BIOINFORMATICS APPLICATIONS NOTE
Vol. 19 no. 0 2003, pages 1–3
DOI: 10.1093/bioinformatics/btg299

PDB file parser and structure class implemented in Python

Thomas Hamelryck[1,2,*] and Bernard Manderick[2]

[1]Department of Cellular and Molecular Interactions, Vlaams Interuniversitair Instituut voor Biotechnologie (VIB) and [2]Computational Modeling Lab (COMO), Department of Computer Science, Vrije Universiteit Brussel (VUB), Pleinlaan 2, 1050 Brussels, Belgium

Received on January 21, 2003; revised on March 3, 2003; accepted on May 10, 2003

ABSTRACT

Summary: The biopython project provides a set of bioinformatics tools implemented in Python. Recently, biopython was extended with a set of modules that deal with macromolecular structure. Biopython now contains a parser for PDB files that makes the atomic information available in an easy-to-use but powerful data structure. The parser and data structure deal with features that are often left out or handled inadequately by other packages, e.g. atom and residue disorder (if point mutants are present in the crystal), anisotropic B factors, multiple models and insertion codes. In addition, the parser performs some sanity checking to detect obvious errors.

Availability: The Biopython distribution (including source code and documentation) is freely available (under the Biopython license) from http://www.biopython.org

Contact: thamelry@vub.ac.be

1 INTRODUCTION

Python (http://www.python.org) is a freely available, object oriented, interpreted language that is easy to use yet very powerful. The biopython project (http://www.biopython.org) provides a set of reusable software modules written in Python that mainly deal with sequence retrieval, analysis and manipulation (Chapman and Chang, 2000). The software described here extends biopython with a powerful set of modules that deal with macromolecular structure, including a PDB (Berman *et al.*, 2000) file parser, a powerful `Structure` class and modules that deal with three-dimensional (3D) neighbor lookup and the identification of polypeptides. Freely available, reusable code to handle macromolecular structure data is far and between. Currently available solutions include the C++ library PDBLib (Chang *et al.*, 1994) and the Python-based MMTK toolkit (Hinsen, 2000). Python is also used in the powerful macromolecular visualization PMV packages (Sanner, 1999; Sanner *et al.*, 1999; Coon *et al.*, 2001) and PyMol (DeLano, 2002). The biopython modules described here should be a valuable addition to the available software.

*To whom correspondence should be addressed.

2 OVERALL DESCRIPTION

A crystal structure is represented using a structure/model/chain/residue/atom (or *SMCRA*) hierarchic data structure (the `Structure` class), whose structure is shown in Figure 1. This data structure forms a very convenient framework to access the atomic data in a PDB file. In the hierarchical SMCRA data structure, a child object (i.e. `Atom, Residue, Chain, Model`) can always be extracted from its parent (i.e. `Residue, Chain, Model, Structure`, respectively) by using an identifier as a key, e.g.

```
model=structure[0]
chain=model['A']
residue=chain[1]
atom=residue['CA']
```

These identifiers are:

- `Model`: The rank of the model in the PDB file, starting from 0. Crystal structures typically only have a single model, while NMR structures typically have several.
- `Chain`: The chain identifier, e.g. `"A"`.
- `Residue`: A tuple composed of three parts: the *hetero field*, the *sequence identifier* and the *insertion code*, e.g. `("H_GLC", 45, "0")`. The hetero field is a string : it is '`W`' for waters, `"H_"` followed by the residue name (e.g. `"H_GLC"` for a glucose hetero residue with residue name `"GLC"`) for other hetero residues and blank for standard amino and nucleic acids. The hetero field is included to avoid problems with hetero and non-hetero residues that have the same sequence identifiers, which is very common in the PDB. The insertion code serves to facilitate the comparison with a reference protein, and is fairly rarely used. For non-hetero residues that have no insertion codes, the sequence identifier alone can also be used

