

2004

A Framework for Implementing Bioinformatics Knowledge-Exploration Systems

John A. Hayes

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Bioinformatics Commons](#)

Recommended Citation

Hayes, John A., "A Framework for Implementing Bioinformatics Knowledge-Exploration Systems" (2004). *Dissertations, Theses, and Masters Projects*. Paper 1539626830. <https://dx.doi.org/doi:10.21220/s2-m9rs-dv47>

This Thesis is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

A FRAMEWORK FOR IMPLEMENTING BIOINFORMATICS
KNOWLEDGE-EXPLORATION SYSTEMS

A Thesis

Presented to

The Faculty of the Department of Applied Sciences

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Master of Science

by

John A. Hayes

2004

APPROVAL SHEET

This thesis is submitted in partial fulfillment of

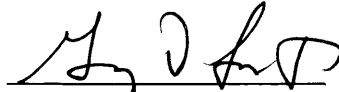
the requirements for the degree of

Master of Science

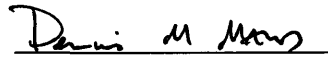


John Ashley Hayes

Approved by the Committee, April 2004



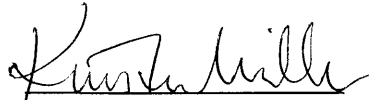
Gregory Smith, Chair



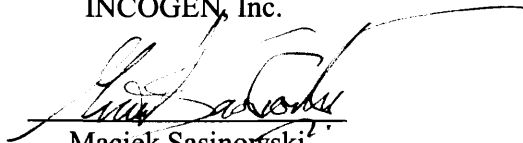
Dennis Manos



Michael Trosset



Krista Miller
INCOGEN, Inc.



Maciek Sasinowski
INCOGEN, Inc.

TABLE OF CONTENTS

	Page
Acknowledgements	viii
List of Tables	x
List of Figures	xi
Abstract	xiii
Introduction	2
I.1. Background	2
I.2. Objectives	4
I.2.1. Web-accessible Pipeline Access and Framework Implementation	5
I.2.2. Knowledge-Exploration	6
I.3. Organization of Thesis	6
Chapter 1: Review of Literature	9
1.1. Bioinformatics Knowledge-Exploration Systems	9
1.1.1. Related Approaches	13
1.1.2. Knowledge-Based Systems	15
Case-Based Reasoning	15
1.2. Case-Related Literature	16
Chapter 2: Framework Structure	17
2.1. Development of Infrastructure	17
2.1.1. User-Interface Elements (UI)	18
Pipeline Visualization	20
Schematic View (SVG)	20
2.1.2. Jakarta Struts and VIBE Web-Access	20
DisplayCaseAction.java	21

ExecutionCaseAction.java	22
ResultsAction.java	22
RecipeSummaryAction.java	22
WikiAction.java	23
GenericCaseForm.java	23
2.1.3. PipelineExecutionHandler.java	24
2.1.4. PipelineService.java	24
2.1.5. PipelineWebViewHandler.java	24
Case-Specific Output	25
Schematic View	25
VPML (VIBE Pipeline Markup Language)	25
Archive	25
2.1.6. Module Services	26
2.1.7. Recipes	26
DefaultRecipeImpl	27
Deploying a Recipe	29
2.1.8. Recipe Support	29
RecipeUtils	29
ModuleFactory	29
ModulePlacers	30
2.1.9. System Configuration	30
Determining the current state of the server	30
Specifying the location for Modules to be Remotely Executed	31
Modifying the layout of the case hierarchy	31
Editing the genericCaseFormToPipeline.jelly file	33

Editing how reports are generated by default	36
Additional System Configuration	37
Automatic Recipe Generation from Exported VIBE Pipeline	37
Chapter 3: Case Implementations	38
3.1. Case Requirements; Case Implementations	38
3.2. Structural Information	39
3.2.1. Nucleotide Information	39
What general information is known from a Nucleotide sequence?	39
Transcription Factors: Find possible Transcription Factor binding sites within the Nucleotide sequence.	40
Genes: Can genes within this sequence be found in silico?	40
Genes: Does my sequence contain a known curated gene?	40
Hidden Markov Models (HMM): Perform an HMM search using nucleotide data to construct our HMM	40
MSA: Generate a Multiple Sequence Alignment from a series of input sequences	41
MSA: Generate a Multiple Sequence Alignment from a series of traces	41
Sequence Manipulation: Convert a nucleotide sequence into another frame	41
Sequence Manipulation: Generate a consensus sequence from a set of nucleotide sequences	41
Sequence Manipulation: Translate a nucleotide sequence into an amino acid sequence	41
Sequence Tagged Sites: Test sequence for presence of STSs.	41
Sequence Tagged Sites: Search the STS database for a nucleotide sequence	42
Trace: Determine the nucleotide sequence from trace information.	42
Trace: Assemble shotgun sequencing data.	42
3.2.2. Amino Acid Information	42
3.2.3. Mass Spectrometry	42

