured data using a rule based database transformation mechanism; one real world biological problem and an illustration of how problem could be addressed using the proposed framework is presented in section 4; section 5 presents a simple implementation as a proof of concept; and section 6 concludes this paper by summarizing our proposal.

2 Related Work

This section covers biological databases, database management systems (DBMS), XML technology, semi-structured DBMS, and software for the analysis of 3-D protein structures.

2.1 Biological Databases and Database Management Systems

2.1.1 Biological Databases

Many databases have been created for the storage of biological data. Sequence databases such as the GenBank sequence data bank [8] facilitate analyses of genomic sequence data. The Protein Data Bank (PDB, [9]) is a database that stores the 3-D co-ordinates of biological macromolecular structures solved by X-ray crystallography, Nuclear Magnetic Resonance Spectroscopy (NMR), or other techniques.

Secondary (derived) databases have been created to cater for specific research areas. Each database concentrates on addressing a particular problem of biological research. For example, the PRINTS database [7] focuses on protein families based on motifs. PROSITE [31] is

a database that focuses on the identification and classification of protein families and domains. ProDom [22] is a cluster database used to aid domain identification. SMART [41] is a database for detecting signalling, extracellular and chromatin-associated proteins.

New databases are also being created concomitantly with the development of new biochemical investigation techniques. DNA microarray technology is a tool for monitoring a multiple genes on a single chip that has been recently adopted by the research community. It gives researchers a clearer view of the simultaneous interactions that occur between the thousands of genes present within any cell. The Stanford Microarray Database (SMD) [46] stores raw and normalized data from microarray experiments.

Interpro [6] is a database that integrates data from a number of primary (experimental) and secondary (derived) biological databases. While this allows biologists an integrated means of access of these diverse databases, there is no scope for the tailoring of the databases to suit the needs of individual researchers who rarely need to consult all the information present in the database concurrently. The extreme size of a database that integrates all known sequence, structural and derived data also raises scalability and performance issues.

2.1.2 Database Management Systems

The integration of heterogeneous biological databases is difficult due to the complicated schema transformations required for each database. A further complication is that many mappings are required between components. Integrating heterogeneous databases also requires the identification of information that is implicitly or explicitly shared. A global schema should be general enough to handle all manner of heterogeneous data models. In these types of situations, the strict schema constraints of relational database model becomes problematic.

Several systems have been developed in the past fews years for utilizing data spread across several heterogeneous biological databases. However, some systems are poorly documented, hardly mention the limitation of the systems, and hence, are difficult to evaluate. These problems have later been addressed and some criteria have been proposed for evaluating such systems in terms of their scalability and architecture [38, 23, 37, 36].

ACeDB [49] is a database manager developed to handle a number of genomic databases. It uses an unstructured data model instead of using relational data model. Its schema only imposes weak constraints upon the database. Some have pointed out that the internal data structure of ACeDB is made of trees and the query language used for querying ACeDB is an object-oriented query language [14]. This limits the system to only selection of objects and pointer traversals, and the system can not perform projections or joins.

2.2 Semi-structured Database Technologies

The eXtensible Mark-up Language (XML) [11] has been adopted as a meta-language for data interchange between different data sources. XML data falls into a category called semi-structured data and it is generally described as such because it does not conform to a rigid schema [4, 13]. The self-describing nature of XML makes it a promising way to define semi-structured data.

Due to the differences in nature of semi-structured and relational databases, a re-thinking of all aspects of database system implementation have been required in order to build a semi-structured DBMS. The internal data structure of a semi-structured database is a graph rather than a set of tables. This requires new algorithms to be developed for indexing, searching, updates, and other database operations.

2.2.1 XML Standards and Query Languages

Several query languages have been recently proposed for the querying of XML documents. They differ in terms of language constructs and of expressive power. A detailed comparisons of five XML query languages has been provided by [10]. A new query language called XQuery [20] proposed by the World Wide Web Consortium (W3C) has drawn attention from the community regarding the future directions of XML query languages. XQuery has a rich set of language constructs including joins, FLWR expressions, transformations, etc. It is, however, less concise and compact than XQL [42] that was first proposed back to 1998. With respect to managing biological data such as protein structures and sequence annotations, we found that the compact XQL serves their applications well. Therefore, XQL is assumed and used to illustrate the ideas throughout this paper. An extension to XQuery to provide support for updates has been presented by [48], which is very similar to the update constructs available in the extended XQL implemented in [2, 51]. Finally, [26] presents an extensive survey on most of the issues related to semistructured and Web data [4], ranging from data models to query languages to database systems.

Various industry standards have been setup for XML data handling such as querying XML and XML path expressions. They include XPath [18], XPointer [19]. New methods such as XFilter [5] have been developed for faster filtering of XML data streaming in.

