The Bioperl Toolkit: Perl Modules for the Life Sciences

Jason E. Stajich, David Block, Kris Boulez, et al.

Genome Res. 2002 12: 1611-1618
Access the most recent version at doi:10.1101/gr.361602

The Bioperl Toolkit: Perl Modules for the Life Sciences

Jason E. Stajich,[1,18,19] David Block,[2,18] Kris Boulez,[3] Steven E. Brenner,[4] Stephen A. Chervitz,[5] Chris Dagdigian,[6] Georg Fuellen,[7] James G.R. Gilbert,[8] Ian Korf,[9] Hilmar Lapp,[10] Heikki Lehväslaiho,[11] Chad Matsalla,[12] Chris J. Mungall,[13] Brian I. Osborne,[14] Matthew R. Pocock,[8] Peter Schattner,[15] Martin Senger,[11] Lincoln D. Stein,[16] Elia Stupka,[17] Mark D. Wilkinson,[2] and Ewan Birney[11]

[1]University Program in Genetics, Duke University, Durham, North Carolina 27710, USA; [2]National Research Council of Canada, Plant Biotechnology Institute, Saskatoon, SK S7N OW9 Canada; [3]AlgoNomics, B 9052 Gent, Belgium; [4]Department of Plant and Molecular Biology, University of California, Berkeley, California 94720, USA; [5]Affymetrix, Inc., Emeryville, California 94608, USA; [6]Open Bioinformatics Foundation, Somerville, Massachusetts 02144, USA; [7]Integrated Functional Genomics, IZKF, University Hospital Muenster, 48149 Muenster, Germany; [8]The Welcome Trust Sanger Institute, Welcome Trust Genome Campus, Hinxton, Cambridge, CB10 1SA UK; [9]Department of Computer Science, Washington University, St. Louis, Missouri 63130, USA; [10]Genomics Institute of the Novartis Research Foundation (GNF), San Diego, California 92121, USA; [11]European Bioinformatics Institute, Welcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD UK; [12]Agriculture and Agri-Food Canada, Saskatoon Research Centre, Saskatoon SK, S7N 0X2 Canada; [13]Berkeley Drosophila Genome Project, University of California, Berkeley, California 94720, USA; [14]Cogina, New York City, New York 10022, USA; [15]Center for Biomolecular Science and Engineering, University of California, Santa Cruz, California 95064, USA; [16]Cold Spring Harbor Laboratory, Cold Spring Harbor, New York 11724, USA; [17]Institute of Molecular and Cell Biology, 117609 Singapore

The Bioperl project is an international open-source collaboration of biologists, bioinformaticians, and computer scientists that has evolved over the past 7 yr into the most comprehensive library of Perl modules available for managing and manipulating life-science information. Bioperl provides an easy-to-use, stable, and consistent programming interface for bioinformatics application programmers. The Bioperl modules have been successfully and repeatedly used to reduce otherwise complex tasks to only a few lines of code. The Bioperl object model has been proven to be flexible enough to support enterprise-level applications such as EnsEMBL, while maintaining an easy learning curve for novice Perl programmers. Bioperl is capable of executing analyses and processing results from programs such as BLAST, ClustalW, or the EMBOSS suite. Interoperation with modules written in Python and Java is supported through the evolving BioCORBA bridge. Bioperl provides access to data stores such as GenBank and SwissProt via a flexible series of sequence input/output modules, and to the emerging common sequence data storage format of the Open Bioinformatics Database Access project. This study describes the overall architecture of the toolkit, the problem domains that it addresses, and gives specific examples of how the toolkit can be used to solve common life-sciences problems. We conclude with a discussion of how the open-source nature of the project has contributed to the development effort.

[Supplemental material is available online at www.genome.org. Bioperl is available as open-source software free of charge and is licensed under the Perl Artistic License (http://www.perl.com/pub/a/language/misc/Artistic.html). It is available for download at http://www.bioperl.org. Support inquiries should be addressed to bioperl-l@bioperl.org.]

Computational analysis is an integral part of modern biological research. Numerous computer software tools exist to perform data analyses, but it is not simple to automatically combine data and results from multiple sources without the use of computer software designed to read and write data specific to the biological domain. The day-to-day work in a typical bioinformatics laboratory consists largely of writing program logic to achieve this data integration.

Perl is one of the most widely used programming languages for these tasks and is commonly thought of as the language most easily grasped by newcomers to the field. Perl has been extremely successful for connecting software applications together into sequence analysis pipelines, converting file formats, and extracting information from the output of analysis programs and other text files.

Much of the Perl software in bioinformatics is specific to

[18]Present address: Genomics Institute of the Novartis Research Foundation (GNF), San Diego, California 92121, USA.
[19]Corresponding author.
E-MAIL jason.stajich@duke.edu; FAX (919) 681-1035.
Article and publication are at http://www.genome.org/cgi/doi/10.1101/gr.361602.

a particular laboratory or institution and is written for immediate utility rather than reusability. This results in considerable inefficiency, as the same software is rewritten multiple times. The Bioperl toolkit brings together reusable Perl modules containing generalized routines specific to life-science information. A primary motivation behind writing the toolkit is the authors' desire to focus energies on a solution whose components can be shared rather than duplicating effort. In our minds, once a routine is written for parsing and interpreting sequence from EMBL and GenBank format sequence files, no one should ever have to write this routine again. In this spirit, we chose to make our code freely available under an open-source license (Open Source Initiative 2001), so that others could extend routines already in the Bioperl library and contribute their own routines as well. Just as the Human Genome Project was facilitated by public sharing of data, so has the open nature of the Bioperl project reduced the time for solutions and new tools to reach the community (Waterston et al. 2002).