Raw Signal Spectrum: Smoothing: Apply Gaussian smoothing to Spectra	43
Raw Signal Spectrum: Smoothing: Apply Moving Average smoothing to Spectra	43
Variable: Selection/Dimension Reduction: Determine Principal Components	43
Variable: Selection/Dimension Reduction: Calculate Discriminant Coordinates	43
Classification & Cross-validation: Determine Leave-1-Out Cross-validation Error	43
Classification & Cross-validation: Determine Random Permutation Cross-validation Error	44
Classification & Cross-validation: Measurement-Out Cross-validation Errors	44
3.2.4. Organism Information	44
Homology: What organisms have a sequence with a strong likelihood of homology?	44
3.2.5. Statistical	44
Matrix: Generate a biplot from an input data matrix	45
Matrix: Generate a screeplot from an input data matrix	45
Dissimilarity: Determine a 2D representation for an input dissimilarity matrix	45
Clustering: Perform agglomerative clustering on an input dissimilarity matrix	45
Correlation: Display correlation matrix in one dimension	45
Correlation: Display correlation matrix in one dimension and separated by groups	46
Chapter 4: A Tour of the System	47
4.1. End-User Tutorial Introduction	47
4.1.1. Example 1: Translate a nucleotide sequence into an amino acid sequence.	48
4.1.2. Example 2: What organisms have a sequence with strong likelihood of homology?	53
4.1.3. Example 3: Executing Several Cases Concurrently	58
4.2. Developer Tutorial Introduction	59
4.2.1. Part 1: Implementing a simple case	59
Statically-defined Analysis Pipelines	60

Dynamically-defined Analysis Pipelines	63
4.2.2. Part 2: Allowing users to modify parameters from the minimal input view	65
4.2.3. Part 3: More Advanced Case Implementations	68
Preparing the Input Form	69
Dynamically generate the Analysis Pipeline	69
Modify the Input View	71
4.2.4. Part 4: Post-Processing of Analysis Pipelines	72
Changing the Output View	73
Using the Element Construction Set (ECS)	74
4.2.5. Part 5: Implementing a Cookbook	76
Chapter 5: Conclusions	81
5.1. Framework Conclusions	81
5.2. Case Conclusions	83
5.3. Future Work	83
5.3.1. Framework Improvements	83
5.3.2. Improvements involving the VIBE Client	85
5.3.3. Case Improvements	86
Appendix A. Case Hierarchy	88
Appendix B. Example Input.jsp	90
Appendix C. Selected Javadocs	94
Bibliography	113
Vita	117

ACKNOWLEDGEMENTS

Many thanks to Dr. Gregory Smith, Dr. Maciek Sasinowski, and Krista Miller for all their useful comments on the project and this paper.

I appreciate the feedback received from Kristina Gleason regarding the TESS module as well as general comments on the system.

I'd like to acknowledge Dr. Susan Martino-Catt of Virginia Tech for her helpful suggestions on how researchers may like to interact with the interface.

Thanks to Dr. Joan Burnside at the University of Delaware for her feedback on the best way to implement several of the selected cases.

Thanks to Marco Huertas for demonstrating to me how to use xmgrace (Grace).

Thanks to Dr. Michael Trosset for providing instructions on how to write several of the R scripts utilized in the Statistical cases.

Many of the Matlab scripts were developed in large part by Dr. Dasha Malyarenko, and I appreciate her letting me add cases that incorporate them as part of this research.

Thanks to Tom Crockett at the College of William and Mary for discussions related to the integration of the system onto their Sciclone server.

I greatly appreciate the suggestions that Clay Campbell provided for me regarding the paper.

I'd like to also thank Jason Miller for providing several of the icons used in this project as well as helping me solve some network issues.

I'd also like to thank Mark Levitt at INCOGEN who provided advice on how to solve several obscure problems I was experiencing.

David Lee Fuquay helped with integrating some of the Matlab-related modules into the VIBE client and deserves many thanks.

Douglas Hawkins also gave me advice on how to implement some Javascript as well as hints on how to prevent browser caching.

Thanks to everyone else at INCOGEN, Inc. for their support in the completion of this project, and I'd like to thank the VIBE team in particular.

This project was funded under the NIST ATP/CTRF grant, and I'd like to thank them for their support.

Finally, I'd like to thank the respective Apache projects for their support libraries. Without these available resources, the project could have never been completed in the time allotted. The Apache projects used by VIBE WA include the following: Jakarta Struts, Jakarta Jelly, Jakarta Commons, Batik, Cocoon, Forrest, Jakarta ECS, Jakarta Tomcat, Xerces, and Xalan.

LIST OF TABLES

Table	Page
1. Document conventions.	7
2. List of abbreviations used in this document.	8
3. List of technologies in software infrastructure.	18

LIST OF FIGURES

Figure	Page
1. Example Analysis Pipeline from VIBE.	4
2. Schematic structure of the project.	5
3. The overall layout of the application infrastructure.	17
4. Action-to-Recipe Function Call-Structure.	27
5. Problem Selection.	49
6. Minimal Input for Case.	50
7. Output Type Selection.	51
8. Case-Specific Output View.	52
9. Location of the Homology Case.	53
10. Alternative Links to the Homology Case.	54
11. Representative View of Pipeline for the Homology Case.	55
12. Representative Module Documentation.	55
13. Minimal Input View.	56
14. Advanced Parameter Modification and Actual Flow of Pipeline.	57
15. Construct a pipeline template in the VIBE client.	60
16. Create a New Case form.	61
17. Case Hierarchy with new Case category.	62
18. Minimal Input form.	62
19. Case Implementation Summary.	63
20. Editing the Case Implementation file.	65

21. Updated Case-Specific Results.	73
22. Create a New Cookbook.	76
23. System Configuration Summary.	77
24. Case Hierarchy Descriptor file.	78
25. Appendix B's Input.jsp as it appears through a web browser.	93

ABSTRACT

This project aims to facilitate the discovery of new biological knowledge through the development of a web-accessible interface for answering specific problems by taking advantage of distributed tools. The target audience includes both biology end-users and software developers. End-users are exposed to problems that the system supports and can view dynamically generated reports specific to answering that problem. The system also allows easy addition and modification of problems and their implementations. The implementation of each problem is meant to reduce many of the difficulties of working with disparate data and analysis resources. The framework was applied to several specific biologically-related problems thereby showing the effectiveness of the system and ease with which new problems could be added. The system may also be used as a learning tool to demonstrate how problems are solved using available resources.

