The Bioverse: An object-oriented genomic database and webserver written in Python

Jason McDermott and Ram Samudrala

Department of Microbiology, University of Washington

mcdermottj@compbio.washington.edu
ram@compbio.washington.edu

The recent success of numerous genomic sequencing efforts have created a great demand for systems to organize, represent and interpret the huge amount of data available. The Bioverse is a database designed to provide a framework for exploring the relationships among the molecular and genomic, proteomic, systems, and organismal worlds. We implemented the Bioverse in the open-source scripting language Python [1] using an object-oriented programming model. The result is a highly flexible framework for the processing and representation of bioinformatics information. We have implemented parallel processing and data schemes for this framework so that the work and data storage can be distributed over our 200+ CPU LINUX cluster. The objects we have created in this framework are reusable, easily extensible and can be configured using control files with a simple syntax. To date, we have applied our framework to 11 genomes including the *Oryza sativa* (rice) [2] and *Homo sapiens* [3] genomes. The current version of the Bioverse (1.0) contains an unsurpassed level of annotation information for over 150,000 unique sequences, including secondary structure predictions, family and domain classifications, homology and protein-protein interaction predictions. A large number of unannotated sequences from the rice genome were assigned annotations by the Bioverse. And on the whole genome scale, networks based on protein-protein interactions and evolutionary relationships have been derived. The webserver represents the user-interface to the database and is available at http://bioverse.compbio.washington.edu.

Overview

The state of post-sequencing genomics mandates a flexible approach to any attempt at representation and organization of the information derived from protein sequences. The vast number of databases, search techniques and prediction methods that are currently available and those that will be developed may need to be integrated into our database. To this end we chose an object-oriented approach for our framework. A generalized view of the dataflow from this framework is shown in Figure 1.

We chose to implement our framework in the open-source scripting language Python. Python has the functionality of low-level compiled languages like C as well as higher level features, such as built in support for complex data types. Importantly, Python is very object-oriented, providing clear and unambiguous class creation, subclassing, multiple inheritance and automatic documentation and is supported on nearly all platforms. As a scripting language, the speed of Python was a concern. However, the most computationally intensive operations being performed are all stand-alone programs (e.g. BLAST [4], HMMPfam [5]) and are accessed through Python's UNIX process

control library. In addition, Python has a simple and easy to use API for wrapping existing C programs or writing computationally intensive subroutines.

Database Structure

Each genome is represented by a Database object which stores all global information pertaining to the genome. The database creates Record objects, each of which represent a unique sequence in the genome and its associated data. Atom objects store the smallest units of bioinformatic data (single BLAST results, secondary structure predictions, etc.) pertaining to a particular record. The modularity of this approach makes the Record easily able to accommodate new types of information which may be added to the framework.

Since all low-level I/O is handled through the Record class, the format of low-level data can be easily changed. We have developed a preliminary implementation of an SQL-format database and an XML schema for storing information in a less flexible but more standardized manner.

Parallel Framework

The computationally intensive nature of many of the methods used to generate data for the database encouraged us to design our framework with parallel processing in mind. A BLAST search on a single protein sequence can take several minutes of CPU time and automated structural prediction by gene threading may take many hours or more, so applying these methods to tens or hundreds of thousands of protein sequences is a non-trivial task. Distribution of processing on our 200+ CPU LINUX cluster is accomplished through a program which reads a database, breaks it into blocks of records of a specified size, spawns remote processes on the cluster that act on the record blocks and then monitors the processes for completion. This approach, though simpler than many other parallel processing approaches, fits well with our data structure, is very flexible and has a very low computational overhead.

A potential pitfall of our parellel processing approach is bottlenecks caused by network I/O from many CPU's at once. To alleviate this problem we distribute the data in our databases over multiple filesystems so that requests to the database are handled by multiple machines. A larger concern than the data from individual records is the data required for the method being applied to the sequences. The NCBI BLAST database [4], for instance, is approximately 200 Mb in size. In a centralized distribution approach, files of this size would have to be transferred to every client filesystem from one master CPU before any processing could proceed. Our solution to this problem is to have each remote process check for a local copy of a library of large data files. If there is no local copy the remote process requests one from one of the multiple server systems and makes a local copy. Version checking has been implemented to ensure that older data is not mixed with newer if any of the data sources is updated. There is a large advantage to this decentralized data distribution approach over a centralized one. For instance, running a BLAST search on a typical bacterial genome (~5000 sequences) using a centralized data distribution scheme and 50 CPUs takes over 30 hours, using our data distribution framework this time is cut to approximately 2 hours. A eukaryotic genome, such as the rice genome with nearly 60,000 sequences, takes approximately 10 days of computation time on 60 CPU's to generate all the data currently included in the Bioverse (running

approximately 15 bioinformatics programs, parsing the results, and generating global hash tables and network views).