To be adopted by the community, our software has to be user friendly. To that end, Bioperl provides extensive documentation of all of the routines in each module, a graphical diagram of the objects in the toolkit, and a set of tutorials that lead the reader through the solutions to common tasks. Additionally, we have created a simplified interface to Bioperl that provides entry-level access to the toolkit. The goal of Bioperl is to help a user focus on her specific problem at hand, such as the logic needed to filter hits in a BLAST (Altschul et al. 1997) report by certain criteria, rather than on the actual mechanics of parsing that BLAST report.

Software Development Methodology

The Bioperl project began in 1995 (Chervitz et al. 1998) at a time when there were few programming toolkits for manipulating biological data or results from sequence analysis programs. Although Perl had already gained widespread popularity in the bioinformatics community for its efficient support of text processing and pattern matching tasks, there were, in fact, no biological toolkits available in this language.

The project grew out of the following observations. First, even though file formats of different analysis programs are different, the information they represent is the same. For example, a pair-wise alignment is always between two sequences and has common properties such as length, score, fraction of identities, start and end of the aligned sequences, and so forth. Second, the number of data structures needed to represent information flow is limited, and common to most applications such as sequences, annotation, features, and alignments. This permits a small set of modules to be reused for a variety of purposes. Third, a set of operations is commonly performed on these data structures. These include reading and writing information to a file, querying a sequence for its features, and translating a coding sequence into protein.

This scenario naturally lends itself to the principle of object-oriented programming, which Perl emulates with modules.

Object-oriented programming is the practice of grouping related tasks together into logical and broadly applicable components. For example, a DNA sequence component could contain methods to retrieve the sequence's accession number, reverse complement the DNA, or translate it into a protein sequence. Object-oriented programming methodology allows accurate modeling of the problem domain, leading to more robust, valid, and reusable code. Instead of attacking a problem by brute force, the problem is analyzed and models are constructed to represent the entities in the problem and solution domains. The entities are represented in code through the use of Perl modules and, where appropriate, form elements in an inheritance hierarchy. The use of inheritance and polymorphism in Bioperl implements the well-established principles of information hiding, modularity, and module cohesion (Tremblay and Cheston 2001).

We designed Bioperl using object-oriented methodology so as to create clean, generic, and reusable modules to represent data structures and operations common to the life sciences. By separating the components into logical groups such as sequences, alignments, and databases, we have been able to add features to a specific module without necessarily affecting the rest of the toolkit library. This separation is a key aspect of object-oriented programming and permits us to produce generic components with a stable interface for the programmer (the so-called API).

At present, the components and operations in Bioperl center around biological sequence analysis and annotation. In the last year, the project has expanded to address new areas including phylogenetics, maps, protein structure, and bibliographic references. The project has >300 modules and comprises >160,000 lines of code and embedded documentation. The Perl modules, illustrated in Table 1, are organized by logical names so that, for example, the Bio::Search hierarchy contains modules related to database searches, and Bio::Graphics contains modules that are related to drawing (Fig. 1). The Bio::Perl module itself is a simplified API that provides access to the most-commonly used Bioperl functions.

When designing Bioperl objects, we sought to provide a pro-

Table 1. Major Bioperl Module Groups

Modules	Description
Bio::Seq	Sequences and their properties
Bio::SeqIO	Sequence data input/output
Bio::Index	Flat-file sequence database indexing and retrieval
Bio::DB	Remote database access for sequences and references via HTTP
Bio::DB::GFF	SQL GFF database for DAS and GBrowse backends
Bio::SeqFeature	Annotations or features that have a sequence location
Bio::Annotation	Generic annotations such as Comments and References
Bio::AlignIO, Bio::SimpleAlign	Multiple sequence alignments and their Input/Output
Bio::LiveSeq, Bio::Variation	Sequence variations and mutations
Bio::Search, Bio::SearchIO	Sequence database searches and their Input/Output
Bio::Tools	Miscellaneous analysis tools
Bio::Tools::Run	Wrapper for executing local and remote analyses
Bio::Tree, Bio::TreeIO	Phylogenetic trees and their Input/Output
Bio::Structure	Protein structure data
Bio::Map, Bio::MapIO	Biological maps and their Input/Output
Bio::Biblio, Bio::DB::Biblio	Bibliographic References and Database retrieval
Bio::Graphics	Graphical displays of sequences

