# PDBTool: A Prototype Object Oriented Toolkit for Protein Structure Verification

C. Pu[1] K. P. Sheka,
J. Ong, L. Chang,
A. Chang, and E. Alessio

Department of Computer Science
Columbia University
New York, NY 10027
calton@cs.columbia.edu

I.N. Shindyalov, W. Chang, and
P.E. Bourne[1]

Department of Biochemistry
and Molecular Biophysics
Columbia University
New York, NY 10032
system@cuhhca.hhmi.columbia.edu

Technical Report Number CUCS-048-92
Department of Computer Science
Columbia University

## Abstract

The Protein Data Bank (PDB) is a large and rapidly growing standard repository of complex biological macromolecules. Existing software tools for the PDB, including storage management, are quite inflexible and work in isolation. We are using object-oriented design and object-oriented database techniques in the construction of a toolbox for the PDB. The main components of the object-oriented toolbox include: (1) a uniform graphical user interface for fast, intuitive, and interactive access to the PDB; (2) several interoperable storage managers, compatible with present and future PDB formats; (3) a new set of efficient, modular, and flexible software tools. The toolbox is being implemented in $C^{++}$. We describe our experience in the the implementation and use of the first prototype, PDBTool, to display, manipulate, and verify the structure of proteins. This project is a successful collaboration of a Biophysics and a Computer Science research and development team.

[1]partially supported by the National Science Foundation under grant IRI 9116798.

# 1 Introduction

The Protein Data Bank (PDB) [1] is the international repository for 3-dimensional structural information on biological macromolecules. PDB includes data derived from X-ray crystallography, Nuclear Magnetic Resonance, and theoretical simulation. The PDB is maintained and distributed by the Chemistry Department at the Brookhaven National Laboratory (BNL). The June 1992 distribution of the PDB contained approximately 1000 structures distributed as a set of ASCII files for a total of 300MB. Like other biological databases such as the Genome Data Bank, the PDB is growing exponentially, and at least 10,000 structures are expected by the year 2000. PDB is a potentially rich source of information for innovative biological research.

Unfortunately, the current distribution of PDB uses a FORTRAN card format makes the information access difficult. In particular, PDB data organization makes query relative to multiple structures, an important inference if structure-function relationships are to be established, difficult and therefore impedes scientific enquiry. Moreover, a very useful AUTHORIN tool, which checks the integrity of a structural submission prior to submission to BNL and also used by BNL to check submissions on arrival, is not publicly available at this time and hence the time to process and register a structure in the PDB is unnecessarily long.

To overcome these difficulties, we are building the Object Oriented Protein Data Bank (OOPDB) [5] toolbox, which will provide structure query capability and operate in a distributed environment over heterogeneous and interoperable databases. Our first goal is to build a prototype persistent and easily queried database for all protein structures and at the same time a verification tool to check the integrity of a single structure. The long-term goal of the project is to support different database back-ends such as object-oriented or extended relational, with a Graphical Users Interface (GUI) front-end to analyze, query, update and graphically represent protein structure. In short, we intend to create a object-based tools using state-of-the-art software engineering techniques; these tools will become an innovative and efficient software instrument for biologists.

This report describes the architecture, initial implementation, and features of PDBTool, a prototype structure verification tool. PDBTool illustrates many of the features in terms of data structures, query methods, GUI, and standard interfaces that we anticipate to include in OOPDB [5]; persistence and methods for managing persistence objects are being included in the adoption of ObjectStore, a commercial object-oriented database management system (OODBMS) supplied by Object Design, Inc. This prototype results from a joint effort between the departments of Computer Science and Biochemistry and Molecular Biophysics at Columbia University. This collaboration is the only funded project between these two departments at this time.

## 2 Related Work

Several efforts are in progress to better manage PDB data to assist in scientific query. SESAM [8] and NDB [2] are based on relational data model, using the Sybase commercial database system. NDB includes exclusively DNA and RNA structures (no complex proteins). Idisis [10] is also relational and originally used the ORACLE commercial database system. Idisis now has its own proprietary storage manager. An alternative approach is used by PKB [4] which although relational in structure is coded in S, a language for statistical analysis. These relational models differ in their database schema, as well as the number and type of derived variables and how those variables are organized. None of these relational databases effectively addresses the impedance mismatch problem between the flat relational tables and complex protein structures. None of these database efforts uses a GUI at this time. To alleviate the difficulties of writing SQL queries, SESAM uses ALI.