2.2.2 Semistructured Object DAtabases (SODA)

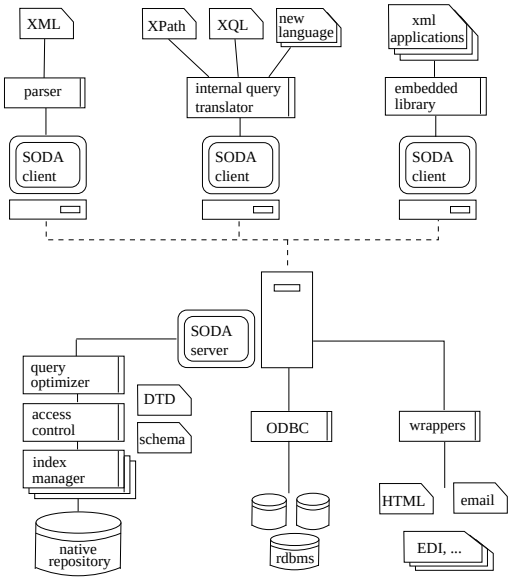

Figure 1: SODA Architecture

Recently, a database system called SODA has been implemented to handle XML databases. Figure 1 is an overview of the architecture of SODA. SODA is a client-server, semi-structured database system which is tailor-made for managing XML information. Query processing and optimization are implemented in and executed by clients while the server is responsible for storing and retrieving XML objects, handling transactions, element locks, garbage collection, database backups and recovery. A lazy XML object conversion approach is used for versioning. Different clients can therefore simultaneously work with different versions of Document Type Definitions (DTD). The novel SODA architecture facilitates several crucial features which are seldom available in other database systems. The SODA query processor is mainly located at the client side. Each query processor contains an internal query translator that maps a query from

one language into a SODA internal micro-query language. SODA therefore supports multiple query languages such as XPath expressions, XQL, and XQuery.

User interactive programs for biological analysis that require XML database access can be built by linking to the SODA client library. The library interface supports embedded query languages such as XPath, XQL, and XQuery for rapid application development. XML parser or loader is itself a database client so that multiple loaders can be run simultaneously to load multiple documents while the database is being updated concurrently by multiple users at the same time. An advanced wrapper system plays an important role in SODA as it provides a bridge between the SODA database and other data sources.

SODA server itself consists of a number of components. Each of these components is responsible for its own task and interacts with other components by means of a strictly defined interface. These components include a storage memory manager, an access control manager, a transaction manager, a page pool manager, an object access manager, and an index manager. The modular design of SODA makes it possible to choose different implementations for each component and also to fine-tune SODA according to the efficiency of various database management algorithms and strategies for specific application requirements. SODA server can interface with other relational database systems such as Oracle or Sybase through an ODBC interface. The underlying physical repository supports standard DOM and SODA Object Model (SOM). SOM provides a system-level interface to the SODA physical storage. Compression and low-level optimization are supported with meta information such as DTD and XML Schema defined by the users or automatically learned from the XML documents.

2.3 Software for the Analysis of 3-D Protein Structures

Many programs have been developed for the analysis of 3-D protein structures, so many so that a complete review here is impossible. A general overview of the algorithms available is presented in [15].

PDB files contain co-ordinates in space for each atom of a protein that was observed experimentally. Almost all other structural information useful to investigators can be derived from this 3-D co-ordinate data. An example of such information is the topology and charge-distribution of the solvent-accessible surface of a protein.

While all amino acid residues present in a protein are important in terms of defining its shape and stability, only residues present on the solvent-exposed surface of the protein are able to participate in interactions with other biological molecules. Many programs are available for the determination of which residues are accessible to solvent [34], some examples being the programs SurfNet [33], Surface (a member of the CCP4 suite of programs [16]), the Crystallography and NMR System (CNS [12]), and SASSY [35]. The program SASSY is notable as it was developed for use in high-throughput calculations. This program provides a fast and efficient method of implementing the dot counting method [47] of calculating solvent accessible surface, allowing for different resolutions depending on speed or accuracy required.

The electrostatic potentials present on the surface of a protein are important in terms of defining how the protein is recognised by other biological molecules. The calculation of charge on the surface of a protein is usually performed by solving the Poisson-Boltzmann equation for the entire surface of a molecule. Examples of programs able to perform this calculations are GRASP [40] and DelPhi [39].

The increase in power and decrease in price of graphical workstations have made the interactive visualization on a standard desktop computer possible. Many different programs are available for the visualization of protein 3-D structures. ViewerPro [3], Swiss PDB Viewer [30], Rasmol [44], PyMol [24] and Python Molecular Viewer (PMV [21]) are generalist protein 3-D structure visualization programs. PyMol and PMV are a particularly interesting programs as they are built around the Python [1] scripting engine. This makes the addition of new functionality to the software suite relatively straightforward, and makes the interfacing of external programs with these programs simple.

3 Data Mapping and Database Transformation

This section describes the proposed framework in detail. It covers the preliminary steps required for data mark-ups and the mechanisms involved in the mapping of simple XML data to more elaborate meta-data. The rule-based database transformation for creating the final expanded database is also discussed. It finally covers issues on how to use the SODA system to provide a consistent data retrieval layer.

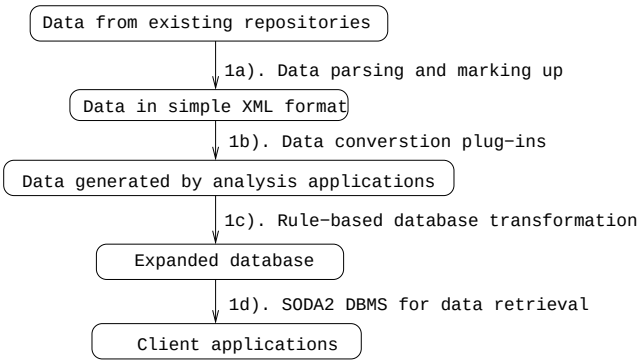

Figure 2: Overall steps involved in the data mapping process

3.1 Preliminary Data Mark-ups

As shown in figure 2 part 1a, data will be translated from existing repositories into semi-structured data. In our case, the default data format for semi-structured data is XML.

