Project and implementation of an object oriented open-source framework for genome sequencing projects

Rodrigo C. M. Coimbra Shana S. Santos Alex V. Barbosa Anderson G. F. Pereira
Fernando A. A. C. de Albuquerque
Maria Emília M. T. Walter
University of Brasilia (UnB)
Department of Computer Science
Darcy Ribeiro Campus, Brasília, Brazil

Abstract

Computational systems play an essential role on genome sequencing projects. Fragments of multiple copies of DNA or RNA are sequenced on molecular biology laboratories and computational systems must group these fragments in order to obtain the original DNA or RNA. These computational systems have three phases: submission, assembly and annotation. There are many efforts on bioinformatics centers around the world to create systems that can be easily adapted to be used on different sequencing projects. In this work, we propose a framework for computational systems that support genome sequencing projects, employing techniques of object-oriented systems, in the context of open-source projects. We also present two case studies of computational systems generated using our framework.

1 Introduction

Genome sequencing projects aim to discover the sequence of bases forming DNA chromosomes, transcript genes or non-coding RNAs of an organism. Due to the large volume of these biological sequences, together with information related to each one of them, the sequencing projects are highly dependent of computational systems. On the laboratories of molecular biology, biologists produce multiple copies of long biological sequences of an organism and cut it in short pieces. These fragments are then sequenced by automatic sequencers, since these machines can not treat long sequences. Sequencing is the task of obtaining the bases — A (Adenine), C (Cytosine), G (Guanine) and T (Thymine) or U (Uracyl) — composing biological sequences. Many fragments composing a plaque are simultaneously sequenced, and each fragment generates an *electropherogram*, having four colored graphics, each one corresponding to one of the four bases, A, C, G and T. When a graphic presents a peak on a certain position of the fragment, a base is identified on this position. If a particular base can not be identified, the character N (uNknown) is associated to the corresponding position. The electropherograms are sent by the biologists to the bioinformatics laboratory, usually using web interfaces. These fragments must be joined in order to reconstruct the original DNA. Besides, biological functions or characteristics must be identified, also a task strongly supported by computational tools.

Typically, a computational system, developed inside a bioinformatics laboratory, processes the fragments on three phases: submission, assembly and annotation. On the **submission phase**, each electropherogram is transformed on a string, called *read* or *sequence*, in which each character is associated to a value measuring its error probability. Usually this task is made by the *Phred* program [11, 10], that generates a file, in *phd* format, containing a string composed by characters A, C, G, T or N for each read, and the error probability associated to each base. Program *Phd2Fasta* [14] converts this *phd* file to two text format files (in *FASTA* format [18]), *read file* and *quality file*. Each sequence is filtered to remove portions probably not belonging to the organism being studied, but to vectors (DNA sequences of organisms used to replicate the DNA of the studied organism) and contaminants (DNA sequences of other organisms), using programs such as *Crossmatch* [14]. Depending on each sequencing project, additional analysis can be made. For example, a redundancy analysis shows if the fragments of a plaque do not overlap, since as fewer is the number of overlapping fragments inside the plaque as better is the quality of the generated fragments. Finally, these files are stored on a database (Figure 1). Other informations also can be stored on databases, according to the sequencing

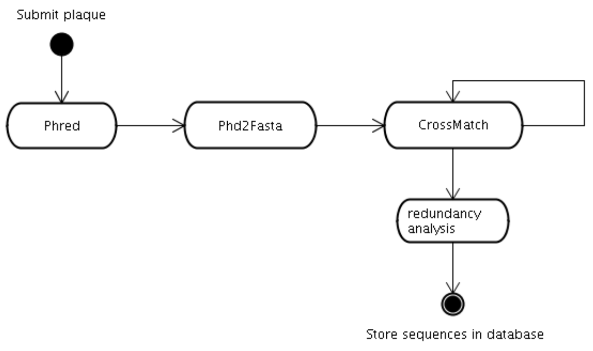


Figure 1. Activity diagram of a pipeline of the submission phase (UML notation [5]).

project needs.

The **assembly phase** consists in grouping *similar sequences* (sequences having "approximately equal" prefixes and suffixes). Two sequences are similar if there are similarities between the suffix from one and the prefix from the other sequence. This clustering aims to put together fragments potentially belonging to the same region of the DNA. Groups formed by more than one sequence are called *contigs* (Figure 2), and groups formed by only one sequence are called *singlets*. From each one of the contigs, a consensus sequence is generated and represents the contig (Figure 2). Programs *Phrap* [14] and *CAP3* [16] are usually used to assembly sequences. Both generate, among others, a file on the *ace* format (containing data about similar suffixes and prefixes of the sequences composing a contig), a file containing data about the contigs, and another one containing the singlets data. Finally, the assembly phase store some statistics, like the total number of groups (contigs and singlets), the total number of identified genes, and some contig visualizations.

Genome sequencing projects can sequence genomic DNA (chromosome DNA) or ESTs (Expressed Sequence Tags) — that are short sequences of transcript DNA. In the first case, it is necessary to identify possible genes on the sequences. This is made by programs like *Glimmer* [22, 8], that identifies beginning or ending positions of a fragment

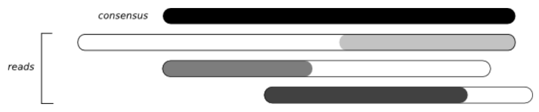


Figure 2. A contig resulting from prefix and suffix similarities, and its consensus sequence.

region possibly coding a gene, known as *open reading frame* (ORF). *Glimmer* has to be used on each contig consensus sequence and on each singlet, and the positions of each candidate gene must be stored on a database. For EST projects there is no need to identify genes, so the consensus sequences of contigs and singlets are directly stored on the database.

The **annotation phase** has the objective of identifying functions of the sequences generated on the assembly phase, and usually is divided on two steps. First, on the automatic annotation step, all the project sequences must be compared with sequences stored on public databases, because their functions are inferred by previously determined functions of similar sequences. The hypothesis is that similar sequences have similar biological functions. The second step, the manual annotation, is done by biologists, that use information of the automatic annotation and their knowledge to decide the biological function to be associated to a sequence. Data generated on both steps, automatic and manual, are stored on the databases of the project.

