Code Generation Through Annotation of Macromolecular Structure Data

John Biggs[1], Calton Pu[1], and Philip Bourne[2]

[1]Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
P.O. Box 91000 Portland, OR 97291-1000
{biggs,calton}@cse.ogi.edu

[2]San Diego Supercomputer Center
P.O. Box 85608, San Diego, CA 92186-9784
bourne@sdsc.edu

Abstract

The maintenance of software which uses a rapidly evolving data annotation scheme is time consuming and expensive. At the same time without current software the annotation scheme itself becomes limited and is less likely to be widely adopted. A solution to this problem has been developed for the macromolecular Crystallographic Information File (mmCIF) annotation scheme. The approach could be generalized for a variety of annotation schemes used or proposed for molecular biology data. mmCIF provides a highly structured and complete annotation for describing NMR and X-ray crystallographic data and the resulting macromolecular structures. This annotation is maintained in the mmCIF dictionary which currently contains over 3,200 terms. A major challenge is to maintain code for converting between mmCIF and Protein Data Bank (PDB) annotations while both continue to evolve. The solution has been to define a simple domain specific language (DSL) which is added to the extensive annotation already found in the mmCIF dictionary. The DSL calls specific mapping modules for each category of data item in the mmCIF dictionary. Adding or changing the mapping between PDB and mmCIF items of data is straightforward since data categories (and hence mapping modules) correspond to elements of macromolecular structure familiar to the experimentalist. Each time a change is made to the macromolecular annotation the appropriate change is made to the easily located and modifiable mapping modules. A code generator is then called which reads the mapping modules and creates a new executable for performing the data conversion. In this way code is easily kept current by individuals with limited programming skill, but who have an understanding of macromolecular structure and details of the annotation scheme. Most important, the conversion process becomes part of the global dictionary and is not open to a variety of interpretations by different research groups writing code based on dictionary contents. Details of the DSL and code generator are provided.

Introduction

Data in many areas of molecular biology are growing exponentially. This massive amount of data begs asking many new questions particularly in the area of comparative analysis. For example, in beginning to compare whole genomes (Hood et al. 1992) for common evolutionary origins and in classifying and comparing protein structures through the presence of common structural motifs (Holm and Sander, 1996; Orengo et al. 1996; Murzin et al. 1995). What is becoming painfully obvious is that good annotation becomes paramount if data are to be compared effectively. We classify such annotation in terms of content and context. Content defines the detail of the description (atomicity) and how much of a given scientific field is covered by the annotation. Context describes the format and computer usability of the annotation.

Context deficiencies are common to many annotation schemes which lack a comprehensive computer readable description for each item of data. All too often the item of data is disjoint from its definition, leading to a complete misinterpretation, or somewhat different interpretations, by different programmers charged with coding applications that read and use that item of data. A complete annotation includes, for each data item, machine readable data type, relationship to other data items, enumeration where applicable, units, and the subject of this paper, a domain specific language (DSL) that calls procedures which operate on that data. The annotation scheme, including the DSL, should be extensible, such that the same core software can be used to read and process an ever expanding set of data items defining the discipline.

Small molecule crystallography has had such a data description since 1990 and an extension has recently been defined for macromolecular crystallography. The small molecule description is referred to as the Crystallographic Information File (CIF; Hall, Allen, and Brown, 1991), and the macromolecular extension mmCIF (Bourne et al. 1996). While improved content and context are welcome there remains the issue of maintaining software which uses this annotation, since this annotation evolves very fast while under development and continues to evolve as the knowledge of the discipline increases and experimental methods change. A methodology has been implemented and proposed here as a general way of maintaining software for handling an evolving annotation scheme. To understand how it is used it is first necessary to understand the basic structure of mmCIF. A more detailed description is given in Bourne et al. (1996) and references therein.

