This is a summary and detailed notes from the *Chado Middleware Bake Off* held at the January 2007 GMOD Meeting.

Executive Summary

Although GMOD uses the Chado schema for its underlying database, each group has developed a separate interface to their databases. This meeting was designed to review the experiences each group has had with its particular solution and to recommend a "best practice."

All participants had developed some type of Object-Relational Mapping (ORM) tool. The tools fell into two general classes. The first class, which includes Hibernate (Java), iBatis (Java), and Chado::AutoDBI (Perl) examines the relational schema and configuration files to create the middleware automatically, or create a relational schema and middleware automatically from a data model (InterMine). The second class involved hand-coding an API to create an object-oriented interface to Chado. Of the tools presented, Modware, built on top of Chado::AutoDBI, was the most mature and feature-rich.

The consensus from the meeting was that while fully automatic tools are convenient, that there is considerable value in creating a hand-crafted predictable API that reflects the biological data closely. The approach taken by Modware, which starts from an auto-generated interface and then adds a hand-coded API layer, is both comprehensible and powerful. The hand-edited layer will serve the common cases in detail while the auto-generated layer retains the full flexibility of the Chado schema for the cases that fall outside the hand-edited API. We recommend that implementors of Perl-based GMOD tools seriously consider Modware for their middleware interface to Chado, and that the implementors of Java-based tools collaborate to create a Java API that parallels Modwares.

Contents

Middleware for Chado databases

Participants

On January 19 2007, representatives of the major model organism databases (MODs) and other interested parties met to discuss and compare Middleware packages used by developers working for the MODs. The workshop attendees were tasked to make specific recommendations. Such recommendations will focus the efforts of the MODs on specific packages and lead to code re-use and more feature-rich middleware.

After an introductory presentation attendees listened to a series of presentations on the different Middleware packages. Each presenter described the features of the package, both positive and negative, and showed how the package would be used to address a specific set of test problems. The attendees reassembled in a general discussion group to discuss the presentations and to develop a consensus statement. This document represents the consensus of the workshop and shows the individual presentations.

Introduction

A group of some 50 GMOD developers gathered at the annual meeting to discuss middleware. This one day meeting had the following general goals:

- To educate GMOD programmers on methods and practices for Middleware
- To facilitate discussion on the best methods
- To guide GMOD to a uniform Middleware layer
- To generate this central reference document for Middleware projects, including:
  - Platform information
  - Strengths & weaknesses of different Middleware packages
  - Specific examples of how one would use a given middleware package

One of the key characteristics of the GMOD software project is the variety of approaches and components that it supports. This applies to applications, database schemas, as well as to middleware, a software *layer* that mediates the exchange of data between the applications and the databases. Despite this diversity certain applications and schemas have emerged as key supported components in GMOD, such as the GBrowse application and the Chado schema, to name just two. However, a consensus view has not emerged with respect to middleware, and there are certainly a number of different middleware packages that have been used in the GMOD world, coming from within this world and from the larger world of open source.

In late 2006 the GMOD developers took note of the large number of middleware packages in use and elected to embark on a short-term study to evaluate and compare these packages. The primary motivation here was to select or recommend certain packages over others specifically within the GMOD context. The assumption is that making such recommendations will serve to focus the developers' effort on a smaller number of packages. Clearly it is also assumed that such a focus will inevitably lead to greater support for and use of those recommended packages, and that all within GMOD will benefit.

Another purpose of this study is to educate GMOD programmers on best practices concerning the use and development of middleware. It is expected that common agreement on these practices will lead to the development of more effective software as well as the best use of the software in practice. Finally, this study should generate a central reference document on these different middleware packages used in GMOD. This reference will contain platform- and language-specific information as well as descriptions of the strengths and weaknesses of the packages that can be used by GMOD developers when considering middleware.

General Evaluation Criteria

The GMOD developers proposed that each presenter provide some basic information about each middleware package, both general and technical. In addition each middleware application was asked to address a set of sample problems, shown below. These example problems are thought to typify some of the common functions that a scientist may need when working with their own database. It was understood that not all software would be able to handle all aspects of the sample problems and this demonstration was not intended to be *live*.