In the automatic annotation, programs like *BLAST* [2] and *Fasta* [20] are often used. *BLAST* (Basic Local Alignment Search Tool) is broadly used to search similar sequences on DNA and protein databases. *BLAST* is a family of computational and statistical methods, also employing heuristics, to search best alignments between two sequences. An *alignment* of two sequences can be obtained putting one sequence above the other, showing correspondences between two characters, one of each sequence, or between a gap and a character, such that both sequences, with gap inserted, have equal lengths. Each alignment has associated values, to express reliability and error probability. Higher reliabilities and lower error probabilities indicate good alignments, and are called *best hits*. *BLAST* has many programs depending on the query sequence (bases or aminoacids) and on the sequence database (bases or aminoacids). *Fasta* is also a family of programs for searching similarities between a query sequence and each of the sequences of a database, both composed by bases or aminoacids, using particular algorithms and strategies to obtain similarities and best alignments. It also associates reliability and error probability to each alignment. These programs generate output in HTML format, that can be directly stored on the database, or in *text* format, that can be processed later. These data are visualized by biologists usually using web pages.

There are many projects to assist and support the development of computational systems for genome sequencing projects. Some of the systems covers the three phases of pipeline (submission, assembly and annotation), but most of them were specifically developed to support the annotation phase.

ACeDB (A *C. elegans* DataBase) [26], developed by the

Sanger Institute, was originally written to analyze and to store data from the *C. elegans* genome project. *ACeDB* was concerned on data storage more than on annotation pipeline, and its first development was not based on techniques of software engineering — its code style was "baroque, sparsely comented and idiosyncratic" (*apud* [15]). *ACeDB* is distributed under the GNU General Public Licence except for a few parts which are distributed under the GNU Lesser General Public License.

Examples of systems that contemplate the three phases of the pipeline are *SABIA* and *GARSA*. *SABIA* (System for Automated Bacterial Integrated Annotation) [1] is composed by an automated web enviroment and a set of Perl/CGI scripts for data manipulation on a relational database (MySQL). It is distributed under proprietary license. *GARSA* (Genomic Analyses Resources for Sequence Annotation) [7] aims to integrate, to analyze and to present information of several bioinformatics tools and genome databases. Its architecture is based on Perl scripts, resulting in code effort to add new tools to the pipeline. It executs some specific programs, accepting the definition of programs parameters, but not plug-ins. Although *GARSA* is distributed under GPL license, download is allowed by request.

Examples of systems mainly developed for the annotation phase include *Genotator* [15], used by several groups in LBNL (Lawrence Berkeley National Laboratory), as well as in Stanford University and other genome study centers. It was developed for the automated execution of some programs for sequence analysis and annotation. *GAIA* (Genome Annotation and Information Analysis) [19, 3], developed by the Bioinformatics Center of Pennsylvania University, supports semiautomated annotation, being first designed to support the human genome annotation. Its components include a configurable pipeline, an storage manager of relational information and a Java-based user interface. Its annotation mechanism is based on autonomous components, called *sensors*, each one performing a specific analysis, and the communication among sensors is made using the annotation database. *BASys* [9] generate more than 60 separate annotations for a gene, but the computacional effort of its developers is more focused on data graphic presentation. *Manatee* [24], created by the Bioinformatics Department of The Institute for Genomic Research (TIGR), is an open source web based interface for interactive editing of annotation data. It is used after the submission of the sequence data to TIGR Annotation Engine, that is an automated annotation pipeline.

Another example of a system that aims to model bioinformatics pipelines is BioWMS [4]. This system does not support genome sequecing projects as a whole, but introduces an elegant way to define bioinformatics pipelines. BioWMS uses UML activities diagrams to model pipelines and translates these diagrams into multiagent systems that will execute the pipeline.

Most of the existing systems are not focused on project modelling nor on free software concept. Some of them can be obtained by request to the authors or can be available for nonprofit organizations, therefore they are not open source in the wide sense of the word. An analysis regarding the architecture of those systems is harmed by the unavailability of the source code of most of them.

In this context, the objective of this work is to propose a framework that supports the three phases of a computational system supporting a genome sequencing project, developed employing techniques of object-oriented development (class modelling is in the core of the project), in the context of open source projects. We want to offer a framework with the most common computational systems functionalities, but with a simpler system configuration and architecture, when comparing with other systems.

Funcionalities of the framework are presented on Section 2. The system architecture is described on Section 3. Two case studies are presented and discussed on Section 4. Finally, on Section 5 we conclude and make some suggestions of future work.

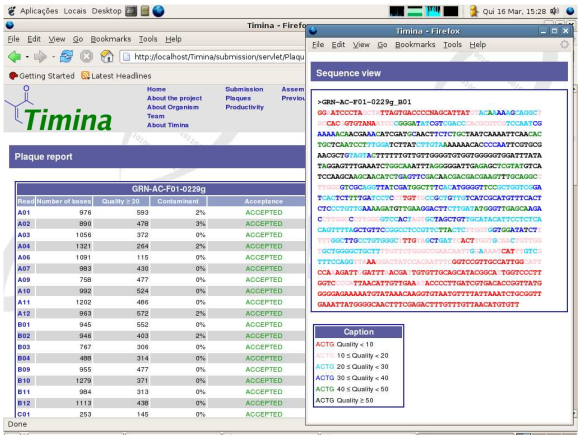
2 The framework and its functionalities

Our framework, called *Timina*, is an object-oriented web application developed in Java. It is a free software (distributed under the GNU General Public License [12]) developed on the SourceForge.net. The project is a free software seed at this moment, but it will migrate to a colaborative free software environment soon. It is available at `http://timina.sourceforge.net`. *Timina* can be downloaded and installed to be used on UNIX environments.

The web interface of *Timina* follows the W3C recommendations, which allow its correct visualization on different browsers. We also used Cascading Style Sheets (CSS), that allows changes on the style of the interface without changing the source code. Whenever possible we make web content accessible to people with disabilities [25]. There are translations for Brazilian Portuguese and English, but more translations can be easily added.

The framework provides support for plaque submission and for reports containing details of the reads from submitted plaques. It also reports, for each user, how many plaques were submitted. Total number of submitted plaques and total number of users that submitted plaques are other useful informations available (Figures 3 and 4).