An Overview of CIF and mmCIF

The key elements of CIF and mmCIF are the dictionaries that define each data item used by the discipline. Small groups of dedicated researchers have defined comprehensive dictionaries that have been opened for community review and subsequently modified before a final approval process by the International Union of Crystallography (IUCr), the scientific body assigned to oversee dictionary maintenance. Currently the small molecule dictionary contains approximately 400 data items and the macromolecular dictionary approximately 3,200. A testament to both the quantitative nature of the field and the dedication of the individuals working on the dictionaries. A typical entry taken from the macromolecular dictionary is:

```
save__atom_site.cartn_x
   _item_description.description
;  The x coordinate of the atom site position specified as
   orthogonal Angstroms. The orthogonal Cartesian axes
   are related to the cell axes as specified by the description
   given in _atom_sites.cartn_transform_axes.
;
   _item.name                    '_atom_site.cartn_x'
   _item.category_id             atom_site
   _item.mandatory_code          no
   _item_sub_category.id         'cartesian_coordinate'
   _item_aliases.alias_name      '_atom_site_cartn_x'
   _item_aliases.dictionary      'cifdic.c94'
   _item_aliases.version         '2.0'
    loop_
   _item_dependent.dependent_name
         '_atom_site.cartn_y'
         '_atom_site.cartn_z'
   _item_related.related_name    '_atom_site.cartn_x_esd'
   _item_related.function_code   'associated_esd'
   _item_type.code               float
   _item_type_conditions.code    esd
   _item_units.code              'angstroms'
    save_
```

This is the definition of the Cartesian x coordinate used in the representation of an atom site. The definition consists of name value pairs with names beginning with an underscore (_). In a dictionary these names form the Dictionary Definition Language (DDL) and are themselves defined in yet another dictionary - the DDL dictionary - which conforms to the same rules. Hence, the description is self-defining and the rules themselves are referred to as a Self-defining Text Archival and Retrieval (STAR) representation (Hall, 1991; Hall and Spadacinni, 1994). Within an mmCIF data file the names have corresponding entries in the mmCIF dictionary and the values are the actual data values.

Each data item belongs to a category of data items. In the above example atom_site is the category as defined by _item.category_id. Data names begin with the category name, followed by a period (.), followed by the unique part of the name. As will be shown subsequently, the category is itself defined and data items within the same category can be grouped into a STAR loop_ structure.

A significant amount of software has been written based upon STAR syntax rules and the DDL (Bourne, 1993; NDB, 1996). Additional dictionaries are being developed which immediately benefit from existing software tools (IUCr, 1997).

To our knowledge no other scientific discipline has developed such a comprehensive annotation scheme. The comprehensive dictionaries and software for handling these dictionaries inspired the logical next step in this work - a domain specific language (DSL) which maps to specific procedures that become part of the standard annotation. It is the use of a prototype DSL to maintain a concordance between mmCIF and PDB data representations which is described here.

Motivation

The advent of mmCIF requires that easily maintainable software exist to convert between the PDB and mmCIF data representations.

Application

The approach described here, the principles of which could impact any detailed annotation scheme, is to include within the dictionary, using standard STAR syntax rules, a simple DSL which represents the concordance between mmCIF data items and fields within PDB formatted records. The DSL includes mapping modules that describe the concordance for each data items within each mmCIF category. A code generator uses the annotations and builds an executable by establishing lower level function calls to the appropriate mapping modules.

This design approach hides the intricacies of the code needed to make the translation from PDB to mmCIF. The domain scientist need only add the appropriate mapping to the mapping modules and regenerate the code needed for correct conversion.

pdb2ocif is an application that uses this code generation scheme and is based upon the Perl translator pdb2cif (Bernstein, Bernstein, and Bourne, 1997).

Details of the DSL

There were two design goals for the DSL. First, it should be a simple language that biologists could comprehend and easily extend as the PDB and mmCIF data representations evolve. Second, a close correlation between the DSL and the mmCIF dictionary should be maintained by including the DSL in the dictionary.

In the early stages of the design, actual C++ code was contained within mmCIF categories. This led to large amounts of code spread throughout the dictionary and we were far from our goal of easy maintenance. The next

evolution was to move the C++ code from the dictionary into modules and design a relatively simple DSL to access those modules. This way the dictionary would be minimally affected and simple to understand. The DSL would be parsed and appropriate calls to translation modules would be handled via a parser module.