Problem 1

Enter the information about the following three novel genes, including the associated mRNA structures, into your database. Print the assigned `feature_id` for each inserted gene.

Note:

- The coordinates are given in exact coordinates.
- Use the `organism_id` for your organism
- Store description in the chado table `featureprop`
- A sequence in fasta format (see Fake Chromosome) should be loaded as genomic sequence, either chromosome or contig -- this will be used as a `srcfeature` in `featureloc`

Gene descriptions:

```
symbol: xfile
synonyms: mulder, scully
description: A test gene for GMOD meeting
mRNA Feature
     exon_1:
       start: 13691
       end: 13767
       strand: 1
       srcFeature_id: Id of genomic sample
     exon_2:
       start: 14687
       end: 14720
       strand: 1
       srcFeature_id: Id of genomic sample

symbol: x-men
synonyms: wolverine
  mRNA Feature
```

```
     exon_1:
       start: 12648
       end: 13136
       strand: 1
       srcFeature_id: Id of genomic sample

symbol: x-ray
synonyms: none
     exon:
       start: 1703
       end: 1900
       strand: 1
       srcFeature_id: Id of genomic sample
```

Problem 2

Retrieve and print the following report for gene **xfile** (the coding sequence and exon coordinates are derived from the associated mRNA feature). The results should resemble the following:

```
symbol: xfile
synonyms: mulder, scully
description: A test gene for GMOD meeting
type: gene
exon1 start: 13691
exon1 end: 13767
exon2 start: 14687
exon2 end: 14720
>xfile cds
ATGGCGTTAGTATTCATGGTTACTGGTTTCGCTACTGATATCACCCAGCGTGTAGGCTGT
GGAATCGAACACTGGTATTGTATAAATGTTTGTGAATACACTGAGAAATAA
```

Problem 3

Update the gene **xfile**: change the name symbol to **x-file** and retrieve the changed record. Regenerate the report from Problem 1. The results should resemble the following:

```
symbol: x-file
synonyms: mulder, scully
description: A test gene for GMOD meeting
type: gene
exon1 start: 13691
exon1 end: 13767
exon2 start: 14687
exon2 end: 14720
>x-file cds
ATGGCGTTAGTATTCATGGTTACTGGTTTCGCTACTGATATCACCCAGCGTGTAGGCTGT
GGAATCGAACACTGGTATTGTATAAATGTTTGTGAATACACTGAGAAATAA
```

Problem 4

Search for all genes with symbols starting with *x-*. With the results produce the following simple result list (organism will vary):

```
1323    x-file     Xenopus laevis
1324    x-men   Xenopus laevis
```

```
1325    x-ray     Xenopus laevis
```

Problem 5

Delete the gene **x-ray** using the `geneId`. Run the search and report in Problem 4 again to show the delete has taken place, with a result resembling the following:

```
1323    x-file     Xenopus laevis
1324    x-men   Xenopus laevis
```

Problem Results

All presenters paid attention to the assigned problems and all packages could perform the required operations if appropriate: the GBrowse (DasI) Adaptor and InterMine, being read-only packages, did not carry out the edit/ delete tasks. Clearly one can see many differences between packages in how the the problems were solved, please see the presentations themselves for these details.

The Middleware Packages

The one day meeting heard presentations from developers using both Perl and Java middleware and a number of satisfactory solutions were described. The focus in almost all cases was some sort of system that connected to the Chado relational database. Although other databases are encountered in the GMOD world (e.g. BioSQL) the Chado schema is popular and serves as a good test schema for this exercise given its complexity. The primary focus in the talks was on functionality from the perspective of writing code and extending the software and less attention was given to performance. Each presenter focussed on their middleware and few side-by-side comparisons were made (for one comparison please see Comparison of XORT and Hibernate for Chado Reporting by Josh Goodman, written before this meeting).

