8-3-2007

Protein Structure Data Management System

Yanchao Wang

Follow this and additional works at: https://scholarworks.gsu.edu/cs_diss

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.gsu.edu/cs_diss/20

This Dissertation is brought to you for free and open access by the Department of Computer Science at ScholarWorks @ Georgia State University. It has been accepted for inclusion in Computer Science Dissertations by an authorized administrator of ScholarWorks @ Georgia State University. For more information, please contact scholarworks@gsu.edu.
PROTEIN STRUCTURE DATA MANAGEMENT SYSTEM

by

YANCHAO WANG

Under the Direction of Rajshekhar Sunderraman

ABSTRACT

With advancement in the development of the new laboratory instruments and experimental techniques, the protein data has an explosive increasing rate. Therefore how to efficiently store, retrieve and modify protein data is becoming a challenging issue that most biological scientists have to face and solve. Traditional data models such as relational database lack of support for complex data types, which is a big issue for protein data application. Hence many scientists switch to the object-oriented databases since object-oriented nature of life science data perfectly matches the architecture of object-oriented databases, but there are still a lot of problems that need to be solved in order to apply OODB methodologies to manage protein data. One major problem is that the general-purpose OODBs do not have any built-in data types for biological research and built-in biological domain-specific functional operations. In this dissertation, we present an application system with built-in data types and built-in biological domain-specific functional operations that extends the Object-Oriented Database (OODB) system by adding domain-specific additional layers Protein-QL, Protein Algebra Architecture and Protein-OODB above OODB to manage protein structure data.

This system is composed of three parts: 1) Client API to provide easy usage for different users. 2) Middleware including Protein-QL, Protein Algebra Architecture and Protein-OODB is designed to implement protein domain specific query language and
optimize the complex queries, also it capsulat es the details of the implementation such that users can easily understand and master Protein-QL. 3) Data Storage is used to store our protein data.

This system is for protein domain, but it can be easily extended into other biological domains to build a bio-OODBMS. In this system, protein, primary, secondary, and tertiary structures are defined as internal data types to simplify the queries in Protein-QL such that the domain scientists can easily master the query language and formulate data requests, and EyeDB is used as the underlying OODB to communicate with Protein-OODB. In addition, protein data is usually stored as PDB format and PDB format is old, ambiguous, and inadequate, therefore, PDB data curation will be discussed in detail in the dissertation.

INDEX WORDS: Protein-QL, Protein-OODB, Domain Specific Query Language, Protein Domain Specific Object-Oriented Database Management System, Protein Algebra, Protein Ontology, Protein Wrapper
PROTEIN STRUCTURE DATA MANAGEMENT SYSTEM

by

YANCHAO WANG

Major Professor: Rajshekhar Sunderraman
Committee: Yanqing Zhang
           Xiaolin Hu
           Yichuan Zhao

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
August 2007
ACKNOWLEDGMENTS

Firstly, I want to give special thanks to my advisor Dr. Rajshekhar Sunderraman for his insightful comments, supportive and valuable advice throughout my degree period. The dissertation would not have been possible without his guidance and help.

Secondly, I would like to thank my committee members, Dr. Yanqing Zhang, Dr. Xiaolin Hu and Dr. Yichuan Zhao for their support and advice.

Finally, my heartfelt thanks go out to my parents, my husband, Zhaosong, my son, James and my friends for their beliefs and support.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................ iv

LIST OF TABLES .............................................................................. vii

LIST OF FIGURES ........................................................................... viii

CHAPTER

1  INTRODUCTION ........................................................................... 1

   1.1 Problem Statement ............................................................... 2

   1.2 Our proposed Solution ....................................................... 4

   1.3 Specific Aims ..................................................................... 10

   1.4 Organizations .................................................................. 10

2  BACKGROUND ............................................................................. 12

   2.1 Flat File Database ............................................................. 12

   2.2 Relational Database .......................................................... 13

   2.3 Object-Relational Database ............................................... 14

   2.4 Object-Oriented Database .................................................. 14

   2.5 EyeDB ............................................................................. 15

   2.6 XML Database .................................................................. 16

   2.7 Database Management System ......................................... 17

   2.8 User Interface ................................................................... 18

   2.9 Domain Specific Language ............................................... 18

3  DOMAIN SPECIFIC OBJECT-ORIENTED DATABASE
   ARCHITECTURE FOR PROTEIN STRUCTURE DATA .................. 20
LIST OF TABLES

Table 2.1 Resident data in database ................................................................. 14
Table 3.1 The Terminal and Definition for Protein-QL Grammar ...................... 26
Table 3.2 Data Dictionary .................................................................................. 32
Table 3.3 The operations and interpretation of some query operators in Protein-QL 36
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Protein Data in PDB file</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>The architecture of domain specific OODB management system for protein structure data</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Java Client API</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Protein-QL Plus API</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>Visualization API</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Data Browser API</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>Domain specific object-oriented architecture for protein structure data</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>The Architecture of Protein-QL</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>The result for example 1</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>The result for PDBFormat of example 1</td>
<td>27</td>
</tr>
<tr>
<td>3.5</td>
<td>The result for example 3</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>The result for PDBSecondary in example 3</td>
<td>29</td>
</tr>
<tr>
<td>3.7</td>
<td>Protein Extended ER</td>
<td>30</td>
</tr>
<tr>
<td>3.8</td>
<td>The execution of Protein-QL</td>
<td>31</td>
</tr>
<tr>
<td>3.9</td>
<td>The alpha-carbon of each amino acid in secondary structure of hexokinase</td>
<td>34</td>
</tr>
<tr>
<td>3.10</td>
<td>The tertiary structure of hexokinase</td>
<td>35</td>
</tr>
<tr>
<td>3.11</td>
<td>The result for sequence(“HIV-1”)</td>
<td>38</td>
</tr>
<tr>
<td>3.12</td>
<td>Login interface</td>
<td>40</td>
</tr>
</tbody>
</table>
Figure 5.4 PDB Data Clean Services in PDB Data Curation System

Figure 5.5 PDB Data Identification in PDB Data Curation System

Figure 5.6 Author Spelling Error in PDB Data Identification

Figure 5.7 The Architecture PDB Data Curation System with XCML

Figure 6.1 Concept Architecture of NeuroMD

Figure 7.1 The architecture of high level programming environment system for protein structure data

Figure 8.1 The architecture of domain-specific database management system for protein structure data

Figure 8.2 Domain specific object-oriented database management system for biology
Chapter 1

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 experimental 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 and computer scientists have to face and solve. This is why more and more database applications are built for research purpose now.

Presently, there are a variety of traditional databases used to manage biological data such as flat files, relational databases and so on.

Flat files usually manage data by using strings and some other simple tools that can be easily mastered, but leave users difficulty to manipulate data. In addition, it does not support complex data types, which makes it not be able to meet the requirements of the management of complicated biological data.

Relational databases are mature and are successfully applied in many areas, and one of the major reasons is that the relational data model 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.
Although the developers of these traditional databases designed and provided biologists special tools to manage biological data, they need to develop new tools or recode previous software after a short time use in order to meet the changes of data formats or data types which wastes not only the time but also the energy since the biologists have to learn new query language and master how to use new tools or software. Therefore, using an efficient database management solution for the biological data is becoming an urgent task for most biologists and computer scientists. Since the object-oriented nature of life science data perfectly matches the architecture of object-oriented databases, biologists and computer scientists are switching their attention to object-oriented databases (OODB). Although OODB has above advantage over traditional 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 that makes OODB not perfectly suitable to biological data.

1.1 Problem Statement

Protein data, one type of biological data, has been growing rapidly due to more and more advanced experimental techniques. So, the problem of efficiently storing, organizing, and analyzing protein data is becoming an important task for most protein scientists and computer scientists. Although object-oriented database provides an architecture that can model the complexity present in protein data, it is quite a tedious task to query such databases and program applications that access such databases since the lack of the built-
in data types for biological research and built-in biological domain-specific functional
operations. Therefore we have to design specific OODB management system
(OODBMS) by adding some additional work over on original OODB to make it perfectly
match the requirements of protein data, also this system must be able to be easily
extended to other biological domains.

As we know, protein data are usually stored in PDB file as flat text formats (shown
on the following screen) that make clients difficult to use, so we designed our system to
provide clients convenient access protein data. In addition, PDB data have a lot of issues

Figure 1.1 Protein data in PDB file
such as inconsistency, heterogeneity and so on, thus we designed PDB Data Curation systems to clean PDB data.

1.2 Our Proposed Solutions

In this dissertation, we designed a Domain Specific OODB Management System (DSOODBMS) to solve the problem of protein data storage, data retrieval and data modification. This system extends the object-oriented database (OODB) system by

Figure 1.2 The architecture of domain specific OODB management system for protein structure data
adding additional layers: Protein-QL, Protein Algebra Architecture and Protein-OODB (other detailed parts) above OODB, it is designed specifically for protein domain, but it is a first step in building a general Bio-OODBMS for biological applications.

The architecture of this new domain specific OODB management system for protein structure data is a three-layer architecture shown in figure 1.2 that consists of the following components: a Client API, a Middleware and a Data Server.

I. Client API layer can have a variety of APIs:

   a. Java Client API can easily be viewed and mastered by protein domain

Figure 1.3 Java Client API
scientists who know requested parameter without much computer background. Clients can be able to formulate Protein-QL queries and have them sent to the server for execution. We also provide help functions help the clients send queries, display results in a domain friendly manner, etc. A simple client that performs some queries should look like the above figure 1.3.

b. PQL Plus Client is much like SQL Plus interaction that allows clients to send protein-QL queries directly to Protein-QL without any java code.

c. Visualization Client is like Rasmol tool that allows clients to view protein data structure and functions. We also support linking the protein to RCSB PDB to get 3D view.
d. Data Browser provides clients to view protein data in PDB format or object format.
II. Middleware layer is designed to better serve clients by using protein domain specific programming query language and protein OODB:

a. Server/Listener provides the basic services of our system. The users should be able to know the types of parameters, results and functions from Server/Listener which hides the details of implementation. The following example shows possible services.

```java
//return any object
public Object[] executeQuery(String request);
public Protein[] executeProteinQuery(String request);
//data structure in PDB format
public String PDBFormat(String proteinName);
//PDB format of primary structure
public String PDBPrimary(String proteinName);
```

b. Protein-QL is designed as domain specific high-level query language and be able to provide convenience for users to store, retrieve, and modify data. It defines some basic operations such as SELECT, INSERT, DELETE, UPDATE that can be executed on basic data types as well as on protein data types. Protein-QL defines a list of queries in protein terms (such as nearest neighbor, subparts of protein and so on) which enables domain scientists to query information in their own language without much syntactical restriction.