Finally, the DSL was tightly bound to the mmCIF dictionary categories. This reduced the DSL to a small set of software resource calls needed by pdb2ocif - LOAD, UNKNOWN, OUTPUT, and pdb2ocif_control_items (described below) for every category found within the mmCIF dictionary. The following is the ATOM_SITE category description, of which the data item _atom_site.cartn_x introduced above is a member, modified to include the DSL.

```
save_ATOM_SITE
   _category.description
;  Data items in the ATOM_SITE category record details
   about the atom sites in a macromolecular crystal
   structure, such as the positional coordinates, atomic
   displacement parameters, magnetic moments and
   directions, and so on.
;
PDB2OCIF_ OUTPUT
   _category.id                  atom_site
   _category.mandatory_code      no
   _category_key.name            '_atom_site.id'
    loop_
   _category_group.id            'inclusive_group'
                                 'atom_group'
    loop_
   _category_examples.detail
   _category_examples.case
   ...
```

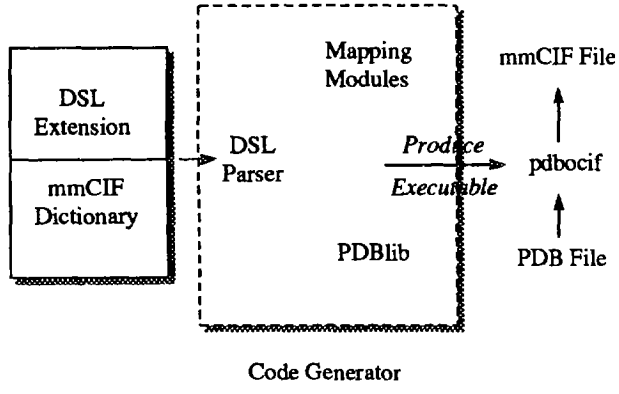

Figure 1. Software Components

The DSL Parser reads the mmCIF dictionary calls the appropriate mapping modules which are compiled into a new executable along with PDBlib, a general purpose library for representing macromolecular structure information. PDBlib includes a PDB parser. In this prototype application an mmCIF file is produced.

The only addition to the dictionary is the DSL statement PDB2OCIF_ OUTPUT. The code generator reads the additional annotation, calls the appropriate mapping module and generates the code needed for the translation. These steps are summarized in Figure 1.

Code Generator

The Code Generator parses the DSL and translates the parsed annotations into calls to the low-level mapping modules. The Code Generator is a C++ class designed to enable simple registry of new mapping modules thereby facilitating the maintenance and extensibility of pdb2ocif.

The Code Generator is also responsible for maintaining the dynamic program state. That is, it manages the DSL for loading PDBlib (see below) with a PDB file and also maintains the program control variables.

Mapping Modules

The mapping modules in the current version of pdb2ocif are written in C++. A mapping module exists for every mmCIF dictionary category. The "UNKNOWN" translation creates translations with placeholder data values for each key data item of the category. This is used to indicate in the resulting mmCIF that no equivalent PDB fields map to the mmCIF data items, but that they should be included if a comprehensive mmCIF data representation is to be used. This mapping is contained within a single module to minimize the number of mapping modules needed for full mmCIF dictionary coverage.

The mapping modules use the PDBlib representation of a macromolecular structure (Chang et al. 1994). PDBlib is a C++ class library which has been used extensively to represent and navigate data parsable from a PDB file (Shindyalov et al. 1994; Biggs et al. 1996).

The following code is taken from the SYMMERTY mapping module for the mmCIF category symmetry and which gets included in the code generated to convert the PDB SYMMETRY record type to its mmCIF equivalent.

```
/* Example: Translation for mmCIF category _symmetry */
   void p2cTranslator::SYMMETRY() {
   char head_PDB_code[5];
   pdblibExt::getHEADER_PDBCode(head_PDB_code);
   printf("_symmetry.entry_id  %s \n", head_PDB_code);
   printf("_symmetry.space_group_name_H-M  '%-11s' \n
   \n",
   cmpound>MyPDBInfo()>MySpaceGroupName());
   }
```

PDBlib provides a rich and efficient set of iterators to navigate the currently loaded PDB structure. Moreover, PDB data are easily accessible from PDBlib and the translation code is much cleaner because the representational details are handled by PDBlib. Also, by having a centralized molecular representation used to interface to the data, representational errors are minimized

since PDBlib has been extensively tested as part of other applications. Finally, a critical and understated role of PDBlib, and software like it, is the ability to parse the complete PDB making allowances for idiosyncrasies that exist in the data..

It is straightforward for researchers not versed in C++ to extend pdb2ocif because other software languages can be used to code translator modules. It is possible to write a mmCIF dictionary category translation in a language like FORTRAN or JAVA and then call the translation from the Code Generator appropriately (either as a script or as a linked object code function call).