The only object-based effort is P/FDM [9]. P/FDM is implemented in Prolog and supports DAPLEX, a functional language for queries into the database. The main purpose of P/FDM is to investigate schema definition and rigorous modeling of protein structures. Being more practical and concerned with efficiency and usability, OOPDB is seen to complement the more formal P/FDM approach. At this time there are no object-based verification tools for testing the reliability of a macromolecular structure, although a number of individual FORTRAN programs exist. A subset of these FORTRAN programs are available in the Crystallographic Workbench (CW) [3] and are used to validate the new methods developed for PDBTool.

## 3 Internal Structure and Interface

### 3.1 User Requirements

One advantage in our collaboration is the presence of biologists in the definition of user requirements for the project. Since we are concerned with the practical use of the OOPDB toolbox, the biologist views are taken very seriously. The requirements on the toolbox can be divided into three parts: user interaction with PDB, programmer interface with the toolbox, and interoperability. We will briefly summarize each one in turn.

User interaction with the PDB must be natural, with the software as imperceptible as possible. To take the software "out of the way" of biologists, a simple GUI was chosen. The GUI must present protein structures at different abstraction levels, namely: polypeptide chain, secondary structure, amino acid residue, and atom. Furthermore, the GUI must be responsive, with quick real-time feedback to user commands to view or edit the molecular structure displayed. From this uniform GUI users should be able to invoke the software tools from the toolbox through simple menus and the results shown immediately on the screen.

Similarly to the user interaction requirements, the programmer interface to the tools must be simple and natural. The different abstraction levels enumerated above should be accessible to programmers without details of particular data structures. The OOPDB toolbox should hide the technical details of data representation and invocation interfaces as much as possible. Therefore, scientific programmers that write the tools can concentrate on the specific scientific mission of their tool, instead of syntax for input/output, query language, or data structure parsing. Furthermore, the integration of tools into the toolbox should be simple and easy, if the programmers follow our programming interface.

Finally, to reach the widest possible segment of the user community, we want interoperability at all levels. At the basic level we support open system hardware and software foundations for our toolbox, UNIX systems running on SUN and SGI workstations. The other components of our system are also standard: FORTRAN-90, ANSI C, and AT&T $C^{++}$ programming languages, plus X windows and Motif for windows. However, we have to support a number of storage managers since different tools adopt different formats: PDB file format, the new Crystallographic Information File (CIF) format, the relational data stored in Sybase (SESAM), and some representative OODBMS.

### 3.2 Object Oriented Approach

From the requirements above, we have chosen an object oriented approach to manage our data. The complex structure of protein molecules map well into the structuring capabilities of object-oriented systems, particularly OODBMS. Object identities, object and data encapsulation, data abstraction, as well as inheritance for its hierarchical structure, have helped us manage and understand protein structures in a much simpler way.

Our PDB schema definition (Figure 1) is derived from our experience in manipulating PDB data and previous work, notably P/FDM [9]. We start with the definition of a separate class for each corresponding abstraction level of proteins: protein, chain, secondary structure (derived classes: helix, strand, turn), residue, atom. The relationships between these basic data classes are captured in operational classes that manipulate data classes. For example, each protein can enumerate its chains through a $C^{++}$ `iterator`, and chains can enumerate their residues, and residues can enumerate their atoms.

One advantage of object-oriented approach is extensibility. Currently we don't define data source specific information in separate classes. For example, attributes in PDB files or CIF generated files are attached to each molecule. In the future we will separately define those classes, so we may have *PDB_Protein* which will inherit, both, from Protein and *PDB_Protein_Info*, or