For example, the query `getPrimary(proteinName)` should return the primary structure of protein named “proteinName”.
c. Protein Algebra Architecture is domain specific and independent of Protein-QL and Protein-OODB. It has three parts -- Protein Ontology, Protein Algebra and Protein Wrapper:

- Protein Ontology is able to define and identify object classes, object properties and remove ambiguity, incompatibility and inconsistency of protein data.
- Protein Algebra is different from original mathematical algebra which should provide an extensible set of high-level genomics and protein data types and collection of appropriate genomics and protein operations or functions.
- Protein wrapper hides the knowledge of Protein-OODB, which makes our Protein Algebra independent of DBMS such that we only need to recode our Protein Wrapper without changing Protein Algebra when Protein Algebra is connected to other data sources.

d. Protein-OODB can provide possible method to solve some protein data sources’ problems and is used to connect Protein-QL and OODB which makes Middleware independent of the underlying OODB. In addition, in order to simplify the queries in Protein-QL, we define the protein, primary, secondary and tertiary protein structures as internal data types such that domain scientists can easily formulate complex requests for data without much of a learning curve.

III. In Data Server layer, OODB will provide users basic operations. We used EyeDB as our underlying OODB. Data storage will hold our protein data.
IV. PDB Data Curation layer also can be considered as Client API which is designed to provide the possible ways to curate data and improve the PDB (most protein data are stored in PDB format) data quality. In this layer, it deals with the data identification, data errors, data redundancy, data ambiguity, data heterogeneity, data consistency, conflict data and obsolete data of PDB files. In addition, in order to make the underlying OODB more powerful, we may use more constraints to check the input data and strengthen the accuracy of data.

1.3 Specific Aims

In this dissertation, we have several specific aims based on our research as follows:

I. Design protein domain specific query language--Protein-QL which is a high-level language and is easily extended to other biological domain

II. Design protein domain specific object-oriented database--Protein-OODB to store, retrieve and manipulate protein data

III. Build PDB data curation system to help improve PDB data quality

IV. Implement Protein-QL and Protein-OODB

V. Design Protein Algebra Architecture

VI. Implement PDB Data Curation System and connect it into our domain specific OODBMS

1.4 Organizations

The rest of the dissertation is organized as follows: The background will be talked about in section 2. Section 3 presents an overview of the domain specific object-oriented database architecture for protein structure data and the protein data types. Section 4 describes Protein Algebra Architecture. Section 5 discusses data curation issues. Related
work will be shown in section 6. Conclusion and future work are presented in Section 7 and section 8.
Chapter 2

BACKGROUND

As we know, everywhere needs data (or information) to be stored such as hotels, airplanes, super stores to identify users, reservations and the details of goods. If there are no such data be stored, above places will be in a chaos because of the difficulty of the identification. Therefore, data or information is stored in somewhere called database is indispensable for all public and private service apartments. In addition, how to store, retrieve and modify data is very important because users need to keep all data consistent and clean in order to avoid wrong information and inconvenience. This kind of data handling is called as data management. Presently, there is a variety of data management software available, such as flat file database management systems, relational database management systems (RDBMSs), object-relational database management systems, and object database management systems (OODBMS). If user wants to make use of above software, he/she has to be familiar with corresponding databases. In the following parts, we will give basic introduction and some examples for these databases.

2.1 Flat File Database

All the information is stored in text files in a flat file database. A flat file is a file containing records, generally one record per line. There are no structural relationships.

[1] Let’s consider a simple example, storing a person’s name, phone number, gender and address in a flat file database for a community.

<table>
<thead>
<tr>
<th>Name</th>
<th>phone</th>
<th>gender</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim Huang</td>
<td>1234567890</td>
<td>M</td>
<td>Apt 456</td>
</tr>
</tbody>
</table>
The information of each line belongs to one person. It looks good, but it is difficult to manage. For example, user requests the phone number of Jim Huang, he/she has to look at the file from the beginning until he/she find it (it will get result very quickly if the requested result is at the beginning of the file, otherwise query has to search the file line by line). Database will send whole row which name is Jim Huang to the user. Therefore, most users use other database software to manipulate their data instead of flat file except for specific purpose.

2.2 Relational Database

A relational database is a database that conforms to the relational model. It is defined as a set of relations. A relation is defined as a set of tuples that all have the same attributes. This is usually represented by a table, which is data organized in rows and columns. In a relational database, all of the data stored in a column should be in the same domain (data type). In the relational model, the tuples should not have any ordering [1]. Here I will give a simple example and SQL statement to show how to create table to save and query data from relational database. The row from a relational table is analogous to a record, and the column to a field [1].

```
CREATE TABLE resident ( 
    name     varchar2(30),
    phone    int(10),
    gender   char(1),
    address  varchar2(25)
)
```
Table 2.1 Resident data in Database

<table>
<thead>
<tr>
<th>Name</th>
<th>phone</th>
<th>gender</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim Huang</td>
<td>1234567890</td>
<td>M</td>
<td>Apt 456</td>
</tr>
<tr>
<td>Ford Smith</td>
<td>1234589010</td>
<td>M</td>
<td>Apt 320</td>
</tr>
<tr>
<td>Amy White</td>
<td>4567294707</td>
<td>F</td>
<td>Apt 128</td>
</tr>
</tbody>
</table>

Now you can get phone number of one person by sending query `select phone from resident where name='Jim Huang'`, then you will get 1234567890. By this way, user can get desired result very easily because query only searches name column and sends phone number back to the user. Therefore it is more convenient than using flat file database and it can keep all data in very good formats.

2.3 Object-Relational Database

An object-relational database is a relational database management system that allows developers to integrate the database with their own custom data types and methods [1]. The table create statement is similar as relational database except for user’s own data types and methods (that means user can define his/her own data types and methods according to his/her preference, convenience and knowledge).

2.4 Object-Oriented Database (OODB)

Object is an individual unit which is used as the basic building block of programs. Each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent little machine or actor with a distinct role or responsibility [1].
The information is represented in the form of objects like those in object oriented programming in object-oriented database. It is designed to work well with object-oriented programming languages such as Java, C#, Visual Basic .NET, C++ and Smalltalk. It uses exactly the same model as object oriented programming languages [1]. OODB uses Object Query Language (OQL) to fetch the query from and send results back to the users. And it supports complex structures, object identity, extensibility and inheritance between classes. Therefore, it is a great idea to use OODB in biological domain with complex data. The examples will be shown in the section 3--Syntax of Protein-QL Layer which shows the formats by using EyeDB to manage object-oriented data.

2.5 EyeDB

EyeDB [2] is one type of OODB. The key features of the EyeDB OODBMS include:

1. Standard OODBMS features: persistent typed data management; client/server model; transactional services; recovery system; expressive object model; inheritance; integrity constraints; methods; triggers; query language; application programming interfaces.

2. Language orientation: a definition language based on the ODMG Object Definition Language (ODL); a query language based on the ODMG Object Query Language (OQL); several manipulation language bindings (at least C++ and Java).

3. Genericity and orthogonality of the object model: object models (i.e. every class derives from the class object and can be manipulated as an object); type polymorphism; binary relationships; literal and object types; transient and persistent objects; method and trigger overloading; template-based collections such as set, bag and array; multi-dimensional and variable size dimensional arrays.
4. Support for large databases: databases up to several Tb (tera-bytes).

5. Efficiency: database objects must be directly mapped within the virtual memory space; object memory copy must be reduced to the minimum; clever caching policies must be implemented.

6. Scalability: programs must be able to deal with hundred of millions of objects without loss of performance.[2]

EyeDB is very suitable for our system according to its features, so we use it as our underlying OODB for implementation.

2.6 XML Database

Data in XML format is extracted from databases and put into XML documents and vice versa which may be more efficient and easier to store the data [1]. XML Database has two major classes:

1. XML-enabled: mapping all XML to a traditional database (such as relational database, flat file, Object-relational database) accepting XML as input and rendering XML as output. The database has to do the conversion itself as opposed to middleware.

2. Native XML (NXD): is based on XML and the fundamental unit of storage is an XML document. Many NXDs are not really standalone databases and don’t really store the XML in true native form. The formal definition of a Native XML Database is:

   a. Defines a logical model for an XML document and stores and retrieves documents according to that model.

   b. Has an XML document as its fundamental unit of storage.
c. *Is not required to have any particular underlying physical storage model.*

*It can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format.* [1]

Now I will give an example to show our resident data in XML database.

```
<Community CommunityName="Flower Garden">
    <Name>Jim Huang</Name>
    <Phone>1234567890</Phone>
    <Gender>M</Gender>
    <Address>Apt 456</Address>
    <Name>Ford Smith</Name>
    <Phone>1234589010</Phone>
    <Gender>M</Gender>
    <Address>Apt 320</Address>
    <Name>Amy White</Name>
    <Phone>4567294707</Phone>
    <Gender>F</Gender>
    <Address>Apt 128</Address>
</Community>
```

XML database can store data in a very uniform format, but it needs parser to convert document if user wants to query data.

2.7 Database Management System (DBMS)

*A database management system (DBMS) is a software designed to manage a database, and run operations on the data requested by numerous clients. A DBMS is a complex set*
of software programs that controls the organization, storage and retrieval of data in a database. A DBMS includes:

1. A modeling language to define the schema of each database hosted in the DBMS, according to the data model.
2. Data structures (fields, records and files) optimized to deal with very large amounts of data stored on a permanent data storage device.
3. A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the use privileges on data.
4. A transaction mechanism, which ideally guarantee the ACID (Atomicity, Consistency, Isolation and Durability) properties, in order to ensure data integrity, despite concurrent user accesses and faults. [1]

2.8 User Interface

User has to communicate with above underlying databases or other tools by User Interface when he/she uses them. The User Interface is the aggregate of means by which people (users) interact with a particular machine, device, computer program or other complex tools. The user interface provides means of input (allowing the users to manipulate the system) and output (allowing the system to produce the effects of the user’s manipulation). [1] User Interface provides a way which connects user and underlying database management system.

2.9 Domain Specific Language

In order to send requests and get results from databases, user has to make use of query language (computer language used to make queries into databases and information
systems [1]) to fetch the information to and from databases. One of query language is called Domain Specific Language (DSL) which is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.\[1\] The key characteristic of DSLs is they focused on expressive power. Our Protein Domain Specific Query Language (Protein-QL) is designed for protein domain such that protein scientists to query information in their own language without knowing much computer science syntax.
Chapter 3

DOMAIN SPECIFIC OBJECT-ORIENTED DATABASE ARCHITECTURE FOR PROTEIN STRUCTURE DATA

The domain specific object-oriented database architecture for protein structure data without Protein Algebra Architecture is a three-layer architecture that consists of the following components: a client API, A middleware (including a RMI server, a query language for protein structures (Protein-QL)), and an object-oriented database for protein structures (Protein-OODB), and an object-oriented database EYEDB layer. Figure 3.1 illustrates this architecture.

The clients can use this system to send domain specific requests and manage the database. The Client writes simple domain specific queries according to Protein-QL and sends them to the Server. The Server receives the queries and communicates with Middleware using java RMI and checks the grammar of queries according to the syntax of Protein-QL. Then the interpreter between Protein-QL and Protein-OODB converts queries into EYEDB queries. Finally EYEDB sends the results back to the server. This system provides clients convenient access and is easily mastered.

Figure 3.1 Domain specific object-oriented architecture for protein structure data
3.1 Protein-QL of Middleware Layer

In recent years, protein scientists have been accumulating data in an exponential rate. Conventional data query languages, such as SQL, OQL, XQuery are dependent of underlying databases and all of them have limitations on their expressive power because they can not capture domain semantics. Therefore, how to use databases to query, store, modify, organize and analyze protein data in clear and efficient ways is becoming more and more important. Although some biological data and conceptual modeling methods have been developed for genomics domain, there is no further research on protein domain. In this dissertation, we proposed Domain Specific Query Language for Protein Structure Data (Protein-QL) to manipulate and handle with protein domain queries.

