User's guide of X3DNA Parser

Yurong Xin

January 17, 2008

1 Introduction

The X3DNAParser is created to process data generated by 3DNA software package. The UML diagram in Figure 1 illustrates the classes and their associations in X3DNAParser. Codes for processing PDB files (enclosed in dashed box in Figure 1) use the Bio.PDB module in Biopython (*http://www.biopython.org/*) with some modifications. An RNA structure is processed hierarchically by a SMCRA scheme (Structure, Model, Chain, Residue, and Atom) to construct a structure object. The object is then passed to the X3DNA parser which interprets the output files of 3DNA. The X3DNA parser contains four child classes which serve different purposes. The *Secondary* class is used to create helix objects which are composed of base pairs. The *Tertiary* class constructs base-pair objects that are not embedded in helical regions. Objects of base-pair steps are generated by the *StepWrapper* class, and higher-order base interactions are parsed by the *MultiWrapper* class.

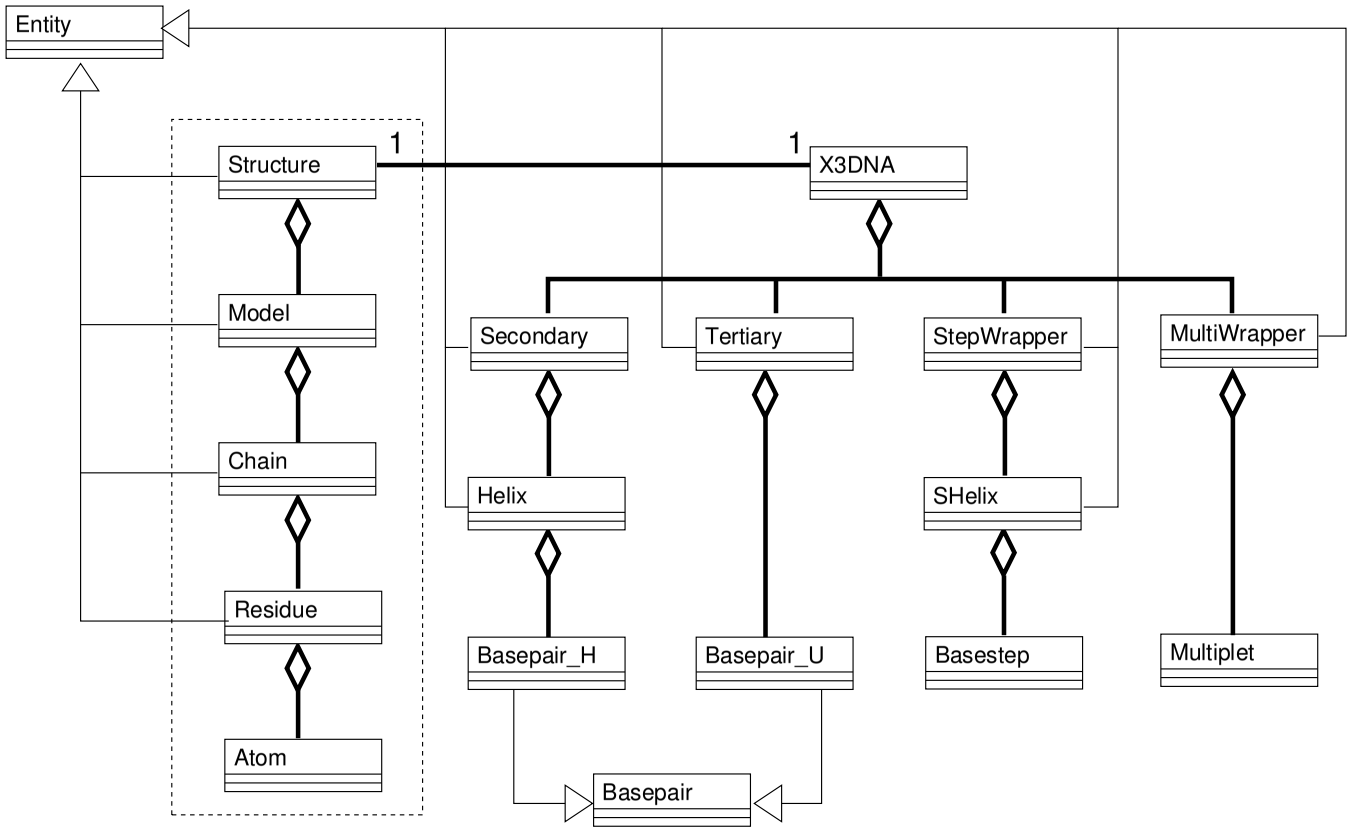

Figure 1: UML diagram of the X3DNAParser package. Classes enclosed in the dashed box are derived from the Bio.PDB package. Bold lines with diamonds denote aggregation, bold lines with numbers denote associations, and lines with triangles denote inheritance.

One useful feature of this parser is to show structural context of residues and base pairs. We use a set of terms to describe structural context in terms of helical and single-stranded secondary elements (Figure 2).

H Continuous helical region in the given helix.

$\mathbf{H}^e$ End of the given helix and in a continuous helical region.

$\mathbf{H}_i^e$ End of the given helix and in a inserted base pair.

$\mathbf{H}_q^e$ Break of a continuous helical region in the given helix.

$\mathbf{H}_i$ Inserted base pair in the given helix.

$\mathbf{H}_l$ Lone helix (with only one base pair).

$\mathbf{S}_b$ Internal loop.

$\mathbf{S}_h$ Terminal loop.

