BIOINFORMATICS APPLICATIONS NOTE Vol. 27 no. 17 2011, pages 2431–2432
doi:10.1093/bioinformatics/btr411

Genome analysis Advance Access publication July 19, 2011

# Simple high-throughput annotation pipeline (SHAP)

Matthew Z. DeMaere[1,*], Federico M. Lauro[1], Torsten Thomas[1,2], Sheree Yau[1] and Ricardo Cavicchioli[1,*]

[1]School of Biotechnology and Biomolecular Sciences and [2]Centre for Marine Bio-Innovation, The University of New South Wales, Sydney, NSW 2052, Australia

Associate Editor: Alex Bateman

**ABSTRACT**

**Summary:** SHAP (simple high-throughput annotation pipeline) is a lightweight and scalable sequence annotation pipeline capable of supporting research efforts that generate or utilize large volumes of DNA sequence data. The software provides Grid capable analysis, relational storage and Web-based full-text searching of annotation results. Implemented in Java, SHAP recognizes the limited resources of many smaller research groups.

**Availability:** Source code is freely available under GPLv3 at https://sourceforge.net/projects/shap.

**Contact:** matt.demaere@unsw.edu.au; r.cavicchioli@unsw.edu.au

Received on March 17, 2011; revised on 3 July, 2011; accepted on 7 July, 2011

## 1 INTRODUCTION

The development of SHAP began in early 2006, after it became apparent that the needs of our environmental genomics (metagenomics) program would not be satisfied by available annotation tools of the day. Automated pipelines were focused primarily on the analysis of single microbial genomes (Almeida *et al*., 2004; Meyer *et al*., 2003; http://manatee.sourceforge.net) and often pursued analysis strategies that became impractical when the volume of sequencing data grew substantially beyond their original design scope. SHAP was designed around five primary requirements: relational storage, scalable high-throughput analysis, a simple extensible core, platform independence and free availability; requirements that were not met by any program evaluated at the time.

Recently, Ergatis (Orvis *et al*., 2010) and the derived annotation-specific server ISGA (Hemmerich *et al*., 2010) have become available. Streamlining the capacity of Ergatis, ISGA addresses many of the issues that motivated the creation of SHAP. However, the ambitious nature of the system makes it complex and demanding to deploy as a monolithic application. As the name implies, SHAP provides a relatively simple means of annotating high-throughput DNA sequencing datasets while, at the same time, allowing for customization and expansion.

## 2 DESIGN AND IMPLEMENTATION

SHAP is implemented entirely in Java, utilizing Spring (http://www.springsource.org) as an application framework, Hibernate (http://www.hibernate.org) for relational storage and search integration and Apache Lucene (http://lucene.apache.org) for full-text indexing and search capabilities (Fig. 1). The design of SHAP pays attention to the rapid and continued development of metagenomics by avoiding implementation of complex or fine-scale features. A layered separation of concerns allowed the isolation of domains with the greatest potential for change. Driven by developments in methodology and platform selection, if left unconsidered this change would represent a significant cost in ongoing codebase maintenance. The selection and behaviour of the underlying analysis tools (BLASTALL, HMMER, etc.) was identified as one such area and therefore their execution detail and subsequent result parsing was isolated within a single layer.

By way of Inversion of Control, much of SHAP's instantiation logic of collaborating objects is XML configuration detail, providing flexibility in differing post-deployment environments without the need for coding changes. By introducing an ORM (Hibernate), the persistence layer is only loosely coupled to the underlying relational database provider. Full-text search facilities (Hibernate Search/Apache Lucene) simplifies user-driven retrieval of results for downstream analysis. High-throughput analysis is achieved by supporting both conventional multi-threaded local execution and Grid computing by way of DRMAA.