In the Protein-QL, we provide some basic operations to store, retrieve, and modify protein data information using EYEDB such as SELECT, INSERT, DELETE, UPDATE, these operations can be executed on basic data types as well as on protein data types. Protein-QL defines a list of queries in protein terms, which enables domain scientists to query information in their own language without much syntactical restriction. For example, if you want to get primary structure of protein named “proteinName”, you can use query `getPrimary(proteinName)`. We also provide other queries such as the nearest neighbor, 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.

3.1.1 The Architecture of Protein-QL

The architecture of Protein-QL has 3 parts shown in figure 3.2, Protein-QL Interpreter, translator and DSDB query execution engine.
Protein-QL Interpreter accepts the domain specific query requests from users. In this part, users can use Protein-QL grammar, Protein-QL data types, Protein-QL operators and Protein-QL functions. Translator receives the queries which are 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.

![Figure 3.2 The architecture of Protein-QL](image)

In our architecture, we implement the features of OODBMS using EYEDB to store complex protein data as object format. As we know, EYEDB provides Object Query Language (OQL) to optimize the queries and serve the users’ requests, and our Protein-QL queries can be easily converted to OQL queries of EYEDB. In our Protein-QL, we define protein data type and its three data structures Primary, Secondary and Tertiary as internal data types, thus biologists can easily request the information with these domain specific formats.
3.1.2 Operators of Protein-QL

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 Protein, Primary, Secondary and Tertiary.

//Protein Object:

class Protein {
    string proteinName; string types; string functions;
    string remark;
    set<words *> domainOfPrimaryStru;
    Primary * primary;
    Secondary * secondary;
    Tertiary * tertiary;
    constraint<unique> on proteinName;
    constraint<not null> on proteinName;
    index on proteinName;
};

struct words {
    string name;
};

//Primary Structure:

class Primary {
    string proteinName;
    array<PrimaryPartition *> primary;
};

class PrimaryPartition{
    string name; int serNum; string chainName;
long elementNo; string seq;

//Secondary Structure:
class Secondary {
    string proteinName;
    array<SecondaryPartition *> secondary;
};
class SecondaryPartition {
    string name; int serNum; string ID; int numStrands;
    string initResName; string initChainID; int initSeqNum;
    string endResName; string endChainID; int endSeqNum;
    int sense; string curAtom; string curResName;
    string curChainID; int curResSeq; string prevAtom;
    string prevResName; string prevChainID; int prevResSeq;
    string comments; int length;
};

//Tertiary Structure:
class Tertiary {
    string proteinName;
    array<TertiaryPartition *> tertiary;
};
class TertiaryPartition {
    string name; int serNum; string atom1; string resName;
    string chained; int resSeq; double x; double y; double z;
    double possibility; double pos; string atom2;
};
Everything as an object is stored, retrieved and modified in database of our Protein-QL, so it is very convenient to execute operations for clients. Actually we had planned to use lists to implement protein four structures, but they have not implemented in EYEDB, we used arrays instead lists, later on, we will change them into lists once they are available.

3.1.3 Syntax of Protein-QL

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

Protein-QL_query ::= [ delete | insert | update | create | ]“(“<result_format>“)”

[“(“<condition> “)”] ;  /*[] optional and empty means select*/

<result_format> ::= <result_item> {, <result_format>}  /*{} zero or more occurrences*/

<result_item> ::= DATATYPE_NAME | DATATYPE_NAME.PROPERTY  /*like Protein or Protein.proteinName*/

<condition> ::= <condition_item> {, <condition>}

<condition_item> ::= <comparison_item> | <operator “(”parameter{,parameter}“)”>|  /*operator can be any operator on Protein-QL, such as Protein.Tertiary*/

<comparison_item>::= <operand> COMPARE <operand> { AND | OR

<operand> ::= NAME.PROPERTY | CONST | <exp>

<operator> ::= DS_OPERATOR | NAME.DS_OPERATOR  /*DS means domain specific*/

<parameter> ::= <operand> | /*empty*/

<operation-item> ::= DATATYPE_NAME [.PROPERTY] | <operand>

<operand> ::= NAME.PROPERTY | CONST | <exp>
/*CONST can be any constant of basic data types*/

<exp> ::= <exp> “+” <exp> | <exp> “–” <exp> | <exp> “*” <exp> | <exp> “/” <exp>
| CONST | NAME. PROPERTY

Table 3.1 The terminal and definition for Protein-QL grammar

<table>
<thead>
<tr>
<th>Terminal</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>{name</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>{p</td>
</tr>
<tr>
<td>COMPARE</td>
<td>{'&lt;=', '&lt;', '=', '!=', '&gt;', '&gt;=', 'LIKE' 'NOT LIKE', 'IN', NOT IN}</td>
</tr>
</tbody>
</table>

We now present some examples using our Protein-QL:

Example 3.1: Gets the protein named “HIV-1". (“HIV-1” is an abbreviation of “HIV-1 protease”)

(Protein)(Protein.proteinName = “HIV-1")

Then the interpreter generates object query in EYEDB:

select p from p in Protein where p.proteinName = “HIV-1";

The result is:
The PDB format result may be obtained using the service `PDBFormat(proteinName)`, the result is shown in the following figure 3.4:
**Example 3.2:** delete a protein object whose name is “HIV-1”

delete (Protein)(Protein.proteinName = “HIV-1”);

Then the interpreter generates object query in EYEDB:

```
select Protein.proteinName = “HIV-1”;
```

=5738.5.854641:oid

delete 5738.5.854641:oid;

= 5738.5.854641:oid

Or using following query:

```
delete p from p in Protein where p.proteinName = “HIV-1”;
```

The result is:

= 5738.5.854641:oid

**Example 3.3:** Gets the secondary structure of protein named “HIV-1”.

(Protein.secondary)(Protein.proteinName = “HIV-1”);

Our interpreter will generate object query in EYEDB:

```
select p.secondary from p in Protein
where p.proteinName = “HIV-1”;
```

The result is:
Figure 3.5 The result for example 3.3

PDB format by using PDBSecondary — PDBSecondary ("HIV-1") from our Java Client API. Then the server returns result in the following figure 3.6.

Figure 3.6 The result for PDBSecondary in example 3.3
Example 3.4: create a new protein object named ‘HIV-1’

create (Protein)(Protein.proteinName = “HIV-1”);

Then the interpreter generates object query in EYEDB:

new Protein(proteinName:“HIV-1”);

The result is:

= 5767.5.586687:oid

3.1.4 Protein ER

Our Protein-QL is complete since all connections (paths) in the following Extended ER

![Figure 3.7 Protein Extended ER](image-url)
(Figure 3.7) can be accessed. Users can access (or connect) all data to handle with the queries.

We give an example to get proteins which have the same subsequence “ARG VAL DCL PHE” as protein “HIV-1”. Here we can use function location(proteinName, “ARG VAL DCL PHE”). If the location returned is greater than zero, then this protein named “proteinName” has the same subsequence “ARG VAL DCL PHE” as protein “HIV-1”. The completion of Protein-QL can be shown in the following figure 3.8. The connection of these two proteins can be finished through the path of protein-primary-location.

![Figure 3.8 The execution of Protein-QL](image)

3.2 Protein-OODB of Middleware Layer

With advancement in the development of the new laboratory instruments and experimental techniques we have seen an explosion protein data. These protein data may come from different computational techniques, experiments, and interpretation of
primary data and data sources themselves may include a variety of different information that may result in heterogeneous protein data sources. Therefore, our system will provide following ways to solve these heterogeneous problems:

1. We will use the most confident data sources by identifying the confidence of data sources to solve multiple sources problems.

2. We will provide a data dictionary (or called controlled vocabulary) to list synonyms, homonyms, common misspellings, and abbreviations such that system can search them quickly in table 3.2. Our data dictionary will define some domain specific concepts, terms, and interpretation, which make databases easier and more efficient to search without any guessing work. It also should be able to solve the problem that the same object has different names of the life science.

<table>
<thead>
<tr>
<th>Synonym</th>
<th>Homonym</th>
<th>Common Misspelling</th>
<th>Abbreviation</th>
<th>Same Object with Different Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIV-1/ Human immunodeficiency virus 1/</td>
<td>HIV-1/ Human Immunodeficiency Virus Type 1/</td>
<td>HIV-1</td>
<td>HIV-1</td>
<td></td>
</tr>
<tr>
<td>HIV-1/ Immunodeficiency Virus Type 1, Human</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>
3. We will develop a schedule table to store the queries needing very long time to execute. These queries and their results are updated with latest data to keep them up-to-date. Clients can get them from local memory to save time.

We defined data dictionary as follows:

```cpp
class DataDictionary{
    string proteinName;
    set<words *> synonym;
    set<words *> homonyms;
    set<words *> cmp;  //Common Misspelling
    set<words *> abbreviation;
    set<words *> sodn;  //Same Object with Different Name};
```

### 3.3 Protein Data Types

As we know, proteins have multiple levels of structure, although they are made up from 20 different L-α-amino acids. Usually protein has four structures, primary, secondary, tertiary and quaternary structures. But protein scientists usually work on the first three protein structures, we implement them as the internal data types of protein object in our architecture, which users can directly request the protein structures from our database.

For our protein object, we define protein characteristics such as name, type, function, synonyms, remark, domains of primary structure, primary structure, secondary structure and tertiary structure.

#### 3.3.1 Primary Structure

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 these sequences
are very important to predict the 3D structure of protein. We provide operations to show the basic information of protein sequences. For example, the following is sub-sequence of a molecule of hexokinase which is a metabolic protein existing in almost all living organisms and composed of approximately 6000 atoms:

A A S X D X S L V E V
H X X V F I V P P X I L Q A V V S I A T T R X D D X D S A A
A S I P M V P G W V L K Q V X G S Q A G S F L A I V M G G G

3.3.2 Secondary Structure

Secondary structure is a local structure of linear segments of the polypeptide backbone

Figure 3.9 The alpha-carbon of each amino acid in secondary structure of hexokinase taken from [4]

Atoms without regard to the conformation of the side chains [3]. Secondary structure usually includes about 1/3 alpha helix, 20-28% beta sheet, and loops (turns and random coil). We implement some operations about helix and sheet to show secondary structure of protein. The following figure shows secondary structure of hexokinase.
3.3.3 Tertiary Structure

Tertiary structure is *the overall shape of a single protein molecule and the spatial relationship of the secondary structural motifs to one another* [3]. The scientists can find out the functionality of protein according to this structure. In the 3D structure, we provide the methods to show nearest neighbor and clusters in a specific range for specific protein. The following figure gives an example of tertiary structure:

![Tertiary Structure of Hexokinase](image)

Figure 3.10 The tertiary structure of hexokinase taken from [4]

Above three protein structures are very important to protein researchers. Therefore, our system provides some operators for these structures. The operators of three protein structures implemented in our architecture are shown in table 3.3.

