Biopython-corba How-To

Brad Chapman (chapmanb@arches.uga.edu)

Last Update–23 September 2001

Contents

1 Introduction

This tutorial is designed to get you up and running quickly with biopython-corba. Biopython-corba does all of the following good stuff:

- Provides a complete implementation of the BioCorba (http://www.biocorba.org) specification that allows interoperation between the different bio projects (biopython, bioperl, biojava). This specification is currently merging with the Life Science Research groups' spec, called BSANE (http://industry.ebi.ac.uk/~muilu/OMG/BSANE/index.html), so it is even more standard (and thus useful!).
- Provides a biopython-like set of interfaces on top of the BioCorba specification, so that using biopython-corba is a lot like using biopython (less to learn, easier to get started!).
- Works interchangably with three different python ORB implementations (omniORBpy, ORBit-python and Fnorb), allowing you to switch ORBs just by changing a line of text in the configuration file.
- Makes it easy to get standard biopython objects "CORBA-ready," so that little work is required to make your biopython code usable as a CORBA server.
- Provides all the standard good stuff that comes with CORBA: position and language independent code (connect easily across a network; talk between Perl and Python and Java).

2 Installation

2.1 Requirements

In order to use biopython-corba, you need to install Biopython and a CORBA ORB implementation.

Biopython is available from (http://www.biopython.org). Currently, you need a current version of CVS to run biopython-corba, which you can check out with instructions from http://cvs.biopython.org. Once a new release becomes after 1.00a3 becomes available, you can get one of those from the standard Download place (http://www.biopython.org/Download/).

You will also need an ORB implementation. Currently you can choose from one of the following:

1. omniORB - Python bindings are available from http://www.uk.research.att.com/omniORB/omniORBpy/ and omniORB itself is available from http://www.uk.research.att.com/omniORB/index.html. omniORBpy is an excellent python ORB under continuous development. It is quite fast and interoperable. It is quite a large C++ program though, so it can take some effort and time to install.

2. ORBit and ORBit-python – ORBit (http://orbit-resource.sourceforge.net/) is the standard ORB for the Gnome project. If you run Gnome or a standard Linux distribution, it may already be installed. Try typing `orbit-config --version`. If you get back something like ORBit 0.5.8, then it's already installed and you just need the ORBit-python bindings, available from http://orbit-python.sault.org/. Currently, you need a CVS version of ORBit-python but 0.3.1 or later should work with biopython-corba. An addtional item that needs to be done is to get ORBit working in an interoperable manner. To do this, you need to create a `.orbitrc` file in your home directory with the line: `ORBIIOPIPv4=1` in it.

3. Fnorb – Fnorb is an ORB implementation written almost entirely in python, and is available from http://www.fnorb.org/. It is slower than the other ORBs, but is relatively easy to install since it is all python. One catch is that Fnorb needs some patches to work with biopython-corba and with python versions later than 2.0. You will need to get and apply the patch at http://bioinformatics.org/bradstuff/bc/fnorb.patch for everything to work happily.

2.2 Getting biopython-corba installed

Now that you've got the requirements installed, we're ready to get this installed. Here we go:

1. Get biopython-corba, which is available by anonymous cvs at biopython (see http://cvs.biopython.orgfor instructions), or via download at http://www.biopython.org/Download/. Presumedly you already have this though, if you are reading this :)

2. Edit the configuration file. This file is located in BioCorba/biocorbaconfig.py, and contains the options available for building and using biopython-corba. Basically, you will just want to edit the variable `orb_implementation` to match the python ORB implementation you installed above.

3. Change into the main biopython-corba directory and excecute, as root, the command `python setup.py install`. This will do all the necessary magic and install everything.

2.3 Testing the installation

There are a number of tests available in the Tests directory. This describes how to run 'em. Feel free to keep running them until you are satisfied that everything installed smoothly.

- Unittests – There are several modules which test individual parts of the libraries. You can run these tests simply with: `python test_Sequences.py`, `python test_Features.py`, `python test_Collections.py` and `python test_PythonBioInterfaces.py`. If these work you should see a lot of 'ok's printed, and no error messages.

- Complete Server tests – There are also more "real-life" tests available for a python server. First, you need to start up the server, so change to the `Scripts` directory and run `python collections_server.py`. Now, in a different xterm window, run the test: `python test_PythonServers.py`. You should see a whole lot of output produced and no errors.