A FRAMEWORK FOR IMPLEMENTING BIOINFORMATICS
KNOWLEDGE-EXPLORATION SYSTEMS

INTRODUCTION

For the last several years the emerging field of bioinformatics has improved the scientific community's access to biological information and the ability to transform this information into usable knowledge about biological organisms. Unfortunately, much of the information that is of interest to biological researchers is not easily accessible. The National Institute of Standards and Technology Advanced Technology Program (NIST ATP) [1] project at INCOGEN [2] is aimed at solving this problem by allowing the simple integration of distributed analysis tools. The goal of this research project is to extend the NIST ATP development and implement a web-accessible interface for researchers to quickly answer particular biological questions.

I.1. Background

A data analysis pipeline consists of a series of steps where each step represents a particular data analysis operation. Researchers frequently encounter what is termed an "analysis pipeline" when they find themselves following a similar order of operations when handed a set of data. Defining the steps they follow in a given situation and the criteria used to determine the flow of the pipeline can be a difficult problem. However, if these steps can be clearly identified and well-defined then the data analysis pipeline has the potential for automation.

The Visual Integrated Bioinformatics Environment (VIBE) [3] developed at

INCOGEN allows users to graphically create data analysis pipelines and execute them. These pipelines are constructed from various modules, each of which performs a specific operation on the data. Currently, VIBE's primary focus is biological sequence analysis. For example, in the data analysis pipeline shown in Figure 1, a `blastn` module runs a Basic Local Alignment Search Tool (BLAST) [4] similarity search using nucleotide sequences against various nucleotide target databases. The high-scoring pairs produced by BLAST are passed on to a Multiple Sequence Alignment (MSA) algorithm such as ClustalW [5]. The aligned sequences that are produced by ClustalW are then used to construct a Hidden Markov Model (HMM) using the Hmmer package [6]. The HMM may then be used as an input into another program of the Hmmer package called 'hmmsearch'. This program will use the HMM to perform a more sensitive search through a sequence database. The user's motivation for constructing this data analysis pipeline may be to determine a biologically relevant similarity that could indicate sequence homology or common ancestry. Without the HMMSearch results, a significant match may have gone undetected with the less sensitive `blastn` search.

VIBE allows users to graphically construct pipelines that may be repeatedly executed; without such a tool the user would have to write Perl [7] scripts or cut and paste among many web pages. The architecture of VIBE is arranged so that the CPU-intensive tasks such as database searches can be off-loaded to a VIBE Server on a physically separate computer. The VIBE client can coordinate with a separate server for each module that is executed.

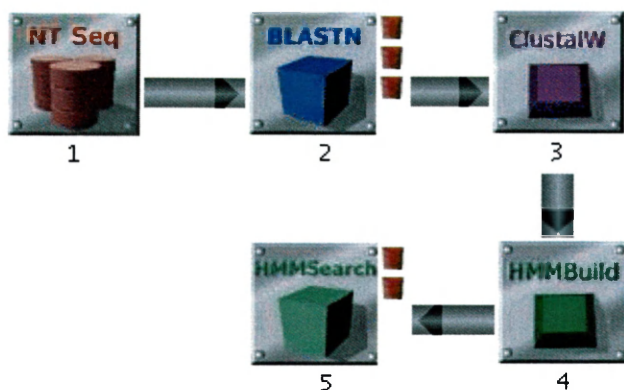


Figure 1. Example Analysis Pipeline from VIBE.

I.2. Objectives

The goal of this project was to facilitate the discovery of new biological knowledge by providing decision-support. This was accomplished through the development of a web-accessible interface that allows researchers to take advantage of distributed tools without requiring that they be familiar with the tools before using them -- a key goal of the NIST ATP project. The code developed under this project both cooperates with the VIBE server and also includes a data-mining portion that provides researchers with an easy mechanism for examining large amounts of information. As part of this project, I consulted with researchers to determine the data analysis pipeline and data mining routines most relevant to their given interests. A schematic depicting the general structure of the project's implementation may be viewed in Figure 2.

