Fast Access to Big Molecules:
An Introduction to the OMG/LSR Macromolecular Structure API

Douglas S. Greer
University of California, San Diego

Alexy Khrabrov
Rutgers University

Philip E. Bourne
University of California, San Diego

John D. Westbrook
Rutgers University

In February 2002 the Board of Directors of the Object Management Group (OMG) voted to approve the Macromolecular Structure Corba specification developed under the auspices of the Life Sciences Research (LSR) task force. This Corba specification [1] provides an open, standardized object-oriented interface that allows fast and efficient remote access to binary data structures containing macromolecular structure data. While at first glance the large size and complexity of this Application Programming Interface (API) may seem intimidating, once the basic principles and core classes are understood, it is relatively simple to write programs with the easy-to-use interface. This paper introduces the fundamental concepts and provides some simple program fragments to illustrate the how the API can be used in developing client applications that can quickly access the rich and extensive set of available data.

The Research Collaboratory for Structural Bioinformatics, which manages the Protein Data Bank (PDB) [2] will support Corba, XML and Relational Database representations of Macromolecular Structure (MMS) data that have been designed to address the needs of a wide range of applications. These include many types of services and high performance programs that require high-speed access to molecular structure information. The Java source code that implements an OMG/LSR compliant Corba server is publicly available as part of the OpenMMS toolkit [3][4]. The OpenMMS toolkit also includes a relational database loader and XML converter. A second implementation of a compliant Corba server written in C++ is currently under development at Rutgers University.

The Corba interface, the relational database Structured Query Language (SQL) schema and the XML representations all use the same terminology definitions and interrelationships defined by a standard scientific ontology developed under the auspices of the International Union of Crystallography (IUCr). This ontology known as the macromolecular Crystallographic Information File (mmCIF) standard [5][6][7] includes both data files and several dictionaries, which define thousands of scientific terms and their attributes. While the amount of existing software that uses the legacy PDB formatted files is so extensive as to insure that this format will never go away, new applications can greatly benefit from the more precise and more extensive mmCIF standard.

The standard representation that ties together the Corba, XML and SQL expressions of the MMS data is the set of mmCIF flat files. Any errors or discrepancies in the expressed forms are resolved by consulting these reference files. A very high quality set of “native” mmCIF files for entire PDB contents is now available at the RCSB beta ftp site [8]. These new files are distinct from the mmCIF files translated from the legacy PDB files by the software tool, pdb2cif. The new API and supporting software tools described here should only be applied to the new beta set of mmCIF data.

The overall view illustrating the MMS dataflow is shown in Figure 1. Among the four methods for accessing MMS data shown in the figure, the mmCIF parsers are the most general purpose and also provide the lowest level and most detailed access to the underlying data. XML files provide a simple and powerful method for interchanging MMS data in a widely used and understood

format. However because of the open and close tags, the XML files tend to be many times larger than the corresponding mmCIF files. Also shown in this figure is a SQL-92 compatible relational database, which provides an appropriate interface for many applications, particularly ones that require extensive string searches.

The delivery of data from a Corba server to client applications, show in figure 1 with thick outlines, potentially provides the highest performance access to MMS data. The object-oriented interface is used to define structures independent of platform and programming language, and yet may be optimized to copy binary data quickly and efficiently across the network. When reading the XML or mmCIF flat files, users are required to retrieve and parse the complete file in order to use even a small portion of it. However the Corba and SQL interfaces provide granular access, which allows client applications to quickly retrieve only a single small data element from the server.