This process is unique for each different data source. Take for example the 3-D protein structural data present in a PDB file. Only the minimal set of data required for further investigation needs to be excised. In the example of a PDB file, this data could be the ”ATOM” rows which contain (amongst others) the fields residue name, residue number, atomic *x*, *y*, *z* co-ordinates, occupancy, and crystallographic temperature factor; the only data required to build a 3-D model of the protein. Further information may be deduced by using the programs presented in section 2.3.

Data transformation will be performed using a technology called LSX/LSX-T. LSX/LSX-T is a SODA-specific translation language that is essentially the reverse of the XSL/XSLT process. It is responsible for mapping external, non-XML data sources to XML so that they can be queried by SODA. The LSX-T plug-in is a data extraction and transformation tool for converting any data stream into XML. Examples of such data streams include the output from biological database queries and output from protein structure analysis programs. LSX-T maintains as much consistency as possible with the XSLT technology, providing a consistent interface for applications and users. LSX syntax looks almost identical to XSL syntax, the difference being that a few extensions have been added. A generic wrapper that can transform data from (any) one format to another is created by integrating LSX-T with XSLT.

To illustrate the basic LSX idea, consider the following short example. Assume we would like to convert the file

2PCY.pdb from the PDB web site to XML format. The PDB file looks like the following:

```
HEADER ELECTRON TRANSPORT PROTEIN(CUPROPROTEIN)...
COMPND APO-PLASTOCYANIN ($P*H 6.0)
...
REMARK 3 REFINEMENT. RESTRAINED ...
...
REMARK 4 THE STRUCTURE OF...
...
TURN 1 T1 ALA 7 GLY 10 ...
...
ATOM 1 N ILE 1 -1.453 19.554 26.971...
...
```

The corresponding LSX to perform the above transformation would be:

```
<?xml version='1.0'?>
<PDB>
<Title.Section>
<?soda_lsx:for-each-line
  group="^HEADER.{4}(.{40})(.{9}).{3}(.{4})"?>
<HEADER>
<classification>
<?soda_lsx:value-of $group[1] strip="1"?>
</classification>
<depDate>
<?soda_lsx:value-of $group[2]  strip="1"?>
</depDate>
<IDcode>
<?soda_lsx:value-of $group[3]  strip="1"?>
</IDcode>
</HEADER><?soda_lsx:end-loop?>
...
<REMARK><?soda_lsx:for-each-line
  re1="^REMARK   1(.{0,60})"?>
<REMARK.1>
<remarkNum>1</remarkNum>
<text>
<?soda_lsx:value-of $re1[1] strip="1"?>
</text>
</REMARK.1><?soda_lsx:end-loop?>
<?soda_lsx:for-each-line
  re2a="^REMARK   2 RESOLUTION.(.{37})"?>
<REMARK.2>
<remarkNum>2</remarkNum>
<?soda_lsx:for-each-line
  re2="^REMARK   2 RESOLUTION.(.{5}) ANGSTROMS."?>
<resolution>
<?soda_lsx:value-of $re2[1] strip="1"?>
</resolution>
<?soda_lsx:end-loop?>
<comment>
<?soda_lsx:value-of $re2a[1] strip="1"?>
</comment>
</REMARK.2>
<?soda_lsx:end-loop?>
<?soda_lsx:for-each-line re3="^REMARK   3(.{0,60})"?>
<REMARK.3>
<remarkNum>3</remarkNum>
<text><?soda_lsx:value-of $re3[1] strip="1"?></text>
</REMARK.3><?soda_lsx:end-loop?>
<?soda_lsx:for-each-line
  re4="^REMARK (  [4-9]| \
  [1-9][0-9]|[1-9][0-9][0-9])(.{0,60})"?>
<REMARK.4_999>
<remarkNum>
<?soda_lsx:value-of $re4[1] strip="1"?>
</remarkNum>
<text>
<?soda_lsx:value-of $re4[2] strip="1"?>
</text>
</REMARK.4_999><?soda_lsx:end-loop?>
</REMARK><?soda_lsx:end-loop?>
</Title.Section>
...
</PDB>
```

The full .lsx file is available for download at http://www.cse.unsw.edu.au/~wshui/ and the lsx program is available from the author by request. The XML output from the program looks like the following:

```
<?xml version='1.0'?>
<PDB>
<Title.Section>
<HEADER>
<classification>
ELECTRON TRANSPORT PROTEIN(CUPROPROTEIN)
</classification>
<depDate>03-NOV-83</depDate>
<IDcode>2PCY</IDcode>
</HEADER>
...
<COMPND>
<continuation></continuation>
<compound>APO-PLASTOCYANIN ($P*H 6.0)</compound>
</COMPND>
...
<AUTHOR>
<continuation></continuation>
<authorList>
T.P.J.GARRETT,J.M.GUSS,H.C.FREEMAN
</authorList>
</AUTHOR>
...
<REMARK>
<REMARK.1>
<remarkNum>1</remarkNum>
<text>
TITL   X-RAY CRYSTAL STRUCTURE ANALYSIS
OF PLASTOCYANIN
</text>
</REMARK.1>
</REMARK>
...
</Title.Section>
...
<Primary.Structure.Section>
<SEQRES>
<serName>1</serName>
<chainID></chainID>
<numRes>99</numRes>
<resName>ILE</resName>
<resName>ASP</resName>
<resName>VAL</resName>
<resName>LEU</resName>
<resName>LEU</resName>
<resName>GLY</resName>
<resName>ALA</resName>
<resName>ASP</resName>
<resName>ASP</resName>
<resName>GLY</resName>
<resName>SER</resName>
<resName>LEU</resName>
<resName>ALA</resName>
</SEQRES>
...
</Primary.Structure.Section>
<Heterogen.Section>
<FORMUL>
<compNum>2</compNum>
<hetID>HOH</hetID>
<continuation></continuation>
<asterisk>*</asterisk>
<text>42(H2 O1)</text>
</FORMUL>
</Heterogen.Section>
...
<Coordinate.Section>
<ATOM>
<serial>1</serial>
<name>N</name>
<altLoc></altLoc>
<resName>ILE</resName>
<chainID></chainID>
<resSeq>1</resSeq>
<iCode></iCode>
<x>-1.453</x>
<y>19.554</y>
<z>26.971</z>
<occupancy>1.00</occupancy>
<tempFactor>15.93</tempFactor>
<segID>2PCY</segID>
<element></element>
<charge>78</charge>
</ATOM>
..
</Coordinate.Section>
...
</PDB>
```