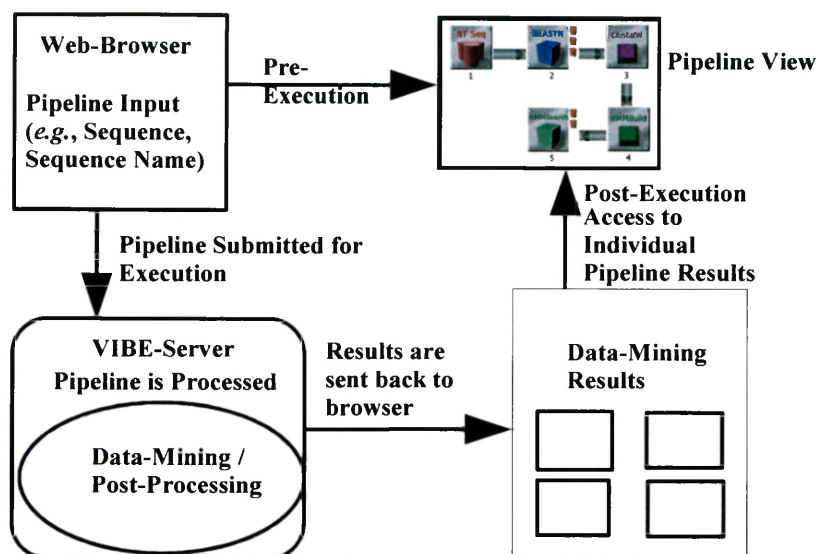


Figure 2. Schematic structure of the project.

The explicit objectives of this project are described in the following sections.

I.2.1. Web-accessible Pipeline Access and Framework Implementation

This objective included the development of a mechanism for executing goal-oriented bioinformatics pipelines from a web-browser. This mechanism makes it easier for researchers to execute instances of a pipeline remotely using pipeline templates. These pipeline templates reside in a repository on the VIBE Server back-end (server-side). It is possible to export a pipeline template from the VIBE client and easily develop new case implementations that answer specific biological questions using that template. In addition to using pipeline templates, the system allows the dynamic generation of pipelines to accommodate multiple input types and their particular analyses. This functionality uses the server-side and effectively eliminates the need for the VIBE client application to execute pipelines. Additionally, this makes the pipelines more portable by making them accessible through any web-browser that can contact the server. This web-

accessible framework will be referred to as VIBE Web Access (VIBE WA).

I.2.2. Knowledge-Exploration

This objective was to develop one or more data-mining methods and implementations that retrieve data from each one of the pipeline modules and correlate information of biological interest. The data-mining module may be thought of as a meta-module that resides outside the pipeline and intelligently filters and orders the results. This was implemented in the framework in the form of case recipes (see Chapter 2). Some specific cases that this project addresses are discussed in Chapter 4 and listed in Appendix A.

I.3. Organization of Thesis

The following chapter describes the literature and resources related to this project and how this project fits into the broader context of bioinformatics knowledge-exploration systems. Chapter 2 elaborates on the development and structure of the web-accessible framework. Chapter 3 discusses the cases that were implemented as part of this project. Chapter 4 gives a brief tour of the system exhibiting many of its capabilities. The final chapter summarizes the conclusions of the project and suggests directions for continued research and development. The document conventions and abbreviations used throughout this paper are indicated in Tables 1 and 2.

Convention	Description	Example
<i>Class names</i>	Source code class names are bold and italics.	<i>RecipeImpl</i>
<i>File names</i>	Files are indicated with the same convention as class names except they will always contain a file extension.	<i>RecipeImpl.py</i>
<i>Method names</i>	Function/method names will be indicated by italics.	<i>generateView</i>
<i>Qualified method names</i>	Function/method names that specify the class they belong to follow the C++ convention.	<i>RecipeImpl::generateView</i>
<i>Package names</i>	Class-package names are indicated by a . delimited series of packages and are set in the class name style. This follows the Java convention.	<i>com.incogen.vibe.server.webaccess.recipes.Recipe</i>
Titles	Case titles and headings are indicated in bold.	Translate a nucleotide sequence into an amino acid sequence.
3rd-Party Tools	3 rd -party tools are indicated in bold.	GeneCards

Table 1. Document conventions.

Abbreviation	Description
AA	Amino acids
API	Application Programming Interface
CBR	Case-based Reasoning
CTRF	Commonwealth Technology Research Fund
HMM	Hidden Markov Model
JAR	Java Archive File
JSP	Java Server Pages
NCBI	National Center for Biotechnology Information
NIST ATP	National Institute of Standards and Technology Advanced Technology Program
NT	Nucleotide
PCR	Polymerase Chain Reaction
SDK	Software Development Kit
SEALS	System for Easy Analysis of Lots of Sequences
SNP	Single nucleotide polymorphism
SOAP	Simple Object Access Protocol
STS	Sequence Tagged Sites
SVG	Scalable Vector Graphics
TSS	Transcription Start Site
VIBE	Visual Integrated Bioinformatics Environment
VIBE WA	VIBE Web Access
VPML	VIBE Pipeline Markup Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

Table 2. List of abbreviations used in this document.

CHAPTER 1

REVIEW OF LITERATURE

1.1. Bioinformatics Knowledge-Exploration Systems

It is becoming increasingly difficult to synthesize useful biological information from large and rapidly growing data repositories [8]. Extraction and integration of useful information is challenging because of the sheer quantity of data, the heterogeneity of data types, and the distribution of resources on various platforms. This chapter will review several different knowledge management and exploration systems and briefly compare them to the VIBE WA project.

GeneCards [9], developed at the Weizmann Institute of Science by Dr. Doron Lancet *et al.*, is an attempt "to develop computational tools that may help to establish an electronic encyclopedia of biological and medical information based on intelligent knowledge navigation technology and a user-friendly presentation of information." The intention of Lancet's group is to give a comprehensive view of biologically relevant information. This approach is different from VIBE WA in that they draw their data from a number of different databases and reconstitute the data into separate "gene cards" that summarize information on a specific gene in their database. A problem with **GeneCards** is the overwhelming amount of information that is presented to the user. This makes it difficult to tell what is directly relevant to an individual's research. VIBE WA addresses

this problem by attempting to return and display only the most relevant information to the given case and pipeline that the user selected.