Timina supports sequence assembly, starting the processing from a shell script. It provides reports containing information such as execution date, total number of accepted sequences, total of contigs, total of singlets and the base av-



Plaque report

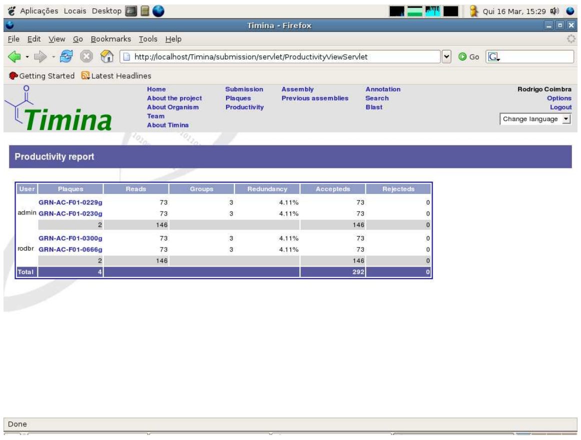
GRN-AC-F01-0229g				
Read	Number of bases	Quality ≥ 20	Contaminant	Acceptance
A01	976	593	2%	ACCEPTED
A02	890	478	3%	ACCEPTED
A03	1056	372	0%	ACCEPTED
A04	1321	264	2%	ACCEPTED
A06	1091	115	0%	ACCEPTED
A07	983	430	0%	ACCEPTED
A09	758	477	0%	ACCEPTED
A10	992	524	0%	ACCEPTED
A11	1202	486	0%	ACCEPTED
A12	963	572	2%	ACCEPTED
B01	945	552	0%	ACCEPTED
B02	946	403	2%	ACCEPTED
B03	767	306	0%	ACCEPTED
B04	488	314	0%	ACCEPTED
B09	955	477	0%	ACCEPTED
B10	1279	371	0%	ACCEPTED
B11	984	313	0%	ACCEPTED
B12	1113	438	0%	ACCEPTED
C01	253	145	0%	ACCEPTED

Sequence view

```
>GRN-AC-F01-0229g_B01
GGAATCCCTAGCTATTAGTGACCCCNAGCATTATGTACAAAAAGCAGGCT
GCCACGGTGTANAATCCCGGGATATCGTCGACCCACGCGTCGTCCAATCG
AAAAACAACGAAACATCGATGCAACTTCTCTGCTAATCAAAATTCAACAC
TGCTCAATCCTTTGGATCTTATCTTGTAAAAAAACACCCCAATTCGTGCG
AACGCTGTAGTACTTTTTTGTTGTTGGGGTGTGGTGGGGGTGGATTTATA
TAGGAGTTTGAAATCTGGCAAATTTAGGGGGATTGAGAGCTCGTATGTCA
TCCAAGCAAGCAACATCTGAGTTCGACAACGACGACGAAGTTTGCAGGCC
TTGGCGTCGCAGGTTATCGATGGCTTTCACATGGGGTTCCGCTGGTCGGA
TCACTCTTTTGATCCTCCTTGTTGCCGCTGTTGTCATCGCATGTTTCACT
CTCCCTGTTGAAAAGATGTTGAAGGACTTCTTGATATGGGTTGAGCAAGA
CCTTGGCCCTTGGGGTCCACTAGTGCTAGCTGTTGCATACATTCCTCTCA
CAGTTTTAGCTGTTCCGGCCTCCGTTCTTACTCTTGGTGGTGGATATCTT
TTTGGCTTGCCTGTGGGCTTTGTAGCTGATTCACTTGGTGCAACTGTTGG
TGCTGGGGCTGCTTTTGTTCTGGGCCGAACAATTGGAAAATCATTTGTCG
TTTCCAGGTTAAAGGACTATCCACAATTTCGGTCCGTTGCCATTGGCATT
CCAAAGATTGGATTTAACGATTGTGTTGCAGCATACGGCATTGGTCCCTT
GGTCCCCATTAACATTGTTGAAATACCCCTTGATCGTGACACCGGTTATG
GGGGAGAAAAATGTATAAACAAGGTGTAATGTTTTATTAAATCTGCGGTT
GAAATTATGGGGCAACTTTCGAGACTTTGTTTGTTAACATGTGTT
```

Caption
ACTG Quality < 10
ACTG 10 ≤ Quality < 20
ACTG 20 ≤ Quality < 30
ACTG 30 ≤ Quality < 40
ACTG 40 ≤ Quality < 50
ACTG Quality ≥ 50

Figure 3. A plaque report generated by Timina.



Productivity report

User	Plaques	Reads	Groups	Redundancy	Accepteds	Rejecteds
admin	GRN-AC-F01-0229g	73	3	4.11%	73	0
	GRN-AC-F01-0230g	73	3	4.11%	73	0
	2	146			146	0
rodbr	GRN-AC-F01-0300g	73	3	4.11%	73	0
	GRN-AC-F01-0666g	73	3	4.11%	73	0
	2	146			146	0
Total	4				292	0

Figure 4. A productivity report.

erage of all contigs. *Timina* shows the history of previous assemblies. As discussed before, in the case of genomic DNA projects, identification of possible genes in contigs and singlets is made.

Automatic annotation on *Timina* is supported by a process also started by a shell script. *Best hits* are listed in order to facilitate biological analysis. A form for manual annotation of sequences is disposed by a web interface. It is possible to search the fields of manual annotation form using keywords and to search hits produced by the automatic annotation.

Submission, assembly and automatic annotation phases are configurable. Using properties files, it is possible to define which programs must be executed, their execution order and parameters (Figures 14 and 15). It is possible to define the fields of the manual annotation form (Figures 16 and 17).

Timina provides a management system for project users. It is possible to insert, remove and modify user informations by a web interface. Permissions are given to each user, so access to parts of the system can be controlled. Authentication security, as well as navigation, must be done by a server supporting HTTPS.

The architecture of the framework, to be described on the next section, provides an interface for plug-ins and extensions, especially for annotation, so that it is easy to implement and add new funcionalities. The project of the framework also permits that a unique servlet container supports one or more sequencing projects.

3 The architecture of the framework

The design of *Timina* is based on a layered architecture: presentation, business and persistence. Our project makes a clear distinction among layers, defining interfaces and builder classes, and using design patterns like *Builder* and *Singleton* [13], such that the three layers composing the software can be distributed in distinct packages, that can be easily replaced or shared.

Presentation layer is composed by Servlets. *Timina* can be executed in any servlet container, such as Tomcat, in which it was developed and tested. The database management system used by *Timina* is PostgreSQL, adopted due to its reliability, good performance attributes and because it is a free software. Project and implementation of presentation and persistence layers are not the focus of this article, so in this work we will detail only the business layer.

The processing of sequencing project phases — submission, assembly and annotation — requires three steps: 1) to generate files, 2) to execute programs using previously generated files (pipeline), and 3) to parse resulting files of executed programs. These data must be stored on the database created for the project. For each phase, there is a subsystem[1] to control its steps containing modules to generate files, to execute programs and to parse resulting files. Figure 5 shows the general structure of modules from the business layer.

From the general description of the business layer, we will specify the classes for each subsystem. Next subsections will present the modules of submission, assembly and annotation subsystems. But first, we will define the classes corresponding to the system entities.

3.1 Entities

We first investigated the possibility to use BioJava [21], a framework containing classes to represent, for example, a DNA sequence and parsers to files commonly used in bioinformatics. But, its data structures would lead to a worse

[1]A subsystem is an aggregate of correlated modules.

