Proceedings of the 28th IEEE
EMBS Annual International Conference
New York City, USA, Aug 30-Sept 3, 2006

A Domain Specific Data Management Architecture for Protein Structure Data

Yanchao Wang, Rajshekhar Sunderraman and Hao Tian

Abstract— **In this paper, we propose an architecture that extends the Object-Oriented Database (OODB) system architecture by adding domain specific additional layers to manage protein structure data. The two layers introduced above OODB are Protein-QL, domain-specific query language and Protein-OODB, a domain-specific data layer. This architecture is designed specifically for the protein domain, but it is the first step in building a general Bio-OODBMS for biological applications. Three internal data types are defined for the primary, secondary, and tertiary protein structures, respectively, to simplify queries in Protein-QL. This enables the domain scientists to easily formulate data requests. We use lambda-DB as the back-end database to implement Protein-QL. Queries in Protein-QL are compiled into OQL which are then executed against the database. In order to make the underlying OODB system (lambda-DB) more powerful, we introduce additional constraints to check the integrity of protein data.**

Index Terms— **Domain Specific Object-Oriented DataBase, Protein Query Language, Lambda-DB Constraint Language**

I. Introduction

The Human Genome project has resulted in an explosive research agenda in bioinformatics that deals with extracting knowledge from protein and DNA sequences and structures. In recent years, the life science data has been growing in exponential rate due to more and more advanced experimental techniques. Obviously, accumulating literatures and experiment data in flat files cannot satisfy biologists' needs. How to store, organize and analyze the huge amount of data in a clear and efficient way is becoming a challenging issue that most biologists have to face and solve. This is why more and more database applications are built for research purpose now.

It is a fact that the most successful type of database system is the relational database. One of the major reasons is that the relational data model [5] is much simpler than others. However this advantage becomes a big issue in the life science database applications because of the lack of support for complex data types. Therefore, object-oriented databases (OODB) are getting more and more attention from biologists. Although the object-oriented nature of life science data perfectly matches the architecture of object-oriented databases, there are still a lot of problems that need to be solved in order to apply OODB methodologies to manage biological data. One major problem is that the database management systems (DBMS) of OODBs are designed for general-purpose applications and they do not have any built-in data types (ex. protein and nucleic acid) for biological research and built-in biological domain-specific functional operations.

In this paper, we propose an architecture that extends the object-oriented database (OODB) system by adding additional layers: Protein-QL and Protein-OODB above OODB. This architecture is designed specifically for protein domain, but it is a first step in building a general Bio-OODBMS for biological applications. This new architecture has four components: a Users Interface, a Protein-QL, a Protein-OODB, and an OODB. Protein-QL provides convenience for users to store, retrieve, and modify data, and defines some basic operations. Protein-OODB solves some protein data sources' problems and is used to connect Protein-QL and OODB layers. Three internal data types are defined for the primary, secondary, and tertiary protein structures respectively to simplify the queries in Protein-QL. Domain scientists can easily formulate complex requests for data without much of a learning curve.

In this paper, we also introduce more constraints to check the input data and strengthen the accuracy of data to make the underlying OODBMS (lambda-DB [2]) more powerful. Overall, this paper gives a general idea to develop a new architecture for a certain biological domain. It is easy to extend this architecture into other biological domains.

The rest of the paper is organized as follows: Section 2 presents an overview of the domain specific object-oriented database architecture for protein structures. The protein data types are described in Section 3. Section 4 discusses data constraint issues. Conclusion and future work are presented in Section 5 and Section 6.

II. Domain Specific Object-Oriented DataBase Architecture for Protein Structures

The domain specific object-oriented database architecture for protein structures is a four-layer architecture that consists of the following components: a users interface, a query language for protein structures (Protein-QL), an object-oriented database for protein structures (Protein-OODB), and an object-oriented database (OODB) layer. Figure 1 illustrates this architecture.

A. Protein-QL Layer

In this layer, we provide some basic operations to store, retrieve, and modify protein data information using Lambda-DB (similar to SELECT, INSERT, DELETE, UPDATE in

Y. Wang and H. Tian are with the Computer Science Department, Georgia State University, 34 Peachtree Street, Suite 1454, GA 30303, USA ywang17@student.gsu.edu, htian1@student.gsu.edu

R. Sunderraman is with Faculty of Computer Science Department, Georgia State University, 34 Peachtree Street, Suite 1452, GA 30303, USA raj@cs.gsu.edu