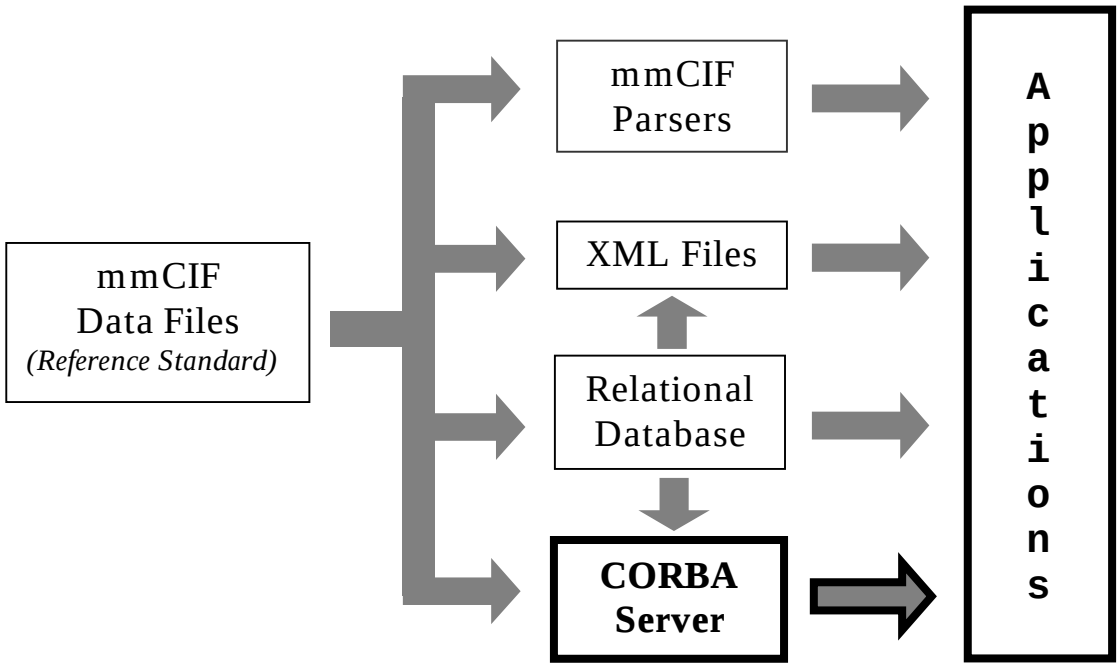

Figure 1. Dataflow through derived Expressions of the mmCIF Data

The OpenMMS toolkit contains two Corba servers, which correspond to two data paths shown in figure 1: a “Reference” server that reads the data from the mmCIF flat files and a much faster “DB” server that reads the data from a relational database. The C++ MMS server also reads data from a relational database. Since in practice, some structures are accessed much more often that others, and some types of data are read more often than others, a Corba server can gain significant performance improvements by the appropriate caching of commonly read data.

Cobra’s Portable Object Adaptor (POA) manages association of objects and object references. It comes with a POA manager, and allows for various policies to enable efficient implementation of object invocations, including creating objects and starting servers on demand. The POA may contain an active object map and do other forms of memory management transparently, allowing the architect to specify a range of behaviors and memory/space tradeoffs with a list of high-level policies[9]. The POA and Corba services allow for persistent and relocatable objects, load-balancing and other infrastructural support for high efficiency and availability.

Data Model

An *Entry*, which represents a molecular structure, is the central object in the data model. An Entry usually corresponds to the contents of a single PDB file (or an XML or mmCIF file). In the Corba API, the Entry object is used to retrieve information about the structure such as the

residue sequence or atom locations. An Entry object for a particular Structure ID e.g. “4HHB” can be obtained from a Corba `EntryFactory` as shown in the example code in the next section.

The mmCIF ontology defines information about an Entry using an Entity-Relationship model [10][11]. While there are several hundred tables containing several thousand scientific terms defined in the mmCIF ontology, only a few of these, including those listed in Table 1 are required inorder to describe molecular structure. The first column of Table 1 lists the name used by mmCIF and the MMS API. In general, names that begin with Entity describe essential biological and chemical features that are known before a structure determination experiment is performed. Names that begin with Struct denote information determined during the experiment (the AtomSite table is also in this group). Both of these groups are contained in the “Core” MMS API module `DsLSRMacromolecularStructure`. Separate modules are used to store the details of the X-Ray Crystallography experiment and information about citations and bibliographic references regarding a particular structure. A new module that will contain information about Nuclear Magnetic Resonance (NMR) experiments is under development.