Data Generation

We expected the nature of our database and the nature of the bioinformatics methods required to evolve over time. To this end, we designed a highly flexible processing environment to perform the back-end generation and processing of Record-related data. Action objects perform operations in the Database and can be specified through a formatted definition file. Since each Action object defines its user-accessible parameters and calling keyword this approach defines a flexible programming language for database operations.

The Bioverse Server

An important measure of the success of an integrated bioinformatics database like the Bioverse is how useful it is to the scientific community in general. One part of this is the quality of the data in the database, the other, equally important part is how well this information is presented. We designed the Bioverse webserver with certain goals in mind. From our perspective the server must be flexible to allow incorporation of new bioinformatics data as it is added to the Bioverse and to allow easy modification of how this data is presented. From the users' perspective the server must be fast, robust and easy to use and must allow for customization of what data is displayed and how it is displayed.

The heart of our server is the ElementServer class, a server class based on Python's BaseHTTPServer, a simple yet robust multi-threading http server class. Similar to the Actions framework described above, the server is comprised of a hierarchy of Element objects which processes a user's request and returns an HTML page based on the request. An Element object has methods for initialization of the object and any dependent Element objects, for processing incoming server requests passed from its parental element, and for describing how the element's data is passed back to its parent in the form of HTML.

In the Bioverse server, the presentation of global database information is established when the server is started. Element hierarchies pertaining to individual genomic records are created dynamically when requested and are kept in a cache to allow quick access to recently requested records. At the lowest level of these hierarchies are HTMAtom elements, objects based on corresponding database Atom classes but which have methods specifying how their data should be presented in HTML. A hierarchy defining the meta-groups of sequence, structure and function are built on top of the HTMAtom objects (Figure 2). Display of these meta-groups and groups below them in the hierarchy are controlled by the user through their interaction with icons on the record display HTML page (Figure 3).

Conclusions

We have implemented a fully object-oriented framework for processing, integration and presentation of a large number of different bioinformatics approaches and have applied this approach to the *Oryza sativa* genome as well as 10 other eukaryotic and prokaryotic genomes. Our framework is built of reusable components which can be easily reconfigured or extended as need demands. The feedback cycle of development,

implementation, and evaluation dictating further development is thus expedited. Suggestions from the scientific community can easily be incorporated into our structure and new bioinformatics methods can be added with a minimum of coding. Because of the object-oriented nature of our framework we are able to easily transfer it to a standard object-oriented database management system (OODBMS) [6]. An advantage of storing our data in an OODBMS is that it can improve performance for complex data such as complicated graphs, by 10-1000% compared to standard relational databases [7, 8].The current version of the Bioverse (version 1.0) has parallel, distributed data generation and storage, incorporates data from approximately 15 external data sources and 7 prediction methods, and provides a powerful interface to access the information generated by these methods.

Acknowledgements
This work was supported in part by a Searle Scholar Award to Ram Samudrala and NSF Grant DBI-0217241.

References
[1] Python home page <http://www.python.org>.
[2] Yu J., Hu S., Wang J., Wong G., et. al. 2002. A draft sequence of the rice genome (Oryza sativa L. ssp. indica.). Science 296:79-92.
[3] Lander E.S., Linton L.M., Birren B., Nusbaum C., et. al. 2001. Initial sequencing and analysis of the human genome. Nature 409: 860-921.
[4] Altschul SF, Madden TL, Schaeffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ. 1997. Gapped BLAST and PSI-BLAST: A new generation of database programs. *Nucleic Acids Research* 25: 3389-3402 (1997)
[5] Eddy SR. 1998. Profile hidden Markov models. *Bioinformatics* 14: 755-763.
[6] PostgreSQL <http://www.postgresql.org>.
[7] Harrington J. 1999. *Object-oriented Database Design Clearly Explained.* organ Kaufmann Publishers, 1st edition.
[8] Object-oriented Databases <http://www.odbmsfacts.com>.