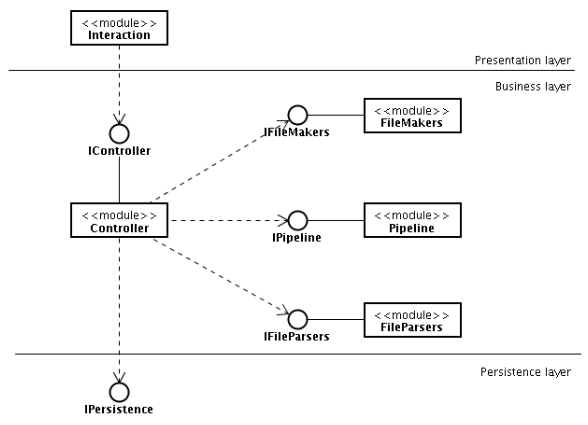


Figure 5. Module view of a subsystem.

performance when compared to a system developed specifically to manage a sequencing project by a group with expertise in Java and bioinformatics. For example, BioJava needs to load `SymbolList` and `Alphabet` objects [21] to represent a DNA sequence while *Timina* represents a sequence using a simple `char[]`.

In Figure 6 we present the entities of submission subsystem (UML notation [5]). `Sequence` represents a string of subunits (bases or aminoacids) of DNA, RNA or protein. `BaseSequence` is a DNA or a RNA `Sequence` with error probabilities (qualities) related to each base. The string of subunits is stored with a `char[]` because the Java class `String` has more resources than needed and its use would require more memory. `BaseSequence` has methods to translate and transcribe a sequence. `Read` consists of a base sequence and other useful informations computed by the system during submission phase, such as the number of bases with acceptable qualities, and the percentage of contaminant bases in the sequence. `Plaque` class contains a set of `Read` objects. `Submission` class represents a particular submission of DNA fragments to the system, and it is identified by two users, one that has submitted the plaque and another that has confirmed its inclusion. To confirm a plaque means that its reads will be used on the assembly phase.

Figure 7 shows the entities of the assembly subsystem. `AlignedBaseSequence` class represents a base sequence involved on an alignment. This class contains informations about gaps and orientation of a `BaseSequence`. `AlignedRead` class represents a sequence belonging to a contig. It stores the initial position and the list of pieces forming the consensus sequence. `Contig` class contains sequences forming the contig (`AlignedRead`) and the consensus sequence representing the contig (`AlignedBaseSequence`). A `Contig` or a `Singlet` can have one or more genes (for genomic DNA), represented by the `ORF` class. The results of an assembly program and its associated statistics are represented by the `Assembly` class.

Figure 8 shows the entities for the annotation subsystem. `AutomaticAnnotation` represents the results obtained from comparisons done by a program, like *BLAST*. `AutomaticAnnotationHit` class stores *best hits* produced by a program. `ManualAnnotation` class represents an annotation made by a biologist. A properties file defines the form fields (represented by `ManualAnnotationField`) that can be used by a biologist in his annotations.

3.2 The business layer

Definition of the entity classes allowed us to project and to implement the subsystems — submission, assembly and annotation — using the structure shown in Figure 5.

Submission subsystem is composed by the classes and interfaces presented on Figure 9. Note that this subsystem have no module to generate files. The `Pipeline` class allows execution of any sequence of programs following the definitions found in the *properties file*. On Section 4 we will show how to specify a pipeline in a properties file. So, `SubmissionPipeline` executes processes of the submission phase, such as to unzip a plaque and to adjust the name of the reads of a plaque.

The classes and interfaces of the assembly subsystem are described on Figure 10. `Pipeline` is also the core of the processing, here executing assembly programs. During the assembly phase, ORFs can be identified. So modules with the same structure for assembly processing can be used to identify ORFs: file makers, pipeline and file parsers.

Annotation subsystem has the structure of modules and interfaces shown on Figure 11. The class structure of this subsystem is the same of the assembly subsystem, having classes to generate files, to execute programs and to parse files, but without specific classes for dealing with ORFs.

Module `FileMakers` (exemplified by `AssemblyFileMakers` in the assembly subsystem)

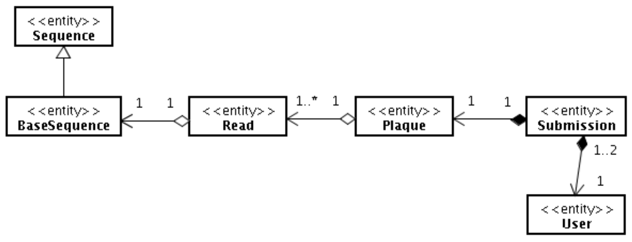


Figure 6. Entity classes for the submission subsystem.

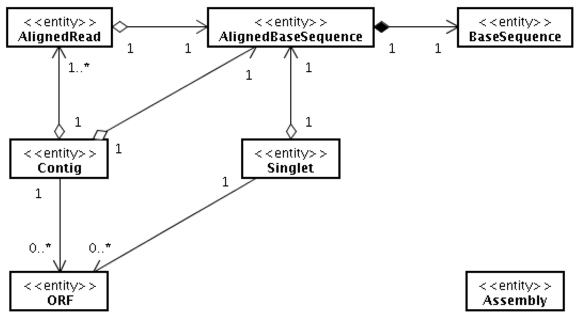


Figure 7. Entity classes for the assembly subsystem.

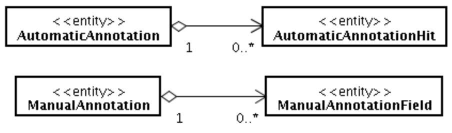

<<entity>> AutomaticAnnotation
<<entity>> AutomaticAnnotationHit
<<entity>> ManualAnnotation
<<entity>> ManualAnnotationField

Figure 8. Entity classes for the annotation subsystem.

hides classes defined in an inheritance tree that effectively generates the files. In fact, module `FileMakers` manages the use of these classes. For each file format, a class to write an object on the file can be implemented using the `FileMaker` abstract class. *Timina* includes a class that generates files on *FASTA* format (`FastaQualityMaker`, an extension of `FileMaker`), that writes the sequence and quality files.