In many ways, the **Genome Analysis Pipeline** (GAP) [10], designed by Manesh Shah and Edward Uberbacher, is similar to the portion of the VIBE WA project that allows users to create and execute pipelines through the web. The **GAP** is a web-based tool developed by the Computational Biology group at Oak Ridge National Laboratory that allows users to specify various elements to include in an analysis pipeline. These elements include **GrailEXP** [11], **NCBI BLAST** [4], **Genscan** [12], **e-PCR** [13], and several other sequence analysis methods intended to give the user a comprehensive view of the results from these methods. However, **GAP** does not allow much flexibility in adding new analysis types and does not answer particular biological questions. The pipelines used in VIBE WA allow much more flexibility and are expressed as particular problems. In fact, the framework that is developed here could be used to create an interface such as the one provided by **GAP** in a relatively straight-forward manner. Both the **GAP** and VIBE WA are trying to address similar problems by allowing flexibility in the ways the data are analyzed and viewed.

GAP does some other interesting things that VIBE WA handles in a similar manner. If the user provides a nucleotide sequence, it will search for possible genes and take those results and pipe them into **blastp** and **HMMPfam** algorithms which operate on protein data. This is very similar to several of the case implementations for this project. The ability to execute a series of common algorithm-arrangements seems to be a desirable trait for an exploratory system.

An alternative exploratory approach is that used by **GeneLynx** [14], which attempts to be a "portal to a collection of hyperlinks for each human gene." It was originally sponsored by the Pharmacia Corporation who was later acquired by Pfizer, and it was principally authored by Wyeth W. Wasserman and Boris Lenhard. It pulls the information on each gene from a variety of resources and is structured as a front-end for a relational database. Recently, the developers of **GeneLynx** have also added separate links for searching genes within rat and mice genomes. This approach is very different from this project because it attempts to extract and pre-process a considerable amount of data from existing remote/distributed sources and store it in one location. The approach in VIBE WA is more flexible because new data can be added to remote resources without requiring a local update. However, it also is potentially less stable since 3rd-party resources may change day-to-day and are not managed locally.

OmniGene [15] is an Open Source platform originally sponsored by the Whitehead Institute at MIT and lead by Brian Gilman. More recently, it is sponsored by Panther Informatics Inc. **OmniGene** is comprised of several different software frameworks that are functionally similar to the VIBE server in that they allow developers to wrap a command-line program on a potentially remote computer with an interface that allows them to easily access the resource. Currently, the frameworks use a SOAP interface for communicating with the analysis engine (the analog of the VIBE server). The individual services are implemented as Enterprise Java Beans (EJBs) [16] and require an application-server such as **JBoss** [17]. Gilman and others are essentially implementing their own infrastructure similar to the VIBE server architecture. In a sense, this is a lower-level approach than what has been accomplished during this project, and

they could probably implement similar case-logic on top of the lower-level infrastructure that they are building.

Likewise, **Biopipe** [18] is an Open Source framework for building workflows developed by Shawn Hoon and others. These workflows could be used to design and build analysis pipelines similar to VIBE. **Biopipe** appears to rely on a Perl back-end that is not very mature. The most recent release of their software is from May 2003. Additionally, it requires a **PostgreSQL** database to manage jobs. This seems to be a work-in-progress that will ultimately be a lighter-weight **OmniGene**.

The **W3H** task framework [19] is designed to seamlessly provide additional functionality over **W2H**. **W2H** is a free web interface to a large set of sequence analysis tools that is currently maintained by DKFZ [20] and EMBL-EBI [21]. The **W3H** task framework allows users to combine these tools together. It is similar to the VIBE project in that it provides access to independent tools that can be combined in novel ways. Both utilize the concept of meta-data (data about data) to describe how the systems should interact with the tools. Currently in the **W3H** framework, the meta-data must be written in a Perl-like language. This is similar to building an analysis pipeline within the VIBE client or programmatically from the browser (see Chapter 2) with the VIBE system. Like VIBE WA, users typically interact with **W3H** through a web interface. The system also seems to strictly rely on interaction through a web-browser or the command-line. This distinguishes it from the VIBE system which can have a rich client interface.

The approach taken while developing VIBE WA is distinct in that it represents a top-down approach. It is problem-centric rather than tool-centric. It is reminiscent of the

approach taken by the Gene Ontology group's flagship project, **AmiGO** [22], in that it presents the user with a hierarchical tree to navigate the resources. In the case of **AmiGO**, the targets of the navigation are gene ontologies, while in this project they are specific cases that a life science researcher may encounter. Since there are a vast number of potential problems that could be treated, it was critical to emphasize the flexibility and extensibility of the framework.

This project also does not attempt to store all of the data in a data warehouse like **GeneLynx** and **GeneCards**. Rather, it has the potential to adapt to additional tools and resources being added to the system easily and remotely like **OmniGene** and **Biopipe**. This is primarily accomplished by leveraging the VIBE architecture.