The Chado::AutoDBI middleware package is built on Perl's Class:DBI, a module that acts as an ORM tool. The Chado::AutoDBI interface is autogenerated directly from the Chado schema using a template adapted from the Turnkey project. This greatly reduces the amount of time developers need to spend maintaining the API since the code can just be regenerated when the schema changes. Modware is built on top of Chado::AutoDBI. Both Chado::AutoDBI and Modware are built specifically for the Chado schema and aren't general tools. However, the Turnkey project can be used to produce an AutoDBI equivalent for other database schemas.

A key difference is that Modware uses BioPerl as its programmatic interface whereas the Chado::AutoDBI API resembles that of Class::DBI. Furthermore Chado::AutoDBI wraps the entire Chado schema where Modware objects only address a subset of the Chado schema that maps to Bioperl objects. Neither of these packages require configuration, they are pre-configured for Chado and all one needs to do is provide connection information (server, username, password, etc). This schema, in theory, could be running on any popular RDBMS (PostgreSQL, MySQL, Oracle, etc.), this flexibility is built into Class:DBI.

The Java packages, Hibernate and iBatis, are far more general than the two packages discussed above and one could use them with any relational schema. One distinguishing feature is the languages that these packages use: Hibernate tends to present more Java to the programmer whereas iBatis presents both Java and XML. Both allow

you to address the schema with SQL should you choose to do so. Hibernate also introduces its own language, HQL, a Java-SQL hybrid. Both would need some configuration to connect to the Chado schema but this should not be considered a significant barrier. Both packages can be used with any popular RDBMS (Postgres, Mysql, Oracle, etc.).

XORT is not a typical ORM tool like these other packages but has been included here because of its utility in bulk operations. This capability is one that the ORM tools are thought not to do very well as the serial construction and destruction of objects is typically not fast, and constructing very large numbers of objects simultaneously consumes quite a bit of memory. This is not to say that one shouldn't use an ORM tool for bulk operations but that one should test the tool in question and not assume its performance is adequate to the given task. Such a test may show an entirely different approach, like that of XORT, is more appropriate.

InterMine is another Java-based generic ORM tool that differs from the others by being tuned for read-only performance. It has been designed to support large complex queries and bulk data production efficiently. Performance of live databases can be tuned through the generation of pre-computed tables: a query optimiser intercepts all queries and re-writes them to make use of the available pre-computed tables. Like the other ORM packages, maintenance is minimised through extensive automatic code generation - when the database object model changes recompilation updates Java classes, the relational schema, the web application and mappings between them. Currently InterMine systems require Postgres but could be extended to other platforms that support the 'Explain' used by the query optimiser. Apart from Java and pilot Perl APIs, a variant of OQL, IQL, is provided.

Java Middleware

The Java packages all used Java plus XML, to some degree. In addition iBatis exposes SQL to the developer and it was argued that this could be viewed either as an advantage (allows tuning of underlying SQL) or a disadvantage. Both Hibernate and iBatis are designed to operate with any relational schema, not just Chado.

Both Abator and iBatis share one limitation which is that one can not auto-generate higher level objects, such as Genes. In fairness one must add that this would not really be possible to do accurately for any system. There is not sufficient information present in the database schema to derive all of the relationships necessary. In both packages the auto-generation code only mimics the schema and it would have take additional work to get a package to automatically create and relate subsets of data based on "join tables" like feature_relationship property. This mostly has to do with the circular nature of the schema.

Hibernate

From the Hibernate Web site:

Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. Hibernate allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API. Unlike many other persistence solutions, Hibernate does not hide the power of SQL from you and guarantees that your investment in relational technology and knowledge is as valid as always.

Hibernate is thought to work best when used in conjunction with a schema that's been designed with objects in mind, an *object-oriented schema*.

Abstraction

Hibernate is the more abstracted of the 2 Java packages, it allows you to work with the relational database with the least exposure to SQL if you choose to do this. It is probably considered the more flexible of the 2 with respect to language since one can program in Java or HBL (Hibernate Query Language), a hybrid between SQL and Java.