Figure 1.
Bioverse Dataflow Diagram. Generalized information flow in the Bioverse database.

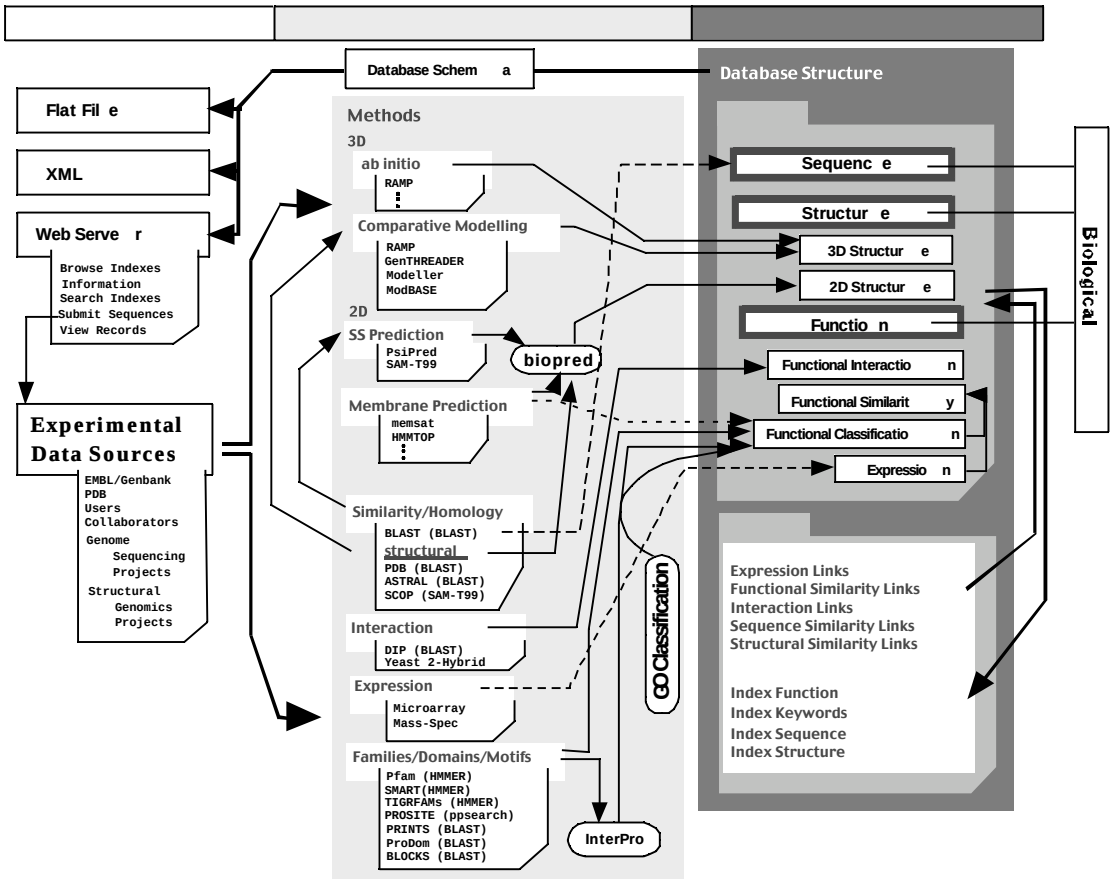

Figure 2.
Database Schema. A simplified version of the Bioverse database object-oriented schema. The schema will be extended as more data sources and methods are integrated into the database.

Figure 3.
Bioverse Screen Shot. An example of a Bioverse session showing the HTML display of a Record object. All colored icons to the left of the data field can be used to open or close subsections of the Record heirarchy. The current Record is shown with the Information section open under the Sequence heading (displaying identifiers from other databases for this sequence), data contributing to the two-dimensional structure prediction in the Structure heading, and evidence for the Usp Interpro function (HMMPfam match) under the Function heading.