1.1.1. Related Approaches

The knowledge discovery object model API for Java (KDOM API) [23] is a software framework for relating biological data developed by Scott Zuyderduyn and Steven Jones at Canada's Michael Smith Genome Sciences Centre. The KDOM API allows developers to easily express relationships between different sets of data to build ontologies. The developers of KDOM are also building a repository of KDOM data definitions that will “provide structure for common biological concepts.” This API may be used by additional projects to store these relations.

Newer technologies are being developed that, like VIBE, will facilitate the combination of disparate resources. These include technologies such as **myGrid** [24] and **BioMOBY** [25]. **BioMOBY** is an open source project to develop “an architecture for the discovery and distribution of biological data through web services.” This is similar to the

NIST ATP project's goals. The **myGrid** project objectives are very similar to **BioMOBY**'s but are stated in a slightly different manner. The developers of **myGrid**, lead by Dr. Carole Goble, are attempting to build open-source middle-ware that collaborates over a grid. **myGrid** is comprised of many separate projects including **Taverna** [26], a tool to build workflows analogous to what the VIBE client does for VIBE services, and **FreeFluo** [27], which executes a subset of WSFL [28] and the workflows created using **Taverna**. The latter functionality is similar to the *PipelineExecutionService*'s function in VIBE WA (see Chapter 2). Interactive viewers and modules like the VIBE client provides are not immediately available through **BioMOBY** or **myGrid**. Instead, the focus appears to be on the discovery of new web services.

Another tool that integrates webservices is **ToolBus** from the Virginia Bioinformatics Institute. It provides the ability to add modules that manipulate remote data services, and includes visualization modules. This functionality is very similar to VIBE. However, it differs from the VIBE WA project in that the system requires a separate installation of the **ToolBus** client. It is also tool-centric rather than case-centric.

This concludes our survey of various approaches to manipulating the lower level tools and data that frequently occur in the bioinformatics field. This area of research appears to be moving toward the development of newer techniques that integrate these tools in a cohesive manner to draw knowledge. The next section discusses some higher-level approaches that can potentially incorporate the power of these lower-level components.

1.1.2. Knowledge-Based Systems

The broad term knowledge-based systems refers to an area of software-engineering that encapsulates data-mining and artificial-intelligence. It can apply to decision-flow-visualization (see **Pipeline Visualization**) and the use of statistics in the context of answering particular questions.

Other knowledge based systems, such as expert systems, are designed to provide decision-support to users by the construction of a knowledge base made up of rules [29]. These rules are applied to a given problem, such as medical diagnosis, and used to infer solutions [30]. The construction of such a knowledge base can be a tedious undertaking and requires a considerable amount of time.

Case-Based Reasoning

A more recent area of active artificial intelligence (AI) research is called case-based reasoning (CBR) [31][32]. These are systems that contain a repository of cases and their solutions that form the knowledge base. A user provides the system with the problem and a software-based reasoner determines if there are cases which represent a similar circumstance. The user's criteria may potentially revise the case and provide a solution to the problem based on past solutions. This differs from the traditional expert system in that the knowledge base is made up of example cases that the researchers provide instead of declarative knowledge. Many applications of CBR are being investigated for medical research such as diagnosis and prognosis applications [33]. However, to date there are few applications of CBR within the bioinformatics field.

The VIBE WA system relies on the user to find similar cases to their problem, and therefore the user is the “reasoner” mentioned in the previous paragraph. However, it

makes the addition and modification of cases easy so that a large repository of biological cases may be developed. Therefore, the VIBE WA system forms a solid foundation on which further automated reasoning capabilities can be added, and it also has the added benefit that it is immediately useful to a researcher.

1.2. Case-Related Literature

Each of the cases that were implemented has its own set of related literature. The appropriate citations and rationale for implementing each case are included in their description below. See the appropriate section for each case under Chapter 4 for more details.

CHAPTER 2

FRAMEWORK STRUCTURE

2.1. Development of Infrastructure

The first stage of this project involved developing the system for executing VIBE pipelines from a web-browser. This provides the ability to use the VIBE system without a VIBE client. After that, it was necessary to develop the framework for easing the integration of novel case-specific implementations. This required the development of support libraries to make the job easier for developers of new cases (see Section 2.1.8).

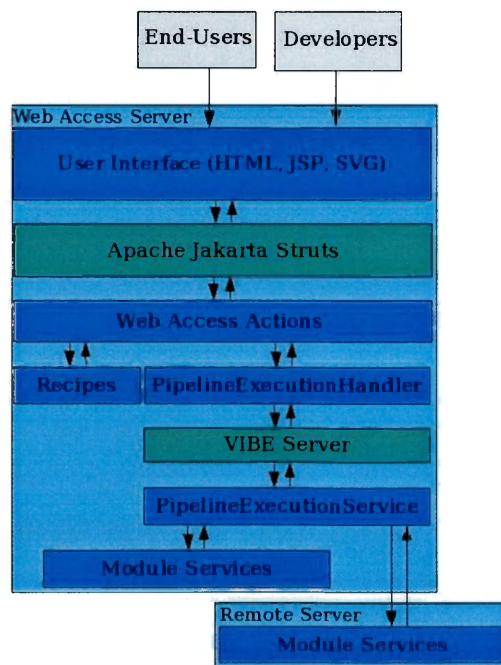


Figure 3. The overall layout of the application infrastructure.

A software-based agent that works on behalf of the user in the form of *Recipe* implementations was created (see Section 2.1.7). This agent will accept input parameters and generate an appropriate VIBE pipeline for the given parameters. This agent also has the ability of generating custom reports that only display the results pertinent to the case.

