Persistent Bioperl

BOSC 2003
Hilmar Lapp
Genomics Institute
Of The Novartis Research Foundation
San Diego, USA
• Bio* contributors and core developers
  ▪ Aaron, Ewan, ThomasD, Matthew, Mark, Elia, ChrisM, BradC, Jeff Chang, Toshiaki Katayama
  ▪ And many others
• Sponsors of Biohackathons
  ▪ Apple (Singapore 2003)
  ▪ O’Reilly (Tucson 2002)
  ▪ Electric Genetics (Cape Town 2002)
• GNF for its generous support of OSS development
Overview

• Use cases
• BioSQL Schema
• Bioperl-DB
  ▪ Key features and design goals
  ▪ Examples
• Status & Plans
• Summary
• ‘Local GenBank with random access’
  ▪ Local cache or replication of public databanks
  ▪ Indexed random access, easy retrieval
  ▪ Preserves annotation (features, dbxrefs,...), possibly even format
• ‘GenBank in relational format’
  ▪ Normalized schema, predictably populated
  ▪ Allows arbitrary queries
  ▪ Allows tables to be added to support my data/question/...
• ‘Integrate GenBank, Swiss-Prot, LocusLink, …’
  ▪ Unifying relational schema
  ▪ Provide common (abstracted) view on different sources of annotated genes

• ‘Database for my lab sequences and my annotation’
  ▪ Store FASTA-formatted sequences
  ▪ Add, update, modify, remove various types of annotation
• Persistent storage for my favorite Bio* toolkit
  ▪ Relational model accommodates object model
  ▪ Persistence API with transparent insert, update, delete
• Normalized relational schema designed for Bio* interoperability

• Toolkit-specific persistence API
BioSQL

- Interoperable relational data store for Bio*
  - Language bindings presently for Bioperl, Biojava, Biopython, Bioruby
- Very flexible, normalized, ontology-driven schema
  - Focal entities are Bioentry, Seqfeature, Term (and Dbxref)
- Schema instantiation scripts for different RDBMSs
  - MySQL, PostgreSQL, Oracle
- Release of v1.0 imminent
  - Schema has been stable for the last 3 months
  - Relatively well documented (installation, how-to, ERD)
- Mailing list (biosql-l@open-bio.org), CVS (biosql-schema), links at http://obda.open-bio.org
BioSQL: Some History

- Ewan Birney started BioSQL and Bioperl-db in Nov 2001
  - Initial use-case was to serialize/de-serialize Bio::Seq objects to/from a local sequence store (as a replacement for SRS)
- Schema redesigned at the 2002 Biohackathons in Tucson and Cape Town
  - Series of incremental changes later in 2002
- Full review at the 2003 Biohackathon in Singapore
  - Changed Taxon model to follow NCBI’s
  - Full ontology model, resembles GO’s model
  - Features can have dbxrefs
  - Consistent naming
Object-Relational Mapping connects two worlds
• Object model (Bioperl) \ Relational model (Biosql)
• Object and relational models are orthogonal (though ‘correlated’)
  • E.g., inheritance, n:n associations, navigability of associations, joins

General goals of the OR mapping are
• Bi-directional map between objects and entities
• Transparent persistence interface reflecting all of INSERT, UPDATE, DELETE, SELECT

Generic approaches exist, most of which are commercial
• TopLink, CMP (e.g., Jboss), JDO, Tangram
# get persistence adaptor factory for database
my $db = Bio::DB::BioDB->new(-database => 'biosql',
                          -dbcontext => $dbc);

# open stream of objects parsed from flatfile
my $stream = Bio::SeqIO->new(-fh => "/STDIN",
                           -format => 'genbank');

while(my $seq = $stream->next_seq()) {
    # convert to persistent object
    $pobj = $db->create_persistent($seq);
    # insert into datastore
    $pobj->create();
}
• Bioperl-db is a sub-project of Bioperl
  ▪ Email to bioperl-l@bioperl.org
    • but biosql-l@open-bio.org will often work, too
  ▪ CVS repository is bioperl-db under bioperl (/home/repository/bioperl/bioperl-db)

• No release of the current codebase yet
  ▪ But v0.2 is imminent
• Transparent persistence API on top of object API
  ▪ Persistent objects know their primary keys, can update, insert, and delete themselves
    • Full API in Bio::DB::PersistentObjectI
  ▪ Persistent objects speak both the persistence API and their native tongue