$\mathbf{S}_l$ Single-stranded segment that links two (quasi-continuous) helices.

$\mathbf{S}_e$ Single-stranded segment at the end of the molecule.

2 Input files

The X3DNAParser needs three files: PDB, "bp", "out". The PDB file is a format used in Protein Data Bank and Nucleic Acid Database. The "bp" file is generated using following command:

```
find_pair -p a.pdb a.bp
```

The "out" file is generated as follows:

```
find_pair a.pdb stdout | analyze
```

It will automatically generate a file named "`a.out`" in most cases. This file is optional because some structures will not have it.

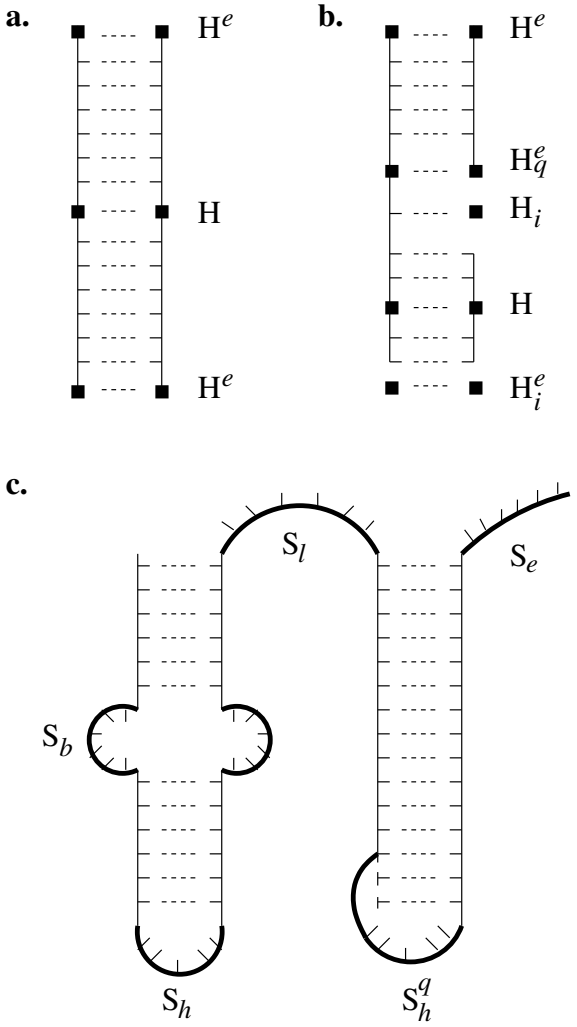

Figure 2: Schematic illustration of the structural context of base pairs. Helices are represented by continuous or quasi-continuous paired lines. Ticks connected by dashed lines denote paired bases in helices. Small black boxes illustrate nucleotides involved in different helical contexts. Single-stranded segments are denoted by bold lines.

3 Examples

The X3DNAParser provides information of 3D structure (based on PDB coordinates) and interaction (based on 3DNA output files). Users can extract information from different levels, such as residue level and base-pair level. A few examples will demonstrate usage of the parser.

3.1 Get all base pairs for a given structure

Base pairs are classified into two groups: those embedded in helices and those outside helices. The example below shows how to get base-pair list in helical regions. If users want to get a list of non-canonical base pairs not embedded in helices, they can use the second example.

Example 1.

```
from PDBParser import PDBParser
from X3DNAParser import X3DNAParser
pdb_parser = PDBParser()
pdb_parser.parse_pdb('2gdi', '2gdi.pdb')
structure = pdb_parser.get_structure()
x3dna_parser = X3DNAParser('2gdi', structure)
x3dna_parser.parse_x3dna('2gdi.bp', '2gdi.out')
x3dna = x3dna_parser.get_x3dna()
secondary = x3dna['secondary']
for element in secondary.get_iterator():
    if element.get_sec_class() == 'helix':
        print "helix #%d: %d base pairs" % (element.get_id(), \
                                            element.get_number())
        for basepair in element.get_child_list():
            res1, res2 = basepair.get_respair()
            tag1 = res1.get_resname() + ":" + str(res1.get_resseq()) + \
                   ":" + res1.get_full_id()[2]
            tag2 = res2.get_resname() + ":" + str(res2.get_resseq()) + \
                   ":" + res2.get_full_id()[2]
            context1 = res1.get_continuity()
            context2 = res2.get_continuity()
            print "\t%s|%s\t%s|%s" % (tag1, context1, tag2, context2)
```

Example 2.

```
from PDBParser import PDBParser
from X3DNAParser import X3DNAParser
pdb_parser = PDBParser()
pdb_parser.parse_pdb('2gdi', '2gdi.pdb')
structure = pdb_parser.get_structure()
x3dna_parser = X3DNAParser('2gdi', structure)
x3dna_parser.parse_x3dna('2gdi.bp', '2gdi.out')
x3dna = x3dna_parser.get_x3dna()
tertiary = x3dna['tertiary']
for basepair in tertiary.get_iterator():
    res1, res2 = basepair.get_respair()
    tag1 = res1.get_resname() + ":" + str(res1.get_resseq()) + \
           ":" + res1.get_full_id()[2]
    tag2 = res2.get_resname() + ":" + str(res2.get_resseq()) + \
           ":" + res2.get_full_id()[2]
    context1 = res1.get_continuity()
    context2 = res2.get_continuity()
    sec1 = res1.get_sec_id()
    sec2 = res2.get_sec_id()
    print "\t%s|%s|#%d\t%s|%s|#%d" % (tag1, context1, sec1, \
                                      tag2, context2, sec2)
```