Technology	Location in the system
Adobe SVG Plug-in	Renders SVG documents in web browser
Batik	SVG rendering library for SVG Viewer in VIBE client
Cocoon	Used by Forrest
ECS	Used to generated well-formed HTML in output reports
Forrest	Used to build basic web interface and some documentation
Java	Most prevalent programming language in system. General-purpose programming.
Javascript	Expands /collapses case hierarchy in browser. Checks for SVG plug-in.
Jelly	Used to map basic input parameters to pipeline
Jetspeed	Provides portal API for cookbook reports
JSP	Used to generate HTML pages
Jython	Used for case implementations
Python	Used for splitting index.html into header & footer during project build
Struts	Framework for developing web applications
SVG	Preview pipelines, advanced views, and graphical output
Tomcat	Servlet container for VIBE server and VIBE WA server
XML	Used throughout VIBE for data & meta-data transmission and storage
XSLT	Preview pipeline generation, algorithms

Table 3. List of technologies in software infrastructure.

This chapter discusses the development of the software infrastructure of VIBE WA. The general structure of the software is indicated in Figure 3 and will be described in more detail below. Table 3 lists the assortment of technologies that are part of the infrastructure and where they were used.

2.1.1. User-Interface Elements (UI)





The target audience of the system are end-users and developers who interact

through the web browser with the user-interface (UI). Currently, this is made up of HTML, SVG, and JSP.

The basic interface of the website was generated using Forrest [34]. Forrest is an XML-backed documentation framework based on the Apache Cocoon [35] project. It was primarily used to statically generate the tutorial pages and index page. It was also used to generate static header and footer HTML pages that are added to the beginning and end of each JSP page. This creates the consistent look and feel over most of the web pages.

Java Server Pages (JSP) [36] is a technology that allows developers to generate web pages dynamically by creating JSP files. JSP files are formed from a markup language similar to XML. It also allows developers to build their own tags that generate HTML based on values of Java objects. For example, this is what the Struts Tag libraries provide.

Several project-specific JSP tags were developed to aid the developers of cases. Of primary interest are the following:

- **<incogen:recipeTitle/>** - Places the contents of the case's description file where this element is located in the JSP file.
- **<incogen:caseLinkBar/>** - Adds links to the example input () , preview pipeline () , help () , and case implementation summary page () for the associated case.
- **<incogen:wikiLink/>** - Adds a link to a Wiki-style edit page for this file.

Pipeline Visualization

Pipeline visualization is comprised of two key components. These are the schematic representation of the pipeline and the ability to modify specific parameters for each module within the pipeline. The schematic view offers the user a quick view of how the pipeline is structured and is also intended to give the user an idea of how to create a similar pipeline within the VIBE client. The parameter modification allows an advanced user to change parameters not available from the minimal case input view. In this case the pipeline is not guaranteed to do what the original case implementor intended the pipeline to do.

Schematic View (SVG)

The schematic views of the pipeline were completed using several different technologies including Scalable Vector Graphics (SVG) [37] and XSLT [38]. SVG is a recent technology recommended by the World Wide Web Consortium (W3C) [39] and developed by many technology-industry leaders. The SVG directed graphs in the preview, advanced case page, and graphical output views are generated using the program *graphviz* [40]. An XSL [41], called *pipelineToDot.xsl*, is used to transform the pipeline's VPML file using an XSLT [38] processor into the syntax for graphviz. A system call is then made that generates the SVG. The SVG plots require that a free browser plug-in from Adobe be installed to handle the rendering of the SVG documents [42].

2.1.2. Jakarta Struts and VIBE Web-Access

The Struts [43] project is an open source framework for developing web applications from Java using the Model-View-Controller (MVC) Model 2 approach. It takes care of some of the more mundane responsibilities of receiving HTTP [44] requests,

setting request variables, and dispatching requests to application-specific classes. It also mediates the response back to the client. This is done by using the Struts-provided *ActionServlet* that runs within a servlet container as the Controller portion of an MVC application. This is what distinguishes Struts as a Model 2 implementation.

The user interface (UI) allows the user to make requests to the server which is facilitated by the Apache Jakarta Struts library. Struts will dispatch the requests to application-specific classes called Actions based on the configuration in the *struts-config.xml* file. This is discussed extensively in the Struts documentation at the project's website [43]. Developers interacting with the Web Access framework should not need to alter this configuration file, but the documentation discussing the Struts JSP tag library is a useful additional reference. The JSP tag libraries makes it easier for developers to interact with Struts from their JSP pages.

The following are the primary Web-Access Actions (these classes all belong to the *com.incogen.vibe.server.webaccess.struts* package) :

DisplayCaseAction.java

This handles the initial interaction of the server with a given case. It has several functions that provide basic functionality common to all cases. These functions are the following:

- **Example** - Loads the basic input view for the case with any example input data. This is accomplished by calling *Recipe::getExampleInputView* on the respective recipe.
- **Help** - Retrieves the case-specific documentation. This is accomplished by calling *Recipe::getDocumentationForward* on the respective recipe. This will typically take the user to the *Help.jsp* file associated with this case.

- **Preview** - Returns an SVG schematic demonstrating the typical flow of the analysis pipeline associated with this case. This will not necessarily be the actual flow (because of changes due to parameter selection). The pipeline that is depicted comes from *Recipe::generatePreviewPipeline*.
- **Display** - Returns the basic input view for this case. This is generated by calling *Recipe::getInputView*. The basic input view is usually the *Input.jsp* file in the case's directory.