mmCIF Category / SQL Table Name / Corba Struct Name	*Common Designation*	*Description*
AtomSite	Atoms	List of all atoms in an Entry with their atomic coordinates. One atom per row.
Entity	Molecule “Classes”	List of molecular entities, each having a unique sequence or chemical formula that are contained in this particular Entry.
StructAsym	“Instances” of Molecules	List of molecular entities contained in the asymmetric unit as determined during the experiment. A particular Entity may appear in this list multiple times.
StructBiolGen	“Instances” of Biomolecules.	List of molecules comprising a functional biological assembly. A particular Entity may appear in this list multiple times.
ChemComp	Residue types	List of unique monomers or ligands contained in the structure. E.g. Glycine, Tyrosine...
ChemCompAtom	Residue atom types	E.g. alpha carbon, gamma carbon, ...
EntityPolySeq	Residue sequence	Actual sequence of monomers that make up a particular polymer Entity.
StructConf	Alpha helices and turns	List of structural confirmations that include helices and turns
StructSheet and StructSheetOrder	Beta sheets	Lists of beta sheets and their ordering

Although our primary focus here is the Corba API, the same model applies to the XML, mmCIF and relational database representations. In a relational database, the information about an Entry is stored in a collection of tables. In mmCIF the tables are called “Categories” and the table columns are called “Items”. Within an mmCIF file, the “_loop” construct is used to represent tables with more that one row. In the Corba specification each table corresponds to a specific type of data structure. An instance of type X represents a single data structure of that type, which in a relational database would correspond to a single row of table X. The columns of the table correspond to specific fields in the data structure. Tables with more than one row correspond to

Arrays of structures of that type. If in the relational database representation a table has N rows then in a Corba representation the Array will have N elements. The Corba Interface Definition Language (IDL) construct "sequence<X>" is used to specify the abstract type representing an array of type X. This construct compiles to actual objects of type X[] (array of X) in Java and C/C++. For details on individual categories the complete set of Java Documentation for the API is available online [12], as are the mmCIF dictionaries [7].

Program Examples

A primary requirement of the design was that it present an interface that was clearly defined and easy to use from the point of view of developing new applications. The code examples in this section illustrate how client programs can use the API to quickly access macromolecular structure data. For the sake of brevity, many details such as exception handling, error checking and some of the required declarations have been left out. A compete test client application, `TxClient.java`, is included with the OpenMMS software.

A single instance of the AtomSite structure stores the Cartesian coordinates and other information about an atom just as a single ATOM record does in this legacy PDB format. The complete list (an IDL sequence) of all atoms in a macromolecular structure is returned by invoking the `get_atom_site_list()` method on an instance of the Entry interface object. As a simple example the following Java code fragment will print out the atom identifier, atom type and the Cartesian (x, y, z) position for all atoms in the macromolecule 4hhb.

Example 1. Retrieving the AtomSite list for hemoglobin (4HHB) and printing the atomic coordinates.

```
Entry e = entryFactory.get_entry_from_id("4hhb");
AtomSite[] a = e.get_atom_site_list();
for (int i = 0; i < a.length; i++)  {
    System.out.println(a[i].id + " " + a[i].type_symbol.id
        + " (" + a[i].cartn.x + ", " + a[i].cartn.y
        + ", " + a[i].cartn.z + ")");
    }
```

Here is an excerpt of the output produced by the code in Example 1:

```
...
6 C (7.002, 20.127, 5.418)
7 C (5.246, 18.533, 5.681)
8 N (9.096, 18.040, 3.857)
9 C (10.60, 17.889, 4.283)
...
```

Note that in the Example 1 code fragment above, only the first two lines are required to retrieve a reference to an instance of a "4HHB". The first line fetches an Entry object corresponding to hemoglobin (4hhb) using the EntryFactory method `get_entry_from_id()`.
The second line then uses this Entry reference (stored in the variable "e") to obtain the complete list of atomic coordinates using the method `get_atom_site_list()`. Once a program has a Corba EntryFactory object, it can execute these two lines over and over to quickly obtain the atomic coordinates (in IEEE floating point format) for any structure in the PDB.

The following example shows a program that lists all of the alpha helices in the structure 4HHB and for each helix lists all of the residues that it comprises. The first line in the Example 2 program fragment obtains an `Entry` object as above. The three following lines fetch the StructConf, EntryPolySeq and ChemComp arrays from the server. The program then iterates through the entire list of structure confirmations and for each one lists all of the residues it contains. Note that the StructConf table may contain other information such as "TURNs" but information about beta sheets is contained in a separate table since it is useful to build a related table expressing the beta sheet connectivity.