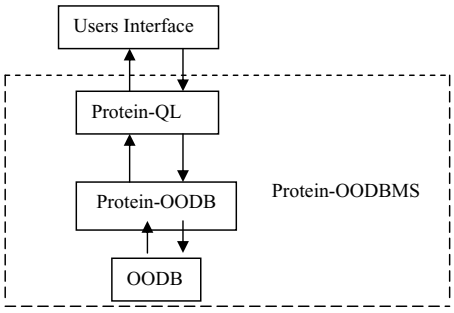

Fig. 1. Domain Specific Object-Oriented Architecture for Protein Structures

SQL), these operations can be executed on basic data types as well as on protein data types. Protein-QL defines a list of operators in protein terms, which enables domain scientists to query information in their own language without much syntactical restriction. For example, you input two proteins and use *globalAlignment(protein1, protein2)* to get sequence alignment to find the similarity of two proteins, then use alignment to get similarity. We also provide other queries such as neighbors, subparts of proteins and so on. All these protein queries are translated into the above operations so that they are independent from OODBs. Therefore, we can easily extend this architecture into other biological domains.

1) The Architecture of Protein-QL Layer:: The architecture of Protein-QL has 3 parts shown in figure 2, Protein-QL Interpreter, translator and DSDB query execution engine.

Protein-QL Interpreter accepts the domain specific query requests from users. In this layer, users can use Protein-QL grammar, Protein-QL data types, Protein-QL operators. Translator receives the queries sent from protein-QL interpreter and translates them into Object Queries (OQs) of OQL, then sends OQs to DSDB (Domain Specific DataBase) query execution engine. DSDB query execution engine executes queries and retrieves data from Protein-OODB, and sends the results back to protein-QL interpreter. Protein-OODB interacts with OODB to provide the operations on basic data types without redefining.

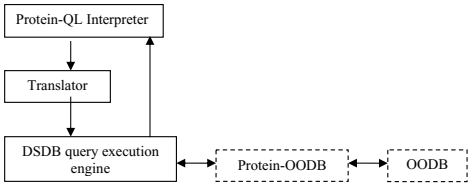

Fig. 2. The Architecture of Protein-QL

In our architecture, we implement the features of OODBMS using lambda-DB [2] to store complex protein data as object format. Lambda-DB provides Object Query Language (OQL) to optimize the queries and serve the users' requests. Our Protein-QL's queries can be easily converted to OQL's queries of Lambda-DB. In our Protein-QL, we define three protein data structures in Protein class, Primary, Secondary and Tertiary as internal data types, thus biologists can easily request the information with these domain specific formats.

2) Operators of Protein-QL Layer:: Protein-QL provides basic operators for OODB such as +, -, *, / and so on just as other query languages, it also supports domain specific operators on protein data types Primary, Secondary and Tertiary.

```
class Protein (extent Proteins key
proteinName){
attribute string proteinName;
attribute string type;
attribute string function;
attribute Vector[string] synonyms;
attribute string remark;
attribute Vector[string] domainOfPrimary;
attribute list<PrimaryPartition> Primary;
attribute list<SecondaryPartition>
   Secondary;
attribute list<TertiaryPartition>
   Tertiary;};
```

Primary Structure (We only show this one)

```
struct PrimaryPartition {
string name; short serNum;
string chainName; long elementNo;
string seq;};
```

3) Syntax of Protein-QL:: We define the BNF grammar of Protein-QL as follows:

```
Protein-QL_query ::="("<result_format>")"
                    ["("<condition>")"];
<result_format> ::= <result_item>
                     {, <result_format>}
<result_item> ::= DATATYPE_NAME |
                  DATATYPE_NAME.PROPERTY
<condition> ::= <condition_item>
                {, <condition>}
<condition_item> ::= <comparison_item> |
  <operator"("parameter{,parameter}")">|
  <operation_item>
<comparison_item>::= <operand> COMPARE
  <operand>
  {AND | OR <operand> COMPARE <operand>}
<operator> ::= DS_OPERATOR |
               NAME.DS_OPERATOR
<parameter> ::= <operand> | /*empty*/
<operation_item>::= <operand> |
            DATATYPE_NAME[.PROPERTY]
<operand>:::=NAME.PROPERTY | CONST |<exp>
<exp> ::= <exp>"+"<exp> |<exp>"-"<exp> |
          <exp>"x"<exp> |<exp>"/"<exp> |
          CONST | NAME.PROPERTY
```

We now present some examples using our Protein-QL: Example 1: Gets the secondary structure of protein named "HIV-1".