Performance

No pairwise comparisons between Hibernate and iBatis were made.

Configuration

Hibernate has the ability to read a database schema and generate Java Code representing the database objects or tables. In this exercise the developer chose not to use this method because one must create a file telling the code auto-generation what to use for indexes, which fields should be complex objects, and so on. It was judged to be less work to just go ahead and make the code by hand.

For each object mapped to the database, one can create an XML file that tells Hibernate which database fields to map to specific object properties. Additionally, one can implement equals() and hashcode() functions for each object based on the table unique constraints. After doing that, Hibernate takes care of all the Create-Retrieve-Update-Delete (CRUD) and transaction functionality.

Documentation

Hibernate is a popular and well-supported tool with extensive documentation.

iBatis

From the iBatis Web site:

The Data Mapper framework (a.k.a. SQL Maps) will help to significantly reduce the amount of Java and .NET code that is normally neededto access a relational database. This framework maps classes to SQL statements using a very simple XML descriptor. Simplicity is the biggest advantage of iBATIS over other frameworks and object relational mapping tools. To use iBATIS you need only be familiar with your own application domain objects (basic JavaBeans or .NET classes), XML, and SQL. There is very little else to learn. There is no complex scheme required to join tables or execute complex queries. Using iBATIS you have the full power of real SQL at your fingertips. The iBATIS Data Mapper framework can map nearly any database to any object model and is very tolerant of legacy designs, or even bad designs. This is all achieved without special database tables, peer objects or code generation.

Abstraction

iBatis does not attempt to achieve abstraction in the way that other Java ORM tools do and assumes that viewing SQL is an advantage, not a disadvantage.

Performance

No pairwise comparisons made between iBatis and Hibernate.

Configuration

The following general steps are performed:

1. Create an XML file of tables the developer wants to create objects for
    - Including specifying the method for retrieving auto-generated sequence values from inserts
    - This file could als be easily auto-generated
2. Make some type adjustments for some variables
    - For example, iBatis mapped some things to Java strings that should have been Integers

Documentation

iBatis is a popular and well-supported tool with extensive documentation.

InterMine

From the InterMine Web site:

InterMine is an general-purpose object-oriented data warehouse system developed as part of the FlyMine: project and made available as stand-alone open-source software. It is able to create a query-optimised data warehouse with a powerful web interface for any data model.

InterMine.bio is an extension to InterMine that defines a biological data model (based on the Sequence Ontology) and is able to integrate data from many standard formats and databases used in biology. Instances of InterMine.bio are created by specifying the sources to load and configuring the particular organisms and data files required. A framework is provided for adding new sources to load custom data, and each new source can easily extend the data model.

Abstraction

An InterMine data model is defined at the object level and a database schema is automatically generated. The database schema is entirely hidden, queries are performed on the object model by a Java API, an OQL-like query language (IQL), or a pilot Perl API.

Performance

Following build, InterMine-based systems are read-only. In contrast to transactional systems this makes it easier to focus on query performance. In particular, InterMine makes it straightforward to manage controlled denormalisation of data to enhance query performance: a generic query optimiser intercepts queries and transparently re-writes them to make use of precomputed tables. New precomputed tables can be added at any time, allowing performance tuning of the live database. Performance is also enhanced through the use of a large object cache in the web application.

Configuration

An InterMine object model is defined as an XML file, Java business objects and a relational schema are automatically generated from it. The mapping between objects and the database is thus handled automatically with no additional configuration. Generation of appropriate indexes is also automatic. InterMine cannot access an existing database schema but could import data from one by defining an object model and importing the data.

Documentation

Documentation on InterMine's functionality and instructions for setting up a new instance are provided at http://trac.flymine.org

Perl Middleware

The Perl] approaches used only the Perl language (the Java packages all used Java plus XML, to some degree). The XORT application is not, strictly speaking, *middleware* but has proven to be very useful in bulk operations using the Chado schema and Chado XML though in principle it can be used with any relational schema.