`Pipeline` uses classes for configuring the execution of a program that uses a class structure similar to the *Command* design pattern [13]. `ProgramRunner` controls the execution of a program or command defined in `ProgramRunnerParameters`, using the

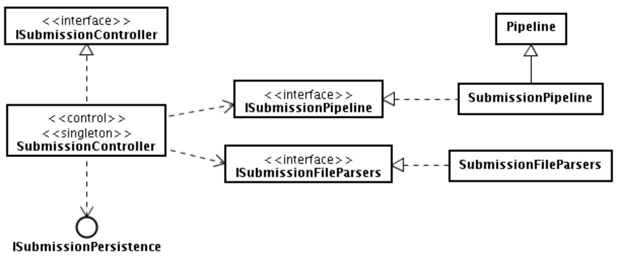

<<interface>> ISubmissionController
<<control>> <<singleton>> SubmissionController
ISubmissionPersistence
<<interface>> ISubmissionPipeline
<<interface>> ISubmissionFileParsers
Pipeline
SubmissionPipeline
SubmissionFileParsers

Figure 9. Classes and interfaces for the submission subsystem.

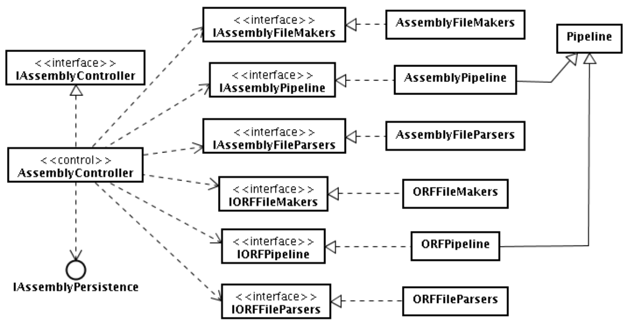

<<interface>> IAssemblyController
<<control>> AssemblyController
IAssemblyPersistence
<<interface>> IAssemblyFileMakers
<<interface>> IAssemblyPipeline
<<interface>> IAssemblyFileParsers
<<interface>> IORFFileMakers
<<interface>> IORFPipeline
<<interface>> IORFFileParsers
AssemblyFileMakers
AssemblyPipeline
AssemblyFileParsers
ORFFileMakers
ORFPipeline
ORFFileParsers
Pipeline

Figure 10. Classes and interfaces for the assembly subsystem.

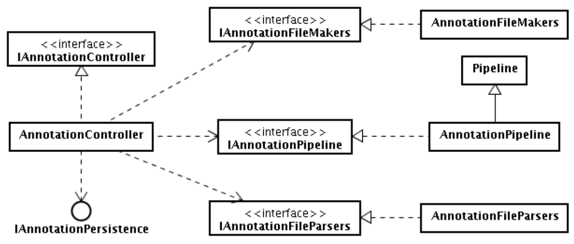

<<interface>> IAnnotationController
AnnotationController
IAnnotationPersistence
<<interface>> IAnnotationFileMakers
<<interface>> IAnnotationPipeline
<<interface>> IAnnotationFileParsers
AnnotationFileMakers
Pipeline
AnnotationPipeline
AnnotationFileParsers

Figure 11. Classes and interfaces for the annotation subsystem.

Java class `Process` to execute it. For each program executed by *Timina*, there is an extension of `ProgramRunnerParameters` for properly configure it (Figure 12). If necessary, any command can be executed by setting it in `ProgramRunnerParameters`.

Analogously to the `FileMakers` module, the `FileParsers` module manages the use of classes that effectively analyze files, iteratively parsing files to specific objects. For example, `FastaParser` sets `Sequence` objects from *FASTA* files and `AceParser` sets `Contig` objects from *ace* files (Figure 13).

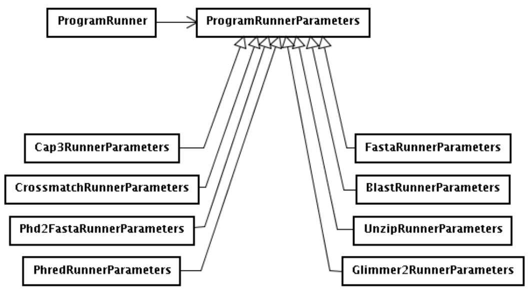


Figure 12. Classes for executing programs.

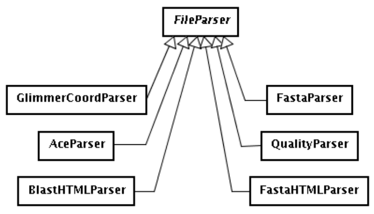


Figure 13. Classes for parsing files.

3.3 Extension points

As described before, business layer is the core of *Timina*. Interfaces provided by this layer — `ISubmissionController`, `IAssemblyController` and `IAnnotationController` — must be used by an implementation of the presentation layer, not described here. In fact, presentation is not the objective of *Timina*, so a user of the framework could project a particular implementation. Likewise, the persistence layer was designed as an independent module that can be replaced. A framework user would project a specific persistence layer, for example, to access other database systems, instead of the PostgreSQL. This layer can be replaced by implementing the persistence interfaces — `ISubmissionPersistence`, `IAssemblyPersistence` and `IAnnotationPersistence` — to provide the services required by the business layer and to implement the interface that defines a framework module: `ITiminaModule`. Finally, for loading a persistence module of *Timina*, the user must implement a builder class (realizing `ITiminaModuleBuilder` interface) and must add the name of the builder in the main properties file.

Another important extension point is the interface for additional processing during the annotation phase. Implementing `IAnnotationPlugin` (an extension of the `ITiminaModule`), a module can modify data in the `ManualAnnotation` objects during the automatic annotation processing. The `ManualAnnotation` objects will be used as pre-annotations that may help biologists in the manual annotation task. Using the best hits of the automatic annotations, a particular annotation plug-in can use some other methods (heuristics or data mining, for example) to infer informations for the fields of the manual annotation. Multiple plug-ins can be executed during the automatic annotation processing. Loading an annotation plug-in is similar to load a persistence module, just adding the name of the class that implements the plug-in in the properties file.

New screens in the web interface can be designed for annotation plug-ins. For example, projects that need to study genes categories [23] could implement an annotation plug-in to cross data and to generate a corresponding report.

```
pipeline = phred, phdFst, cmatch, cap3

phred.class
 = br.unb.timina.programs.PhredRunnerParameters
phred.input = esd
phred.output = phd
phred.parameterFile = /usr/local/phred/phredpar.dat

phdFst.class
 = br.unb.timina.programs.Phd2FastaRunnerParameters
phdFst.input = phd
phdFst.output = reads.fst

cmatch.class
 = br.unb.timina.programs.CrossmatchRunnerParameters
cmatch.input = reads.fst
cmatch.minMatch = 12
cmatch.minScore = 20
cmatch.filter = /var/timina/data/vectors.fst

cap3.class
 = br.unb.timina.programs.Cap3RunnerParameters
cap3.input = reads.fst.screen

minimumQuality = 20
minimumNumberOfBasesWithMinimumQuality = 100

readsFastaToParse = reads.fst.screen
readsQualityToParse = reads.fst.screen.qual
readsAceToParse = reads.fst.screen.cap.ace
```

Figure 14. Properties file of the submission phase for the Jararaca Project.