In this dissertation, we also define above three structures as internal basic data types for our protein OODB, users can conveniently and easily send the queries without any extra learning.
3.3.4 The Operators for Protein

As we know, protein data has very large size and clients may need some part of a certain protein data, not the whole data. So our protein-OODB layer provides some operators based on protein domain, which clients can conveniently and easily send the queries without any extra learning. We now define some operators in table 3.3, users can easily add more operators and functions into our system later.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence(String ProteinName)</td>
<td>sequence: {s</td>
</tr>
<tr>
<td>oneLetterCode(String ProteinName)</td>
<td>oneLetterCode: {olc</td>
</tr>
<tr>
<td>lengthOfSequence(String ProteinName)</td>
<td>lengthOfSequence: {l</td>
</tr>
<tr>
<td>subSequence(String ProteinName, int start, int end)</td>
<td>subSequence: {sub</td>
</tr>
<tr>
<td>location (String ProteinName, string subSequence)</td>
<td>location: {pos</td>
</tr>
<tr>
<td>locationOneLetter(String ProteinName, String subSequence)</td>
<td>locationOneLetter: {pos</td>
</tr>
<tr>
<td>noOfSub(String ProteinName, String subSequence)</td>
<td>noOfSub: {n</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>noOfSubOneLetter(String ProteinName, String subSequence)</code></td>
<td>`noOfSubOneLetter : {n</td>
</tr>
<tr>
<td><code>globalAlignment(String ProteinName1, String ProteinName2)</code></td>
<td>`globalAlignment: {ga</td>
</tr>
<tr>
<td><code>mostSimilarity(String ProteinName)</code></td>
<td>`mostSimilarity: {ms</td>
</tr>
<tr>
<td><code>getPrimary (String ProteinName)</code></td>
<td>`getPrimary: {ps</td>
</tr>
<tr>
<td><code>noOfHelix(String ProteinName)</code></td>
<td>`noOfHelix: {n</td>
</tr>
<tr>
<td><code>lengthOfHelix(String ProteinName, int helix)</code></td>
<td>`lengthOfHelix: {l</td>
</tr>
<tr>
<td><code>lengthOfEachHelix(String ProteinName)</code></td>
<td>`lengthOfEachHelix: {l(List)</td>
</tr>
<tr>
<td><code>noOfChain(String ProteinName)</code></td>
<td>`noOfChain: {n</td>
</tr>
<tr>
<td><code>noOfStrand(String ProteinName, String chain)</code></td>
<td>`noOfStrand: {n</td>
</tr>
<tr>
<td><code>noOfStrandEachChain(String ProteinName)</code></td>
<td>`noOfStrandEachChain: {n(List)</td>
</tr>
<tr>
<td><code>senseOfEachStrand(String ProteinName)</code></td>
<td>`senseOfEachStrand: {sense(List)</td>
</tr>
<tr>
<td><code>getSecondary(String ProteinName)</code></td>
<td>`getSecondary: {ps</td>
</tr>
<tr>
<td><code>center3D(String ProteinName)</code></td>
<td>`center3D: {d</td>
</tr>
<tr>
<td><code>nearestNeighbor3D(String ProteinName)</code></td>
<td>`nearestNeighbor3D: {p1</td>
</tr>
</tbody>
</table>
We can combine these operators in our domain specific queries, for example

$\text{(Protein)(lengthOfSequence("HIV-1") > 100)}$. We also can use them separately. The following shows some examples using our Protein-QL:

**Example 3.5**: gets the sequence of protein named “HIV-1”.

```
sequence("HIV-1");
```

The result is shown in following figure:

![Figure 3.11 The result for sequence(“HIV-1”)](image)

**Example 3.6**: gets the number of helix of protein named “HIV-1”.

```
```
noOfHelix("HIV-1");
The result: 2

**Example 3.7:** gets the nearest neighbor of protein named “HIV-1”.

nearestNeighbor3D("HIV-1");
The result:

Protein{5738.5.854641:oid }={proteinName=......}

### 3.4 Details on Using System Based on EyeDB

It’s very easy to use our system. The following figures show the application of our system. Client only needs to input corresponding parameters for different services. He/she can use our online help if client is not familiar with our system. The next part will introduce how to use our system.

Firstly, client needs to login which he/she inputs user name and password from the following interface.
Clients will see the following interface after they login. We provide different services for different users such as Java Client API, PQL API (Protein-QL API similar as SQL), Data Browser, Visualization and Help.
If clients have difficulty to use services of our system, they can click Help button to get information about how to use it.
In Java Client API, the Java Clients only need to know service name, input parameters and output since we also put Protein-QL services on the web site such that Java Clients do not need to learn Protein-QL. The details of implementation do not affect the usage of clients. In the implementation, the system develops algorithm to create an alignment from a number of sequences such that the system does not need to extract the data for each protein from the database to increase performance (because less data has to be read from and written to the database) and reduce storage needs. We also design algorithm to calculate center of protein and store them in the file, therefore, the system only needs to calculate the center once for each protein.

![Figure 3.15 Java Client API](image-url)

Figure 3.15 Java Client API
Protein-QL clients can make use of PQL Plus to send queries and get results from our database. Clients only need know the syntax of our Protein-QL which is available on the Help part.

Figure 3.16 Protein-QL Plus API

In the Data Browser, clients input Protein name to get PDB or object format view of protein. We already put all protein names on the interface, clients just copy and paste into input area.
Clients can input PDB ID (listed on the web) for specific protein in visualization to get 3D view.
Then clients click “Click me” to be taken to RCSB PDB web site to view 3D structure of Protein.

Figure 3.19 Visualization API connected to RCSB PDB

3.5 Details on Using System Based on MySQL

In our previous work, we implemented Protein-QL and Protein-OODB for Protein Domain Specific Object-Oriented Database Management System (Protein-DSOODBMS) for protein structure data in OODB by using EyeDB. In order to make our work be applicable in other database systems, we only need to code one adapter if we want to connect our domain specific Protein-QL to other database. We can convert Protein-OODB queries to relational database such as MySQL which can
avoid translating Protein-QL to MySQL by making use of Protein-OODB. In this case, we need to convert Objects of EyeDB into String of object content in order to match data in table of MySQL. The figure 3.20 shows this application.

Figure 3.20 The architecture of DBMS for protein structure data based on Different DBs

In MySQL, we define our protein data structure as follows:

```sql
create table Protein(
    proteinName varchar(30) not null primary key,
    types varchar(30),
    functions varchar(40),
    remark varchar(200),
    domainOfPrimaryStru varchar(400)
);

create table PrimaryPartition(
    proteinName varchar(30) not null, 
    porder int, 
    name varchar(6), 
    serNum int,
);```
CREATE TABLE ProteinNames (
  proteinName varchar(30) NOT NULL,
  sorder int,
  name varchar(6),
  serNum int,
  ID varchar(4),
  numStrands int,
  initResName varchar(3),
  initChainID varchar(2),
  initSeqNum int,
  endResName varchar(3),
  endChainID varchar(2),
  endSeqNum int,
  sense int,
  curAtom varchar(1),
  curResName varchar(3),
  curChainID varchar(2),
  curResSeq int,
  prevAtom varchar(1),
  prevResName varchar(3),
  prevChainID varchar(2),
  prevResSeq int,
);
create table TertiaryPartition(
    proteinName varchar(30) not null,
    torder int,
    name varchar(6),
    serNum int,
    atom1 varchar(3),
    resName varchar(3),
    chained varchar(2),
    resSeq  int,
    x float,
    y float,
    z float,
    possibility float,
    pos float,
    atom2 varchar(1),
    primary key (proteinName, torder)
);

Example 3.8: Gets the protein named “HIV-1” (Abbreviation “HIV-1 Protease”).

(Protein)(Protein.proteinName = “HIV-1”);

Then the interpreter generates Protein-OODB query:

select p from p in Protein where p.proteinName=“HIV-1”;

Then the interpreter generates query in MySQL:

select * from Protein where proteinName=“HIV-1”;

select name, serNum, chainName, elementNo, seq from PrimaryPartition where proteinName="HIV-1" order by porder;
select name, serNum, ID, numStrands, initResName, initChainID, initSeqNum, endResName, endChainID, endSeqNum, sense, curAtom, curResName, curChainID, curResSeq, prevAtom, prevResName, prevChainID, prevResSeq, comments, length from SecondaryPartition where proteinName="HIV-1" order by sorder;
select name, serNum, atom1, resName, chained, resSeq, x, y, z, possibility, pos, atom2 from TertiaryPartition where proteinName="HIV-1" order by torder;

The result is shown in figure 21:

Figure 3.21 The result for example 3.8

Example 3.9: Gets the secondary structure of protein named “HIV-1”.

Our interpreter will generate Protein-OODB query:

```sql
select p.secondary from p in Protein where p.proteinName = "HIV-1";
```

It will generate query in MySQL:

```sql
select name, serNum, ID, numStrands, initResName, startResName, initChainID, initSeqNum, sense, curAtom, curResName, curChainID, curResSeq, prevAtom, prevResName, prevChainID, prevResSeq, comments, length from SecondaryPartition where proteinName = "HIV-1" order by sorder;
```

The result is shown in the following figure:

![Figure 3.22 The result for example 3.9](image)

**Example 3.10:** delete a protein object whose name is “HIV-1”

```sql
delete (Protein)(Protein.proteinName = "HIV-1");
```
Then the interpreter generates object query in EYEDB:

```
select Protein.proteinName="HIV-1";
=5738.5.854641:oid
delete 5738.5.854641:oid;
= 5738.5.854641:oid
```

It will generate query in MySQL:

```
delete from Protein, PrimaryPartition, SecondaryPartition,
TertiaryPartition where proteinName = "HIV-1";
```

**Example 3.11:** gets the sequence of protein named “HIV-1”.

```
sequence("HIV-1");
```

```
ARG VAL GLU PRO GLU GLU ALA XLE
ARG VAL GLU PRO GLU GLU ALA XLE
PRO GLU ILE THR LEU TRP LYS ARG PRO LEU VAL THR ILE
LYS ILE GLY GLY GLY LEU LYS GLU ALA LEU LEU ASP THR
GLY ALA ASP ASP ASP VAL ILE GLU GLU MET SER LEU PRO
GLY ARG THR LYS PRO ILE MET ILE GLY GLY ILE GLY GLY
PHE ILE ILE GLY VAL ARG GLU TYR ASP GLU ILE ILE ILE GLU
ILE ALA GLY HIS LYS ALA ILE GLY THR VAL LEU VAL GLY
PRO THR PRO VAL ASN ILE ILE GLU ARG ASN LEU LEU LEU
GLN ILE GLY ALL THR LEU ASN PHE
PRO GLU ILE THR LEU TRP LYS ARG PRO LEU VAL THR ILE
LYS ILE GLY GLY GLY LEU LYS GLU ALA LEU LEU ASP THR
GLY ALA ASP ASP ASP VAL ILE GLU GLU MET SER LEU PRO
GLY ARG THR LYS PRO ILE MET ILE GLY GLY ILE GLY GLY
PHE ILE ILE GLY VAL ARG GLU TYR ASP GLU ILE ILE ILE GLU
ILE ALA GLY HIS LYS ALA ILE GLY THR VAL LEU VAL GLY
PRO THR PRO VAL ASN ILE ILE GLU ARG ASN LEU LEU LEU
GLN ILE GLY ALL THR LEU ASN PHE
```

Figure 3.23 The result for example 3.11

**Example 3.12:** gets the number of helix of protein named “HIV-1”.