SHAP's minimalist persistent domain model begins with a Project which contains one or more Samples. Following this template, biological data is organized hierarchically via cascading one-to-many associations; where instances of Project and Sample are user defined; instances of Sequence are defined by imported sequencing data; and Feature and Annotation instances are the derived result of computational analyses. The database schema emitted by the Hibernate mapped object model parallels a subset of BioSQL (http://www.biosql.org), with the notable exception that, for the sake of simplicity, SHAP makes only light use of ontological terms and adopts no official ontology (Eilbeck *et al*., 2005).

A computational analysis is represented by a Job that is parcelled into computationally independent Tasks. Job and Task store historical runtime metadata as well as execution state, permitting work to be stopped, restarted and reattempted. Jobs are divided into two classes: detection, which determine Features from Sequences, and Annotation, which determine Annotations from Features. Within each class, the workload is embarrassingly parallel, allowing for concurrent execution of all outstanding work at the task level.

SimpleDetector and SimpleAnnotator are primary objects supporting analysis within SHAP, designed for simplicity and reusability with underlying analysis tools by imposing a basic contract. The contract stipulates that an analyzer acts on a specific target object type (Sequence for SimpleDetector, Feature for SimpleAnnotator) and produces a specific result object type

*To whom correspondence should be addressed.

(Feature and Annotation respectively), and that the underlying tool accept FASTA formatted sequence as input and produce results as a file. Analysis then becomes a generic process with the exception of result parsing.

Result parsing is delegated to implementations of the DetectorParser and AnnotatorParser interfaces. The number of implementations required to support a desired workflow is not as high as might be expected. In the case of BLASTALL, one implementation would suffice for all choices of reference database.

Jobs are user defined by a plan. Expressed in XML, a plan defines the targets of analysis and the chosen analysers. Plans are submitted to the pipeline via the command line.

Data import and export can be accomplished on the command line, with output formats being FASTA, Genbank or CSV tables. For more expedient multi-user access, a web application is provided. Deployed to a standard Servlet 2.5 container, the interface provides browse, search and export functions.

Search queries follow current convention where only plain text input of words of interest is necessary to define an effective query. The Lucene query language supports further sophistications, wildcards (?,*), fuzzy searches (~), logical operators (AND, OR), explicit field references (name:Project01) and relevance boost factors (^N). Boost factors permit particular terms to be emphasized, altering the ranking of search results. For example, to boost the relevance of 'rna' over 'dna' while searching for polymerases, a possible query could be 'polymerase rna^2 dna'.

## 3 RESULTS

SHAP does not expect any prerequisite analysis tools be installed or any particular Analyser be defined prior to use, but does provide a set of example definitions. In this way, SHAP can be considered both a framework and a skeletonized pipeline ready to be fleshed-out by the user. Although SHAP was designed to employ an external database provider and web server, both are capable of being embedded within the application itself. This fully self-contained version requires little to no configuration by the end-user and demonstrates the value of SHAP's layered design (Fig. 1). Its simplicity is a major advantage compared with other similar software packages when custom or new tools need to be incorporated in the pipeline.

In comparison, ISGA requires 42 subordinate tools to be installed as part of the installation process, which, although no doubt offering a comprehensive analysis, remains a daunting task.

A naïve SLOC (source lines of code) size metric (Nguyen *et al*., 2007) applied to the core codebases of SHAP and ISGA finds that ISGA comprises 26 867 physical and 17 305 logical lines of Perl, whereas SHAP comprises 16 395 and 10 683 lines of Java, respectively. Ignoring a possible further normalization to account for the relatively higher level nature of PERL to Java (6:2.5) (Prechelt, 2000), there remains a 38% reduction in codebase size. This comparative reduction could be greater still if ISGA's dependence on the larger Ergatis framework (73 227 physical LOC) for workflow support was included in the metric. However, the SHAP and Ergatis frameworks should not be compared as simplicity of purpose brings forth a feature disparity particularly in the subtly of process.