4 Two case studies

In this section, we present the computational systems for the *Anaplasma Project* [17] and the *Jararaca Project* [6], both generated using our framework.

Figures 14 and 15 show examples of how a user can create a pipeline for submission phase using *Timina*. These definitions will be used by `Pipeline` class to instance `ProgramRunnerParameters`, configuring external programs to be executed in the order specified on the properties file. The pipeline defined at Figure 15 is the same presented on Figure 1. Basically the main differences among the pipelines are the filters (files with contaminants and vectors sequences) and reports required by each project.

Some computational systems for genome sequencing projects were previously developed on the Bioinformatics Laboratory of the University of Brasilia, and these pipelines were defined on the source code, particulary using the JAVA language. A class was used to control the execution of several programs by invoking an `execute()` method for specific classes of each external program.

Figures 16 and 17 show examples of how a user can define fields for the manual annotation using *Timina*. These definitions will be used to create columns in the database table that stores manual annotations. For each defined field, an instance of `ManualAnnotationField` will be created for a `ManualAnnotation` object. The definitions

```
pipeline = phred, phdFst, cmatch1, cmatch2 cmatch3, cap3

phred.class
 = br.unb.timina.programs.PhredRunnerParameters
phred.input = esd
phred.output = phd
phred.parameterFile = /usr/local/phred/phredpar.dat

phdFst.class
 = br.unb.timina.programs.Phd2FastaRunnerParameters
phdFst.input = phd
phdFst.output = reads.fst

cmatch1.class
 = br.unb.timina.programs.CrossmatchRunnerParameters
cmatch1.input = reads.fst
cmatch1.minMatch = 12
cmatch1.minScore = 20
cmatch1.filter = /var/timina/data/vectors.fst

cmatch2.class
 = br.unb.timina.programs.CrossmatchRunnerParameters
cmatch2.input = reads.fst.screen
cmatch2.minMatch = 12
cmatch2.minScore = 20
cmatch2.filter = /var/timina/data/bostaurus.fst

cmatch3.class
 = br.unb.timina.programs.CrossmatchRunnerParameters
cmatch3.input = reads.fst.screen.screen
cmatch3.minMatch = 12
cmatch3.minScore = 20
cmatch3.filter = /var/timina/data/ecoli.fst

cap3.class
 = br.unb.timina.programs.Cap3RunnerParameters
cap3.input = reads.fst.screen.screen.screen

minimumQuality = 20
minimumNumberOfBasesWithMinimumQuality = 100

readsFastaToParse = reads.fst.screen.screen.screen
readsQualityToParse = reads.fst.screen.screen.screen.qual
readsAceToParse = reads.fst.screen.screen.screen.cap.ace
```

Figure 15. Properties file of the submission phase for the Anaplasma Project.

of the fields are also used to show them for the biologists on the web interface. There is also the option to specify external links and translations for the field names.

For the previous computational systems, not generated by *Timina*, these fields were defined on the source code of the web interface and they were added manually in the database definition. In order to add new fields, the class representing the manual annotation also require changes in the source code.

Figure 19 shows the main page of the *Anaplasma Project* as generated by *Timina*. Note that *Timina* allows changes on the layout of the user interface by changing the style sheet. This is an interesting feature when comparing with the previous systems, in which changes in the layout definitions required modifications on the source code. Figure 18 shows some annotation pages in the *Jararaca Project*.

The use of properties files and CSS by *Timina* makes easier to adapt the system to the needs of each project.

```
annotationFields = ec, cat1, cat2, cat3

ec.name = ec
ec.name.en_US = EC (GO)
ec.name.pt_BR = EC (GO)
ec.link = yes
ec.link.name.en_US
 = GO "enzymes" to Enzyme Commission Numbers
ec.link.name.pt_BR
 = "Enzimas" GO para Enzyme Commission Numbers
ec.link.url
 = http://www.geneontology.org/external2go/ec2go

cat1.name = cat1
cat1.name.en_US = Category 1
cat1.name.pt_BR = Categoria 1
cat1.link = no

cat2.name = cat2
cat2.name.en_US = Category 2
cat2.name.pt_BR = Categoria 2
cat2.link = no

cat3.name = cat3
cat3.name.en_US = Category 3
cat3.name.pt_BR = Categoria 3
cat3.link = no
```

Figure 16. Properties file of the annotation form fields for the Jararaca Project.

Previous implementations of other genome projects were not focused in software architecture. For example, the class representing a plaque also controled the submission processing and the access to database. These characteristics implied on reuse problems on these systems — there was not a shared core between them. This implied that common funcionalities of these systems now have different implementations.

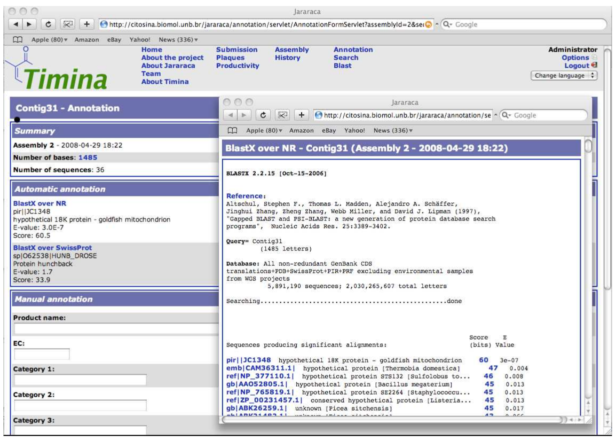
Timina tried to solve these problems with its software architecture (Section 3). Presentation, business and persistence are distributed on different packages, increasing reuse and reducing evolution problems. Business and persistence packages should be placed on the path of the libraries of the servlet container application. Presentation package and properties files must be placed on the specific path of a project in the servlet container (Figure 20).