```
noOfHelix("HIV-1");
```

The result is
Example 3.13: gets the nearest neighbor of protein named “HIV-1”.

nearestNeighbor3D(“HIV-1”);

The result is shown in the following snapshot:

3.6 Details on Using System Based on XML Database

We also converted Protein-OODB queries to XML database such as Orcale 10g so that we only need to convert Objects of EyeDB into String of object content in order to match data.

Protein schema file:

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="Protein" minOccurs="1" maxOccurs="1">
    <complexType>
      <sequence>
      </sequence>
    </complexType>
  </element>
</schema>```
<element name="proteinName" type="CHAR_30"/>
<element name="types" type="CHAR_30"/>
<element name="functions" type="CHAR_40"/>
<element name="remark" type="CHAR_200"/>
<element name="domainOfPrimaryStru" type="CHAR_400"/>
<element name="PrimaryPartition" minOccurs="1"
  maxOccurs="unbounded">
  <complexType>
    <sequence>
      <element name="proteinName" type="CHAR_30"/>
      <element name="porder" type="integer"/>
      <element name="name" type="CHAR_6"/>
      <element name="serNum" type="integer"/>
      <element name="chainName" type="CHAR_2"/>
      <element name="elementNo" type="long"/>
      <element name="seq" type="CHAR_200"/>
    </sequence>
  </complexType>
</element>
<element name="SecondaryPartition" minOccurs="1"
  maxOccurs="unbounded">
  <complexType>
    <sequence>
      <element name="proteinName" type="CHAR_30"/>
      <element name="sorder" type="integer"/>
      <element name="name" type="CHAR_6"/>
      <element name="serNum" type="integer"/>
    </sequence>
  </complexType>
</element>
<element name="ID" type="CHAR_4"/>
<element name="numStrands" type="integer" minOccurs="0" maxOccurs="1"/>
<element name="initSeqName" type="CHAR_3"/>
<element name="initChainID" type="CHAR_2"/>
<element name="initSeqNum" type="integer"/>
<element name="endSeqName" type="CHAR_3"/>
<element name="endChainID" type="CHAR_2"/>
<element name="endSeqNum" type="integer"/>
<element name="sense" type="integer" minOccurs="0" maxOccurs="1"/>
<element name="curAtom" type="CHAR_1" minOccurs="0" maxOccurs="1"/>
<element name="curResName" type="CHAR_3" minOccurs="0" maxOccurs="1"/>
<element name="curChainID" type="CHAR_2" minOccurs="0" maxOccurs="1"/>
<element name="curResSeq" type="integer" minOccurs="0" maxOccurs="1"/>
<element name="prevAtom" type="CHAR_1" minOccurs="0" maxOccurs="1"/>
<element name="prevResName" type="CHAR_3" minOccurs="0" maxOccurs="1"/>
<element name="prevChainID" type="CHAR_2" minOccurs="0" maxOccurs="1"/>
<element name="prevResSeq" type="integer" minOccurs="0" maxOccurs="1"/>
<element name="comments" type="CHAR_40" minOccurs="0" maxOccurs="1"/>
<element name="length" type="integer" minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element name="PrimaryPartition" minOccurs="1" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="proteinName" type="CHAR_30"/>
<element name="torder" type="integer"/>
<element name="name" type="CHAR_6"/>
<element name="serNum" type="integer"/>
<element name="atom1" type="CHAR_3"/>
<element name="resName" type="CHAR_3"/>
<element name="chained" type="CHAR_2" minOccurs="0" maxOccurs="1"/>
<element name="resSeq" type="integer" minOccurs="0" maxOccurs="1"/>
<element name="x" type="float" minOccurs="0" maxOccurs="1"/>
<element name="y" type="float" minOccurs="0" maxOccurs="1"/>
<element name="z" type="float" minOccurs="0" maxOccurs="1"/>
</sequence>
</complexType>
</element>
<element name="possibility" type="float" minOccurs="0"
maxOccurs="1"/>
<element name="pos" type="float" minOccurs="0"
maxOccurs="1"/>
<element name="atom2" type="CHAR_1" minOccurs="0"
maxOccurs="1"/>
</sequence>
</complexType>
</element>
</sequence>
</complexType>
</element>
</schema>
Protein DTD file:
<!ELEMENT Protein (proteinName, types*, functions*, remark*,
domainOfPrimary, PrimaryPartition*, SecondaryPartition*,
TertiaryPartition*)>
<!ELEMENT proteinName (#PCDATA)>
<!ELEMENT types (#PCDATA)>
<!ELEMENT functions (#PCDATA)>
<!ELEMENT remark (#PCDATA)>
<!ELEMENT domainOfPrimary (#PCDATA)>
<!ELEMENT PrimaryPartition (proteinName,porder, name,
serNum, chainName, elementNo, seq)>
<!ELEMENT porder (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT serNum (#PCDATA)>
<!ELEMENT chainName (#PCDATA)>  
<!ELEMENT elementNo (#PCDATA)>  
<!ELEMENT seq (#PCDATA)>  
<!ELEMENT SecondaryPartition (proteinName, sorder, name, serNum, ID, numStrands*, initResName, initChainID, initSeqNum, endResName, endChainID, endSeqNum, sense*, curAtom*, curResName*, curChainID*, curResSeq*, prevAtom*, prevResName*, prevChainID*, prevResSeq*, comments*, length*)>  
<!ELEMENT sorder (#PCDATA)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT serNum (#PCDATA)>  
<!ELEMENT ID (#PCDATA)>  
<!ELEMENT initResName (#PCDATA)>  
<!ELEMENT numStrands (#PCDATA)>  
<!ELEMENT initResName (#PCDATA)>  
<!ELEMENT initChainID (#PCDATA)>  
<!ELEMENT initSeqNum (#PCDATA)>  
<!ELEMENT endResName (#PCDATA)>  
<!ELEMENT endChainID (#PCDATA)>  
<!ELEMENT endSeqNum (#PCDATA)>  
<!ELEMENT sense (#PCDATA)>  
<!ELEMENT curAtom (#PCDATA)>  
<!ELEMENT curResName (#PCDATA)>  
<!ELEMENT curChainID (#PCDATA)>  
<!ELEMENT curResSeq (#PCDATA)>  
<!ELEMENT prevAtom (#PCDATA)>  
<!ELEMENT prevResName (#PCDATA)>
Example 3.14: Gets the protein named “HIV-1” (Abbreviation “HIV-1 Protease”).

(Protein)(Protein.proteinName = “HIV-1”);

Then the interpreter generates Protein-OODB query:

select p from p in Protein where p.proteinName = "HIV-1";

Then the interpreter generates query in XPath:
```sql
select extractValue(object_value,'/PrimaryPartition/name/text()'),
extractValue(object_value,'/PrimaryPartition/serNum/text()'),
extractValue(object_value,'/PrimaryPartition/chainName/text()'),
extractValue(object_value,'/PrimaryPartition/elementNo/text()'),
extractValue(object_value,'/PrimaryPartition/seq/text()')
from ProteinXML
where existsNode(object_value, '/PrimaryPartition')=1
and extractValue(object_value,'/proteinName')= 'HIV-1';

select extractValue(object_value,'/SecondaryPartition/name/text()'),
extractValue(object_value,'/SecondaryPartition/serNum/text()'),
extractValue(object_value,'/SecondaryPartition/ID/text()'),
extractValue(object_value,'/SecondaryPartition/numStrands/text()'),
ex}
```
Example 3.14: Gets the secondary structure of protein named “HIV-1”.

(Protein.secondary)(Protein.proteinName = "HIV-1");
Our interpreter will generate Protein-OODB query:

```plaintext
select p.secondary from p in Protein where p.proteinName = "HIV-1"
```

It will generate query in XPath:

