Rbbt: A Framework for Fast Bioinformatics Development with Ruby

Miguel Vazquez, Rubén Nogales, Pedro Carmona, Alberto Pascual, and Juan Pavón

Abstract In a fast evolving field like molecular biology, which produces great amounts of data at an ever increasing pace, it becomes fundamental the development of analysis applications that can keep up with that pace. The Rbbt development framework intends to support the development of complex functionality with strong data processing dependencies, as reusable components, and serving them through a simple and consistent API. This way, the framework promotes reuse and accessibility, and complements other solutions like classic APIs and function libraries or web services. The Rbbt framework currently provides a wide range of functionality from text mining to microarray meta-analysis. The source code for the framework, as well as for several applications that use it, can be accessed at `http://github.com/mikisvaz`.

1 Background

Molecular biology produces data in many different fields, and in great amounts. There is a great potential for developing new analysis methods and applications by either merging data from different sources, or by taking advantage of the large scale of several data repositories. These applications are, however, hindered by the complications in developing such systems, as they may require complex data processing pipelines just to set up the application environment. The fact is that many application that implement similar or closely related functionality end up having to roll up their own processing back-end pipelines, which are often very similar to one another and

Miguel Vazquez · Juan Pavón
Dep. Ingeniería del Software e Inteligencia Artificial, Universidad Complutense Madrid

Rubén Nogales · Pedro Carmona
Dep. Arquitectura de Computadores y Automática, Universidad Complutense Madrid

Alberto Pascual
Biocomputing Unit, National Center for Biotechnology-CSIC

may actually account for a great portion of the development time. It would increase the turnout on these applications if these processing pipelines where implemented in a clean and structured way such that they could be shared and reused between different applications.

In fact, this approach would not only allow for sharing just these pipelines, but would also allow for more sophisticated functionality to be exported as APIs, thanks to the possibility to easily install and process the supporting data. Things like identifier translation, gene mention recognition and normalization, or annotation enrichment analysis are good examples. The problem of providing APIs for these types of functionality, those that are very data dependant, is partially solved by allowing the necessary data files to be shipped along with the API. This solution, however, does not apply well when the data is to large, or is not static, either because it must be updated periodically, or because it must allow for some type of customization on how it is produced.

One solution to this problem is to build our own ad-hoc back-end systems, and export the functionality through the use of web services, such as those following the SOAP and REST communication protocols. This approach has several advantages, in particular, that the complexity of installing and administering the system falls only over the system developers and not over the API users. It does, however, have one important drawback with respect to classic APIs, other than possible reliability or latency problems, and that is that open source API can be modified and adapted to particular needs, while these web services are static, and their behaviour is entirely controlled by the developers and maintainers of the application.

The solution proposed in this work, and implemented by the Rbbt framework, is to develop the back-end processes in such a way that they can be shipped with the API so that they can install, automatically, the complete execution environment on the users system. This way, the users will have local access to the functionality while still maintaining the possibility of altering the code to fit their needs. Furthermore, by encapsulating all the complexity of the system behind a friendly to use top level API interface, the user can include this functionality, otherwise very complex, easily into their own scripts and applications, thus expanding the range of possibilities in their own analysis workflows.

Another important benefit of this approach is that the users will have direct access to the data used to provide the functionality, which can, in turn, allow for developing other functionality over this same data. Actually, by having access to the actual processing pipelines, these themselves can be adapted to process the data into any other format that suits better to the needs of the user.

By following a set of design practices, the process of implementing new analysis applications may leave behind a collection of such high-level functionality that can then be reused in other applications. In this fashion, the SENT and MARQ applications [10, 11] ended up producing the functionality in Rbbt, which where then expanded by including the functionality in Genecodis [6]. These functionality can now be used to develop new applications very fast, as much of the heavy lifting has already been done.

2 Implementation

Since one of the main objectives was to offer a cohesive top level API that provided access to all the functionality, we chose the Ruby language to implement that API. Ruby is a versatile language that can be used for small processing scripts, very much like Perl, but can also be used for web development using state-of-the-art frameworks such as the popular Ruby-on-Rails. This way, the usefulness of the API is maximized, since it can be used over the complete application stack, from small maintenance scripts to large web sites.