This data transformation layer can support arbitrary number of structured formats and different types of data. The resulting data (in XML format) is loaded directly into SODA or may be saved on file systems for further processing.

3.2 Meta-data Creation

This stage creates a set of meta-data from the original data set, the resulting meta-data being stored in the XML

database system. To avoid confusion in naming different data sets in this paper, we define the initial data sources from different repositories (pre-XML mark-ups) as *db-1*, the XML data produced from preliminary data conversion as *db-1a*, and the meta-data created from *db-1a* as *db-1b*. Starting from *db-1a*, a set of algorithms or applications can be applied to derive more specific data. The transition from *db-1a* to *db-1b* includes actions such as secondary structure detection, calculating the dihedral angles, calculating molecular surfaces, and so on. Generally, this process tries to generate as much extra information as is relevant from a single set of data values. We can view this process as a one-to-one or one-to-many mapping of *db-1a* to *db-1b*.

One could argue that it is simpler to extract these information from the existing PDB files. What we are trying accomplish, however, is the creation of an automatic cascading mechanism such that whenever *original* data (in our case, the simple XML mark-ups) change, the system automatically updates all the data derived from this original data. The advantages of this mechanism include the following:

1. Derived data can be kept consistent with the raw data.
2. At times when better algorithms for feature detection are developed, we can apply them as additional plug-ins to derive the meta-data. This makes the process process of meta-data creation more stream-lined and easier to manage.
3. Different sets of data generated by using different algorithms can be compared and examined.

The simple XML data is first parsed through a filter that uses a fast filtering mechanism to selectively apply different algorithms on data that fits descriptions from query expressions such as XPath expressions. Techniques such as XFilter allows such functionality. Schema transformation from *db-1a* to *db-1b* is also performed as part of this process.

3.3 Rule-based Database Transformation

Up until this stage we have created a process that uses one-one or one-many mapping to generate meta-data from simple marked-up data. The mapping functions used are simple algorithms or applications. Extra information may be generated by using a many-to-one or many-to-many mappings. This layer of information is generated using rule-based database transformations. The following sections describe the details of rule-based database transformations.

The rules for data mapping can be a set of algorithms, one-to-one, one-to-many, many-to-one or many-to-many data mappings, or a separate job to be processed by external programs. This step may therefore not be as efficient as a database system should be. Further optimization of this step can be explored in later experiments.

3.3.1 Data-structures

Schema transformation between one semi-structured database to another is different to the conventional relational database transformation. This is mainly due to the nature of semi-structured database. That is to say that there are no strict constraints on the database, which stands in contrast to the strict constraints enforced by relational database schema. As a result of this, all nodes that fit our criteria must be found by using an XPath expression before they can be mapped to another database.

Data-structures involved in this stage include a Directed Acyclic Graph (DAG) to handle the rule dependency and a hash table to index the XPath expression to corresponding data nodes.

Rules for transformation may or may not be independent of each other. For example, consider there are *n* rules for mapping database *a* to database *b*. Then, it is possible for rule *j* to be dependent on the completion of rule *i* and rule *k*, where ($i > 0, j > 0, k > 0, i \leq n, j \leq n, k \leq n, n > 0$, and $i \neq j \neq k$). Rule dependency must therefore be considered when implementing rule-based transformation.

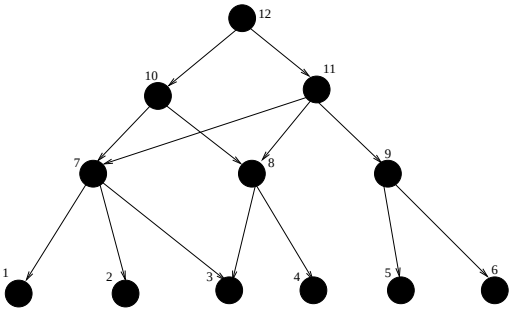

Figure 3: Rules are saved in a DAG and traversed in reverse level order.

One solution to this is to set rules into a data structure such as a DAG. Each leaf node contains a rule that is independent of others. The immediate parent nodes of the leaf nodes are contain rules that require the completion of their child nodes' rules first before executing itself. The same principle applies for their ancestor nodes. As shown in figure 3, the numbers next to each nodes indicates the order in which the graph is traversed. This is a reverse level-ordered traversal of the graph and it can guarantee the correct order of execution of each rules without violating rule dependencies.

A hash table is also used to map the hashed value of an XPath expression to its corresponding data nodes. This allows faster access to the data nodes. Consider the XPath expression $//atom[secondary = "helix"]$. The corresponding data nodes are stored in a list, ready to be processed by the mapping functions or programs.