3 Getting Started

This section is designed to get you up and running quickly with biopython-corba. After reading this section, the goals are that you should:

1. know how to make a Biopython Dictionary object (like a GenBank file) available as a python server
2. Understand how to use a biopython-corba client to connect to a Dictionary-like object and deal with it.

This section assumes some basic familiarity with biopython and CORBA.

3.1 Setting up a BioCorba server from a Biopython Dictionary

The big advantage of the biopython-corba libraries are that they make it easy to take a biopython object and make it available through CORBA. This example shows how to take GenBank information represented as a Biopython Dictionary object, and turn it into a BioCorba BioSequenceCollection.

3.1.1 Setting up the biopython Dictionary

First, we need to create a Biopython Dictionary. Biopython Dictionary objects act like standard python dictionaries, except that they provide biological information. In this example, we are going to create a dictionary to a GenBank file of drought related proteins in Arabidopsis.

We do this in the exact same way it is done in biopython. First we index the GenBank file, then we set up a dictionary that will return `SeqRecord` objects for each GenBank item in the dictionary:

```
from Bio import GenBank

gb_file = "a_drought.gb"
index_file = "a_drought.idx"
GenBank.index_file(gb_file, index_file)
feature_parser = GenBank.FeatureParser()
gb_dictionary = GenBank.Dictionary(index_file, feature_parser)
```

Now we've got a dictionary that is ready to be fed into a BioCorba server.

3.1.2 Making the Dictionary a BioCorba server

The biopython-corba server code does all of the hard work of taking the dictionary you give it and making it follow the BioCorba standards. All you need to do is follow the right way of presenting your objects. In BioCorba the standard database-like object is called a `BioSequenceCollection`. You can give a `BioSequenceCollection` any Dictionary-like object from Biopython (ie. Fasta dictionaries, etc) or any object of your own that behaves like a Dictionary (implements `__getitem__` and `keys`). We give `BioSequenceCollection` our Dictionary object, the name of the database, a version and the description, and it does the rest.

Next, we need to make a reference to the server available. This reference is analagous to a web URL, except it is more complicated because it needs to specify the entire path to get to the host, port and object itself. In CORBA these references are called Interoperable Object References (IORs). In this example, we will write a stringified version of the IOR for the server to a file so a client can get it. In real life, you might put this IOR on a web site, e-mail it to a collaborator, or somehow get it to someone else.

Once the reference to the server is written out, all we need to do is get the server running so it can sit and wait for clients to connect to it. Whew, all that seemed like a lot of work, but the following small amount of code will get it done:

```
from BioCorba.Server.Seqcore.CorbaCollection import BioSequenceCollection

server = BioSequenceCollection(gb_dictionary, "Arabidopsis Drought Database",
           1.0, "Putative Drought Resistance Genes in Arabidopsis")

print "Writing IOR to a file and starting the server..."
server.string_ior_to_file("a_drought.ior")
server.run()
```

Whoo hoo – with just this little bit of code we've taken a GenBank file and made it available through the BioCorba interface. With the server you've set up, people can write code in any language to attach to this server and retrieve the information out of it. Isn't CORBA a great thing?

3.2 Using a BioCorba client though a Biopython-like interface

This section describes retrieving information through BioCorba. The recommended method to use the biopython-corba interfaces is through the Bio-like interfaces, which emulate the biopython interfaces. It's better to write your code against these interfaces for a couple of reasons. First, they're easier to learn since they resemble, as close as possible, biopython code. Second, these APIs will be more stable between biopython-corba releases. The underlying BioCorba interfaces are likely to change, and this has been especially true as we've been working to get the CORBA interfaces "right."

We'll demonstrate how to take a BioCorba `BioSequenceCollection` server and get information from it through a GenBank-like interface. This will show retrieving SeqRecord objects from a Dictionary-like interface, and getting features on these SeqRecords.

3.2.1 Getting the BioCorba server and connecting to it

The trickiest part of learning how to use BioCorba is understanding how to connect to a server. The biopython-corba interfaces try to make this job easier by makine it involve only 2 steps:

1. Get a "loader" object that can be used to get connections with servers of a specific type. In our case, we are going to be creating a loader that will give us a `BioSequenceCollection` from a python based server.