5 Conclusions and future work

In this work we presented the framework *Timina*, that builds the three phases of a computational system supporting genome sequencing projects — submission, assembly and annotation — employing techniques of object-oriented development, in the context of open-source projects. *Timina* offers the most common functionalities of computational systems for sequencing projects, but with a simpler configuration and architecture, when compared to other systems. We also presented and discussed two case studies of bioinformatics systems generated by our framework.

```
annotationFields = cogCat, cogName, ec, cat1, cat2, cat3,\\
 defGO1, defGO2, defGO3, defGO4, defGO5, ortoGene, ortoName

cogCat.name = cogCategory
cogCat.name.en_US = COG Category
cogCat.name.pt_BR = Categoria COG
cogCat.link = yes
cogCat.link.name.en_US = COG table
cogCat.link.name.pt_BR = Tabela do COG
cogCat.link.url
 = ftp://ftp.ncbi.nih.gov/pub/COG/COG/fun.txt

cogName.name = cogName
cogName.name.en_US = COG Name
cogName.name.pt_BR = Nome do COG
cogName.link = no

ec.name = ec
ec.name.en_US = EC (GO)
ec.name.pt_BR = EC (GO)
ec.link = yes
ec.link.name.en_US
 = GO "enzymes" to Enzyme Commission Numbers
ec.link.name.pt_BR
 = "Enzimas" GO para Enzyme Commission Numbers
ec.link.url
 = http://www.geneontology.org/external2go/multifun2go

cat1.name = cat1
cat1.name.en_US = Category 1
cat1.name.pt_BR = Categoria 1
cat1.link = no

cat2.name = cat2
cat2.name.en_US = Category 2
cat2.name.pt_BR = Categoria 2
cat2.link = no

cat3.name = cat3
cat3.name.en_US = Category 3
cat3.name.pt_BR = Categoria 3
cat3.link = no

defGO1.name = defGO1
defGO1.name.en_US = GO Definition 1
defGO1.name.pt_BR = Defini\u00E7\u00E3o GO 1
defGO1.link = no

defGO2.name = defGO2
defGO2.name.en_US = GO Definition 2
defGO2.name.pt_BR = Defini\u00E7\u00E3o GO 2
defGO2.link = no

defGO3.name = defGO3
defGO3.name.en_US = GO Definition 3
defGO3.name.pt_BR = Defini\u00E7\u00E3o GO 3
defGO3.link = no

defGO4.name = defGO4
defGO4.name.en_US = GO Definition 4
defGO4.name.pt_BR = Defini\u00E7\u00E3o GO 4
defGO4.link = no

defGO5.name = defGO5
defGO5.name.en_US = GO Definition 5
defGO5.name.pt_BR = Defini\u00E7\u00E3o GO 5
defGO5.link = no

ortoGene.name = ortoGene
ortoGene.name.en_US = Ortholog gene on E. coli
ortoGene.name.pt_BR = Gene ort\u00F3logo em E. coli
ortoGene.link = no

ortoName.name = ortoName
ortoName.name.en_US = Name of ortholog gene on E. coli
ortoName.name.pt_BR = Nome do gene ort\u00F3logo em E. coli
ortoName.link = no
```

Figure 17. Properties file of the annotation form fields for the Anaplasma Project.


Jararaca
http://citosina.biomol.unb.br/jararaca/annotation/servlet/AnnotationFormServlet?assemblyId=2&se
Google
Apple (80) Amazon eBay Yahoo! News (336)
Timina
Home
About the project
About Jararaca
Team
About Timina
Submission
Plaques
Productivity
Assembly
History
Annotation
Search
Blast
Administrator
Options
Logout
Change language
Jararaca
http://citosina.biomol.unb.br/jararaca/annotation/se
Google
Apple (80) Amazon eBay Yahoo! News (336)

Contig31 - Annotation

Summary

Assembly 2 - 2008-04-29 18:22

Number of bases: 1485

Number of sequences: 36

Automatic annotation

BlastX over NR
pir||JC1348
hypothetical 18K protein - goldfish mitochondrion
E-value: 3.0E-7
Score: 60.5

BlastX over SwissProt
sp|O62538|HUNB_DROSE
Protein hunchback
E-value: 1.7
Score: 33.9

Manual annotation

Product name:

EC:

Category 1:

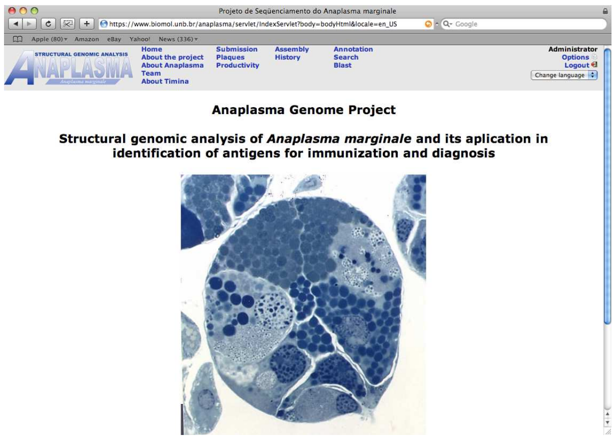
Category 2:

Category 3:

BlastX over NR - Contig31 (Assembly 2 - 2008-04-29 18:22)

```
BLASTX 2.2.15 [Oct-15-2006]


Reference:
Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schäffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs",  Nucleic Acids Res. 25:3389-3402.

Query= Contig31
         (1485 letters)

Database: All non-redundant GenBank CDS
translations+PDB+SwissProt+PIR+PRF excluding environmental samples
from WGS projects
           5,891,190 sequences; 2,030,265,607 total letters

Searching..................................................done



                                                                 Score    E
Sequences producing significant alignments:                      (bits) Value

pir||JC1348  hypothetical 18K protein - goldfish mitochondrion     60   3e-07
emb|CAM36311.1|  hypothetical protein [Thermobia domestica]          47   0.004
ref|NP_377110.1|  hypothetical protein STS132 [Sulfolobus to...     46   0.008
gb|AAO52805.1|  hypothetical protein [Bacillus megaterium]          45   0.013
ref|NP_765819.1|  hypothetical protein SE2264 [Staphylococcu...     45   0.013
ref|ZP_00231457.1|  conserved hypothetical protein [Listeria...     45   0.013
gb|ABK26259.1|  unknown [Picea sitchensis]                          45   0.017
```

Figure 18. Manual and automatic annotation pages in the Jararaca Project, respectively from left to right.


Projeto de Seqüenciamento do Anaplasma marginale
https://www.biomol.unb.br/anaplasma/servlet/IndexServlet?body=bodyHtml&locale=en_US
Google
Apple (80) Amazon eBay Yahoo! News (336)
STRUCTURAL GENOMIC ANALYSIS
ANAPLASMA
Home
About the project
About Anaplasma
Team
About Timina
Submission
Plaques
Productivity
Assembly
History
Annotation
Search
Blast
Administrator
Options
Logout
Change language

Anaplasma Genome Project

Structural genomic analysis of *Anaplasma marginale* and its aplication in identification of antigens for immunization and diagnosis

Figure 19. Layout of the homepage of the Anaplasma Project, as generated by Timina.