From the outset of development, the goal was to build a system that could deal with large volumes of data, scale both in the persistence domain and analysis throughput, not become overly complex or difficult to adapt in a developing field of research and allow migration between platforms. Through a layered design, the areas of greatest change are isolated from the rest of system and thereby insulated from their changing habits. SHAP's Web interface provides simple and expedient access to the annotation results, presenting users with a familiar means of access.

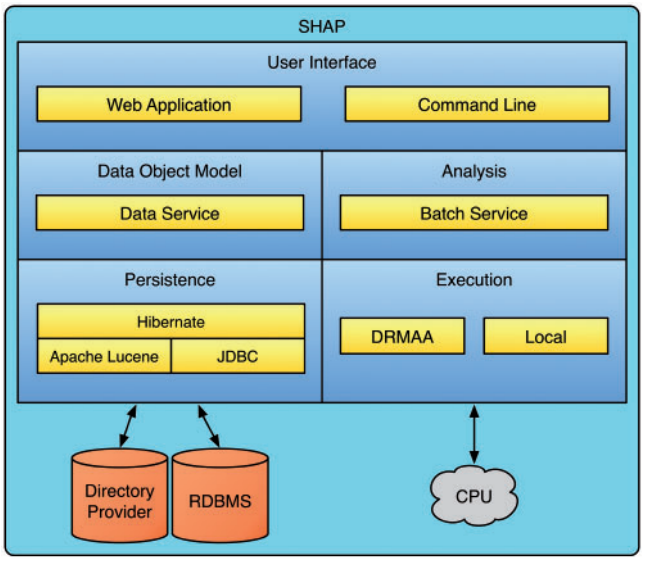


**Fig. 1.** A simplified representation of the application layers within SHAP. The Data Service façade provides abstracted transactional access to persistent storage, retrieving information conventionally by reference or full-text search. The Batch Service façade offers a similar abstraction for job processing, either by local execution or queue submission via DRMAA.

SHAP has been in active use within the group, having supported recent research activities (Ng *et al*., 2010; Lauro *et al*., 2011; Yau *et al.*, 2011). Deployed on our in-house 64 core cluster, the running system has accumulated over 13 years of computational time in 97 days, analysing 2.6 million DNA contigs (2.1 billion bases), producing 4.1 million ORF predictions (578 million amino acids) and recording 7.7 million significant annotation results.

Future development may involve converting SHAP to OSGi dynamic modules to further enhance platform independence and ease of deployment.

*Funding*: Australian Research Council.

*Conflict of Interest:* none declared.

## REFERENCES

Almeida,L.G. *et al*. (2004) A system for automated bacterial (genome) integrated annotation–SABIA. *Bioinformatics*, **20**, 2832–2833.

Eilbeck,K. *et al*. (2005) The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biol.*, **6**, R44.

Hemmerich,C. *et al*. (2010) An Ergatis-based prokaryotic genome annotation web server. *Bioinformatics*, **26**, 1122–1124.

Lauro,F.M. *et al*. (2011) An integrative study of a meromictic lake ecosystem in Antarctica. *ISME J.*, **5**, 879–895.

Meyer,F. *et al*. (2003) GenDB–an open source genome annotation system for prokaryote genomes. *Nucleic Acids Res.*, **31**, 2187–2195.

Ng,C. *et al*. (2010) Metaproteogenomic analysis of a dominant green sulfur bacterium from Ace Lake, Antarctica. *ISME J.*, **4**, 1002–1019.

Nguyen,V. *et al*. (2007) A SLOC counting standard. *COCOMO II Forum*.

Orvis,J. *et al*. (2010) Ergatis: a web interface and scalable software system for bioinformatics workflows. *Bioinformatics*, **26**, 1488–1492.

Prechelt,L. (2000) An empirical comparison of seven programming languages. *IEEE Computer*, **33**, 23–29.

Yau,S. *et al.* (2011) Virophage control of Antarctic algal host–virus dynamics. *Proc. Natl Acad. Sci. USA*, **108**, 6163–6168.