```plaintext
select extractValue(object_value,'/SecondaryPartition/name/text()'),
extractValue(object_value,'/SecondaryPartition/serNum/text()'),
extractValue(object_value,'/SecondaryPartition/ID/text()'),
extractValue(object_value,'/SecondaryPartition/numStrands/text()'),
extractValue(object_value,'/SecondaryPartition/initResName/text()'),
extractValue(object_value,'/SecondaryPartition/initChainID/text()'),
extractValue(object_value,'/SecondaryPartition/initSeqNum/text()'),
extractValue(object_value,'/SecondaryPartition/endResName/text()'),
extractValue(object_value,'/SecondaryPartition/endChainID/text()'),
extractValue(object_value,'/SecondaryPartition/endSeqNum/text()'),
extractValue(object_value,'/SecondaryPartition/sense/text()'),
extractValue(object_value,'/SecondaryPartition/curAtom/text()'),
extractValue(object_value,'/SecondaryPartition/curResName/text()'),
extractValue(object_value,'/SecondaryPartition/curChainID/text()'),
extractValue(object_value,'/SecondaryPartition/curResSeq/text()'),
extractValue(object_value,'/SecondaryPartition/prevAtom/text()'),
extractValue(object_value,'/SecondaryPartition/prevResName/text()'),
extractValue(object_value,'/SecondaryPartition/prevChainID/text()'),
extractValue(object_value,'/SecondaryPartition/prevResSeq/text()'),
extractValue(object_value,'/SecondaryPartition/comments/text()'),
extractValue(object_value,'/SecondaryPartition/length/text()')
from ProteinXML where existsNode(object_value, '/SecondaryPartition')=1 and
extractValue(object_value,'/proteinName')= 'HIV-1';
```

The result is shown in the following figure:
Example 3.16: delete a protein object whose name is “HIV-1”

delete (Protein)(Protein.proteinName = "HIV-1");

Then the interpreter generates object query in EYEDB:

select Protein.proteinName="HIV-1";

=5738.5.854641:oid

delete 5738.5.854641:oid;

= 5738.5.854641:oid

It will generate query in XPath:

delete from ProteinXML where extractValue(OBJECT_VALUE, '/proteinName/text()') like 'HIV-1';

delete from ProteinXML where extractValue(OBJECT_VALUE, '/PrimaryPartition/proteinName/text()') like 'HIV-1';
delete from ProteinXML where extractValue(OBJECT_VALUE, '/SecondaryPartition/proteinName/text()') like 'HIV-1';
delete from ProteinXML where extractValue(OBJECT_VALUE, '/TertiaryPartition/proteinName/text()') like 'HIV-1';

Example 3.17: gets the sequence of protein named “HIV-1”.

sequence("HIV-1");

Figure 3.28 The result for example 3.17

Example 3.18: gets the number of helix of protein named “HIV-1”.

noOfHelix("HIV-1");

The result is
Example 3.19: gets the nearest neighbor of protein named “HIV-1”.

nearestNeighbor3D(“HIV-1”);

The result is shown in the following snapshot:
Chapter 4

PROTEIN ALGEBRA

Like we illustrated in chapter 3, our Domain Specific Object Oriented DataBase Management System (DSOODBMS) is used to manipulate Protein Data by providing Protein-QL and Protein-OODB to deal with the queries in protein domain which can be easily extended into other biological domains. In our application system, we can have two ways to match Protein-QL to Protein-OODB. One is to directly interpret Protein-QL syntax to Protein-OODB (shown in section 3), but the other uses Protein Algebra Architecture to connect them to optimize the queries in order to provide better performance for protein data management.

In this dissertation, we design Protein Algebra Architecture to optimize protein data queries. It has three components: Protein Ontology, Protein Algebra and Protein Wrapper. The Protein Algebra provides an extensible set of high-level genomics data types (GDTs) (e.g., genome, gene, chromosome, protein) and Protein Data Types (PDTs) (e.g. primary, secondary, tertiary) together with a comprehensive collection of appropriate genomic functions (e.g., translate, transcribe, decode) and protein functions (e.g., sequence, getPrimary, nearestNeighbour), it also provides genomics and protein operations to deal with protein domain specific object queries. Protein Ontology which is designed as a dictionary is used to map Protein Algebra to Protein-QL. Protein Wrapper connects Protein Algebra and Protein-OODB which makes Protein Algebra independent of Protein-OODB.
4.1 Protein Algebra Architecture

In our Protein-DSOODBMS, there are two ways to map Protein-QL and Protein-OODB, direct mapping and Protein Algebra Architecture mapping. Direct mapping does not provide good query optimization, so we make use of Protein Algebra Architecture to optimize object queries such that we can provide better performance for protein data management in green in the following figure.

Figure 4.1 The architecture of protein domain specific OODB management system
Protein Algebra Architecture is used to map Protein-QL and Protein-OODB which consists of three components: the Protein Ontology, the Protein Algebra, and the Protein Wrapper. The Protein Algebra module maps Protein-QL queries to Protein-OODB OQL requests and provides excellent object query optimization.

4.2 Protein Algebra

Our Protein Algebra is a domain specific high-level algebraic methodology which is different from traditional mathematical algebra. It can incorporate multiple data sources, high-level genomics and protein concepts and terminology. The algebra provides several domain specific data types (genomic data types – GDTs and protein data types – PDTs). In addition, it is independent of the software (OO-DBMS) that provides persistence and can be easily extended in case new data types or operations.

4.2.1 Protein Algebra Format

The formalization of our Protein Algebra is as follows:

\[
\begin{align*}
\text{sorts} & \quad \text{GDTs/PDTs} \mid \text{normal data types (such as string, int...)} \\
\text{ops} & \quad \text{genomics/protein operators}
\end{align*}
\]

Genomics operators can be *translate, transcribe and splice*. Protein operators ([2]) include *sequence, getPrimary*, etc. which have already been implemented in our Protein-OODBMS. By using this format, our Protein Algebra can easily query on genomics and protein domain data. For example:

\[
\begin{align*}
\text{sorts} & \quad \text{Protein, Sequence} \\
\text{ops} &
\end{align*}
\]
sequence: Protein \rightarrow Sequence

This simple algebra contains two PDTs for Protein and amino acid Sequence of protein primary structure as well as one operator, sequence, which returns its amino acid Sequence for a given Protein. The algebraic expression can be syntactically expressed as sequence(Protein).

4.2.2 Protein Algebra Data Types and Operations

Protein Algebra supports an extensible set of high-level genomics data types [5] (e.g. genome, gene, protein) and provides protein data types (PDTs) and collection of appropriate genomics and protein operations or functions such as transcribe, splice, sequence, getPrimary, nearestNeighbour and so on.

Firstly, we design new sorts called Protein Data Types (PDTs) (e.g. Protein, Primary, Secondary, Tertiary) and ops (protein operations, e.g. sequence, getPrimary, noOfChain, nearestNeighbor) which have same format as sorts and operators (ops) of GenAlg [5]. Some examples of sorts and ops are shown below:

i. sorts

    Protein, Sequence, length

    ops

    sequence: Protein \rightarrow Sequence

    lengthOfSequence: Sequence \rightarrow length

ii. sorts

    Protein, Primary

    ops

    getPrimary: Protein \rightarrow Primary

iii. sorts
iv. sorts

Protein, nearestNeighbour

ops

nearestNeighbour3D: Protein \rightarrow \text{Protein}

Secondly, we extend Genomics Data Types (GDTs) and Genomics Operations of GenAlg to protein’s structures in order to get more detailed information by providing more sorts and ops on protein data types (PDT) and protein operations. For example, we can extend sorts and ops of GenAlg as follows:

sorts

Gene, PrimarymRNA, mRNA, Protein, Primary, Sequence

ops

transcribe: Gene \rightarrow \text{PrimarymRNA}
splice: PrimarymRNA \rightarrow mRNA
translate: mRNA \rightarrow \text{Protein}
getPrimary: Protein \rightarrow Primary
sequence: Primary \rightarrow Sequence

Thirdly, our algebra can return multiple types in the same queries instead of single returned type result of GenAlg as following examples shows.

**Example 4.1:** Get the type and function of protein “HIV-1”. (“HIV-1” is an abbreviation of “HIV-1 Protease”)

sorts

Protein, String union String
ops
getTypes: Protein → String
union
getFunctions: Protein → String

Type and function can be returned in one query even though they are strings in different formats.

**Example 4.2:** Get the sequence and secondary structure of protein “HIV-1”.

sorts

Protein, Sequence union Secondary

ops

sequence: Protein → Sequence
union

getSecondary: Protein → Secondary

In our Protein-OODBMS, Sequence is string and Secondary is an object.

Finally, we can add some conditions or constraints on sort and ops which are very important for queries.

**Example 4.3:** Get the sub-sequence from position 0 to position 50 of protein.

sorts

Protein, subSequence

ops

subSequence: Protein → subSequence(0, 50)

**Example 4.4:** Get the sequence of protein which has the same sub-sequence as protein “HIV-1”.

sorts

Protein, Sequence
4.2.3 Protein Algebra Optimization

Our Protein Algebra provides query optimization for large database and complex queries to improve system performance. The basic idea of the optimization is as follows: Suppose that the query contains several constraints, Protein Algebra checks them starting from the inner-most one. It will stop query if present condition does not pass the checking, which saves time and optimizes queries. The following two examples illustrate how Protein Algebra optimizes the complex queries by using optimizer inside of Protein Algebra.

**Example 4.5:** Get the sequence of protein which the length is greater than the length of protein “HIV-1” and has the same sub-sequence as protein “HIV-1”.

```plaintext
ops
sequence:Protein(location(Protein.proteinName,
    subSequence("HIV-1",5,20))>=0) \rightarrow Sequence
```

Protein Algebra decides whether checking will go through next condition depending on the first condition `lengthOfSequence(Protein.proteinName) > lengthOfSequence("HIV-1")`, `location(Protein.proteinName, subSequence("HIV-1",5,20))>=0` \rightarrow Sequence

**Example 4.6:** Get the type of protein which has the same number of helix as protein “HIV-1” and the number of chain is greater than protein “HIV-1”.

```plaintext
sorts
    Protein, Sequence
ops
sequence:Protein(lengthOfSequence(Protein.proteinName)>
    lengthOfSequence("HIV-1"), location(Protein.proteinName,
    subSequence("HIV-1",5,20))>=0) \rightarrow Sequence
```

Protein Algebra decides whether checking will go through next condition depending on the first condition `lengthOfSequence(Protein.proteinName) > lengthOfSequence("HIV-1")` is true or not. Example 4.6 is similar as this one.

**Example 4.6:** Get the type of protein which has the same number of helix as protein “HIV-1” and the number of chain is greater than protein “HIV-1”.

```plaintext
sorts
```
Protein, String

ops

getTypes: Protein(noOfHelix(Protein.proteinName) ==
             noOfHelix("HIV-1"), noOfChain(Protein.proteinName) >
             noOfChain("HIV-1")) → String

4.3 Protein Ontology

Ontology is a controlled vocabulary to describe the functions, process and components
for specific domains and used by people, databases, and applications to share domain
information [6]. In the computer world, ontology is known as a machine-readable
vocabulary that is specified with enough precision to allow differing terms to be precisely
related. Ontology enables users to share data, reuse and analyze domain data, especially
for complicated biological data. But due to different goals and/or shortcomings of
existing ontologies, we attempt to design an ontology called Protein Ontology to resolve
syntactic, terminological and semantic differences which are induced by multiple protein
data sources. Our Protein Ontology is capable of defining and identifying genomics and
protein data objects, data operations and terminologies. It is also able to solve the
problems of identical protein information represented differently in different data sources
and same name used in the distinct concepts in different research to remove protein data
ambiguity, incompatibility and inconsistency.

The Protein Ontology is designed as a dictionary to map Protein-QL [2] to Protein
Algebra. The following examples show how it works.

Example 4.7: Get the primary structure of protein “HIV-1”.

Protein-QL query is:

(Protein.primary)(Protein.proteinName="HIV-1");
Protein Ontology will map it into Protein Algebra as follows:

```
sorts
    Protein, Primary

ops
    getPrimary: Protein → Primary
```

**Example 4.8:** Get the type and function of protein “HIV-1”.

Protein-QL query is:

```protein-ql
(Protein.types,Protein.functions)(Protein.proteinName= "HIV-1");
```

It will be translated by Protein Ontology into Protein Algebra as follows:

```
sorts
    Protein, String union String

ops
    getTypes: Protein → String
    union
    getFunctions: Protein → String
```

**Example 4.9:** Get the sequence of protein which the length is greater than the length of protein “HIV-1” and has the same sub-sequence as protein “HIV-1”.

Protein-QL query is as follows:

```protein-ql
(sequence(Protein.proteinName))(lengthOfSequence(Protein.proteinName)>lengthOfSequence("HIV-1"),
location(Protein.proteinName,subSequence("HIV-1",5,20))>= 0)
```

It should be mapped to Protein Algebra as follows:

```
sorts
    Protein, Sequence
```
The Protein Ontology has two important goals. The first one is to identify the objects in genomics and protein domains. The second one is to interpret Protein-QL queries to Protein Algebra and remove the data ambiguity, incompatibility and inconsistency by defining genomics and protein domain specific terminologies to describe the syntax and semantics.

4.4 Protein Wrapper

The Protein Wrapper encapsulates the knowledge of Protein-OODB and provides a pathway from Protein Algebra to Protein-OODB, which makes our Protein Algebra independent of underlying database. Thus we only need to recode our Protein Wrapper without changing Protein Algebra if Protein Algebra is integrated into other data sources. In addition, Protein Wrapper can interpret Protein Algebra with query optimization to Protein-OODB. Consider the following two query examples:

**Example 4.10:** Get the primary structure of protein “HIV-1”.