As future work we intend to develop: (i) a system that helps biologists to decide which function to infer for a gene — it could use the plug-in interface, also described on this work, (ii) a visual interface to make easier the system configuration, and (iii) a performance analysis comparing the implementation using Servlets with an implementation assisted by a presentation framework. Another possible future work is to adopt the aspect oriented paradigm for the design and implementation of *Timina*. The framework could get benefits from this technique considering that its subsystems have very similar code.

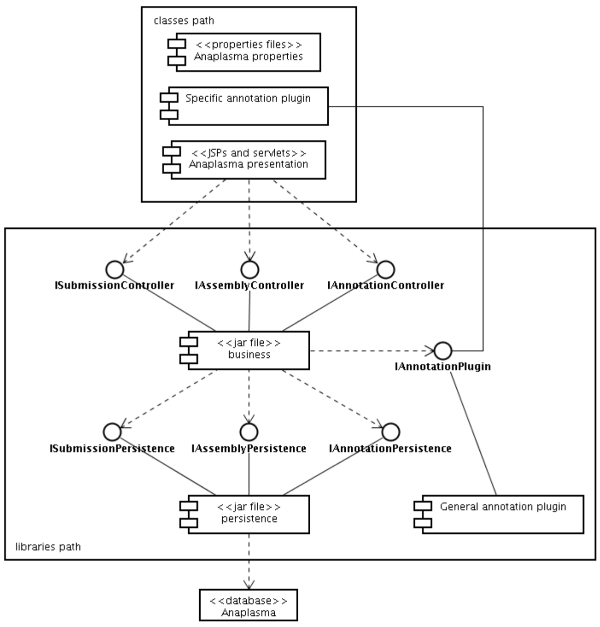


Figure 20. Distribution of the components of Timina in a server for the Anaplasma Project.

References

[1] L. G. P. Almeida, R. Paixão, R. C. Souza, G. C. da Costa, F. J. A. Barrientos, M. T. dos Santos, D. F. de Almeida, and A. T. R. Vasconcelos. A System for Automated Bacterial (genome) Integrated Annotation – SABIA. *Bioinformatics*, 20(16):2832–2833, 2004.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[3] L. C. Bailey, Jr., S. Fischer, J. Schug, J. Crabtree, M. Gibson, and G. C. Overton. The GAIA Software Framework for Genome Annotation. Technical report, Center of Bioinformatics - University od Pennsylvania, USA, 1998.

[4] E. Bartocci, F. Corradini, E. Merelli, and L. Scortichini. BioWMS: a web-based Workflow Management System for bioinformatics. *BMC Bioinformatics*, 8(1), March 2007.

[5] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley Longman, Inc., Reading, Massachussetts, 1999.

[6] Brazilian Amazon Consortium for Genomic Research. Genomic Analysis of *Bothrops atrox*, 5 2008. Available at `http://citosina.biomol.unb.br/jararaca` (accessed in May 2008).

[7] A. M. R. Dávila, D. M. Lorenzini, P. N. Mendes, T. S. Satake, G. R. Sousa, L. M. Campos, C. J. Mazzoni, G. Wagner, P. F. Pires, E. C. Grisard, M. C. R. Cavalcanti, and M. L. M. Campos. GARSA: genomic analysis resources for sequence annotation. *Bioinformatics*, 21(23):4302–4303, 2005.

[8] A. L. Delcher, D. Harmon, S. Kasif, O. White, and S. L. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Research*, 27(23):4636–4641, 1999.

[9] G. H. V. Domselaar, P. Stothard, S. Shrivastava, J. A. Cruz, A. Guo, X. Dong, P. Lu, D. Szafron, R. Greiner, and D. S. Wishart. BASys: a web server for automated bacterial genome annotation. *Nucleic Acids Research*, 33(Web Server issue):W455–W459, 2005.

[10] B. Ewing and P. Green. Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities. *Genome Research*, 8(3):186–194, 1998.

[11] B. Ewing, L. Hillier, M. C. Wendl, and P. Green. Base-Calling of Automated Sequencer Traces Using Phred. I. Accuracy Assessment. *Genome Research*, 8(3):175–185, 1998.

[12] FSF. The Free Software Definition, 7 2006. Available at `http://www.gnu.org/philosophy/` (accessed in Februrary 2006).

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Padrões de Projetos: Soluções reutilizáveis de software orientado a objetos*. Bookman, Porto Alegre, 2000.

[14] P. Green. Phred, Phrap, Consed, 2 2006. Available at `http://www.phrap.org/phredphrapconsed.html` (accessed in Februrary 2006).

[15] N. L. Harris. Genotator: A Workbench for Sequence Annotation. *Genome Research*, (7):754–762, 1997.

[16] X. Huang and A. Madan. CAP3: A DNA Sequence Assembly Program. *Genome Research*, 9(9):868–877, 1999.

[17] Midwest Network on Bioinformatics. Genomic Analysis of *Anaplasma marginale*, 5 2008. Available at `https://www.biomol.unb.br/anaplasma` (accessed in May 2008).

[18] NCBI. FASTA format description, 2 2006. Available at `http://www.ncbi.nlm.nih.gov/blast/fasta.shtml` (accessed in Februrary 2006).

[19] G. C. Overton, C. Bailey, J. Crabtree, M. Gibson, S. Fischer, and J. Schug. GAIA: Framework Annotation of Genomic Sequence. *GENOME RESEARCH*, (8):234–250, 1998. Center for Bioinformatic – University of Pennsylvania.

[20] W. R. Pearson and D. J. Lipman. Improved Tools for Biological Sequence Comparison. *Proceedings of the National Academy of Sciences of the USA*, 85(8):2444–2448, 1988.

[21] M. Pocock, T. Down, and T. Hubbard. BioJava: open source components for bioinformatics. *SIGBIO Newsletter*, 20(2):10–12, 2000.

[22] S. L. Salzberg, A. L. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated Markov models. *Nucleic Acids Research*, 26(2):544–548, 1998.

[23] R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, E. V. Koonin, D. M. Krylov, R. Mazumder, S. L. Mekhedov, A. N. Nikolskaya, B. S. Rao, S. Smirnov, A. V. Sverdlov, S. Vasudevan, Y. I. Wolf, J. J. Yin, and D. A. Natale. The COG database: an updated version includes eukaryotes. *BMC Bioinformatics*, 4(1):41, 2003.

[24] TIGR. Manatee, 2 2006. Available at `http://manatee.sourceforge.net` (accessed in Februrary 2006).

[25] W3C. Web Content Accessibility Guidelines 1.0, 2 2006. Available at `http://www.w3.org/TR/WAI-WEBCONTENT/` (accessed in Februrary 2006).

[26] R. Waterston and J. Sulstont. The genome of Caenorhabditis elegans. *Proc. Natl. Acad. Sci.*, (11):10836–10840, 1995.