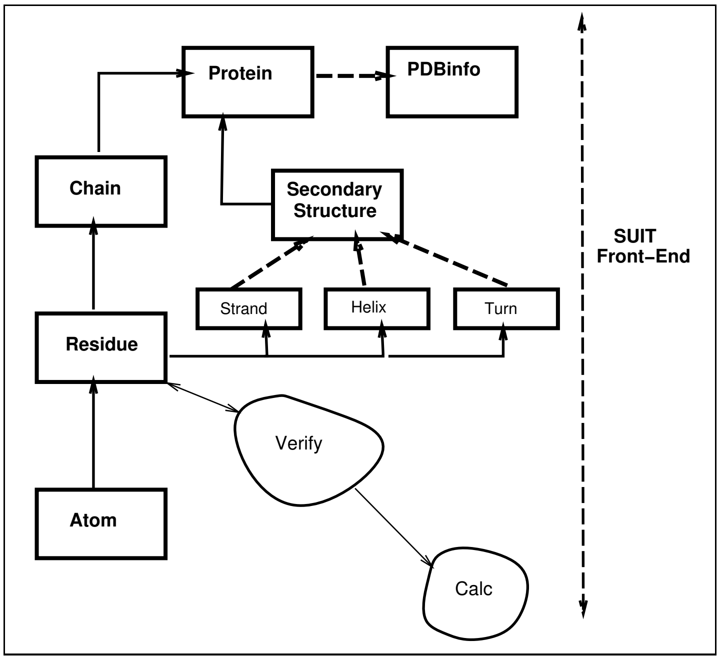


Figure 1: PDBTool Schema

*CIF_Protein* which will inherit from Protein and *CIF_Protein_Info*. Since current tools do not use this information, the decision on the new classes can be postponed.

### 3.3 An Architecture for Scientific Data Management

We have designed a general architecture for the OOPDB toolbox. It contains three main components: software tools, user interaction, and backend storage. The software tools will be accessed through a GUI and they will manipulate data stored in heterogeneous backend storage managers. This architecture is motivated by:

- The requirement to have a uniform GUI for users of the toolbox, which isolates the GUI from the rest of the toolbox. A GUI can be defined as a user interface which uses graphical elements to communicate information to users. GUI environments include Open Look, Motif, the Apple Macintosh interface and Microsoft Windows. GUI environments typically present information with windows and icons, and uses a mouse or pointing device in addition to the standard keyboard for user input.

  GUIs are particularly suitable for *direct-manipulation*, user-centered user interfaces, in which the user directly interacts with objects instead of having to, for example, enter typically obscure commands on a command line like with a specific query language. In addition, available commands are always visible to the user without having to remember complicated sequences of commands in order to perform a certain task.

- The requirement for heterogeneous storage managers, which imposes an abstract interface between them and the software tools. This is the only way to support uniform access to heterogeneous backends. Although we support the standardization of scientific data representation, be it for information access, archival, or interchange, we recognize the reality of different storage managers used in different applications and their proliferation for many valid reasons. Standard formats and storage management will make our task easier, but our approach fully integrates heterogeneous storage managers.

- The requirement for the tools to be interoperable and data to be interchangeable. This includes for example data exchange among heterogeneous storage managers and tool integration with the uniform GUI as well as heterogeneous storage managers.

This architecture with three components has shown an applicability beyond the OOPDB. In the management of other scientific data, such as computational chemistry (*ab initio* calculations), material sciences (waste control), and atmospheric sciences (weather prediction by integration of ocean, ice, and boundary layer models), the same architecture is being used. PDBTool was designed using this architecture, with the three main components described below.

### 3.4 The Implementation

The prototype PDBTool was developed on Sun SPARC workstations using SUNOS version 4 release 1.2. The prototype currently runs on all Sun SPARC workstations. The prototype has been tested on an IBM RS/6000, model 320E under AIX version 3.2 as well as an SGI Elan workstation under Irix version 4.0.1 as X servers. (SUNOS, AIX, and Irix are compatible variants of the UNIX operating system.)

On top of the hardware and operating system, the libraries and utilities used by PDBTool include: SUIT v2.2 (Simple User Interface Toolkit) [11], X Windows V11R5, and Cfg math library. These are all publicly accessible software packages. The toolkit itself was divided into components and modules in order to simplify its implementation and incite future code reusability. The modules are:

- [The Graphical User Interface] For the GUI we used the Simple User Interface Tool (SUIT) [6] software developed at the University of Virginia. SUIT provides a set of graphical widgets to represent data under the X Window System environment. Figure 2 shows how the screen looks like.

- [Verification Tools] The verification tools are exemplified by a Ramachandran plot which illustrates the distribution of the $\phi$ versus $\psi$ dihedral angles in the polypeptide backbone of a protein structure. The plot is divided into a number of allowed and disallowed regions based on geometric constraints within the protein. Instances of $\phi$ versus $\psi$ dihedral angles outside the allowed regions may indicate errors in the structure.