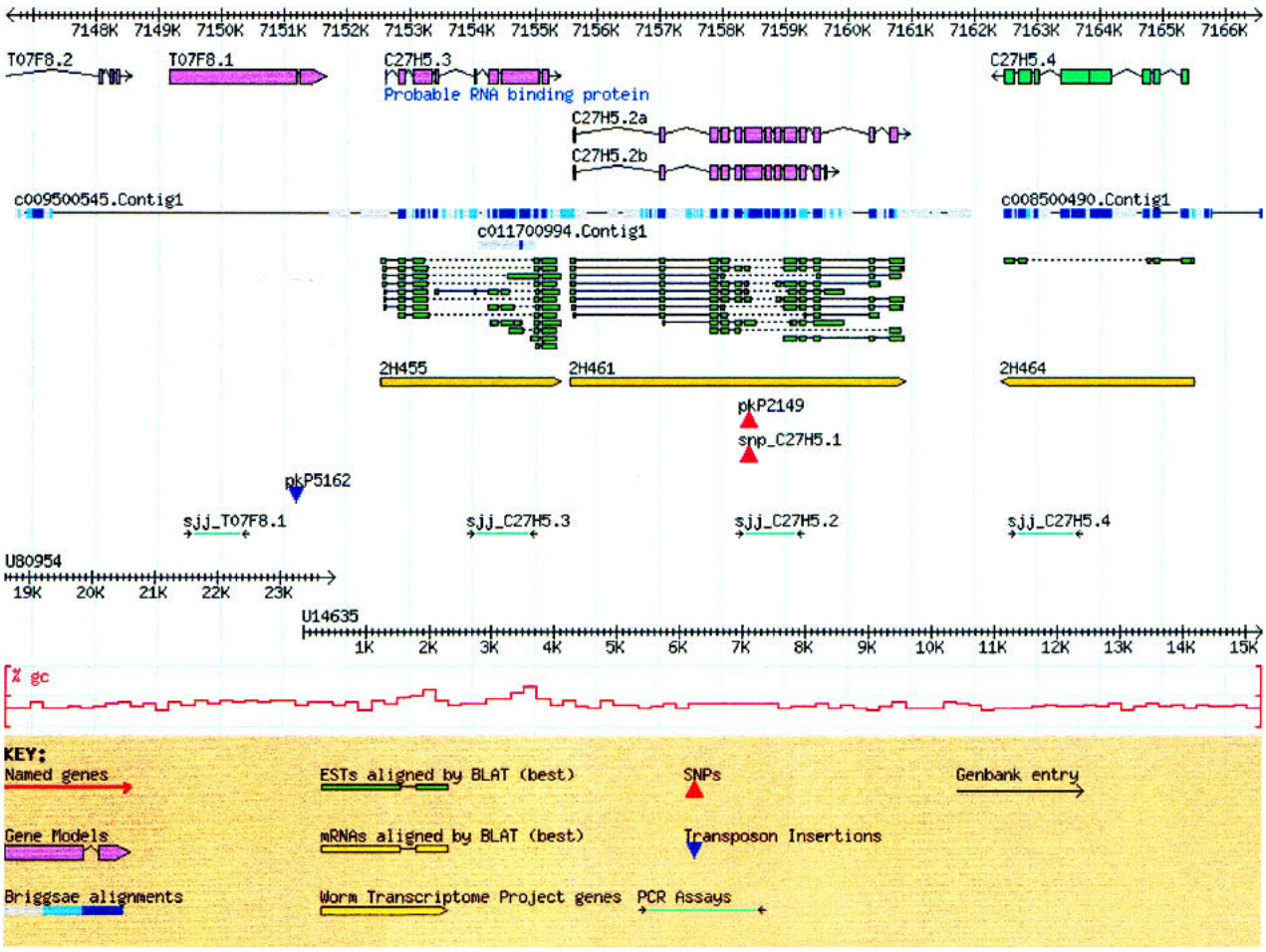

Figure 1 Rendering a sequence graphically with Bio::Graphics. This image represents a 20-Kb segment of the *C. elegans* genome containing annotated genes, a cross-species alignment (*C. elegans* to *C. briggsae*), EST alignments, SNPs, PCR primer pairs, and a GC content histogram. The module's flexible glyph-based architecture allows the application programmer to adjust precisely how to display biological objects. Glyphs allow the programmer to define different symbols for different data types or data sources and each are drawn as a separate track in the image. The module is also suitable for illustrating the extent of protein domains, physical (clone) maps, and horizontal maps.

gramming interface that is very easy to use, but at the same time, could be easily extended in its capabilities and behavior through code reuse. Using an object-oriented paradigm, we followed certain design principles.

First, separate the interface from the implementation. The key information about a component is the method names and their list of accepted arguments. Similar in concept to interfaces in Java, we built interfaces as collections of methods that describe the expected behavior of a module, but do not do any of the work. Child modules implement the interfaces, providing specializations of their parents to perform specific tasks. To help distinguish implementation modules from interface definitions, we used a capital I appended to the object name. This principle is based on the well-established methodology of specifying a given module's Abstract Data Type by defining how a given module will behave without specifying the mechanism by which it achieves this end.

For example, Bio::SeqFeatureI describes the contract for all modules that are features on sequences. This includes methods for start, end, strand, and access to comments and other attributes via tag/value pairs. All modules in the Bio::SeqFeature hierarchy implement this interface (Fig. 2). The power of this design is that operations that expect a Bio::SeqFeatureI, such as operations in the Bio::Graphics modules, can operate on anything that implements the Bio::SeqFeatureI interface. In this manner, sequence annotation that is retrieved from a DAS server (Dowell et al. 2001), a local file, or a database server can all be drawn as an image with the same methods in the Bio::Graphics modules.

Second, generalize common routines into a single module, providing a base framework for the respective operations. As an example, we centralized the basic input/output (IO) operations into an IO object, called Bio::Root::IO. Because all modules that need IO data access use operations from the IO module, these operations are implemented across the entire package in a consistent way. This design choice also provides a single location for applying improvements to the shared methods.