2. Use the "loader" object to get a client connection from a server, based on a server reference. As we mentioned before, the reference to a server is called an IOR, and can be located in a number of possible places (ie. web site, file). In this case we'll be connecting to our server from a reference in a file.

Now that we've said all of that, the code to get a client reference to a `BioSequenceCollection` should (hopefully) make sense:

```
from BioCorba.Client.BiocorbaConnect import PythonCorbaClient
from BioCorba.Client.Seqcore.CorbaCollection import BioSequenceCollection

ior_file = 'a_drought.ior'

client_loader = PythonCorbaClient(BioSequenceCollection)
client = client_loader.from_file_ior(ior_file)
```

Now we've got a `BioSequenceCollection` client connected to a server. This client uses the raw BioCorba interfaces, but we'd like to be able to use the Biopython-like interfaces. To do this, we will load up a GenBank Dictionary object in much the same way we do in the Biopython. The major difference is that instead of using

a file handle to a GenBank file we will use our connected client. So, we can just think of the client/server connection as a way to get info out of a GenBank file. The code to get the GenBank dictionary should hopefully look familiar (except that the imports come from BioCorba.Bio instead of just Bio):

```
from BioCorba.Bio import GenBank
feature_parser = GenBank.FeatureParser()
gb_dict = GenBank.Dictionary(client, feature_parser)
```

Whoo hoo, now we've got our Biopython-like GenBank dictionary. As we'll see in the next section, we can use this exactly like we would a normal Biopython Dictionary object.

3.2.2 Using the Biopython-like interface

Now that we've got our GenBank-like object, we are ready to use it exactly like we would any Dictionary object from Biopython. We don't need to worry about CORBA or anything at all. The following example code, which should be familiar to anyone using the standard SeqRecord and SeqFeature objects from Biopython, will just look at the available accessions in the database, and then retrieve a record and look at it:

```
accessions = gb_dict.keys()
print "Accessions:", accessions

seq_record = gb_dict['BE038456']
print "name:", seq_record.name
print "id:", seq_record.id
print "description:", seq_record.description
print "First 50 bases:", seq_record.seq[:50]
print "First SeqFeature:"
print "\tType:", seq_record.features[0].type
print "\tLocation:", seq_record.features[0].location
print "\tQualifiers:", seq_record.features[0].qualifiers
```

This produces the following output:

```
ccessions: ['BE038008', 'BE038108', 'BE038455', 'BE038456', 'BE038835',
'BE844799']
name: BE038456
id: BE038456.1
description: AA17G06 AA Arabidopsis thaliana cDNA 5' similar to
drought-induced protein di19, mRNA sequence.
First 50 bases:
Seq('GCACGAGCTTCTTCTTCATCTATTCAGGATCGATCCGGCTTGAGATTTAT', RNAAlphabet())
First SeqFeature:
    Type: source
    Location: (0..708)
    Qualifiers: {'clone_lib': ['AA'], 'organism': ['Arabidopsis thaliana'],
'dev_stage': ['12 weeks'], 'cultivar': ['Columbia'],
'tissue_type': ['leaves, flowering plants'],
'db_xref': ['taxon:3702'], 'note': ['20 h 200mM NaCl']}
```

Nice. Now we're set up for both creating server objects and accessing them through a Biopython-like interface. Hopefully this Tutorial gave enough instruction to get you started exploring this further.

4 Other Useful Information

4.1 Learning more about Python and CORBA

Alan Robinson has written a very nice introduction to writing CORBA clients, which has a path on python. You can check this out on line at http://industry.ebi.ac.uk/~alan/CORBA/. The python section is also available in the Doc directory in pdf as `IntroPythonClient.pdf`.

I also wrote a similar paper which is meant as an introduction to writing python servers. It is available in the Doc directory in pdf format in the file `IntroPythonServer.pdf`.

4.2 The Standard Python Mapping

The mapping of IDL to python has been standardized, and documentation describing this official mapping is available (http://www.omg.org/cgi-bin/doc?ptc/00-04-08). Due to recent acceptance of this standard, the degree to which different python ORB implementations follow it vary, unfortunately. The biopython-corba package attempts to standardize "non-compliant" ORB implementations so that they can be used through the same interface. This makes it possible to use any supported ORB with the biopython-corba interfaces without needed to change any code.