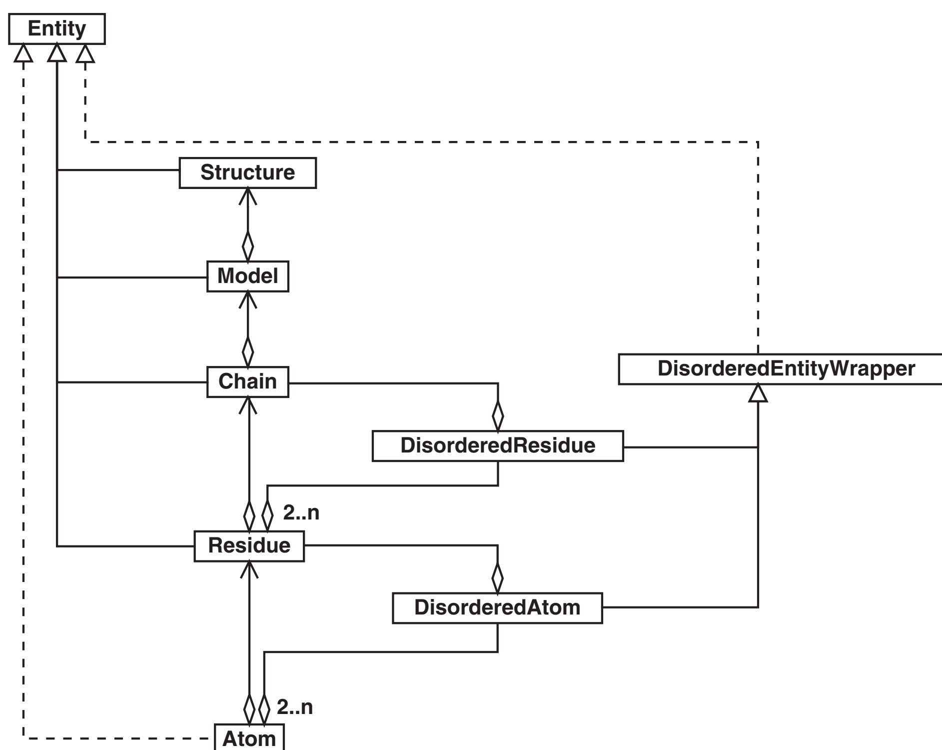

Fig. 1. UML (Fowler, 1999) diagram of the SMCRA data structure used to represent the atomic data. Full lines with triangle arrows indicate inheritance, full lines with diamonds indicate aggregation and dotted lines indicate interface realization.

as the identifier as a shortcut, e.g. `(" ", 120, " ")` can be replaced by `120` alone.

- `Atom`: The atom name, e.g. `"CA"`.

It is also possible to get a list of all children of a parent, e.g.:

```
residue_list=chain.get_list()
```

The parent can also be obtained from a child, e.g.:

```
chain=residue.get_parent()
```

Disordered atoms or residues (i.e. in the case of point mutations) are represented by `DisorderedAtom` and `DisorderedResidue` classes, which hide the complexity associated with disorder by forwarding all method calls to one of the wrapped `Atom` or `Residue` objects. Disordered residues arise when two or more point mutants of one polypeptide chain are present in the crystal (e.g. PDB file 1EN2, which has a Gly/Ser point mutation at position A10). The user can specify which of the disordered residues or atoms is 'seen' when using the SMCRA data structure. This approach corresponds to the chain of responsibility design pattern (Gamma *et al.*, 1995), and effectively shields the user from the complexity of disorder while still providing a full representation.

The data structure described above allows to access the atomic data in a PDB file in a very convenient manner. For example, the `Atom` class contains methods to extract the temperature factor (isotropic or anisotropic), coordinates, occupancy, etc. In addition, code is available to locate all atoms within a certain distance of each other, to perform distance neighbor lookup [both implemented using a KD tree data structure (Bentley, 1975) implemented in C++], and to extract all polypeptides from a PDB file.

It is a well-known fact that many PDB files in the Brookhaven database do not respect the prescribed syntax, or are difficult to interpret unambiguously. Therefore, some common problems are identified and either cause an exception (if the `PERMISSIVE` flag is set to 0, which is the default; see below) or are automatically corrected. The former problems include multiple residues or atoms with the same identifier, while the latter include disordered atoms with blanc altlocs. Although many of these problems are fixed in the `mmCIF` files available from the PDB, wide use of these files likely awaits the corresponding `XML` versions.

3 EVALUATION

The PDB parser was evaluated by parsing 5405 PDB files from the PDB_SELECT database (Hobohm and Sander, 1994), which contains a representative list of protein structures (April 2002 release, 90% sequence identity cutoff). Obsolete PDB entries were left out. The `PERMISSIVE` flag of the parser

was set to 1. Parsing all files took 3 h and 50 min on a 1.6 GHz PC, or on average about 2.5 s per structure. All files could be parsed successfully. Some interesting and non-trivial cases are mentioned below.

Structure 1EN2 contains five fully disordered residues due to the presence of point mutants in the crystal (Gly/Ser A10, Ala/Gly A14, Trp/Arg A16, Gly/Ser A80 and Asn/Lys A81). This is correctly interpreted, and the residues in question are stored in `DisorderedResidue` objects.