Third, use the Factory and Strategy patterns (Gamma et al. 1994) as much as possible. A Strategy pattern defines one or more operations that a particular implementation must support. For example, Bio::SeqIO uses a Strategy pattern by specifying that Bio::SeqIO modules must support the operation next_seq( ). Various parsers implement their own parsing

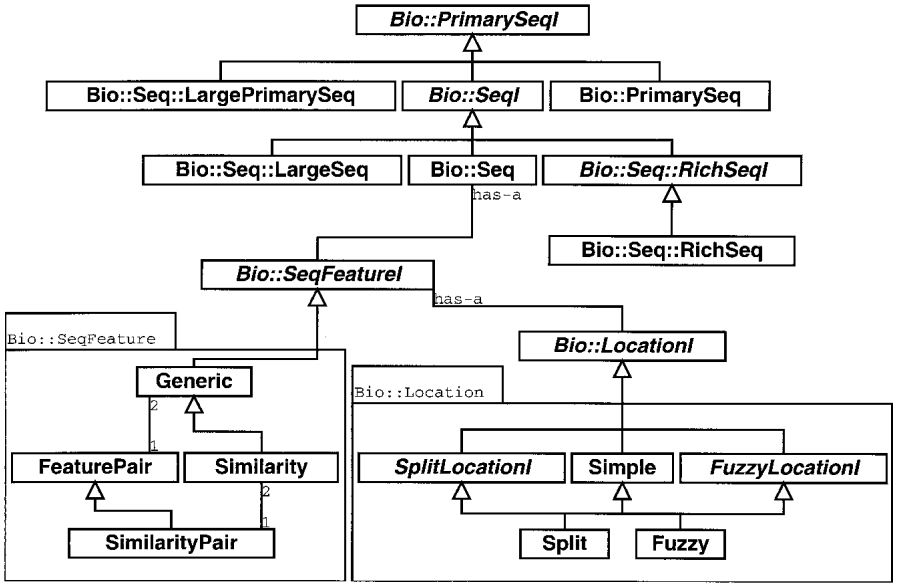

Figure 2 This figure shows a portion of the Bioperl object model including the interfaces (shown in italicized type) for sequences (PrimarySeqI, SeqI, RichSeqI) and their implementations PrimarySeq (general sequence), Seq (sequence with features), RichSeq (sequence with features and rich annotation), LargePrimarySeq (for sequences too large to be held in a program's memory), and LargeSeq (large sequences with features). Also included in the diagram is the sequence feature interface (SeqFeatureI) and its implementations Similarity (manage similarity information), FeaturePair (paired feature information), and SimilarityPair (paired similarity information such as a pair-wise alignment information). Additionally, the diagram shows the location objects that manage Simple (start, end, and strand information), Split (multiple start and end spots on a sequence such as a set of exons), and so-called Fuzzy locations (where start, end or span is not exact) for sequence features.

algorithms, but each parser has a next_seq( ) method that provides consistency and eases usage. Example usage of the Bio::SeqIO module can be seen in Figure 3.

The Factory pattern is a design pattern used when a module must create an object but cannot know what specific subtype must be created. It can serve as an aggregator for a set of modules that implement the same Strategy. For example, the Bio::SeqIO module acts as a Factory that produces specific sequence file parsers for different sequence formats. When the user asks Bio::SeqIO to parse a particular sequence file, Bio::SeqIO determines the correct parser to instantiate and invoke, thereby shielding application code from the technical details of the sequence file format and the Bio::SeqIO object hierarchy. Application code can be written generally to handle sequence input without worrying about whether it is processing a sequence file in GenBank, EMBL, SwissProt, or BSML format, or whether the sequence data is local, or being retrieved dynamically from a remote database via the Internet.

```
use Bio::DB::EMBL;
use Bio::SeqIO;

my $db = new Bio::DB::EMBL();
my $seq = $db->get_Seq_by_acc("U14680");
my $seqout = new Bio::SeqIO( -format => "genbank");
if (defined $seq) {
   $seqout->write_seq($seq);
}
```

Figure 3 Retrieving a sequence from a remote database with Bio::DB::EMBL. This code retrieves an mRNA sequence in EMBL format from the EBI EMBL databank with the accession no. U14680 and writes the sequence out in GenBank format to the terminal. One could replace Bio::DB::EMBL with Bio::DB::GenBank and instead retrieve the sequence from NCBI just as easily, as the software can read and write both EMBL and GenBank formats and is able to connect to both services through the World Wide Web. The retrieved sequence can then be passed to Bio::Graphics for graphical rendering, to the Bio::SeqIO interface for writing to a file, or to the ODBA interfaces for storage in a relational database.

Bioperl is written purely in Perl and requires at least version 5.005 of the Perl interpreter (the current stable version of Perl as of the time of writing is 5.8.0). The toolkit has been validated for cross-platform compatibility on most UNIX and UNIX-like operating systems. In addition, Bioperl has been tested and runs on Macintosh OS X and Microsoft Windows operating systems.

Because the Bioperl toolkit depends on the Perl interpreter, there are a number of rare cases in which its behavior is not consistent across different versions of Perl or between versions of Perl on certain operating systems. Descriptions of these version-specific problems and their solutions are available from the Bioperl Web site.