Chado::AutoDBI

The Chado::AutoDBI tool is based on Class::DBI and maps objects directly to tables in the Chado schema. It provides a very easy perl interface for low-level access to the database. It is currently used by the GMODWeb project, as a bulk loader for the Chado database, and as the underlying ORM tool in Modware. Chado::AutoDBI is automatically generated from the Chado database schema which makes it very easy to update when changes are made. This code autogeneration process was adapted from the Turnkey project which is a generic ORM/website code generation tool.

Abstraction

Chado::AutoDBI objects map directly to the Chado tables so it could be said that Chado::AutoDBI is as abstract as Chado itself. Therefore one needs to become somewhat familiar with Chado itself in order to use Chado::AutoDBI.

Performance

No pairwise comparisons of performance were done using Perl middleware. All packages were deemed to give adequate performance when used to connect UIs to underlying databases. On the other hand the presenters were reluctant to recommend their packages for *bulk* operations.

Configuration

Chado::AutoDBI requires no configuration, it is designed to interact with the Chado schema out-of-the-box. If the schema changes then Chado::AutoDBI can be quickly autogenerated again to adjust to the change.

Documentation

For more information see the schema/chado/README.AutoDBI, the Wiki page, and the Chado::AutoDBI Presentation.

Modware

Abstraction

Modware has higher level of abstraction than that provided by Chado::AutoDBI. The critical point is that it can create BioPerl object representations of features directly from Chado, this could either be highly suitable or not at all appropriate for a given environment.

Performance

No pairwise comparisons of performance were done using Perl middleware. All packages were deemed to give adequate performance when used to connect UIs to underlying databases.

Configuration

Modware requires minimal configuration, it is designed to interact with the Chado schema out-of-the-box. If the schema changes then the underlying Chado::AutoDBI should be able to adjust to the change. Some convenience views are created when installing Modware (i.e. V_MRNA_FEATURES).

Documentation

BioPerl-style documentation at http://gmod-ware.sourceforge.net/doc/, written for POD for all methods.

GBrowse (DasI)

Abstraction

Since it implements the Bio::DasI interface, the abstraction is similar to what one would see with BioPerl/BioDas features.

Performance

Bio::DB::Das::Chado performs relatively well since it is designed to be a database adaptor to drive a Generic Genome Browser (GBrowse) instance. Though it does not perform as well as the Bio::DB::GFF and Bio::DB::SeqFeature adaptors, those are for databases designed specifically for quick retrieval of data for GBrowse, whereas Chado is designed as a complete data warehouse. The main way to get better performance for the Chado GBrowse adaptor is to implement materialized views of the GFF or SeqFeature schema inside Chado.

Configuration

This package needs no configuration, it is pre-configured for Chado.

Documentation

The DasI interface is well-documented, about a dozen methods and three classes, all documented.

The Bio::Chado API

The ParameciumDB group has created an API for Chado based on the Bio::EnsEMBL API. No one was able to give a presentation on this API, more information can be found here:

- http://paramecium.cgm.cnrs-gif.fr/chadoapi/

Getting More Information

The issues around using and developing middleware are of general interest in GMOD. If you have questions about middleware we suggest that you contact the GMOD Development list rather than contacting individual developers. You can sign up for the list here:

https://lists.sourceforge.net/lists/listinfo/gmod-devel

Object-Relational Mapping Principles

Presentation by Sohel Merchant

Sohel Merchant, Bioinformatics Software Engineer at dictyBase, Center for Genetic Medicine, Northwestern University, Chicago. This Wiki section is an edited version of Sohel's presentation.

Also see ORM (Wikipedia).

Outline

- The Problem
- Solutions
- ORM
- Perl ? Class::DBI
- Summary

The Problem

- Developers need to perform Create, Retrieve, Update, Delete (aka CRUD) operations on data inside an application.
- The real world objects represented using a programming language needs to be stored in databases
- Using relational databases to store object-oriented data leads to a semantic gap
- RDBMS have fixed types, but OO can have more complicated user defined types.