• Several retrieval methods on the persistence adaptor API:
  ▪ find_by_primary_key(), find_by_unique_key(), find_by_query(), find_by_association()
  ▪ Full API in Bio::DB::PersistenceAdaptorI
• Extensible framework separating object adaptor logic from schema logic
  ▪ Central factory loads and instantiates a datastore-specific adaptor factory at runtime.
  ▪ Adaptor factory loads and instantiates persistence adaptor at runtime - no hard-coded adaptor names
  ▪ Queries are constructed in object space and translated to SQL at run-time by schema driver
  ▪ Designed with adding bindings to other schemas than BioSQL in mind (e.g., Chado, Ensembl, MyBioSQL, …)
use Bio::DB::BioDB;
# create the database-specific adaptor factory
# (implements Bio::DB::DBAdaptorI)
$db = Bio::DB::BioDB->new(-database => "biosql",
                        -dbcontext => $dbc);

• Step 1: connect and obtain adaptor factory
• Step 2: depends on use case
  ▪ Load sequences:

```perl
use Bio::SeqIO;
# open stream of objects parsed from flatfile
my $stream = Bio::SeqIO->new(-fh     => \
  -format => 'genbank');
while(my $seq = $stream->next_seq()) {
    # convert to persistent object
    $pseq = $db->create_persistent($seq);
    # $pseq now implements Bio::DB::PersistentObjectI
    # in addition to what $seq implemented before
    # insert into datastore
    $pseq->create();
}
```
• Step 2: depends on use case
  ▪ Retrieve sequences by alternative key:

```perl
use Bio::Seq; use Bio::Seq::SeqFactory;
# set up Seq object as query template
$seq = Bio::Seq->new(-accession_number => "NM_000149",
                     -namespace => "RefSeq");
# pass a factory to leave the template object untouched
$seqfact = Bio::Seq::SeqFactory->new(-type=>"Bio::Seq");
# obtain object adaptor to query (class name works too)
# adaptors implement Bio::DB::PersistenceAdaptorI
$adp = $db->get_object_adaptor($seq);
# execute query
$dbseq = $adp->find_by_unique_key(
    $seq, -obj_factory => $seqfact);
warn $seq->accession_number(),
    " not found in namespace RefSeq\n" unless $dbseq;
```
• **Step 2:** depends on use case
  - Retrieve sequences by query:

```perl
use Bio::DB::Query::BioQuery;
# set up query object as query template
$query = Bio::DB::Query::BioQuery->new(
    -datacollections => ["Bio::Seq s",
        "Bio::Species=>Bio::Seq sp"],
    -where => ["s.description like '%kinase%'",
        "sp.binomial = ?"]);

# obtain object adaptor to query
$adp = $db->get_object_adaptor("Bio::SeqI");
# execute query
$qres = $adp->find_by_query($query, -name => "bosc03",
    -values => ["Homo sapiens"]);

# loop over result set
while(my $pseq = $qres->next_object()) {
    print $pseq->accession_number,"\n";
}
```
• Step 2: depends on use case
  - Retrieve sequence, add annotation, update in the db

```perl
use Bio::Seq; use Bio::SeqFeature::Generic;
# retrieve the sequence object somehow ...
$adp = $db->get_object_adaptor("Bio::SeqI");
$dbseq = $adp->find_by_unique_key(
    Bio::Seq->new(-accession_number => "NM_000149",
                  -namespace => "RefSeq"));
# create a feature as new annotation
$feat = Bio::SeqFeature::Generic->new(
    -primary_tag => "TFBS",
    -source_tag => "My Lab",
    -start=>23,-end=>27,-strand=>-1);
# add new annotation to the sequence
$dbseq->add_SeqFeature($feat);
# update in the database
$dbseq->store();
```
• Extensibility: handle my own object by adding my own adaptor. A) Custom sequence class

```perl
package MyLab::Y2HSeq;
@ISA = qw(Bio::Seq);
sub get_interactors{
    my $self = shift;
    return @{$self->{_interactors}};
}
sub add_interactor{
    my $self = shift;
    push(@{$self->{_interactors}}, @_);
}
sub remove_interactors{
    my $self = shift;
    my @arr = $self->get_interactors();
    $self->{_interactors} = [];
    return @arr;
}
```
• Extensibility: handle my own object by adding my own adaptor. B) Custom adaptor class

```perl
package Bio::DB::BioSQL::Y2HSeqAdaptor;
@ISA = qw(Bio::DB::BioSQL::SeqAdaptor);
sub store_children{
    my ($self,$obj) = @_;  
    # call inherited method
    $self->SUPER::store_children(@_);
    # obtain persistent term object for the rel.ship type
    my $term = Bio::Ontology::Term->new(
        -name => "interacts-with",
        -ontology => "Relationship Types";
    my $termadp = $self->db->get_object_adaptor($term);
    my $reltype = $termadp->find_by_unique_key($term) or
        $self->db->create_persistent($term)->create();
    # continued on the next page ...
```
Extensibility: handle my own object by adding my own adaptor. B) Custom adaptor class (cont’d)

```perl
# store the interacting sequences
foreach my $seq ($obj->get_interactors()) {
    # each interactor needs to be persistent object
    $seq = $self->db->create_persistent($seq)
        unless $seq->isa("Bio::DB::PersistentObjectI");
    # each interactor also needs to have a primary key
    $seq = $seq->adaptor->find_by_unique_key() or
           $seq->create();
    # associate the interactor with this object
    $seq->adaptor->add_association(
        -objs => [$obj, $seq, $reltype],
        -contexts => ["object","subject",undef]);
}
return 1; # done
```
• **load_seqdatabase.pl** (bioperl-db/scripts/biosql)
  - Use for loading and updating bioentries and their annotation
  - Supports all Bio::SeqIO supported formats
    - genbank, embl, swiss, locuslink, fasta, gcg, ace, ...
  - Supports all Bio::ClusterIO supported formats
    - Unigene