In addition to pure Perl solutions to bioinformatics problems, Bioperl can take advantage of external data analysis packages. Bioperl is capable of parsing the output from a variety of programs including BLAST [both NCBI and WUBLAST (Gish 2002) versions], HMMer (Eddy 2001), ClustalW (Thompson et al. 1994), T-Coffee (Notredame et al. 2000), Phylip (Felsenstein 1983), many EMBOSS (Rice et al. 2000) programs, Genscan (Burge and Karlin 1997), and 18 others. In addition, it can launch remote analyses using the EMBOSS suite, NCBI BLAST, and the multiple sequence alignment programs ClustalW and T-Coffee. In some cases, when an external package is not available, Bioperl will fall back to using a slower method, either by emulating the package in pure Perl or by invoking a network-based analysis service such as the NCBI BLAST analysis queue. Additional work is in progress to incorporate into the project access to remote analysis services at the European Bioinformatics Institute (EBI) (Novella: http://industry.ebi.ac.uk/novella) and Pasteur Institute (Pise: http://bioweb.pasteur.fr).

For us to produce uniform software code, we established coding guidelines that are extensions of widely accepted object-oriented programming style. All modules were required to meet minimal standards before release. These standards include a complete set of regression tests, well-formed embedded documentation for each method, and a concise example code in the SYNOPSIS section of each module's documentation. We use the Perl embedded documentation format (called POD, or Plain Old Documentation) to interleave documentation and the source code. This documentation can be converted to text, TeX, or HTML. We have used the Pdoc (http://pdoc.sourceforge.net) tool to generate colored and hy-

perlinked documentation in HTML for easy online browsing available at http://doc.bioperl.org.

Our development process often starts when someone presents an idea for a needed tool on our mailing list. Ideally, a prototype or example code is posted, and we discuss ideas for implementation and common scenarios for where the functionality is needed. The developer then writes, or prepares from code he already has, an interface for the proposed module that describes the basic routines the module will implement. For the new module, we require a set of regression tests to be written. This is intended to ensure that the module satisfies its declared interface and can be tested easily later on when other modules that may interact with it have been updated. Bioperl 1.0 >3000 such tests that passed on multiple platforms before the toolkit was declared ready for release. Finally, all Bioperl code is subject to ongoing code review by and with the core developer group. This methodology is derived in large part from the software development strategy called Extreme Programming (Beck 1999). This process is highly iterative and modules are often revisited and improved depending on the needs of the developer. We attempt to always maintain a backward compatibility, so that code that depends on an earlier version of the module will continue to work.

To support multiple developers in different time zones and institutions, the entire Bioperl source code is hosted by the Open Bioinformatics Foundation (OBF) (http://www.open-bio.org) on a server in which our code and documentation is shared among developers and interested users. Information on how to obtain the source directly from our server is available at the project Web site http://www.bioperl.org and the source code server at http://cvs.open-bio.org.

RESULTS

Bioperl has >20 active developers led by a core of five primary developers who ensure that standards are met, prepare code releases, and set the vision for the project. At the time of writing, the mailing lists for the project include 1300 subscribers, and our Web site recieves an average of 10,000 unique visitors each month. The project has been used in a variety of endeavors including genome sequencing, annotation, sequence variation elucidation, disease gene discovery, and comparative genomics. An example using Bioperl modules to complete the task of retrieving sequences from a remote database is shown in Figure 3, and an example of parsing a BLAST report can be seen in Figure 4.

By far, the most advanced use of the Bioperl toolkit has come through the EnsEMBL (Hubbard et al. 2002) project. The basic sequence handling, file format parsing, and sequence features for annotation model have been used as building blocks for automatically annotating the *Danio rerio*, *Drosophila melanogastor*, *Takifugu rubripes*, *Homo sapiens*, *Anopheles gambiae*, and *Mus musculus* genomes (http://www.ensembl.org).

Additionally, the Genquire (Wilkinson et al. 2002) annotation package is built on top of the Bioperl object model and stores sequence and annotation data in a relational database. The interactive sequence-rendering capabilities are partitioned into a specific Bioperl package called bioperl-gui.

The Generic Model Organism Browser (GBrowse) (L.D. Stein, A. Day, T. Harris, A. Arva, S. Shu, S. Lewis, and C. Mungall, in prep.), Distributed Annotation System Perl (DAS) server (Dowell et al. 2001), and TFBS (Lenhard and Wasserman 2002) all use the Bioperl object model to describe sequences and Bioperl tools to complete analyses. The GBrowse system is a Web interface to databases of features for a genome project. The DAS system provides researchers a means to annotate sequences locally and publish the annotations to the community via the DAS XML protocol. TFBS provides a Perl implementation of objects for DNA sequence pattern representation by matrix profiles, with associated methods for searching the sequences for the occurrence of patterns, pattern storage, and generation of new patterns. The implementation uses Bioperl sequence, alignment, sequence features, and feature pair objects.

```
use Bio::SearchIO;

# Let's parse a BLAST report
my $search = new Bio::SearchIO( -format => 'blast',
                                -file   => 'report.bls');
my @HitsToSave = ();
my $cutoff_Evalue = 0.001;
my $cutoff_Len    = 120;
# iterate over each query sequence
while(my $result = $search->next_result) {
  # iterate over each hit on the query sequence
  while(my $hit = $result->next_hit) {
    # iterate over each HSP in a hit
    while( my $hsp = $hit->next_hsp ) {
      if( $hsp->evalue < $cutoff_Evalue &&
        $hsp->length('total') >= $cutoff_Len ) {
        push @HitsToSave, $hit;
        last; # Only want to process each Hit once
      }
    }
  }
}

print "Hits:\n";
foreach my $hit ( @HitsToSave ) {
  print $hit->name, "\n";
}
```

Figure 4 Report parsing with Bio::SearchIO. This code parses a BLAST report from a file called report.bls and saves, in an array called @HitsToSave, only the hits that have High-scoring Segment Pairs (HSPs) meeting an e-value and length threshold. In this case, any hit with e-value >0.001 or length < 120 residues will be excluded. Once the array is built, the names of each of the hits that had a HSP that met the criteria are printed out. To parse a FASTA (Pearson and Lipman 1988) report file one simply changes the format specification from blast to fasta.