The Ramachandran plotting tool was broken up into two parts: calculation and display. Calculation involves deriving the $\phi$ and $\psi$ angle of each residue by traversing through the protein. The functions necessary to find these dihedral angles were translated by hand from FORTRAN IV to $C^{++}$ [7]. The FORTRAN IV code was taken from the program PHIPSI written at BNL. During development, the calculation code could not be tested, because it could not access the protein information. Displaying data involved using SUIT to create a pop-up window that is positioned at a fixed place on the screen. The display programs includes simple windowing primitives, including pop-up displays and refreshing the plotting window spaces. Figure 2 shows the Ramachandran plot as a window on the lower right hand corner of the PDBTool.

- [The Storage Manager] was implemented on PDB flat files, the current standard storage file system for the crystallographers community. The storage manager contains two main modules, the File Importer and the Sequential Access Method.

- [File importer] To implement the file importer we designed several classes and methods to store the data from the PDB file. In this prototype version, the PDBTool can only work with PDB files. Hence the file importer module reads the PDB file into memory-resident classes. Currently only primary and secondary structure information are loaded, as well as some miscellaneous PDB header information such as remarks, authors, and expiration date.

- [Sequential Access Method] Iterator is a facility provided by the toolbox to enumerate objects sequentially. Its main use is for applications to traverse objects of the same class at each level of protein class hierarchy. Each iterator may be treated as a set, since biologists tend to treat protein as a set of residues, atoms, or secondary structures, PDBTool naturally reflects this fact.

  Filters provide sequential access to protein residues which fit a certain residue type criteria. For example, given a specific residue type, a Filter will return an Iterator of its component atoms. Iterators and Filters provide an abstract *set* interface on top of protein classes. When PDBTool is adapted to work with other data sources, like SESAM, CIF, and ObjectStore, the implementation changes will be encapsulated in the iterator and filter interfaces, which remain the same to the software tools.

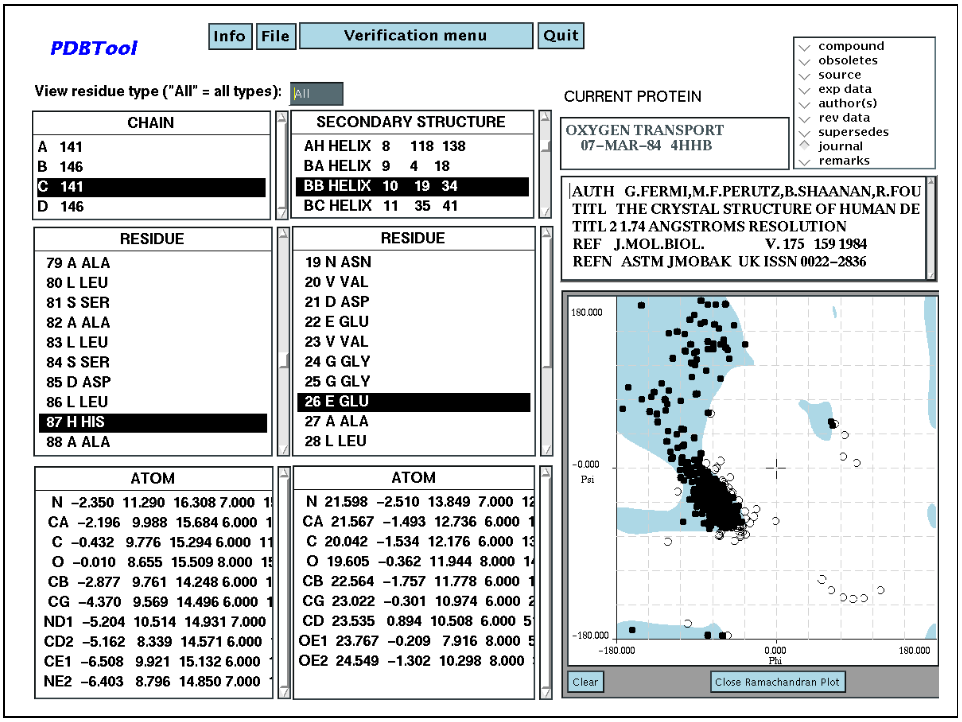


Figure 2: PDBTool Environment

### 3.5 Features

PDBTool represents a novel approach to the goal of structure verification. Traditionally scientists run a single program to test a specific aspect of the structure, analyze the results and begin a new program. PDB offers a more intuitive approach through a continuous interaction with the structure under study. This interaction is facilitated by the GUI which is divided into 2 distinct areas. At the lower right, a square was reserved for the viewing of the results of calculations. The rest of the screen was used for browsing the structural details in a hierarchical manner.