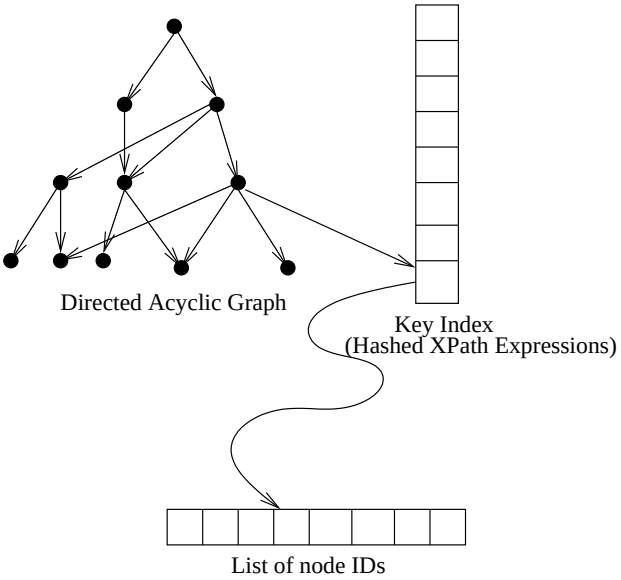

Figure 4: Rule-based database transformation

3.3.2 Transformation Algorithm

We first define the final expanded data layer as *db-1c*. It is generated through the rule-based transformation from *db-1b*.

1. Create a virtual node (as shown in figure 5) and label it *V*. Link node *V* to all of the selected nodes from *db-1b* that are to be mapped to *db-1c*.

2. Create a root node *R* for *db-1c* before any mapping starts. All the output from the mapping will be descendant of node *R*.

3. Let X = $\{e_1, e_2, e_3, ..., e_n\}$ $where$ $n > 0$ and each e is a XPath expression for a specific set of nodes in *db-1b*. For each e ∈ X, we find all the nodes from *db-1b* and save their location in a list. The list is then saved into a hash table *H* as the value with a matching key. The key is the hash value of the XPath expression e.

4. Let the DAG that stores rules be G. Traverse G in reverse level order. For each node traversed, extract the mapping function $\mathcal{F}$ from the node and its corresponding XPath expression.

5. Probe the hash table *H* using the XPath expression and find the corresponding data set $\mathcal{D}$. Apply $\mathcal{F}$ on $\mathcal{D}$ in order to generate new data set $\mathcal{D}'$.

6. Re-organize data set $\mathcal{D}'$ such that all nodes in $\mathcal{D}'$ are descendant of node *R*.

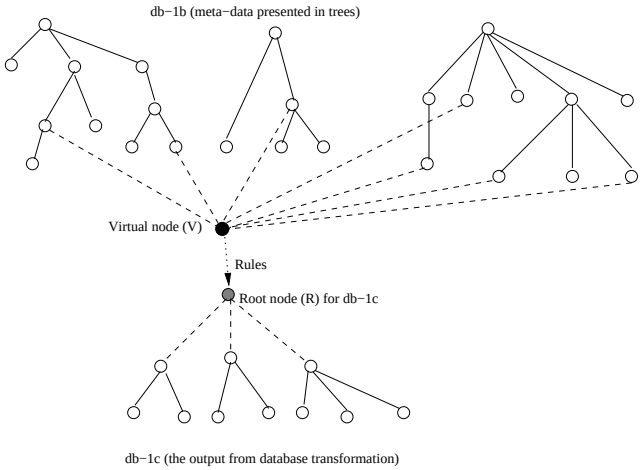

Figure 5: Rule-based database transformation

3.4 Interfacing with Data Layer

As a result of rule-based database transformation, the final data layer *db-1c*, once generated, becomes the superset of *db-1b* and *db-1a*. Users can query *db-1c* for data originally from *db-1b* and *db-1a*. Retrieving information from this data layer requires the use of SODA.

3.4.1 Querying the Database

When documents are loaded to SODA, information such as file name, owner, creation date would be added as system annotations for the document entry point (the document root node). This information is also maintained as meta-data for fast access to a particular document. The corresponding XQuery function for specifying the document name is

```
document(''protein.xml'')//Name
```

This query will find all Name elements from the document protein.xml stored in the database. Since document information is stored as annotation, regular expression matching does apply to this information. For instance, the following will find all Name elements from any XML documents stored in the database with names matching `''.*protease''`.

```
document(''.*protease\.xml'')//Name
```

Therefore, users can tailor their queries just like they can with other relational databases. Instead of writing SQL [32] functions, users can specify their own XQuery functions, note that XQuery is itself a functional programming language. This is flexible for users to develop their own system independent data analysis programs.

SODA also supports querying external data sources, either from file or URL, and store the result under a particular element node of a local database that resides within the SODA system. The following query will retrieve an XML document generated by an external CGI script and find all entries that has an entry name of "T3MO_SALTY". The result will be saved under the root node of the current database named *LocalSwissProt*.

```
//LocalSwissProt[0]!
insert(
   url(''http://au.expasy.org/sprot/getxml.cgi'')
   //Info[EntryName="T3MO_SALTY"]
)
```

The *insert* and *url* operators are system specific functions of SODA.

4 Example Scenario

This section describes a real biological problem and discusses how our proposed system can assist in solving such problem.