A common error in PDB files is the occurrence of residues with identical identifiers (present in 0.8% of the parsed PDB files). Water residues with identical residue identifiers were detected in numerous PDB files (e.g. PDB file 1AE9 contains 97 water residue pairs with identical identifiers, presumably due to missing chain identifiers). In structure 1AZZ, amino acid D27 is defined twice. Upon inspection it turns out that the second residue labeled D27 is located in between residues D46 and D48, and should thus presumably be renamed to D47. A similar situation exists in 1BD2, where Glu B48 should presumably be Glu B47. In 1BE3, a duplicate residue Leu A3 (presumably Leu A203) is present between residues A202 and A204. Duplicated atoms are also abundant (present in 1.6% of the parsed PDB files), in most cases due to missing altloc identifiers. In all those cases, a warning is generated and the duplicated residues or atoms are left out (if `PERMISSIVE=1`) or an exception is generated (if `PERMISSIVE=0`).

4 USAGE EXAMPLE

The following example prints a list of all residues in the PDB file `1fat.pdb` that contain a disordered 'CA' atom:

```
from PDBParser import PDBParser
parser=PDBParser(PERMISSIVE=1)
structure=parser.get_structure("1fat", "1fat.pdb")
for model in structure.get_list():
   for chain in model.get_list():
     for residue in chain.get_list():
        if residue.has_id("CA"):
            ca_atom=residue["CA"]
            if ca_atom.is_disordered():
               print residue
```

5 CONCLUSIONS

The PDB parser and its associated data structure described above is a powerful, complete, very easy-to-use and freely available tool to extract atomic data from a PDB file. This will make it e.g. possible for crystallographers derive simple statistics from the PDB in a straightforward way, or for structural bioinformaticians to build more complex applications based on the available code. One of the main shortcomings is the fact that the modules only deal with the atomic data, and not with the information in the PDB header (which contains, e.g. information on refinement, space group, protein, etc.). Another drawback is speed: since Python is an interpreted language, its execution speed is lower than for compiled languages like C++ (e.g. parsing the structure of the large ribosomal subunit 1FKK which contains about 65 000 atoms takes about 30 s on a 1.6 GHz PC). For most applications however, the modules are fast enough. New structure related features will be added to biopython in the future, and contributors to this open source project are welcome.

ACKNOWLEDGEMENTS

The authors thank all Biopython contributors. T.H. is supported by the *Horizontale onderzoeksactie—inverse protein folding* of the VUB. Additional support from the EU (TEMBLOR, FW5 1999/C381/06) is acknowledged.

REFERENCES

Bentley,J. (1975) Multidimensional binary search trees used for associative searching. *Commun. ACM*, **18**, 509–517.

Berman,H., Westbrook,J., Feng,Z., Gilliland,G., Bhat,T., Weissig,H., Shindyalov,I. and Bourne,P. (2000) The Protein Data Bank. *Nucleic Acids Res.*, **28**, 235–242.

Chang,W., Shindyalov,I., Pu,C. and Bourne,P. (1994) Design and application of PDBlib, a C++ macromolecular class library. *Comput. Appl. Biosci.*, **10**, 575–586.

Chapman,B. and Chang,J. (2000) Biopython: python tools for computational biology. *ACM SIGBIO Newslett.*, **20**, 15–19.

Coon,S., Sanner,M. and Olson,A. (2001) Re-usable Components for Structural Bioinformatics. In *9th International Python Conference*.

Pls provide page range.

DeLano,W. (2002). *The PyMOL Molecular Graphics System*. DeLano Scientific, San Carlos, CA, USA.

Fowler,M. (1999) *UML Distilled*. Addison-Wesley, New York.

Gamma,E., Helm,R., Johnson,R. and Vlissides,J. (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, Addison-Wesley, New York.

Hinsen,K. (2000) The molecular modeling toolkit: a new approach to molecular simulations. *J. Comp. Chem.*, **21**, 79–85.

Hobohm,U. and Sander,C. (1994) Enlarged representative set of protein structures. *Protein Sci.*, **3**, 522–524.

Sanner,M. (1999) Python: a programming language for software integration and development. *J. Mol. Graph Model*, **17**, 57–61.

Sanner,M., Duncan,B., Carrillo,C. and Olson,A. (1999) Integrating computation and visualization for biomolecular analysis: an example using python and AVS. In *Proceedings of the Pacific Symposium in Biocomputing*, pp. 401–412.