```plaintext
sorts
    Protein, Primary

ops
    getPrimary: Protein → Primary
```

This algebraic expression contains two PDTs -- Protein and Primary as well as one operator `getPrimary`. It is translated into Protein-OODB as follows:

```
select p.Primary from Protein p where p.proteinName="HIV-1";
```
Example 4.11: Get the type and function of protein “HIV-1”.

sorts
    Protein, String union String
ops
    getTypes:       Protein \rightarrow String
                     union
    getFunctions:   Protein \rightarrow String

This algebraic expression can be translated into Protein-OODB as follows:

select p.types, p.functions from Protein p where p.proteinName="HIV-1";

Examples 4.10 and 4.11 show a general format that Protein Wrapper translates Protein Algebra to Protein-OODB. The following examples 4.12 and 4.13 illustrate how Protein Wrapper interprets Protein Algebra optimization queries to Protein-OODB queries. In these two examples, Protein Wrapper will interpret and at the same time check the conditions. It will continue to check next condition if present one is satisfied. Otherwise, Protein Wrapper will stop translation and send result back to Protein Algebra.

Example 4.12: Get the sequence of protein which the length is greater than the length of protein “HIV-1” and has the same sub-sequence as protein “HIV-1”.

sorts
    Protein, Sequence
ops
    sequence:Protein(lengthOfSequence(Protein.proteinName)>lengthOfSequence(“HIV-1"),location(Protein.proteinName,subSequence(“HIV-1", 5, 20)) >= 0) \rightarrow Sequence

This query can be translated into Protein-OODB as follows:
select sequence(p.proteinName)
from (select p.proteinName
from Protein p
where (lengthOfSequence(p.proteinName) > 
    lengthOfSequence("HIV-1"))
where location(p.proteinName, subSequence("HIV-1", 5, 20) >= 0);

If the condition  
lengthOfSequence(p.proteinName) > 
lengthOfSequence("HIV-1") is true, then the translation will go through
following conditions. Otherwise the translation will stop for this protein and start next
translation for another protein.

Example 4.13: Get the type of protein which has the same number of helix as the one of
protein “HIV-1” and the number of chain is greater than protein “HIV-1”.

sorts
    Protein, String
ops
    getTypes: Protein(noOfHelix(Protein.proteinName)
    == noOfHelix("HIV-1"), noOfChain(Protein.proteinName)
    > noOfChain("HIV-1")) \rightarrow String

This algebraic expression will be translated into Protein-OODB as follows:

select p.types
from (select p from Protein p
    where (noOfHelix(p.proteinName) > noOfHelix("HIV-1"))
where noOfChain(p.proteinName) > noOfChain("HIV-1");

Protein Wrapper translates algebra according to the order of Protein Algebra constraints
without losing any optimization of Protein Algebra.
### 4.5 Protein Algebra Architecture Conclusion

In our Protein-DSOODBMS, we design Protein Algebra Architecture including Protein Ontology, Protein Algebra and Protein Wrapper to connect Protein-QL and Protein-OODB. This approach is based on performing transformations over an object algebra called Protein Algebra to deal with object-oriented query processing which uses object query optimization to provide better performance for large databases. The Protein Algebra as the basis for query processing to define algebraic transformations provides an extensible set of high-level Genomic Data Types and Protein Data Types together with a comprehensive collection of appropriate genomics and protein functions, it also provides genomics and protein operations to handle with domain specific object queries. It is mapped to Protein-QL through Protein-Ontology which is designed as a dictionary and it is connected to Protein-OODB by Protein Wrapper such that Protein Algebra is independent of Protein-OODB.
Chapter 5

DATA CURATION

As we mentioned, protein data size is very huge, the authors may lose some data or change some part of experiment data without any intention when they use data. These situations may result in the missing or wrong data information, 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 [7].

5.1 Data Curation in EYEDB

EYEDB is one type of Object-Oriented database providing 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 EYEDB Constraint Language. Now we can change PrimaryPartition definition as follows:

```c
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 EYEDB. This module is invoked immediately after a PDB file is converted into object-oriented form in EYEDB thereby ensuring good data quality in the protein OO database. In addition, we can apply the three protein structure information of PDB files to enforce constraints such as the length of each chain, positions, amino acid types, sense of each strand. For example we can check the positions of residues of secondary structure according to tertiary structure checking by EYEDB Constraint Language which will give us more cleaner and confident data.

5.2 PDB Data Curation

As we know, most protein data are stored in PDB format that is public and easy understanding. PDB (Protein Data Bank) is a centralized repository of protein structures founded in 1971 at Brookhaven National Laboratory, USA [8]. The PDB filename ends with .pdb. PDB file specifies the positions in space of every atom in a molecule, which includes atomic coordinates, a header which gives information about the model embodied in the coordinates. The PDB files for X-ray crystallography vary widely in quality, and rarely are they grossly incorrect [8]. Unfortunately, PDB supports several different data formats, thus it suffers from inconsistencies in how the data formats are constructed over time. PDB files do not contain complete bond information for biopolymers, which chemical bonds must be reconstructed by the reading/viewing software based on chemistry tables and known bond rules. Since different software can interpret these rules differently, software can be inconsistent the way it draws bonds in PDB-based structures and may produce undesired redundancy [9]. For example, the sequence given in the PDB SEQRES records is compared against the sequence derived from the coordinate records
that can show redundant data. In addition, the PDB is inconsistent in many other aspects, such as lack of EC numbers, compound naming, inconsistent chain labeling, residue numbering different between related structures and the sequence databases [10]. PDB format is old, ambiguous, and inadequate, but it is still the most widely used format because all relevant software can read it. Although some users claim they do not mind the errors of PDB file, which maybe result in the partial or wrong information to affect the accuracy of derived and propagated data. It is very necessary to develop a system to curate data in PDB file to provide easy understanding and better services for protein scientists and computer scientists.

In this dissertation, we propose two architectures for PDB data curation. One simply uses Checking Filter and Curation Engine to curate data. The other one improves the first one by adding XCML curation part to further curate data.

5.2.1. The PDB Data Curation System

Figure 5.1 shows this PDB Data Curation System. From User Interface, Checking Filter gets PDB file, which checks the errors and reports them to Curation Engine. And then
curated data by Curation Engine are sent to Database storage. The users can get better PDB file from our database after this process.

Most errors are fixed in original PDB file after it is through our PDB Data Curation System. Although some curations are not automatic, they actually help improve the PDB data.

5.2.2. Major Issues of PDB Data

As we described above, the inconsistency, redundancy and other problems may occur in PDB file. Following will show possible solutions for these issues.

5.2.2.1 Data Identification

PDB users usually want to know who generates the data, which paper, which lab, which date so that users can identify the respects of PDB data.

In PDB file, first 6 columns are not changeable in the process of data generation, but some users may change them without any intention such as changing AUTHOR to AUTHRO, TITLE to ITTLE, and propagate these errors to others who pull out data from them. Therefore our system is designed to automatically check those spelling errors and provide users the authority to change them. In addition, our data curation system also gives information about missing data such as no HEADER, no AUTHOR, no Date, and so on. Since header of PDB file is very important that contains information about the original literature citation, full names of ligands, optionally residues constituting various functional sites, etc, it is very necessary to include it in PDB file. And author, paper and lab give us information about respects of different data resources, which can be made use of confirming the confidence of data. Date in PDB file headers shows the time that the atomic coordinates were deposited, modified and published at the Protein Data Bank,
which helps keep data up-to-date. Therefore, identifying these data will give users very useful and helpful information.

5.2.2.2. Data Errors

- Data errors in original PDB file: The users can report to our system if they find the errors of PDB file. After we get original authors’ approval and correct them with marking the reason, author, date for changed part, then we resubmit. The reason we do in this way is that different parts of PDB file have different authorities, some parts are generated automatically by system, some are only used by primary authors, some are managed by PDB staff, others are controlled by submitters.

- Data errors generated when users change data: The size of PDB file is very huge such that users may remove or change some data without any intention when they are using them. We store the original data and changed parts of PDB file in our storage to avoid this kind of errors. If users have changed data, the system sends alert signal which can avoid users to change data without any attention. But if users really want to change data, our system compares the current file and previous file to avoid the users to make mistake. The system will suggest users to use the previous data file if it finds the errors from new data file.

- Data format errors: Our system retains a normal PDB file format which other PDB file can be very easily converted. If the current PDB file has any incompatible items corresponding to it, system will report them to the users such that users can correct them to make data conform to the common format.
5.2.2.3. Redundant Data

PDB data may be redundant which wastes storage and users’ time, therefore our system provides filters to automatically or manually check data redundancy. The system can remove redundant data without any loss of accuracy of data to make sure that data file does not have repeatable data by using following method—the system firstly checks REMARK part of PDB file and sees whether there is redundant data or not, if it has, the system will compare protein sequence (primary structure) and atomic structure (tertiary structure) to get rid of duplicated part in protein sequences by applying structures matching method.

5.2.2.4. Ambiguous data

PDB data may have different names for the same object. For example, some PDB files have ambiguous side chain atoms ("AE1", "AE2" atoms on Gly residues in 1ECO and 5CYT). It can be fixed by defining those ambiguous side chain atoms in a hash table which lists all different names for each object (in this case, showing all ambiguous side chain atoms for same thing). The system will automatically check them when they appear in the protein structures. In addition, the different objects may have the same name in PDB file that are difficult to check. But we can check them by saving all corresponding objects for each name in a table. We manually check them according to the protein structures, if they are not the same thing in the structures, We will give them different names.

5.2.2.5. Heterogeneous Data

There are so many PDB resources on the web sites. Different labs may apply different formats to save their experiments’ results which makes them difficult be understood.
Building a table to store the respects (confidence) of different labs (or submitters) provides the users convenience to use the most trustful data to avoid confusion.

5.2.2.6. Inconsistent Data

The Validation Suite and Server used by PDB can check the coordinate format and validate the overall structure before deposition which provides sequence/coordinate alignment and data inconsistencies [11]. Data inconsistencies can be fixed by changing PDB file to mmCIF format and converting mmCIF back to PDB file in [12]. For the simple inconsistent data such as the same protein has different names, which can be checked by collecting all synonyms in a dictionary, the system will automatically check them when PDB file is submitted into the PDB Data Curation System. For complicated data inconsistencies that are found by Validation Suite and Server of PDB, our system can manually amend according to the protein structures and data information.

5.2.2.7. Conflict Data

Our system provides module checker which applies machine learning combing fuzzy method to check conflict data according to the protein data structures, and automatically correct them without users’ involvement. According to three protein structures, the system can remove the conflict data in any protein structure. For example, the amino acids of primary structure can be checked against protein tertiary structure by using some extra constraints and assertions.

5.2.2.8. Obsolete Data

As we know, life science has a tremendous amount of data and update very often. In order to satisfy the users’ need, our system tries to keep following the pace of PDB data
appearance to update old data and add new data types once they appear. For this part, the system will manually check the new data entries every month and store both if it finds the new one for existing data so that users can conveniently keep track of the provenance for specific data.

The following figures show the user interface of our PDB Data Curation System and some parts of results:

![Image](image_url)

**Figure 5.2 PDB Data Curation Interface**
Figure 5.3 PDB File View in .txt of PDB Data Curation System

The following figure shows the services of PDB Data Clean:
The Data Identification Errors can be handled by the following services:
If PDB file has author spelling errors, our system will deal it with the following service (Header missing error also can be solved by using similar way in our system):

![Figure 5.6 Author Spelling Error in PDB Data Identification](image)

Figure 5.6 Author Spelling Error in PDB Data Identification

### 5.2.3 The Architecture of PDB Data Curation System with XCML

The eXtensible Markup Language (XML) is self-describing and is standardized by the World Wide Web Consortium (W3C). It is human and machine readable, flexible, extensible and has already become the standard format for exchanging information through the network [13]. It usually uses XML Schema or Document Type Definition (DTD) to define the syntax and data types. The data source will generate XML data according to their Schema definition or DTD when data are parsed. But it can not verify semantic constraints, avoid erroneous data and is domain dependent. Therefore, users will not get any better data if system just converts PDB file into XML format as PDBj-ML [14] did. In order to avoid the disadvantages of XML and make the use of XML’s merits to get better data, we apply an XML constraint language, eXtensible Constraint Markup Language (XCML) [15], which is more powerful and supports the specification of dynamic and inter-relationship constraints, to add some extra constraints and assertions in XML format so that the system can further curate data during the data conversion.
The following architecture shows how to improve our PDB Data Curation System by using XCML. Firstly, PDB file is submitted to PDB Data Curation System, after curation PDB file can be converted into XCML format (PDB-XML) by ConverterXML which uses some constraints to further curate. PDB-XML can be in Protein-OODB [16] format after it is handled by ConverterPDB and ConverterOODB. At this moment, users can get better data that are in domain specific object-oriented format.