In addition there are a number of radio buttons at the upper right hand corner, that permit you to review the header information contained in the PDB file, for example, bibliographic information, remarks mad by the author, and revisions to the PDB entry. Finally, pull-down menus contain commands for reading protein data and invoking verification methods. Two additional buttons are available for the user to look at the author credits and copyright information, and to exit PDBTool.

#### 3.5.1 Browsers

The protein data browsers display the protein data in a hierarchical fashion which represents the organization of protein structure. A chain browser subwindow shows a sequential list of chains which make up the protein. If an item in the chain browser subwindow is selected (highlighted), the sequence of residues which make up that chain is displayed in the residue browser subwindow below. Likewise, selecting an item in the residue browser subwindow will cause the atoms which make up that residue to be displayed in the atom browser subwindow below. This atom browser displays atom information in column-wise order used by the PDB; namely, atom name x,y,z coordinates, temperature factor and occupancy. The secondary structure browsers works in a similar fashion.

A feature unique to the primary structure browser is a viewing filter. If the text box labeled "View residue types" contains the word "All", then all residue types are listed in the residue browser subwindow. On the other hand, if the text box contains the three-letter or single-letter code for a particular residue, only residues matching that code will be listed.

### 3.5.2 Viewer

In our first demo in August 1992, PDBTool offered one verification tool, the Ramachandran plot. Because of the modular design of the program, other tools will be simple to add in later. The Ramachandran plot can only be called when a protein has been loaded. When the plot is called, it will bring up a graph in the viewer. The plot is a graph of the $\phi$ versus $\psi$ torsion angles of the residues within the current protein. The plot only displays those residues that match the filter conditions of the browser. The filter condition controls which type of residues are displayed, the default is all. These torsion angles are displayed on an energy contour map, which defines where most $\phi/\psi$ coordinates should lie. Points that fall within the expected contour are marked as filled black circles. Points that fall outside of the contour are marked as empty yellow circles. Points outside of the acceptable regions indicate potential errors in the structure. Within the plotting area zoom-in and zoom-out are allowed in order to seeing high concentration of clustered points more clearly.

Zoom-in is accomplished by depressing the left mouse button and tracing two opposite corners of a rectangle. Zoom-out is accomplished by depressing the right mouse button and holding it down for about a second. Because the yellow points are usually considered of interest, they can be selected by clicking the left mouse button while over the point of interest. Doing so will bring up an alternate pop-up window, which will overlap the secondary structure atom browser. This window contains the name of the protein, the chain, and the residue of the selected point. It also contains the calculated $\phi$ and $\psi$ values for the residue. Like the Ramachandran plot itself, this pop-up window has a close button.

## 4 PDBTool Development

### 4.1 Evaluation of Substrate

For the development of PDBTool, we have decided to use standard components that are widely available. Our experience with SUN and SGI workstations and standard programming languages such as $C^{++}$ has been good. It is not our intention to defend the merits of $C^{++}$ as an object-oriented programming language. We simply found it to be useful in structuring our application, reliable in compiling our code, and helpful in the modular construction and testing of the system. (See next section.)

Our experience with the SUIT user interface builder is largely positive, corroborating the SUIT implementors' experience [11]. SUIT was developed primarily as a teaching tool, but we decided that SUIT was more suitable for PDBTool than, say, an X-windows based toolkit such as Motif or Xt, for several reasons:

- Nearly all members of the team have little or no GUI programming experience.
- SUIT is very easy to learn and use, and has a documented average learning time of two hours for novice GUI programmers.
- SUIT allows graphical editing of the GUI appearance and behavior.
- SUIT has an adequate set of widgets.
- SUIT is free and readily available for several platforms, including Sun and Silicon Graphics workstations, which PDBTool must run on. It also alleviated our need to search for and purchase a commercial product with similar capabilities.