TABLE I
THE TERMINAL AND DEFINITION FOR PROTEIN-QL GRAMMAR

Terminal	Definition
NAME	{name | name ::= [_a−zA−Z][_a−zA−Z0−9]∗ }
PROPERTY	{p | p is a property of a data type in Protein-QL}
COMPARE	{≤, <, =, ≠, ≥, >,LIKE,NOT LIKE,IN,NOT IN}

```
(Protein.Secondary)(Protein.proteinName =
"HIV-1");
```

Our translator will generate object query in Lambda-DB:

```
SELECT p.Secondary FROM p in Protein
WHERE p.proteinName = "HIV-1";
```

The result is:

```
Secondary: (name: "HELIX", serNum: 1,
   numStrands: 1, initResName: "GLY",
   initChainID: "C", initSeqNum:86,
   endResName: "THR", endChainID: "C",
   endSeqNum: 91, sense: 1)  ......
```

Example 2: Selects the type and function of protein named "HIV-1" after verifying the consistency of length of first helix with the start and end ATOM positions.

```
(Protein.type,Protein.function)
(getSecondary("HIV-1").name="HELIX",
getSecondary("HIV-1").serNum=1,
(getSecondary("HIV-1").endSeqNum-
getSecondary("HIV-1").initSeqNum)=
getProtein("HIV-1").Secondary.length);
```

Then the translator generates object query in Lambda-DB:

```
for each pp in SELECT p.type, p.function
(SELECT(x.endSeqNum-x.initSeqNum) FROM x
in p.Secondary)=
(SELECT y.length FROM y in p.Secondary))
FROM p in Protein WHERE p.proteinName =
"HIV-1" and p.Secondary.name="HELIX" and
p.Secondary.serNum = 1;
```

If the data are not consistent, the result will be null, but in our case, the result is:

```
type: "virus"
function: "cleave the nascent polyprotei-
           ns during viral replication"
```

B. Protein-OODB Layer

With advancement in the development of the new laboratory instruments and experimental techniques we have seen an explosion protein data as evidenced in the growth of the public sources of protein data such as PDB. The protein data may come from different experiments, computational techniques, and interpretation of primary data. This usually results in data sources using a variety of formats such as flat files, web sources, or the reference data source. Data sources themselves may include a lot of different information such as data source name, the time data was recorded, the experiment's information such as conditions, steps, exceptions, foundations etc., author, location, and related information. Therefore, these heterogeneous protein data sources may result in the following problems:

1. Some protein data may have several parts, and each part may come from different data sources, we combine them into one format after collecting them separately. Some data may have several data sources, we try to get the primary or the most confident data sources by identifying the confidence of data sources.
2. Data may have several synonyms, homonyms, common misspellings, and abbreviations, we provide a data dictionary to list them and users can search them quickly. Our data dictionary defines some domain specific concepts, terms, and interpretation, which make databases easier and more efficient to search without any guessing work. It also can solve the problem that the same object have different names, and different objects use the same name in the life science.
3. We try to keep track of data sources by recording the provenance of data sources according to sources' information.
4. Protein data has a lot of inherent uncertainty, which may result in inaccurate or wrong results, we define experiments' standards and use "fuzzy" method to identify the reliability of those data, if data are not in the range of threshold, we will discard them to avoid further uncertainty.
5. Protein data is complicated and the data size is very huge, some queries may need very long time to execute. We develop a schedule table to store those queries and their results with latest data in this layer. We schedule queries depending on execution time for users.

III. PROTEIN DATA TYPES

Proteins have multiple levels of structure, although they are made up from 20 different L-α-amino acids. Usually protein scientists use primary, secondary, tertiary structures to analyze the protein functions. Primary structure is the sequences of its amino acids that give information about what and how the protein can do, and the history of the protein. With specific sequences, you can find out more about structure, function, and evolution of protein and they are very important to predict the 3D structure of protein. Secondary structure usually includes about 1/3 alpha helix, 20-28$\%$ beta sheet, and loops (turns and random coil), which is a local structure of linear segments of the polypeptide backbone atoms. Tertiary structure is *three dimensional, native structure of a single polypeptide or protein* [6]. The scientists can find out the functionality of protein according to tertiary structure. The following shows the operators for different structures (we only show a few of them because of the limitation of pages of paper) users can easily add other operators if they want.

- lengthOfSequence(Protein p): {*l* |*p is protein name, l is the length of protein p's primary structure*}

- subSequence(Protein p,int start, int end): {*sub | p is protein name, sub is the subsequence from position start to position end in p's primary structure*}
- globalAlignment(Protein p1, Protein p2): {*ga | p1, p2 are protein names, ga is the global alignment using Needleman-Wunsch method in the primary structures*}
- noOfHelix(Protein p): {*n | p is protein name,n is the number of helix in p's secondary structure*}
- getSecondary (Protein p): {*ps | p is protein name, ps is p's secondary structure*}
- nearestNeighbor(Protein p): {*p1 | p, p1 are protein names in tertiary structure, p1 has the shortest distance among neighbors of p*}
- cluster(Protein p,double d): {*p1 | p, p1 are protein names,p and p1's distance is equal to or less than d*}
- getProtein(Protein p): {*pro | p is protein name, pro is p's all information that is stored in databases*}

IV. Data Constraint

As we mentioned, protein data size is very huge, the missing or wrong data information may occur since the authors lose some data or change some part of experiments' data without any intention when they use data. So curating data is becoming very important for the protein scientists and computer scientists. Data curation is *the activity of managing and promoting the use of data from its point of creation, to ensure it is fit for it contemporary purpose, and available for discovery and re-use* [9]. Now most users apply PDB file to store protein data. PDB file's data curation for format conversion, missing data, redundant data, refining models and other issues will be introduced in our another paper.