Ruby can also interface fairly easily with other tools such as the R environment or Java APIs. This allows the Rbbt API to wrap around these tools and offer their services through the main API, thus helping reduce the complexity of the user code by hiding it behind the API and data processing tools, which can take care of compiling and installing third party software.

Rbbt is designed to serve functionality that otherwise could not be included in classic API due to their strong data dependence. This means that the API must be able to find this data locally on the system, which in turn, requires a configuration tool that can take care of producing that data and installing it in the appropriate locations. We will refer to such locations as local data stores, and may include anything from data files, to third party software, or sandboxes with training and evaluation data to train models for machine learning methods. In fact, a few additional benefits arise by coupling the API functionality with these data stores, for instance, the possibility of implementing transparent caching strategies into the API.

The characteristics of data processing pipelines have similarities to compiling a software program, where certain files are produced by performing particular actions over their dependencies. So, for example, to produce a classifier for biomedical articles, one must build the model from the word vectors for positive and negative class articles, these word vectors are derived in turn from the text of these articles, which are themselves produced by downloading from PubMed the articles corresponding to the PubMed identifiers that are listed as positive and negative classes. Changing the way we compute the word vectors, for example, using a different statistic for feature selection, should only require updating the files downstream. The classic Make tool from UNIX offers a simple and clear way to define these dependencies and production steps. One drawback of Make is its rigid syntax; for this reason in Rbbt we use Rake, a Ruby clone of Make, that implements the same idea, but in 100% Ruby code, so that it can directly harness the full potential of the language, including, of course, the Rbbt libraries.

Data files in the local data stores are preferably saved in tab separated format, which is easy to parse, examine and modify using standard text editors. Only when performance, or some other factor, becomes an issue a database or a key value store should be used. This helps to maintain the interpretability and accessibility of the data.

3 Features

The Rbbt package includes a collection of **core functionality** to help accessing online data, and managing the local data stores. These include access to online repositories such as PubMed, Entrez, BioMart, The Gene Ontology, etc. Some of the resources accessed through the Rbbt API, such as PubMed articles, or gene records from Entrez, are cached to increase speed and reduce load on the remote servers. Access to general online resources which are likely to be updated periodically is also cached, but in a different type of non-persistent cache so that can be purged easily when performing system updates.

For managing the data stores, Rbbt includes functionality to work with tab-separated data files. These files often have an identifier in the first column and a list of properties as the successive columns. The Rbbt API includes functions to load this kind of files as a hash of arrays and merge several of such structures by matching arbitrary columns, which is used for example to build the identifier translation files from data in different sources such as biomart and bach download files from organisms specific databases.

Organism specific information. Rbbt uses a configuration file for each supported organism to list all the details for processing the basic information. One of such resources, for example, is the identifier file, which lists of the gene identifiers in some particular format, followed by other equivalent identifiers for that gene in other formats. In the case of yeast, for example, the native format is the SGD identifier, and other supported formats include Entrez Gene, Gene Symbol, or several Affymetrix probe id formats. These identifiers are retrieve from batch download files from organisms specific databases merged with informations gathered from BioMart and Entrez gene. Also, the Rbbt lists Gene Ontology associations for each gene, common synonyms for gene names as can appear on the literature, and PubMed articles associated to each gene derived from Entrez GeneRIF or GO association files. This files are the seed for the rest of Rbbt functionality for that organism, from simple identifier translation to automatic processing of GEO datasets, or models for gene mention and normalization, it all can be processed automatically once these files are in place, which is done by just complete these simple configuration files. Rbbt currently support 8 model organisms: *Homo sapiens*, *Mus musculus*, *Rattus norvegicus*, *Saccharomyces cerevisiae*, *Schizosaccharomyces pombe*, *Caenorhabditis elegans*, *Arabidopsis thaliana*, and *Candida albicans*. Other organisms can be easily included by just completing the information in their configurations files.

Text mining functionality in Rbbt includes building bag-of-words representations for documents, strategies for feature selection like TF-IDF or Kullback-Leibler divergence statistics, document classification, and several strategies for named entity recognition. In the last category are a general purpose named entity recognition system using regular expressions, which, coupled with the Polysearch thesaurus can be used to find mentions to diseases, cellular locations, and several other things. Rbbt implements a special purpose named entity recognition system for genes, which is also known as gene mention recognition [9], as well as a normalization [5] engine that can be used to map the mentions into the corresponding genes. Both the gene

mention recognition and the normalization tasks have specific sandboxes inside the data stores with training and evaluation data to build and asses the models. Additionally, Rbbt wraps the Abner and Banner software APIs [7, 4] for gene mention, with the same top level interface, so that they can be used interchangeably with its in-house developed system. Our own gene mention system borrows most of its ideas from Abner and Banner; it is based on conditional random fields [3, 8], trained using CRF++ [2], but has a much simpler and flexible configuration system to define the features.