The resultant SUIT-based interface consists of two major components (Figure 2) a browser (on the left) and a visualizer (lower right). Integrating the code was quite smooth due to the coding approach we followed. Programmers were able to use each others' function prototypes without actually having to compile each others code. The main problem during integration was due to the packages we were working with: The SUIT [6] code was written to work under C, not $C^{++}$ [7]. While the standard SUIT calls could be called from a $C^{++}$ block, we had to modify the SUIT function prototypes to allow the $C^{++}$ routines access to the lower-level graphics calls through SUIT.

### 4.2 Testing

Taking advantage of object-oriented implementation, we were able to hierarchically test the prototype PDBTool. Most modules were independently tested in a bottom-up modular approach

[12], in order to check its correct functionality with independent tests run on each, and to check for correct mathematical and geometrical calculations. For example, the file importer was exhaustively tested with already existing PDB files. For the Access Methods module, we tested reachability, meaning, checking that all objects were reached in the proper sequence without missing any of them.

The verification tools module was tested as a subsystem. It was first integrated with the file importer module. Then it was loaded with the available data, passing it through the verification algorithms, (like torsion angle calculation) and comparing its results with already existing outputs, in order to check that the program conformed to its specification. Boundary testing was also used to test for unusual cases. For the GUI we used visual testing to see the proper arrangements of the SUIT [6] widgets on the PDBTool window.

Once all modules were integrated into a single executable unit, system testing was conducted to iron out the interface incompatibilities and system bugs. From a practical viewpoint testing involved a review by crystallographers at the summer 1992 meeting of the American Crystallography Association (ACA). PDBTool was well received at this meeting and the notion of a more intuitive approach to structure verification acknowledged.

### 4.3 Adding New Tools

Since our first demo we have been busy adding new tools to the PDBTool. This section will be written as new tools are completed and incorporated into PDBTool.

### 4.4 Current work

Currently we are extending our prototype classes and structures into an environment managed by an OODBMS such as ObjectStore, ODE or $O_2$. Persistence will allow us to extend the field of action to manipulating more than one protein at a time, with the subclasses chains, residues, secondary structures and atoms in a single workspace. Future enhancement to the PDBTool also include interoperability with other commercial OODBMS and relational databases such as Sybase as back-ends.

Another area of attention is the management of inconsistency and uncertainty in the PDB. Concretely, some structures do not contain all the atoms. So far, each software tool has to handle the missing atom problem by itself. We are studying several alternatives in representing missing atoms and how the tools can deal with them in the least painful way. Another similar problem is the alternative conformation of atoms. Due to thermal movements, some atoms have inherently unstable locations. We are implementing a standard interface to encapsulate the uncertainty and alternatives.

Intuitively the central focus of structure verification is the 3-dimensional representation of the actual protein structure. We have developed PDBview, based on data structures used in PDBTool first as a stand alone structure viewer and later as a new display option in PDBTool. PDBview is based on X/Motif and runs on any X server. PDBview offers real-time rotation and translation and various labeling options. Part of our current efforts is to improve the PDBview towards an interactive molecular editor/browser.

## 5 Conclusion

From the computer science point of view, the project has needed many software engineering techniques. Technically, it started from understanding concepts in structural biology, to taking requirements specifications, designing their classes, designing the prototype in modules, to testing them and expanding them. Managerially, it started from project planning, to scheduling meetings and forming the appropriate software development teams, to defining and meeting deadlines, to making smooth transition between project students. We have found that current software engineering practices can be effectively used in building a non-trivial scientific application (the PDBTool) on time, with good performance and reliability, plus successful continued development.

PDBTool is a result of the first collaboration between the departments of Computer Science and Biochemistry and Molecular Biophysics. It has been successful in introducing better coding practices among basic scientists in biochemistry who had limited experience in designing powerful data structures and implementing object oriented techniques. On the other hand, it has introduced computer scientists to real-world problems from the data rich domain of molecular biology. It is our intention to seek further funding to extend this level of interaction in the production of powerful, usable, maintainable, and extensible software instruments for the molecular biology community to simplify, improve and speedup their scientific research.

## A Glossary

**ALI** A Line Interface. A domain specific 4th generation query tool, which translate pre-defined more common types of query into SQL.