Solutions

- Data Access Object (DAO)
- Developer writes a class which contains one attribute for each field in the table
- Methods for CRUD typically contains JDBC/DBI code with the necessary SQL statements.
- Object Relational Mapping (ORM), WikiPedia:
    - ♦ ?ORM is a programming technique that links databases to object-oriented language concepts, creating (in effect) a virtual object database.?
- Developer needs to configure the ORM
- Less amount of manual coding
- CRUD methods are automatically generated by the ORM layer

ORM

ORM solutions

- Perl
    - ♦ Class::DBI
- Java
    - ♦ EJB
    - ♦ Hibernate
    - ♦ JDO
    - ♦ iBatis

Perl - Class::DBI

- Provides a simple interfaces for wrapping Perl classes around a database tables
- Tables are mapped directly to objects
- The table column name are mapped to the get/set methods
- Can be used with transactions

Class::DBI

Defining a class in Class::DBI to represent a table:

```
CVTERM
cvterm_id
cv_id
name
definition
dbxref_id
```

Corresponding code:

```
package Chado::Cvterm;
use base 'Chado::DBI';
Chado::Cvterm->set_up_table('Cvterm');
```

Class::DBI - CRUD

```
## Create
$term_dbobj = Chado::Cvterm->create({
                                name      => ?DUMMY TERM?,
                                cv_id     => 1,
                                dbxref_id => 125
                           });
## Retrieve
$term_dbobj = Chado::Cvterm->retrieve(2);

## Update
$term_dbobj->name( $term->name() );
$term_dbobj->definition( $term->definition );

## Delete
$term_dbobj->delete();
```

Java - Hibernate

- Hibernate maps Java Objects directly to database tables
- Scalable
- Works well for controlled Data model

Java - iBatis

- iBATIS maps Java Objects to the results of SQL Queries
- XML definitions for queries
- Queries and managing Maps
- Transactions
- Good fit for existing database schema

Summary

- ORM provides painless roundtrip of data between the application and database.
- Reduces the amount of SQL code and allows a programmatic style interface to the RDBMS
- Choice of ORM solution depends on the type of project
- Flybase examined iBatis and Hibernate, both use XML configuration files
    - ♦ Hibernate is better if you're building schema from scratch
    - ♦ Both auto-configure given a schema.
    - ♦ Both have strengths and weaknesses.
- Is Hibernate better when you're in the process of designing a schema?
    - ♦ Hibernate can assist you in making a *Hibernate-compatible* schema.

XORT

Background

- Source: http://sourceforge.net/project/showfiles.php?group_id=27707
- Language: Perl
- Authors: Pinglei Zhou
- Users: Flybase

- Support: Pinglei Zhou
- Third party code: Uses XML::Parser::PerlSAX, Unicode::String, XML::DOM, and DBI from Perl (CPAN)

Technical Overview

- Database connectivity:
- Transaction support:
- Code generation:

Special topics

Comparing Hibernate & XORT

Flybase tried Hibernate, but just creating simple print() statements in the course of doing bulk operations they encountered performance issues. Therer are many caching parameters available in Hibernate but the problem is that Chado is recursive or cyclical. XORT does some simple, and automatic, caching. With XORT you can handle recursive or cyclical operations more easily. In common operations such as merging genes Chado users will encounter this issue routinely.

Also see Comparison of XORT and Hibernate for Chado Reporting.

Limitations

- Database schemas need to follow certain rules
  - ♦ All must have internal int primary key
  - ♦ All must have unique key(s)
- It may take a long path to retrieve certain type of data
  - ♦ Example: gene->allele->genotype->phenotype via feature_relationship
- Structure not stored in memory, you flush out data as it goes

Presentation by Pinglei Zhou and Josh Goodman

XORT Presentation

Chado::AutoDBI

Background

- Source: http://sourceforge.net/projects/gmod/
- Language: Perl
- Authors: Allen Day, Scott Cain, Brian O'Connor, & others
- Users:
- Support:
- Third party code: Based on Class::DBI by Michael Schwern & Tony Bowden