4.1 A Biological Problem

Enzymes are proteins that modify the rate at which chemical reactions occur within biological systems. The active site of an enzyme is defined as the region of the enzyme involved in substrate binding and subsequent catalysis. A study of the 3-D structure of an enzyme, and of the active site in particular, can assist in the elucidation of the precise catalytic mechanism of the enzyme. Detailed knowledge of enzyme catalytic mechanism is important for a number of reasons, one being that a detailed knowledge of the mechanism of an important enzyme could lead to the development of new, more potent therapeutic agents (so-called *rational* drug design).

Whilst the functional regions of many enzymes have been reported with high precision, there is still much that remains to be determined about what makes some sites specific for roles within the context of the complete enzyme. To this end it is proposed that a study of the accessibility of such sites would allow us to infer some general rules or guidelines as to accessibilities of and amino acid compositions within such sites, and therefore to their placement in the overall topology of the enzymes. We anticipate that such a study will provide insight into the structural requirements of particular activities within proteins.

As mentioned in section 1, several consortia have undertaken to determine the 3-D structures of many proteins in a high-throughput manner. Improved knowledge as to what constitutes an active site would simplify the analysis structures determined by these structural genomics consortia. An in-depth knowledge of the structural properties associated with a particular function would also aid in comparisons between families of enzymes related by similar sequence, binding partners or activity.

4.2 A New Approach

As mentioned above, there are currently no complete criteria that can be used to efficiently determine which amino acid residues form part of the functional regions of an enzyme based on 3-D structural data alone. We will use entries from the PROSITE database to help define the functional regions for proteins. LSX-T will be used to import PROSITE into the database. The resulting XML data representation of PROSITE will look like the following (due

to space constraints, we will not show the full translation for PROSITE entries).

```
<PROSITE>
 <Entry id="SUBTILASE_ASP" type="pattern">
  <AC>PS00136</AC>
  <Date>
   <Created>APR-1990</Created>
   <Updated>
    <DataUpdate>NOV-1995</DataUpdate>
    <InfoUpdate>JUL-1998</InfoUpdate>
   </Updated>
  </Date>
  <Description>
    Serine proteases, subtilase family,
    aspartic acid active site.
  </Description>
  ...
  <Pattern>
    [STAIV]-x-[LIVMF]-[LIVM]-D-[DSTA]
    -G-[LIVMFC]-x(2,3)-[DNH]
  </Pattern>
  ...

  <PDBRef>
   <Ref>1AH2</Ref>
   <Ref>1BJR</Ref>
   ...
  </PDBRef>
 </Entry>
</PROSITE>
```

These definitions of active site composition gained from the PROSITE database will be used in a bootstrapping context. It is anticipated that our analysis will provide insight into the characteristics of a the functional sites of proteins such that these human-annotated databases would no longer be required. Solvent accessibility can be retrieved for the residues in the functional region and others for comparison purposes by extracting the relevant marked-up PDB data from the database, creating PDB files to be processed, and then importing the relevant output into the database as meta-data.

The steps are examples of rule-based database transformations. A rule can be created to check how common accessibility is, and labeling certain amino acids as having rare accessibilities as appropriate. A second rule, dependent on the previous one, can be created to test postulated correlations. The results of rule checks could be then imported into the SODA database, making it available to client applications.

Charge distribution within functional sites is also important, especially in the context of binding partner recognition and can be analyzed with the help of programs such as DelPhi. Programs such as SurfNet can also be used to build descriptions of the topology of functional sites. Due to the 3-D nature of the above information and of other problems we are seeking to address, a client that interfaces with the PyMol or PMV molecular visualization packages could also be of assistance.

5 Prototype Implementation

As a proof of concept, we built a small prototype with minimal features to demonstrate the feasibility of our idea. This section describes the mains steps that are needed in setting up this prototype. We also discuss various tests done on the prototype and their results.

5.1 Solvent Accessible Surface Area (SASA)

Our prototype currently includes an implementation of solvent accessible calculations based on the program SASSY [35]. Solvent accessibility has long been recognised [34] and is now widely accepted as a useful means of describing topological and functional properties of experimental protein structures. It plays a key role in computing the solvation energy of proteins, and is central to the hydrophobic effect of protein folding [27]. Solvent accessibility has also been related to the stability of water-soluble proteins [25]. Algorithms calculating accessible surface can be slow and applications often do not provide flexibility of output. By extending the standard PDB entries in the database to include solvent accessibilities, solvent accessible surfaces will be more readily obtainable to users.

The SASA module for this prototype is a modified implementation of the SASA algorithm. This module works on XML data instead of standard PDB files. No significant change has been made to the implementation of the algorithm itself.

5.2 Main Steps

The prototype uses application programming interface (API) provided by SODA version 3 and its backend database system to maintain XML data. SODA3 provides an interface that allows developers to implement their own functional modules and then dynamically load into the system through XQuery calls that starts with an unique namespace other than those that are taken up by the W3C.

For this prototype, it consists of three main steps. They are:

1. Using the LSX module for PDB to XML conversion.
2. Load the modified implementation of the SASA module.
3. Activate the trigger system to link the SASA module to the XML data.

5.2.1 PDB to XML conversion

Writing the LSX file to convert PDB files to XML will be the first step in converting raw data files into XML data. The LSX file is then loaded into the SODA database system. This is done using a local XQuery extension call:

```
soda:load("pdb.lsx")
```

The function ***"soda:load"*** is an extension of the XQuery for SODA system. It executes the loading of a data into the database. The file ***"pdb.lsx"*** contains all the LSX codes for converting PDB files into XML format and itself is a well-formed XML document, so it can be loaded into the database. The advantage of storing the LSX files as a part of the XML database is that it can be reused to convert other PDB files into XML, and it can save time loading the actual LSX file itself.