Language Specific Extensions to the Mapping Modules

In this application Perl code had already been written (Bernstein, Bernstein, and Bourne, 1997) to map PDB record types and their associated fields to mmCIF data items. Rather than recode this information it was encapsulated in C++ classes. An example of this encapsulation was the Perl notion of an associate array written as the C++ class "AssocArray." Associations are established using the class method "associate" and the [] operator is overloaded to provide access.

```
/* Example: AssocArray class usage */
AssocArray cit_title_1;
  ...
cit_title_1.associate("something", 1);
  ...
if (cit_title_1[1] !strcmp(cit_title_1[1], "something")
  ...
```

The PERL extensions allow programmers who are used to scripting languages to use C++ classes and functions more easily. The PERL extensions were designed to facilitate this project only, thus to date only a small number of extensions have been written. In time these extensions could grow to contain more PERL functionality as well as other scripting languages.

Discussion

pdb2ocif is an example of an application that is easily maintainable even while the underlying annotation used by the application is evolving rapidly.

A biologists with a small knowledge of C++, and the PDBLib library, could take on the task of replacing this stubbed output by:

1. removing the translation from the UNKNOWN module.
2. implement a new translation module for this category.
3. compile the new module into the pdb2ocif executable.

Similarly, adding new category translations to pdb2ocif involves minimal programming:

1. add a translation to the UNKNOWN module or implement a new translation module for the new category translation.
2. register the new category translation with the Code Generator.
3. compile the new/revised modules into the pdb2ocif executable.

Software maintenance is easily spread over many people. Each mapping module is separated from every other mapping module and there are no interdependencies.

This work was supported by grants NSF 9310154 and the DOE.

References

Bernstein, H., Bernstein F., and Bourne, P.E. 1997. CIF Applications. *pdb2cif*: Translating PDB Entries into mmCIF Format. Forthcoming.
http://ndbserver.rutgers.edu/NDB/mmcif/software/index.html

Biggs, J., Pu, C., Groeninger, A., and Bourne, P.E. 1996. PDBtool: An Interactive Browser and Geometry Checker for Protein Structures. *J. App. Cryst.* 29(4):484-490.

Bourne, P.E. ed. 1993. *Proceedings of the First Macromolecular CIF Tools Workshop.* Tarrytown NY.

Bourne, P.E., Berman, H.M., McMahon, B., Watenpaugh, K., Westbrook, J., and Fitzgerald, P.M.D. 1996. The Macromolecular CIF Dictionary (mmCIF). *Methods in Enzymology.* Forthcoming.

Chang, W., Shindyalov, I.N. Pu, P and Bourne, P.E. 1994. Design and Application of PDBlib, a C++ Macromolecular Class Library. *CABIOS,* 10(6):575-586.

Hall, S.R. 1991. The STAR File: A New Format for Electronic Data Transfer and Archiving. *J. Chem Inf. Compt. Sci.* 31:326-333.

Hall, S.R., Allen, F.H., and Brown, I.D. 1991. The Crystallographic Information File (CIF): A New Standard Archive File for Crystallography. *Acta Cryst.* A47:655-685.

Hall, S.R, and Spadaccini, N. 1994. The STAR File: Detailed Specifications. *J. Chem Inf. Compt. Sci.* 34:505-508.

Holm, L., and Sander, C. 1996. The FSSP Database: Fold Classification Based on Structure-structure Alignment of Proteins. *Nucleic Acids Res.* 24:206-209.

Hood, L., Koop, B., Goverman, J., and Hunkapiller. T. 1992. Model Genomes: The Benefits of Analysing Homologous Human and Mouse Sequences. *Trends Biotechnol* 10:19-22.

IUCr. 1997. *http://www.iucr.ac.uk/iucr-top/cif/home.html.*

Murzin, A.G., Brenner, S.E., Hubbard, T., and Chothia, C. 1995. scop: A Structural Classification of Proteins. Database for the Investigation of Sequences and Structures. *J. Mol. Biol.* 247:536-540.

NDB. 1996. Macromolecular Crystallographic Information File Home Page. *http://ndb.rutgers.edu/NDB/mmcif/*

Orengo, C.A., Michie, A.D., Jones, S., Swindells, M.B., Jones, D.T., and Thornton, J.M. 1996. CATH Protein Structure Classification *http://www.biochem.ucl.ac.uk/bsm/cath/.*

Shindyalov, I.N., Chang, W., Pu, C., and Bourne, P.E. 1994. MMQL An Object Oriented Macromolecular Query Language: Prototype Data Model and Implementation. *Prot. Eng.* 7(11): 1311-1322.