Example 2. Listing the residues contained in all of the hemoglobin (4HHB) alpha helices.

```
Entry e = entryFactory.get_entry_from_id("4hhb");
...
StructConf[] scf = e.get_struct_conf_list();
EntityPolySeq[] eps = e.get_entity_poly_seq_list();
```

```
ChemComp[] cc = e.get_chem_comp_list();
for (int j = 0; j < scf.length; j++) {
    System.out.println("Structure Conformation " + scf[j].id
        + " in chain " + scf[j].beg_label.asym.id + " contains:");
    int start = scf[j].beg_label.seq.index;
    int end = scf[j].end_label.seq.index;
    for (int i = start; i <= end; i++) {
        System.out.println("    Monomer: " + cc[eps[i].mon.index].name
            + " (" + eps[i].mon.id + ") at position " + eps[i].num);
    }
}
```

The actual output of this program is quite long since there are 32 alpha helices in 4HHB. Below is a small excerpt of the printout for Helix 24 (i.e. when j = 23 in the Example 2 code fragment above):

```
...
Structure Conformation HELX_P24 in chain C contains:
    Monomer: THREONINE (THR) at position 118
    Monomer: PROLINE (PRO) at position 119
    Monomer: ALANINE (ALA) at position 120
    Monomer: VALINE (VAL) at position 121
    Monomer: HISTIDINE (HIS) at position 122
    Monomer: ALANINE (ALA) at position 123
    Monomer: SERINE (SER) at position 124
...
```

Unfortunately, a paper of this size can only provide a brief glimpse into the content and functionality of this API. For more in-depth information the reader is encouraged to consult the online references listed below.

Acknowledgements

This work has been supported by the Protein Data Bank (PDB) and National Partnership for Advanced Computing Infrastructure (NPACI). The Protein Data Bank (PDB) is operated by Rutgers, The State University of New Jersey; the San Diego Supercomputer Center at the University of California, San Diego; and the National Institute of Standards and Technology – three members of the Research Collaboratory for Structural Bioinformatics (RCSB). This work is supported by grants from the National Science Foundation, the Department of Energy, and two units of the National Institutes of Health: the National Institute of General Medical Sciences and the National Library of Medicine.

References

1. http://www.omg.org/technology/documents/formal/macro_molecular.htm
2. H.M. Berman, J.D. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne, (2000) The Protein Data Bank. *Nucleic Acid Research* 28(1), 235-242.
3. http://openmms.sdsc.edu
4. D.S. Greer, J.D. Westbrook and P.E. Bourne, (2002) An Ontology Driven Architecture for Derived Representations of Macromolecular Structure, *Bioinformatics* 18(9): 1280
5. J.D. Westbrook and P.E. Bourne, (2000) STAR/mmCIF: An Extensive Ontology for Macromolecular Structure and Beyond. *Bioinformatics* 16(2) 159-168
6. P.E. Bourne, H.M. Berman, B. McMahon, K. Watenpaugh, J. Westbrook. and P.M.D. Fitzgerald, (1997) The Macromolecular CIF Dictionary. *Methods in Enzymology*. 1997 227, 571-590.
7. http://pdb.rutgers.edu/mmcif
8. ftp://beta.rcsb.org/pub/pdb/uniformity/data/mmCIF/divided/
9. M. Henning, S. Vinoski, *Advanced CORBA Programming with C++* Addison-Wesley, 1999
10. T.M. Connolly, C.E. Begg, A.D. Strachan, *Database Systems, A Practical Approach to Design, Implementation and Management.* Addison-Wesley, 1995
11. C. J. Date, *An Introduction to Database Systems*. Addison-Wesley, 1981
12. http://openmms.sdsc.edu/OpenMMS-1.2.8_LSR-1.0/openmms/docs/api/index.html

Authors Contact Address

Douglas S. Greer
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Drive
La Jolla, CA, 92093-0527, USA
Email: dsg@sdsc.edu