As we know, lambda-DB provides Object Query Language (OQL) to serve the users' requests, OQL constraints can be used to detect abnormal data when users input data, but it does not provide constraints to check the conflict data. For example, if users want to define some attribute as two characters, they have to use string and can not use char[2] that may affect the accuracy of data. So we add some constraint syntax to strengthen this query language called Lambda-DB Constraint Language. So we can change PrimaryPartition definition as follows:

```
struct PrimaryPartition {
   char[6] name; short serNum;
   char[1] chainName; long elementNo;
   char[200] sequence;};
```

By identifying and expressing all such constraints, we can automate the process of verifying if the PDB data file is consistent and has good quality. We propose to implement a constraint checker module that is responsible to enforce all constraints on Lambda-DB. This module is invoked immediately after a PDB file is converted into object-oriented form in lambda-DB thereby ensuring good data quality in the protein OODB.

V. Conclusion

This paper proposes an architecture that is protein domain specific, we also add some new features to existing OODB. This architecture has four components, a Users Interface, a Protein-QL, a Protein-OODB, and an OODB. Protein-QL provides convenience for users to store, retrieve, and modify data, and define some basic operations. Protein-OODB solves some protein data sources' problems and is used to connect Protein-QL and OODB layers. In our architecture, we also define three protein data structures as internal data types to simplify the queries so that the users can easily to understand and send requests. We also introduce some constraints into Lambda-DB to improve the data quality. This paper gives a general idea to develop a new architecture for a certain biological domain. It is easy to extend this architecture into other biological domains. A prototype implementation of this architecture is in progress.

VI. Future Work

We will develop a biological architecture (including Protein, DNA, RNA and so on) like the following figure 3 shown called Bio-OODBMS. Our current architecture is object-oriented, which is not dependent on the existing OODB that makes users easier to add other domains into this architecture. Like Protein-QL and Protein-OODB, we just need to develop new query languages for DNA (RNA and other domains) and DNA (RNA and other domains) Object-Oriented DataBase to connect Users Interface and OODB. We also plan to add some new features into lambda-DB to check the input data so that this language will provide more strong constraints to check the accuracy of data.

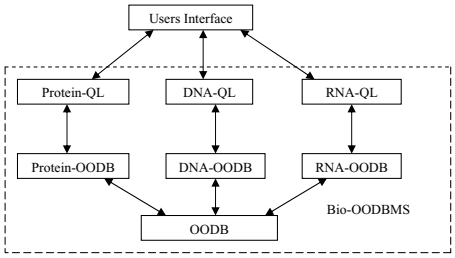

Fig. 3. Domain Specific Object-Oriented Architecture for Biology

References

[1] Esser, R. and Janneck, J.W.: A Framework for Defining Domain-Specific Visual Languages.

[2] Fegaras, L., Srinivasan, C., Rajendran, A. and Maier, D.: lambda-DB: An ODMG-Based Object-Oriented DBMS. SIGMOD Conference 2000, pp. 583.

[3] Krane, D.E. and Raymer, M.L.: Fundamental Concepts of Bioinformatics. 2003 Pearson Education, Inc., publishing as Benjamin Cummings, 1301 Sansome Street, San Francisco, CA 94111

[4] Mount, D. W.: Bioinformatics: Sequence and Genome Analysis. 2001 Cold Spring Harbor Laboratory Press, Cold Spring Harbor, New York.

[5] Codd, E. F.: A relational model of data for large shared data banks. Commun. ACM (1970), Vol. 13, 377–387.

[6] http://www.whatislife.com/reader/protein/protein.html

[7] Branden, C.: Introduction to Protein Structure. Published by Garden Publishing, Inc.

[8] Eidhammer, I., Jonassen, I. and Taylor, W.R.: Protein Bioinformatics: An Algorithms Approach to Sequence and Structure Analysis. Published by John Wiley & Sons.

[9] Lord, P. and Macdonald, A.: e-Science Curation Report: Data curation for e- Science in the UK: an audit to establish requirements for future curation and provision.