Interoperability

Sometimes the best solution for a bioinformatics problem is a hybrid of multiple tools. Providing interoperability between languages allows a programmer using the toolkit to build components using work done in other languages and projects. These tools, written in different programming languages such as C, Java, and Python, can be used within a Perl program simply by invoking them (a process often called shelling out). In some situations, these tools require that data be available in a certain format or within a certain database. Bioperl provides software layers that can, for example, populate a database with sequence information that can be accessed and used to generate an interactive graphical interface provided by the Biojava toolkit. In other cases, Bioperl is used to create files in a format recognized by other programs so that they can perform their analyses.

Bioperl also supports a number of Extensible Markup Language (XML) standard data exchange formats accepted in the Bioinformatics community. Previous work has outlined scenarios in which XML has been useful in a biological context (Achard et al. 2001). XML standards supported by Bioperl

include the sequence markup formats Bioinformatics Sequence Markup Language (BSML; http://www.bsml.org) and Berkeley *Drosophila* Genome Project's (http://www.fruitfly.org) Genome Annotation Markup Elements (GAME; http://www.bioxml.org/dtds/index.html), NCBI BLAST XML for BLAST reports, and the bibliographic standards Medline XML provided by the European Bioinformatics Institute's Bibliographic Query Service (BQS; http://industry.ebi.ac.uk/openBQS/) and Entrez Pubmed XML format (Wheeler et al. 2002). By supporting these XML-based formats, programs using Bioperl are able to process data from a growing number of data sources that have adopted them as their standard. Furthermore, the extensible nature of XML allows new features to be added to the data formats without breaking existing parsers and code.

Software can interoperate not only through the invocation of external programs, but also through invoking methods on remote components possibly written in a different programming language from the calling component. Such a mechanism constitutes the tightest integration of reusable software components in a language-independent way. The Common Object Request Broker Architecture (CORBA) (Object Management Group 2001) provides an architecture for enabling this technology. This technology has been applied to biological data at the EBI in their Radiation Hybrid (Rodriguez-Tomé and Lijnzaad 2001) and EMBL Nucleotide Databases (Wang et al. 2000). CORBA implementations are available from commercial vendors (e.g., Inprise's VisiBroker, IONA's ORBacus) as well as from open-source projects (e.g., ORBit, MICO). Bioperl is compliant with the BioCORBA project (http://www.biocorba.org), one of the proposed standards for CORBA components for biological processes. BioCORBA is also supported by the Biojava (http://www.biojava.org) and Biopython (http://www.biopython.org) projects. The standard is under consideration for adoption by the Management Group's Life Science Research group (http://lsr.ebi.ac.uk) and is included in the proposed Open Bioinformatics Database Access (OBDA; http://obda.open-bio.org) standard for sequence data access. Bioperl's support for BioCORBA allows applications written on top of Bioperl to interact transparently with remote BioCORBA servers to perform operations as diverse as protein domain analysis and bibliographic retrieval, without knowing any of the details of the CORBA protocol.

Last, Bioperl fully supports the recent Open Bioinformatics Database Access initiative (OBDA), a language-independent relational database schema for the storage and retrieval of GenBank/EMBL entry-based sequences. By supporting this common schema, sequence objects that are created and maintained by Bioperl can be accessed and manipulated by OBDA-compliant programs written in Java, Python, Ruby, or C. This provides a level of read/write cross-platform data compatibility unmatched by any other sequence database project, and enables extensive enterprise-level application development.

DISCUSSION

Open-source development has proven to be a valuable and productive mechanism for creation of the toolkit. No single individual owns the project, rather it is owned by the community of contributors. The community approach prevents the death of a project due to loss of interest by the sole developer and does not permit project stagnation in the confines of a single laboratory in which a single individual or group is responsible for the continued vitality of a project. The original Bioperl project team has been replaced completely over the last 7 yr as members leave the project and new contributors join; however, the project aims have remained focused, whereas the functionality has continued to expand.

Throughout the development process, we learned a great deal about appropriate software practices for a diverse group of contributors. Our programming methodology, which includes defining use cases for our software, establishing a comprehensive regression test suite, and utilizing code reviews or audits of contributed source code, helped the community develop code that is compatible and consistent. The principles of good design and good documentation have made it easier for new developers to join the project.

A number of previous collaborative projects to develop comprehensive libraries for the life sciences have produced unsatisfactory results, or have been aborted prior to fruition. We believe that in many of these cases, ongoing development was stymied by a lack of commitment to open-source principles. For example, BioWidgets, an early attempt to create a Java-based biological toolkit, did not adopt an open-source methodology at its inception. This left it open to intellectual property disputes, and ultimately the project dissolved as early contributors withdrew their software from the project (N. Goodman, pers. comm.). In contrast, none of the software placed in Bioperl can be withdrawn from the project, because it has been explicitly donated to the community using a license that expressly permits copying and modification.