Technical Overview

- Database connectivity:
- Transaction support:
- Code generation:

Special topics

- Demonstrations of what your software does well

Limitations

- Performance
  - ♦ Can one read thousands of objects into memory? You could do this but it's not suited to bulk operations
- Joins & complex queries

```
# Add the add_constructor for looking for name lengths

__PACKAGE__ ->add_constructor(long_names => qq{ length(name) > 15 });

# Custom SQL

__PACKAGE__->set_sql(xfiles => qq{
       SELECT FEATURE_ID
       FROM FEATURE
       WHERE NAME = 'xfiles' });
```

Presentation by Brian O'Connor

Chado::AutoDBI Presentation

Modware

Modware seeks to provide an object-oriented perl interface to Chado to allow for rapid application development without worrying about the complex details of the schema on a day-to-day basis.

Background

- Source: http://gmod-ware.sourceforge.net
- Language: Perl
- Authors: Sohel Merchant, Eric Just
- Contact: e-just [at] northwestern.edu
- Users: dictyBase
- Support: Fully supported by authors. Sourceforge infrastructure (bug reports, mailing lists)
- Third party code: GMOD, BioPerl

Technical Overview

- Database connectivity: Uses the Chado::AutoDBI connection from GMOD. No connection configuration necessary since Modware is built on top of GMOD and the coonnection is configured on GMOD install.
- Transaction support: Transactions are fully supported. The database handle is available as a singleton through Modware::DBH. To rollback at any time, simply insert
  ```
  new Modware::DBH->rollback()
  ```
  into script.
- Code generation: No automatic code generation.

Special topics

- Features in Modware map to familiar Bioperl (Bio::Seq and Bio::SeqFeature) objects that can be accessed through a 'bioperl' method.
- Modware internally links Bio::SeqFeature objects to the proper Bio::Seq objects to provide maximum functionality from the BioPerl libraries.
- Modware's API documentation is complete and easily browsed at http://gmod-ware.sourceforge.net/doc/
- Plenty of other documentation (quick starts with simple use cases) at http://gmod-ware.sourceforge.net

Limitations

- Currently Alpha release (hoping for user feedback to create Beta)
- Does not *cover* all of Chado
- Not enough users to get quality feedback yet
- Performance is slower by object-oriented nature
- Language dependent

Presentation by Eric Just

Modware Presentation

GBrowse (DasI) Adaptor

Background

- Source: http://sourceforge.net/project/showfiles.php?group_id=27707&package_id=34513&release_id=433523
- Language: Perl
- Authors: Scott Cain
- Users: Several through GBrowse, none that I know of as a scripting tool.
- Support: GBrowse mailing list
- Third party code: None.

Technical Overview

- Database connectivity: Connects *via* Perl's DBI
- Transaction support: N/A (read only adapter)
- Code generation: N/A

Special topics

Clearly, Bio::DB::Das::Chado is designed for use with GBrowse.

Limitations

- Read-only
- Not generic middleware but if you use Chado and GBrowse may be useful
- Incomplete implementation of Bio::DasI; just enough to make GBrowse work
- Also, despite the name, has never been tested with a Das server.

Presentation by Scott Cain

GBrowse (DasI) Adaptor Presentation

iBatis and Abator

Background

- Source: http://ibatis.apache.org/
- Language: Java
- Authors: Apache group
- Users:
- Support:
- Third party code:

Technical Overview

- Database connectivity:
- Transaction support:
- Code generation:

Special topics

- Demonstrations of what your software does well

Limitations

- Does not hide SQL
- Does not create a whole object model of the database in memory
- Not as widely used as Hibernate
- No Perl version

Presentation by Jeff Bowes

iBatis Presentation

Hibernate

Background

- Source: http://www.hibernate.org
- Language: Java
- Authors: JBoss group
- Users: VectorBase
- Support: JBoss group
- Third party code:

Technical Overview

- Database connectivity:
- Transaction support:
- Code generation:

Limitations

- Issue around Completeness
- Exception Handling
- Performance Tuning

Presentation by Robert Bruggner

Hibernate Presentation

PSU Chado Interface

I think this eventually became what is described in the Artemis-Chado Integration Tutorial. Dave C.