After the loading of the LSX file, a PDB file "2PCY.pdb" is loaded into the main database. This PDB file is a record from the PDB database that descreibes the structure of the electron transport protein plastocyanin. The conversion is invoked by calling:

```
soda:lsx("pdb.lsx", "2PCY.pdb")
```

The function ***"soda:lsx"*** is another SODA defined XQuery function. It executes the LSX code to transform a given source data file into XML and load the XML file into the database system.

5.2.2 The Trigger system

Once the module has been implemented, we need to link it with the source data and the second layer meta-data. Such that whenever the source data layer is changed, the trigger system invokes SASA module to recalculate SASA values and update the meta-data layer. SODA gives users the choices of trigger at either the source data layer or the meta-data layer. That is, we can either use trigger to link SASA to the source data layer, such that whenever the source data is updated, the meta-data is also updated. The other option is less sensitive to the update of source data, this involves the trigger to link to the meta-data layer. So whenever users try to access the meta-data layer, the first thing the system will do is to resynchronize the data with the source data. This could slow down the initial read, but it uses less resources.

5.3 SASA Test On XML Data

For this experiment, we intend to build meta-data on solvent accessible surface areas for each atoms in PDB record *2PCY*. SASA module is the implicit transition rule which generates the final meta-data. After the transition, new nodes in the database can be view by running the following query.

```
let $a := document("2PCY.pdb.xml")//ATOM
return
<?xml version='1.0'?>
<2PCY.SASA.Test>
 {for $i in $a
 <ATOM>
  {$i/serial}
  {$i/name}
  {$i/sasa}
 </ATOM>}
</2PCY.SASA.Test>
```

The output of the query appears as the following:

```
<?xml version='1.0'?>
<2PCY.SASA.Test>
 <ATOM>
   <serial>1</serial>
   <name>N</name>
   <sasa>16.1304</sasa>
 </ATOM>
 <ATOM>
   <serial>2</serial>
   <name>CA</name>
   <sasa>9.05721</sasa>
 </ATOM>
 <ATOM>
   <serial>3</serial>
   <name>C</name>
   <sasa>0</sasa>
 </ATOM>
 <ATOM>
   <serial>4</serial>
   <name>O</name>
   <sasa>0</sasa>
 </ATOM>
 <ATOM>
   <serial>5</serial>
   <name>CB</name>
   <sasa>3.53869</sasa>
 </ATOM>
 <ATOM>
   <serial>6</serial>
   <name>CG1</name>
   <sasa>12.868</sasa>
 </ATOM>
 <ATOM>
   <serial>7</serial>
   <name>CG2</name>
   <sasa>0</sasa>
 </ATOM>
 <ATOM>
   <serial>8</serial>
   <name>CD1</name>
   <sasa>23.484</sasa>
 </ATOM>
 <ATOM>
   <serial>9</serial>
   <name>N</name>
   <sasa>3.82756</sasa>
 </ATOM>
 ...
</2PCY.SASA.Test>
```

6 Conclusion

In this paper we have described a framework for the analysis of biological data from a variety of data sources. The specific problem we propose to address is the analysis of the structural characteristics of enzymes. The SODA DBMS provides a consistent interface for information retrieval and storage. The proposed framework allows for the integration of data retrieved from existing repositories with information creating using existing biological analysis tools. The proposed framework is unique as it allows biological researchers to specify what means will be used to obtain the data that will be used to populate the database. Rule-based transformations provide automated integration of information from these diverse sources. The flexibility of the proposed framework will allow for the integration of data from novel repositories or algorithms as such resources become available.

References

[1] Python programming language. See http://www.python.org/.

[2] Soda3: Semistructured object database. See http://www.sodatech.com/.

[3] Viewerpro. See http://www.accelrys.com/viewer.

[4] S. Abiteboul. Querying semi-structured data. In F. N. Afrati and P. Kolaitis, editors, *ICDT*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1997.

[5] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *VLDB*, pages 53–64. Morgan Kaufmann, 2000.

[6] R. Apweiler, T. K. Attwood, A. Bairoch, A. Bateman, et al. The interpro database, an integrated documentation resource for protein families, domains and functional sites. *Nucleic Acids Research*, 29:37–40, 2001.

[7] T. K. Attwood, M. D. R. Croning, D. R. Flower, A. P. Lewis, J. E. Mabey, and P. Scordis. Prints-s: The database formerly known as prints. *Nucleic Acids Research*, 28:225–227, 2000.

[8] D. A. Benson, I. Karsch-mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. Genbank. *Nucleic Acids Research*, 28:15–18, 2000.

[9] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[10] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–79, 2000.

[11] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language(xml) 1.0 second edition w3c recommendation. Technical Report REC-xml-2001006, World Wide Web Consortium, Oct 2000.

[12] A. T. Brunger, P. D. Adams, G. M. Clore, W. L. DeLano, P. Gros, R. W. Grosse-Kunstleve, J. S. Jiang, J. Kuszewski, N. Nilges, N. S. Pannu, R. J. Read, L. M. Rice, T. Simonson, and G. L. Warren. Crystallography and nmr system (cns): A new software system for macromolecular structure determination. *Acta Crystallographica*, D54:905–921, 1998.

[13] P. Buneman. Semistructured data. In *PODS*, pages 117–121. ACM Press, 1997.

[14] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In H. V. Jagadish and I. S. Mumick, editors, *SIGMOD*, pages 505–516. ACM Press, 1996.

[15] W. C. Carter and M. R. Sweet, editors. *Macromolecular Crystallography*, volume 277. Academic Press, New York, 1997.