**CIF** is a controlled vocabulary used to reference the structure and the experiment that derived the atomic structure. CIF is based on the more general Standard Archival and Retrieval (STAR) format. The base CIF dictionary has been ratified by the International Union of Crystallography and extensions for macromolecular structures are currently being defined. CIF is expected to be endorsed by the scientific community in the next 1-2 years.

**Daplex** is a query language for the functional data model; described by D.W.Shipman in the ACM Transactions of Database Systems 6:140-173, 1981.

**IDISIS** is derived from Biped, a relational database which used the commercial product Oracle. Idisis replaces the commercial backend with a backend developed by Oxford Molecular who market Idisis. Idisis is the work of Janet Thornton and colleagues, London. Idisis is organized to favor searches relating structure to sequence.

**NDB** The Nucleic Acid Database is a relational database, based on SYBASE which contains experimental and derived information on nucleic acid structures, namely single and double stranded DNA and RNA. NDB was developed by Helen Berman and Colleagues at Rutgers University.

**RNA** Ribonucleic Acid. A first step of protein synthesis. It is a transcription which produces a single strand of messenger RNA (mRNA) from the double stranded DNA. It is this single stranded mRNA from which the protein is produced in the process of translation. In short, RNA is vital to protein synthesis.

**PDB** The Protein Data Bank (PDB) is the international repository for macromolecular structure information on molecules of biological interest. The July 1992 release of the PDB contained information on 957 structures with atomic coordinates. The structures are predominately proteins, with a few DNA, RNA, carbohydrates, and complexes thereof. The PDB is maintained by the Chemistry Department at the Brookhaven National Laboratory and distributed on magnetic tape and CDROM on a quarterly basis. Recent releases are available via ftp.

**P/FDM** is an object oriented database developed by Peter Gray and Colleagues, Aberdeen. P/FDM organizes classes following the natural hierarchy in proteins protein, residue and atoms, and uses Prolog and the Daplex query language.

**PKB** provides a file structure and query mechanism for PDB data. It is based on the statistical language S, and hence is good for statistical analysis of the PDB. PKB was developed by Steven Bryant of the National Center for Biotechnology Information.

**SESAM** is based on the relational database SYBASE and contains data obtained from the PDB. Additionally a number of derived fields have been tabulated which are concerned with the energetics of proteins. Unlike the other relational databases which are accessed using SQL, SESAM has ALI. SESAM was developed by Shoshana Wodak and colleagues in Brussels, Belgium.

## References


[1] F.C. Bernstein, T.F. Koetzle, G.J.B. Williams, E.F. Mayer Jr., M.D. Bryce, J.R. Rodgers, O. Kennard, T. Simanouchi, and M. Tasumi. The protein data bank. *J. Mol. Biol.* 122:535-542 (1977).

[2] H. Berman NDB: The nucleic acid database. *unpublished.*

[3] P.E. Bourne and P.L. Marquess The crystallographic workbench. *Insight and Innovation in Data Visualization.* Ed. J.E. Bowie, Manning Pub., 1992.

[4] S.H. Bryant. PKB: A program system and data base for the analysis of protein structure. *Proteins* 5:233-247, 1989.

[5] R.G.G. Cattell Next-Generation Database Systems Communications of the ACM, Oct 1991. Vol 34 Num 10

[6] M.J. Conway. The SUIT Version 2.2 Reference Manual. *Computer Science, GUI* The University of Virginia 1991, 1992

[7] S. C. Dewhurst and K. T. Stark Programming in $C^{++}$ *Computer Science* Prentice Hall 1989.

[8] M. Huysmans, J. Richelle, and S.J. Wodak SESAM: A relational database for structure and sequence of macromolecules. *Proteins* 11:59-76, 1991.

[9] P.M.D. Grey, N.W. Paton, G.J.L. Kemp and J.E. Fothergill An object-oriented database for protein structure analysis. *Protein Engineering* 3:235-243, 1990.

[10] S. Islam, and M. Sternberg. A relational database of protein structures designed for flexible inquiries about conformation. *Protein Engineering* 2:431-432, 1989.

[11] R. Pausch, M. Conway, R. Deline Lessons Learned from SUIT, the Simple User Interface Toolkit Volume 10, Number 4, October 1992, pages 320-344.

[12] I. Sommerville Software Engineering *Software Engineering* Addison Wesley 1990.