Microarray Analysis in Rbbt includes automatic processing pipelines that can compile a database of gene expression patterns from most of the thousands of datasets available in GEO [1], as well as other GEO series or any other microarray repository with a little configuration. This data consists of gigabytes of expression data that can be used in meta analysis applications. It also supports automatic translation of probe ids from different platforms into a organism based common reference format that can be used to perform analysis across experiments using different technologies. The API includes functionality to manage this automatic processing pipelines, an R function library in charge of the actual automatic processing, so that the data is easily used from this environment, and a collection of features to perform rank based comparison for content based retrieval of similar signatures.

4 Examples of use

To illustrate the simplicity of using the Rbbt library we will use two example tasks, identifier translation of gene identifiers and gene mention recognition and normalization. For more complete examples check the applications sample directory, or the code for SENT and MARQ, which represent full blown applications using the complete Rbbt API.

In order to have access to this functionality the user must only retrieve the rbbt package, and run the processing pipelines to set up the necessary data and learning models, which is done using the configuration tool packaged in Rbbt.

The first example, in listing 1, is a command-line application that can translate gene identifiers for a given organism into Entrez Gene Ids. The input list of gene ids, which is provided through standard input one line each, may be specified in any of the supported gene id formats, which, for instance, in the case of *H. sapiens* includes more than 34 different formats such as Ensemble Gene Ids, RefSeq, or Affymetrix probes. The whole scripts boils down to loading the translation index for the organism and the selected target format (Entrez Gene Id in this case), and then using that index to resolve all our input ids.

Listing 1 Translate Identifiers

```
require 'rbbt/sources/organism'
require 'rbbt/sources/pubmed'

usage =<<-EOT
```

```
  Usage: #{$0} organism

  organism = Sce, Rno, Mmu, etc. See 'rbbt_config organisms'

  You will need to have the organism installed. Example:
  'rbbt_config prepare organism -o Sce'.

  This scripts reads the identifiers from STDIN.

  Example:
  cat yeast_identifiers.txt | #{$0} Sce
EOT

organism = ARGV[0]

if organism.nil?
  puts usage
  exit
end

index =
  Organism.id_index(organism, :native => 'Entrez Gene Id')
STDIN.each_line{|l| puts "#{l.chomp} => #{index[l.chomp]}"}
```

The second example, in listing 2 (only relevant part is shown), is a command-line application that, given an organism, and a query string, performs the query in PubMed and finds mentions to genes in the abstracts of the matching articles. In this case the task boils down to loading the gene mention engine (*ner*), the normalization engine (*norm*), performing the query to find the list of articles (*pmid*), and then going through the articles performing the following two steps: use the gene mention engine to find potential gene mentions in the articles text, which is basically the title and abstract of the article, and then using the normalization engine to resolve the mentions into actual gene identifiers, using the article text to disambiguate between ties. The gene mention engine has several possible back-ends, Abner, Banner, and our own in-house development RNER, all based on conditional random fields. If RNER is used, the model for that specific organism must be trained, a process that, while been rather lengthy, is completely automated using the configuration tool, while still having room for customization, and needs only to be performed once. The normalization engine needs no training, and also has ample room for configuration.

Listing 2 Gene Mention Recognition and Normalization

```
# Load data, this can take a few seconds
ner  = Organism.ner(organism, :rner) # Use RNER
norm = Organism.norm(organism)

# Query PubMed. Take only the last 'max' articles
pmids = PubMed.query(query, max.to_i)

# For each article:
PubMed.get_article(pmids).each{|pmid, article|
```

```
  # 1. Find mentions
  mentions = ner.extract(article.text)

  # 2. Normalize them
  codes = {}
  mentions.each{|mention|
    codes[mention] = norm.resolve(mention, article.text)
  }

  # 3. Print results
  puts pmid
  puts article.text
  puts "Mentions: "
  codes.each{|mention, list|
    puts "#{ mention } => #{list.join(", ")}"
  }
  puts
}
```