[16] N. . Collaborative Computational Project. The ccp4 suite: Programs for protein crystallography. *Acta Crystallographica*, D50:760–763, 1994.

[17] T. G. I. S. Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[18] W. W. W. Consortium. Xml path language (xpath) version 1.0. w3c recommendation. Technical Report REC-xpath-19991116, World Wide Web Consortium, Nov 1999. See http://www.w3.org/TR/1999/REC-xpath-19991116.

[19] W. W. W. Consortium. Xml linking language (xlink) version 1.0. w3c recommendation. Technical Report REC-xlink-20010627, World Wide Web Consortium, Jun 2001. See http://www.w3.org/TR/2001/REC-xlink-20010627/.

[20] W. W. W. Consortium. Xquery 1.0. a query language for xml. w3c working draft. Technical Report WD-xquery-20020430, World Wide Web Consortium, April 2002. See http://www.w3.org/TR/2002/WD-xquery-20020430/.

[21] S. I. Coon, M. F. Sanner, and A. J. Olson. Re-usable components for structural bioinformatics. In *Proceedings of the 9th International Python Conference*, 2001.

[22] F. Corpet, J. Gouzy, and D. Kahn. The prodom database of domain families. *Nucleic Acids Research*, 26:323–326, 1998.

[23] S. Davidson, C. Overton, and P. Buneman. Challenges in integrating biological data sources. *Journal of Computational Biology*, 2:557–572, 1995.

[24] W. L. DeLano. Pymol: An open-source molecular graphics tools. *CCP4 NEWSLETTER ON PROTEIN CRYSTALLOGRAPHY*, 40, 2002.

[25] D. Eisenberg and A. D. McLachlan. Solvation energy in protein folding and binding. *Nature*, 319:199–203, 1986.

[26] D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

[27] R. Fraczkiewicz and W. Braun. Exact and efficient analytical calculation of the accessible surface areas and their gradients for macromolecules. *Journal of Computational Chemistry*, 19:319–333, 1997.

[28] C. M. Fraser, J. D. Gocayne, O. White, M. D. Adams, et al. The minimal gene complement of mycoplasma genitalium. *Science*, 270:397–403, 1995.

[29] C. M. Fraser, J. D. Gocayne, O. White, M. D. Adams, et al. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269:496–512, 1995.

[30] N. Guex and M. C. Peitsch. Swiss-model and the swiss-pdbviewer: An environment for comparative protein modeling. *Electrophoresis*, 18:2714–2723, 1997.

[31] K. Hofman, P. Bucher, L. Falquet, and A. Bairoch. The prosite datbase, its status in 1999. *Nucleic Acids Research*, 27:215–219, 1999.

[32] A. N. S. Institute. Iso/iec 9075:1992, information technology — database languages — sql, 1992.

[33] R. A. Laskowski. Surfnet: A program for visualizing molecular surfaces, cavities and intermolecular interactions. *J. Mol. Graph.*, 13:323–330, 1995.

[34] B. Lee and F. M. Richards. The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology*, 55:379–400, 1971.

[35] L. K. Lee and W. B. Church. SASSY - rapid calculation of solvent accessible surfaces. See http://www.mmb.usyd.edu.au/˜bchurch/.

[36] S. Letovsky. Beyond the information maze. *Journal of Computational Biology*, 2:539–546, 1995.

[37] V. M. Markowitz. A strategy for database interoperation. *Journal of Computational Biology*, 2:573–583, 1995.

[38] V. M. Markowitz and O. Ritter. Characterizing heterogeneous molecular biology database systems. *Journal of Computational Biology*, 2:547–556, 1995.

[39] A. Nicholls and B. Honig. A rapid finite difference algorithm, utilizing successive over-relaxation to solve the poisson-boltzmann equation. *Journal of Computational Chemistry*, 12:435–445, 1991.

[40] A. Nicholls, K. Sharp, and B. Honig. Structure, function and genetics. *Proteins*, 11:281–296, 1991.

[41] C. P. Ponting, J. Schultz, F. Milpetz, and P. Bork. Smart: identification and annotation of domains from signalling and extracellular protein sequences. *Nucleic Acids Research*, 27:229–232, 1999.

[42] J. Robie, J. Lapp, and D. Schach. Xml query language (xql), 1998. See http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[43] R. Saiki, S. Scharf, F. Faloona, K. Mullis, G. Horn, and H. Erlich. Enzymatic amplification of beta-globin genomic sequences and restriction site analysis for diagnosis of sickle cell anemia. *Science*, 230:1350–1354, 1985.

[44] R. Sayle and E. J. Milner-White. Rasmol: Biomolecular graphics for all. *Trends in Biochemical Sciences TIBS*, 20:374, 1995.

[45] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270:467–70, 1995.

[46] G. Sherlock et al. The stanford microarray database. *Nucleic Acids Research*, 29:152–155, 2001.

[47] A. Shrake and J. A. Rupley. Environment and exposure to solvent of protein atoms, lysozyme and insulin. *Journal of Molecular Biology*, 79:351–71, 1973.

[48] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating xml. In *SIGMOD*, 2001.

[49] J. Thierry-Mieg and R. Durbin. Syntactic definitions for the acedb database manager - technical report. Technical report, MRC Laboratory for Molecular Biology, Cambridge, England, 1992.

[50] C. J. Venter, M. D. Adams, et al. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

[51] R. Wong. Soda2: An xml database management system. In *the Proceedings of XML Asia Pacific Conference*, 2000.