• Many command line options
  - For flexible handling of updates
    - --lookup, --noupdate, --remove, --mergeobjs
  - For filtering and post-processing sequences
    - --seqfilter, --pipeline
• load_ontology.pl (bioperl-db/scripts/biosql)
  ▪ Use for loading and updating ontologies and terms
  ▪ Supports all Bio::OntologyIO supported formats
    • dagflat (incl. soflat, goflat), InterPro, simplehierarchy
  ▪ Tested for GO and SOFA
• Many command-line options
  ▪ For handling updates and obsoleted terms
    • --lookup, --noupdate, --remove
    • --noobsolete, --updobsolete, --delobsolete, --mergeobjs
  ▪ For (re-)computing the transitive closure
    • --computetc
• **load_ncbi_taxonomy.pl** (biosql-schema/scripts)
  - Use for loading and updating the taxon tables with the NCBI Taxonomy database
  - Downloads the database from NCBI automatically if desired
  - Some options to configure and tune load and update
  - Automatically updates the Nested Set values in the taxon table
Current Status

- BioSQL is stable and release-ready
  - Imminent release of v1.0
  - Well-documented ;-), ER-diagram
  - Supports MySQL, PostgreSQL, and Oracle
  - Toolkit-independent script for populating taxa
- Bioperl-db is stable but documentation is patchy
  - Core APIs stable and documented, but no How-To’s
  - All tests pass on all 3 RDBMS platforms
  - Head revision wants Bioperl >= 1.2.2 (but for RichSeqI attributes you need Bioperl main trunk)
  - Fuzzy locations get transformed to simple locations
- BioSQL & Bioperl-db are used in production and at multiple places
• Persistence Adaptors for more object types
  ▪ Phenotypes (OMIM)
  ▪ Markers (SNPs, STSs, …)
• Increased support for lazy loading
  ▪ Features and annotations for a sequence (sequence itself is already lazy-loaded)
• Write adaptors for other applications to run off of BioSQL
  ▪ Genome browsers: GBrowse, Apollo
  ▪ Ontology editors: DAG-edit
Plans For The Future (II)

• Proof-of-Concept for interoperability
  ▪ Load through Bioperl/Bioperl-db, retrieve through Biojava

• Proof-of-Concept of the architecture’s flexibility
  ▪ Map to schemas different from BioSQL: Chado, Ensembl
BioSQL is a very flexible, ontology-driven, stable relational schema to capture richly annotated databank entries

BioSQL is supported as the persistent storage across the Bio* projects

Bioperl-db is the object-relational mapping for Bioperl objects to BioSQL

Bioperl-db adds a transparent persistence API on top of all supported Bioperl objects

Presently supported areas of the object model are sequences, features, annotations, clusters, ontologies