The following are some examples by using XCML for PDB file:

Example 1. We can add one constraint to make sure that the length of each helix is equal to the difference of end position and begin position (pdbx_PDB_helix_length = end_auth_seq_id - beg_auth_seq_id + 1). The system can check the correctness of length for each helix generated by using this constraint. But the system need to define pdbx_PDB_helix_length, end_auth_seq_id and beg_auth_seq_id as integers instead of strings as PDBj_ML defined.
PDBj_ML format (Assume end_auth_seq_id and beg_auth_seq_id are already defined):

```xml
<xsd:element name="pdbx_PDB_helix_length" minOccurs="0" maxOccurs="1" nillable="true" type="xsd:string" >
</xsd:element>
```

XCML format:

```xml
<Constraint context="pdbx_PDB_helix_length">
  <Parameter>
    <name>pdbx_PDB_helix_length</name>
    <type>number</type>
  </Parameter>

  <Assertion test="pdbx_PDB_helix_length=($end_auth_seq_id-$beg_auth_seq_id+1)"/>
</Constraint>
```

Example 2. The system uses constraints to ensure that the element of each sheet is exactly on the specific chain of protein sequences.

PDBj_ML part:

```xml
<xsd:element name="pdbx_nonpoly_scheme" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:all>
      ......
      <xsd:attribute name="ndb_seq_num" use="required" type="xsd:string" >
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```
The test part of assertion can check whether end_label_seq_id is equal to ndb_seq_num or not which can check the consistency of data.

Example 3. We assume here the three structures of protein in the PDB file are separately generated which tertiary structure is not produced from primary and secondary structures. The system can add some constraints to calculate the number of amino acids
of each chain in primary structure (sequences) and tertiary structure (atoms) by using a counter to calculate how many elements there are in each sheet.

Firstly, the constraint must guarantee they are the same chain. (Here we did not give PDBj_ML format)

```xml
<Constraint context="pdbx_poly_seq_scheme">
    
    <Parameter>
        <name>seq_id</name>
        <type>number</type>
    </Parameter>

</Constraint>

<Constraint context="atom_site">
    
    <Parameter>
        <name>label_comp_id</name>
        <type>number</type>
    </Parameter>

    <Assertion test="count(label_comp_id with distinct label_seq_id)=$seq_id"/>

</Constraint>
```

5.2.4 Data Provenance

Data come from different sources and produced by different methods vary in the degree of real reliability. Also data that has been transformed multiple times is more likely to have been incorrectly transformed or lose an important context element [17]. In order to evaluate data’s reusability, it is necessary to understand the details of its collection. So
data provenance is playing a crucial role in PDB file. In our system, we emphasize on solving the following two aspects of data provenance:

1. Same question different answers from different resources: As we know, 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 or referencing multiple data sources. Data sources themselves may include a lot of different information. Therefore same question may get different answers from different resources when users query from the web sites. The results from our system come from the most confidential data sources of our respects’ table. If the most confidential data is obsolete, the system will use greedy algorithm to search up-to-date data with the more confidence.

Algorithm:

   Get the result set S from data sources
   Check the confidence of set S according to the respects’ table and get the most confidential one MCS
   Loop
      If the MCS is obsolete, then find the next most confidential one MCS
      Until get the more confidential and up-to-date result
   End loop

2. Uncertain probability: The PDB file has a very huge data size and used by different-level users, there may be some errors in the file. So updates are often given to PDB file that may produce erroneous data by faulty experimental procedure or by a breakdown in the process to result in uncertain data and
uncertain probability of data will propagate in the process. In order to solve this problem, we design a method to measure the uncertainty of data by using “fuzzy” method. In this method, the algorithm chooses the most certain one for each data source from certainty table in order to get the most certain data from propagation process.

Algorithm:

For each level data source

   Use the biggest rate certainty/uncertainty from the certainty table

   Until the last level

End for
Chapter 6

Related Work

6.1 NeuroDM

In [18], we proposed a novel neuron data model with a neuron domain specific query language. This data model is composed of five components: neuron data structure, constraints, operations, domain-specific query language (NeuroQL), and controlled vocabulary. It can capture and enforce more domain semantics than traditional data models, such as attributes of attributes and source information. It also enables users to model their special interests and constraints on any entity or its attribute in the data model. NeuroQL defines a list of queries in neuroscience terms so that users can easily master it. A few basic operations are defined in NeuroDM and form the middleware.

Figure 6.1. Concept Architecture of NeuroMD taken from [18]
between NeuroQL and DBMSs. NeuroQL consists of a list of queries that defined in neuroscience terms. NeuroQL enables neuroscientists to query the database using their own language without much syntactical restriction. NeuroQL can be easily extended because it is independent from the DBMSs, on which the database is built. The architecture is shown in the figure 6.1.

6.2 Genomics Project

The U.S. Human Genome Project began from 1990, now it can 1) identify all the approximately 20,000-25,000 genes in human DNA, 2) determine the sequences of the 3 billion chemical base pairs that make up human DNA, 3) store this information in databases, 4) improve tools for data analysis, transfer related technologies to the private sector, 5) address the ethical, legal, and social issues (ELSI) that may arise from the project [19]. It provided a lot of tools to query genomics data, in the similar way we want to have more tools on protein data query.

In Genomics Algebra (GenAlg), it provides an extensible set of high-level genomic data types (GDTs) (e.g., genome, gene, chromosome, protein, nucleotide) together with a comprehensive collection of appropriate genomic functions (e.g., translate, transcribe, decode). It also provides the Unifying Database to manage the semi-structured contents of publicly available genomic repositories and to transfer these data into GDT values. These values then serve as arguments of Genomics Algebra operations, which can be embedded into a DBMS query language.[20] But it did not provide further details on Protein Data Types (e.g. primary, secondary, tertiary) and appropriate protein functions (e.g., sequence, getPrimary, nearestNeighbour).
6.3 Protein Project

The most researchers provide protein data in PDB format which is in flat file. Therefore, users can not conveniently look at protein primary structure, secondary structure and tertiary structure. Thus it is urgent to develop a new database system to analyze PDB files to support protein data structure, and can provide domain specific query language such that protein domain users can use it to do research on protein structure data.
Chapter 7

CONCLUSION

This dissertation shows an application system—high level programming environment system for protein structure data that is protein domain specific and implemented by adding some new features (such as some internal protein data types, protein-QL, Protein Algebra Architecture and protein-OODB) to existing OODB—EYEDB. This system has three components, User Interface, Middleware and Data Storage. Middleware is designed to implement protein domain specific query language and optimize the complex queries, also it encapsulates the details of the implementation such that users can easily understand and master Protein-QL. It includes Protein-QL, Protein Algebra Architecture and Protein-OODB: 1) Protein-QL provides convenience for users to store, retrieve, and modify data, and define four protein internal data types—protein, primary, secondary and tertiary to simplify the queries so that the users can easily understand and send requests, it also defines some basic and domain specific operations. 2) Protein Algebra Architecture is used to optimize the complicated queries to save time and energy. 3) Protein-OODB solves some protein data source problems and is used to connect Protein-QL and EYEDB (Data Storage). Data Storage is used to store our protein data. This application system shows a general idea to develop a new system for a certain biological domain. It is very easy to extend this system into other biological domains.

EYEDB is our underlying database because it is more mature than Lambda-DB and we used RMI to transform the transactions between our system and client as figure 7.1 shows.
Figure 7.1 The architecture of high level programming environment system for protein structure
Chapter 8

FUTURE WORK

In the future, we plan to extend our Protein Domain Specific OODB Management System (Protein-OODBMS) to provide wider services for different kinds of users. That means, the system not only allows users to input and output XML queries, but also provides a few databases for users to choose for their queries by which makes it independent of underlying databases such that users request protein data in Object Oriented format, but data can be stored in multiple formats in OODB, relational DB, XML DB or other data storages.

We will make the system support a mapping that links terms used in the databases to their semantic meaning such that terms appearing in different databases, but representing the same data, can be treated as equals from the point of user API. The system uses ontology in protein domain knowledge to provide the necessary database semantics [21]. The aim of the system is to create domain-specific database management that is user-oriented and independent of underlying databases and Protein-QL supports domain-specific objects, operators, functions, and ontology of the system is able to capture domain semantics to solve ambiguity and heterogeneity of data. We plan to make our new system be able to be easily applied to any other biological domain and users also can integrate this system to other application system as shown in the figure 8.1:

1. Clients use our protein domain specific query language to request results from database which depends on the server provider.
Figure 8.1. The architecture of domain-specific database management system for protein structure data

2. XML Converter and XML Back Converter are used to make our OODBMS independent of databases by which Protein-OODB can be converted into OODB format, relational DB format and XML DB format, and so on. XML Back Converter will translate Protein-QL into ODB, SQL, XQuery and other query languages in XML format according to the client’s requests for specific database.
3. There are specific database schemas in XML Converter such that queries can be translated into specific database.

4. Handler in Protein-OODB is in charge of distributing queries to specific server according to the client’s request.

5. XML Translator is used to translate Protein queries in XML format to Protein-QL.

6. Database ID is sent to Handler of Protein-OODB by Server Listener after client chooses database from which data results are generated. Then Handler decides to send queries to OODB or to XML Converter with Database ID.

7. Improving User API such as Visualization, Data Browser, Java Client, PQL Plus and PDB Expert to provide more convenient service for different users.


9. It is easily integrated to other biological domains.

10. The green parts are already done in figure 8.1.

We also plan to develop a biological system for biology called Bio-OODBMS shown in figure 8.2. Presently, most protein data are stored in PDB format, so we plan to add PDB data curation system [22] into our current system so that the system can get much cleaner and more confident PDB data when clients use them.
Figure 8.2 Domain specific object-oriented database management system for biology
REFERENCES

1. Wikipedia definitions.


3. Protein secondary structure.

4. Protein Data Structure


8. Protein Data Bank.


11. PDB Extract.


31. Controlled vocabulary.


36. Tian, H., R. Sunderraman, and H. Yang, A Domain-Specific Conceptual Data Modeling and Querying Methodology.


42. *eXtensible Markup Language*.


44. Lord, P., A. Macdonald, and e. al., *From Data Deluge to Data Curation*. 2004.


Report on Three Genomes to Life Workshops: Data Infrastructure, Modeling and Simulation, and Protein Structure Prediction”


63. Ontology.


65. PDB from RCSB.


75. Semantics.