Another example of a non-open-source project that has failed to live up to its potential is the NCBI toolkit (ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools/), a powerful and highly functional C-language-based toolkit. Despite its quality, the toolkit has failed to achieve widespread usage because of acknowledged deficiencies in its documentation and a development process that is closed. By limiting the development process to NCBI personnel, the toolkit has evolved to work best within the NCBI environment and to address NCBI-specific operating procedures. Outside of the NCBI, the toolkit is used to run BLAST, whereas other functionality lies fallow. In contrast, Bioperl has a large and active user base because of extensive interaction between the developers and the users of the library. We allow users to contribute bug fixes and documentation improvements directly to the project, thereby keeping the project relevant to their needs.

We feel that much of the success of the Bioperl toolkit can be attributed to the open-source nature of the project that has allowed a diverse group of individuals to participate in a collaborative effort. We have successfully encouraged users of the toolkit to assist in the development by contributing bug fixes, documentation enhancements, and new functionality for the benefit of all users. Contributors are part of academic, governmental, nonprofit, pharmaceutical, and commercial bioinformatics groups on every continent. This collaboration and the guiding principle to get working products written in an extensible manner have made Bioperl an excellent platform for Perl bioinformatics software development. The open sharing and discussion of ideas that embodies the scientific spirit has proven to be successful in the world of scientific software development as well.

The open-source development model also has some drawbacks, which came to light throughout the lifetime of

the project. One drawback is that component development is only focused on what contributors find useful for their own work. Because developers tend to be technically advanced, the code documentation and tutorials have been geared toward these types of advanced users. Further, when a strong developer has left the project, it has not always been immediately possible to find a contributor willing to carry on the portion of the project for which the original developer was responsible. As a result, some parts of the project have been temporarily neglected, and in some cases phased out.

We have addressed these issues by establishing guidelines for contributions that includes a commitment to comprehensive documentation and high standards for released code. When necessary, we gracefully retire unmaintained components by providing deprecation warnings to the community, and at all times endeavor to ensure that there is a clear migration path from deprecated modules to new modules that provide equivalent functionality.

In the future, Bioperl will continue to evolve, addressing more domains of bioinformatics. We plan to create objects to manage sequence assembly information, haplotype maps, gene expression, and protein interaction data. Additionally, projects focusing on multispecies comparisons will build Perl modules to manage alignment and syntenic information. We will create software layers to interact with OBDA databases, develop a generic analysis pipeline system to provide automated analysis components, and expand the supported file formats the toolkit can read and write.

ACKNOWLEDGMENTS

The current Bioperl Core comprises, in alphabetical order, Ewan Birney, Hilmar Lapp, Heikki Lehväslaiho, Jason Stajich, and Lincoln Stein. The authors acknowledge contributions from the following people, in alphabetical order: Brad Chapman, Michele Clamp, Tony Cox, James Cuff, Andrew Dalke, Allen Day, Arne Elofsson, Mark Fiers, Ed Green, Roger Hall, Peter van Heusden, Joseph Insana, Nicolas Joly, Aaron J Mackey, Emmanuel Mongin, Jong Park, Lorenz Pollak, Richard Resnick, Todd Richmond, Gert Thijs, Charles Tilford, Andrew Walsh, Kai Wang, and Alex Zelensky. Additional ideas and help came from other OBF project team members including Jeff Chang, Thomas Down, Keith James, and all of the Bioperl mailing list members. Some parts of the object model, especially locations, were adopted from the excellent work of the Biojava project and its leaders Thomas Down and Matthew Pocock. We acknowledge the former project coordinators, in chronological order: Steven Brenner, Chris Dagdigian, Georg Fuellen, Steve Chervitz, and Ewan Birney, as well as initial contributors Jong Park and Richard Resnick, who provided help establishing the project. Our group owes its early organizational support to its association with the VSNS-BCD BioComputing Courses, http://www.techfak.uni-bielefeld.de/bcd/welcome.html, funded by the Association for the Promotion of Science and Humanities in Germany (Stifterverband fur die Deutsche Wissenschaft). We thank Brian Osborne and Peter Schattner for their documentation and tutorial work, and Chris Dagdigian for his tremendous support as computer systems administrator for the OBF. The Bioperl project and its sister projects (commonly referred to as the Bio{*} projects) are supported under the umbrella of the Open Bioinformatics Foundation (http://www.open-bio.org). OBF has received hardware donations from Compaq and Sun Microsystems, and we accept donated bandwidth and computer server space from Wyeth Research. J.E.S. is supported by NIH Genetics training grant T32 GM07754–22. S.E.B is supported by NIH grant 1 K22 HG00056. I.K. is supported by NHGRI Grant K22 HG-00064-01. L.D.S. is supported by NIH grants HG00739 and P41HG02223. E.B. is support by EMBL core funding. We thank F. Dietrich, M. DeLong, M. Hahn, and two anonymous reviewers for their comments on this work.

The publication costs of this article were defrayed in part by payment of page charges. This article must therefore be hereby marked "advertisement" in accordance with 18 USC section 1734 solely to indicate this fact.

REFERENCES

Achard, F., Vaysseix, G., and Barillot, E. 2001. XML, Bioinformatics, and data integration. *Bioinformatics* **17:** 115–125.

Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. 1997. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res.* **25:** 3389–3402.

Beck, K. 1999. Extreme programming examined: Embrace change. Addison Wesley, Reading, MA.

Burge, C. and Karlin, S. 1997. Prediction of complete gene stuctures in human genomic DNA. *J. Mol. Biol.* **268:** 78–94.

Chervitz, S.A., Fuellen, G., Dagdigian, C., Brenner, S.E., Birney, E., and Korf, I. 1998. Bioperl: Standard perl modules for bioinformatics. Bio Informatics Technology and Systems (BITS), http://www.bitsjournal.com/bioperl.html.

Dowell, R.D., Jokerst, R.M., Day, A., Eddy, S.R., and Stein, L.D. 2001. The distributed annotation system. *BMC Bioinformatics* **2:** 7.

Eddy, S.R. 2001. HMMER: Profile hidden Markov models for biological sequence analysis. http://hmmer.wustl.edu.

Felsenstein, J. 1983. PHYLIP (Phylogeny Inference Package) version 3.5c. Distributed by the author. Department of Genetics, University of Washington, Seattle.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. Design patterns: Elements of reusable -object-oriented software. Addison Wesley, Reading, MA.

Gish, W. 2002. Washington University BLAST. http://blast.wustl.edu.

Hubbard, T., Barker, D., Birney, E., Cameron, G., Chen, Y., Clark, L., Cox, T., Cuff, J., Curwen, V., Down, T., et al. 2002. The Ensembl genome database project. *Nucleic Acids Res.* **30:** 38–41.

Lenhard, B. and Wasserman, W.W. 2002. TFBS: Computational framework for transcription factor binding site analysis. *Bioinformatics* **18:** 1135–1136.

Notredame, C., Higgins, D.G., and Heringa, J. 2000. T-coffee: A novel method for multiple sequence alignments. *J. Mol. Biol.* **302:** 205–217.

Object Management Group 2001. CORBA/IIOP Specification. OMG publications. http://www.omg.org/technology/documents/formal/corba_iiop.htm.

Open Source Initiative 2001. The open-source definition. http://opensource.org/docs/definition.html.

Pearson, W.F. and Lipman, D.J. 1988. Improved tools for biological sequence analysis. *Proc. Natl. Acad. Sci.* **85:** 2444–2448.

Rice, P., Longden, I., and Bleasby, A. 2000. The European molecular biology open source suite. *Trends Genet.* **16:** 276–277.

Rodriguez-Tomé, P. and Lijnzaad, P. 2001. Rhdb: The radiation hybrid database. *Nucleic Acids Res.* **29:** 165–166.

Thompson, J.D., Higgins, D.G., and Gibson, T.J. 1994. CLUSTAL W:Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* **22:** 4673–4680.

Tremblay, J.P. and Cheston, G. 2001. Data structures and software development in an object-oriented domain. Prentice Hall, Upper Saddle River, NJ.

Wang, L., Rodriguez-Tomé, P., Redaschi, N., McNeil, P., Robinson, A., and Lijnzaad, P. 2000. Accessing and distributing EMBL data using CORBA (common object request broker architecture). *Genome Biol.* **1:** research0010.1–0010.10.

Waterston, R.H., Lander, E.S., and Sulston, J.E. 2002. On the sequencing of the human genome. *Proc. Natl. Acad. Sci.* **99:** 3712–3716.

Wheeler, D.L., Church, D.M., Lash, A.E., Leipe, D.D., Madden, T.L., Pontius, J.U., Schuler, G.D., Schriml, L.M., Tatusova, T.A., Wagner, L., et al. 2002. Database resources of the National Center for Biotechnology Information: 2002 Update. *Nucleic Acids Res.* **30:** 13–16.

Wilkinson, M.D., Block, D., and Crosby, W.L. 2002. Genquire: Genome annotation browser/editor. *Bioinformatics* (in press).

WEB SITE REFERENCES

ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools/; NCBI toolkit.
http://bioweb.pasteur.fr; L'Institut Pasteur Bioweb (Pasteur Institute).
http://doc.bioperl.org; Bioperl code documentation page.
http://forkhead.cgr.ki.se/TFBS/; TFBS Project.
http://industry.ebi.ac.uk/novella/; EBI Novella project Institute.
http://industry.ebi.ac.uk/openBQS/; BQS –Bibliographic Query Service.
http://lsr.ebi.ac.uk; Management Group's Life Science Research group.
http://obda.open-bio.org; Open Bioinformatics Database Access.
http://opensource.org/docs/definition.html; Open-Source Initiative–Open-Source definition.
http://pdoc.sourceforge.net; Pdoc home page.
http://www.biocorba.org; BioCORBA Project.
http://www.biojava.org; Biojava Project.
http://www.bioperl.org; Bioperl Project.
http://www.biopython.org; Biopython Project.
http://www.bioxml.org/dtds/index.html; GAME–Genome Annotation Markup Elements.
http://www.bsml.org; BSML–Bioinformatic Sequence Markup Language.
http://www.ebi.ac.uk; EMBL Outstation–European Bioinformatics Institute.
http://www.ensembl.org; EnsEMBL Project.
http://www.fruitfly.org; Berkeley *Drosophila* Genome Project.
http://www.open-bio.org; Open Bioinformatics Foundation.

Received May 4, 2002; accepted in revised form August 9, 2002.