Background

- Source: Pathogen Sequencing Unit, Sanger Institute
- Language: Java
- Authors: Chinmay Patel, Adrian Tivey, Tim Carver
- Users: In the future this will be freely available and used by GeneDB and Artemis
- Support: In development
- Third party code: iBatis and Spring/Hibernate

Overview

We're using a common interface with two different implementations: an iBatis and a Hibernate one. This gives us the ability to choose either implementation depending upon the requirements of the application.

iBatis

See also iBatis and Abator

Technical Overview

- Database connectivity:

Dynamically via Java code or properties file

- Transaction support:

Yes

- Code generation:

Using common interface which was originally automatically generated, but mappings hand-generated

Advantages

- Direct access to SQL

Limitations

- Not completely pluggable between both engines
- Lazy loading is either on or off

Hibernate

See also Hibernate

Technical Overview

- Database connectivity:

Via usual Spring configuration methods

- Transaction support:

Yes

- Code generation:

Interface and Hibernate implementation originally auto-generated, then hand-edited

Advantages

- Complete coverage of core schema (except Phenotype module)
- Choice of writing in either object-level query language or SQL

Limitations

- Not completely pluggable between both engines
- Not currently using subclassing eg just a Feature, not Gene, Exon etc

Presentation by Chinmay Patel

PSU Presentation

GUS Web Development Kit (WDK)

Background

- Source: http://www.gusdb.org/WDK
- Language: XML/Java/JSP
- Authors: GUS WDK development team
- Users: PlasmoDB/ApiDB/CryptoDB/ToxoDB, others under development
- Support: GUS WDK development team
- Third party code: Struts/JSP/Ajax/Axis
- Platform requirements: any Oracle, PostgreSQL or MySQL database; Linux; Tomcat

Technical Overview

- Database connectivity: JDBC
- Transaction support: read-only presentation layer object model
- Code generation: java classes generated from detailed object specification in XML

Advantages

- Configurable coarse grained layer tailored for the presentation layer
- Configurable in XML by non-programmers
- Seemless integration with front-end query engine and web page generator

Limitations

- read-only
- not designed as a general purpose ORT
- configuration is complicated

Presentation by Steve Fischer

GUS WDK Presentation

InterMine

Background

- Source: http://www.intermine.org
- Language: Java
- Authors: FlyMine/InterMine development team
- Users: FlyMine, StemCellMine, others under development
- Support: FlyMine/InterMine development team
- Third party code: Struts/JSP/Ajax

Technical Overview

- Database connectivity: JDBC
- Transaction support: basic during build (doesn't support concurrent writes) as database is optimised for high read-only query performance
- Code generation: extensive use of automatic code generation

Limitations

- slow, but improving, build times (loads and integrates ~25m objects in ~36 hours)
- configuration complicated but being simplfied
- deals well with overlapping features, but currently limited support for querying locations in DNA or protein sequences
- export still limited to tab or comma-delimited or FASTA formats though great flexibility in choice of output columns and their order
- sequences not handled very well
    - ♦ e.g. each chromosome sequence is stored in one big text field in PostgreSQL
- can't yet do queries involving sizes of collections of things
    - ♦ e.g. find the genes with only 1 transcript
- doesn't yet support left outer join behaviour (under development)

Presentation by Gos Micklem

InterMine Presentation

Wiki Authors

- Jeff Bowes, XenBase
- Robert Bruggner, VectorBase

- Scott Cain, GMOD
- Steve Fischer, PlasmoDB/GUS
- Josh Goodman, FlyBase
- Eric Just, DictyBase
- Sohel Merchant, DictyBase
- Brian O'Connor, UCLA
- Brian Osborne, GMOD
- Chinmay Patel, GeneDB
- Pinglei Zhou, FlyBase
- Gos Micklem, FlyMine/InterMine