5 Conclusions

Open source APIs offer an inestimable resource for software developers. However, due to the complexity of bioinformatics analysis, this sharing was unfeasible for many functionality due to the strong dependence in elaborate data processing pipelines required to set up the applications environment. The use of web services opens the possibility of offering an API over this functionality while hiding these complex processes from the API user. However, web servers may suffer from unreliability or latency, and have much less room for adapting them to different circumstances. Rbbt has shown that the approach of constructing the processing pipelines themselves so that they can set up the data stores automatically not only allows to provide these same API locally, but has a number of additional benefits, in particular, more options in terms of modifying and adapting the code, reuse of the data files, and reusing the processing pipelines for other tasks.

Rbbt has been successfully used, in whole and in part, in developing several production ready applications. These applications where developed very fast with a limited development group; Rbbt and agile development practices played a fundamental role in the their fast turnout.

The source code for the framework, as well as for several applications that use it, can be accessed at `http://github.com/mikisvaz`.

Acknowledgments

We acknowledge support from the project *Agent-based Modelling and Simulation of Complex Social Systems (SiCoSSys)*, which is financed by Spanish Council for Science and Innovation, with grant TIN2008-06464-C03-01. We also thank the support from the *Programa de Creación y Consolidación de Grupos de Investigación UCM-BSCH, GR58/08*

References

1. Barrett, T., Suzek, T., Troup, D., Wilhite, S., Ngau, W., Ledoux, P., Rudnev, D., Lash, A., Fujibuchi, W., Edgar, R.: NCBI GEO: mining millions of expression profiles–database and tools. Nucleic acids research **33**(Database Issue), D562 (2005)
2. Kudo, T.: Crf++: Yet another crf toolkit. (2005). URL `http://chasen.org/taku/software`
3. Lafferty, J., McCallum, A., Pereira, F.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. Proceedings of the Eighteenth International Conference on Machine Learning table of contents pp. 282–289 (2001)
4. Leaman, R., Gonzalez, G.: Banner: An executable survey of advances in biomedical named entity recognition. In: Pacific Symposium on Biocomputing, vol. 13, pp. 652–663. Citeseer (2008)
5. Morgan, A., Lu, Z., Wang, X., Cohen, A., Fluck, J., Ruch, P., Divoli, A., Fundel, K., Leaman, R., Hakenberg, J., et al.: Overview of BioCreative II gene normalization. Genome Biology **9**(Suppl 2), S3 (2008)
6. Nogales-Cadenas, R., Carmona-Saez, P., Vazquez, M., Vicente, C., Yang, X., Tirado, F., Carazo, J., Pascual-Montano, A.: GeneCodis: interpreting gene lists through enrichment analysis and integration of diverse biological information. Nucleic Acids Research **37**(Web Server issue), W317–W322 (2009). URL `http://genecodis.dacya.ucm.es`
7. Settles, B.: ABNER: an open source tool for automatically tagging genes, proteins and other entity names in text. Bioinformatics **21**(14), 3191 (2005)
8. Settles, B., Collier, N., Ruch, P., Nazarenko, A.: Biomedical Named Entity Recognition using Conditional Random Fields and Rich Feature Sets. COLING 2004 International Joint workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP) 2004 pp. 107–110 (2004)
9. Smith, L., Tanabe, L., Ando, R., Kuo, C., Chung, I., Hsu, C., Lin, Y., Klinger, R., Friedrich, C., Ganchev, K., et al.: Overview of BioCreative II gene mention recognition. Genome Biology **9**(Suppl 2), S2 (2008)
10. Vazquez, M., Carmona-Saez, P., Nogales-Cadenas, R., Chagoyen, M., Tirado, F., Carazo, J., Pascual-Montano, A.: SENT: semantic features in text. Nucleic Acids Research **37**(Web Server issue), W153–W159 (2009). URL `http://sent.dacya.ucm.es`
11. Vazquez, M., Nogales-Cadenas, R., Arroyo, J., Botías, P., García, R., Carazo, J., Tirado, F., Pascual-Montano, A., Carmona-Saez, P.: MARQ: an online tool to mine GEO for experiments with similar or opposite gene expression signatures URL `http://marq.dacya.ucm.es`. Under submission