ExecutionCaseAction.java

This primarily handles the execution request of a case. It does this by dispatching the browser requests to one of the following functions:

- **Submit** - Submits the case's analysis pipeline to be executed to the locally running VIBE server. This pipeline comes from *Recipe::generatePipeline*.
- **Advanced** - Generates the advanced page for a case. This page is automatically generated by the framework and will list all the modules and their respective parameters. It will also show the actual pipeline that is about to be executed. This pipeline comes from *Recipe::generatePipeline*.

ResultsAction.java

This will take a request including the job ID for a case instance and an output type. The value of the output type indicates which corresponding view will be presented. The output types are currently handled by the *PipelineWebViewHandler* (discussed below).

RecipeSummaryAction.java

The *RecipeSummaryAction* handles operations that are available through the case

implementation summary page. This includes uploading files to a case implementation's directory, cloning of the case, exporting the case, and removing files from the directory.

WikiAction.java

The WikiAction is responsible for allowing developers to edit server-side files. In general, it is primarily used to edit case implementation files. It will load the selected file and initialize it into an editable text area. The developer may make modifications to the contents of the file and submit any changes. Overwritten files are archived into a .zip file with an associated timestamp and may be reloaded if there were problems with the newly submitted file. The file can also be previewed before changes are submitted. This is particularly useful for JSP files. For Jython files, the WikiAction will validate the syntax of the script and report an error message if it detects problems with the submitted file. For example, this will alert the developer of problems with indentation or quotation marks.

GenericCaseForm.java

The *GenericCaseForm* extends the Struts library's *ActionForm* and is used by default for all the cases through VIBE Web Access. *ActionForms* are used by Struts to retrieve and set values that are used in the presentation layer of a web application. For this system, a new instance of *GenericCaseForm* is associated with each case's *Input.jsp* file. This class is used as a mechanism for generically setting and getting variables from the HTML forms. All values stored in forms should fall into two categories: input values and parameter values. These are maintained in a *java.util.Map* so they can be accessed from JSP files like `inputValue(value_key)` and `parameterValue(value_key)`. Appendix B demonstrates this functionality in more detail.

2.1.3. PipelineExecutionHandler.java

The *PipelineExecutionHandler* is responsible for taking the input from the browser and merging it with the selected recipe's pipeline. It will also call the *generatePipeline* method from the selected recipe (see Appendix C) to retrieve the particular pipeline for the case. As the name implies, it handles the execution of the pipeline by sending a *PipelineService* (see below) job to the local VIBE Server. After the completion of the pipeline, it will store the analysis pipeline and its results, along with the associated recipe that was used on the server for later retrieval.

2.1.4. PipelineService.java

This is a VIBE Service implemented using the VIBE SDK's API. It was developed for this project to allow server-side execution of pipelines. It is responsible for taking the pipeline XML file and relaying the particular module jobs to their respective machines. The *PipelineService* executes all modules within the pipeline until they cannot continue. For example, if an upstream module is killed or cannot be executed, its downstream modules cannot be executed and the service returns.

The service takes as input a VPML file containing the pipeline to execute. This, in turn, will contain the series of modules and their respective parameters and data locations. The output of this service is a similar VPML file with any new data locations (*e.g.*, the output of the modules).

2.1.5. PipelineWebViewHandler.java

The *PipelineWebViewHandler* is responsible for returning the appropriate output that the user selects. The researcher currently has the option of selecting four output

types.

Case-Specific Output

With this output type, the final output is handed over to the selected recipe to generate a tailored output for the case. The *PipelineWebViewHandler* will call the *Recipe::generateView* (see Appendix C).

Schematic View

This is an output type common to all cases. It will generate an SVG output view that allows the user to examine the raw results (*e.g.*, XML data) from each module within the pipeline. It will also graphically depict to what extent the pipeline was completed. For example, a green bounded box indicates that the module completed successfully and that results are waiting. A red box indicates that there was a failure in the execution of that particular module. A blue box is an unknown state.

VPML (VIBE Pipeline Markup Language)

This output type will return the raw XML file that contains the structural and parameter information on the case-specific pipeline file that was executed but no output or input data associated with the modules. This may be saved and opened from the VIBE client to make modifications and use visualization modules not available through the web interface.

Archive

This output type returns a .zva file (zipped VIBE archive) that contains a VPML file as well as all the data for the completed modules in the pipeline. This can safely be transferred to a machine that is not accessible to the network that executed the pipeline. Like the VPML output type, this file may be saved and loaded from the VIBE client for

pipeline modification and visualization.

2.1.6. Module Services

The module services shown in Figure 3 are standard VIBE services implemented using the VIBE SDK [45].

2.1.7. Recipes

A recipe is a class that extends *com.incogen.vibe.server.webaccess.recipes.Recipe*. The primary function of a recipe is to implement a particular case. It can be thought of as an agent that works on behalf of the user to manipulate pipelines. This is similar to the role the user would normally fill in using the VIBE client. There is a one-to-one correspondence between a recipe and case.

Recipes allow a programmer to implement specific cases by using the VIBE Server back-end. This allows them to dynamically construct pipelines that may be executed on several distinct machines and format the results to cater to their interests. Recipes are typically written in the Jython [46] programming language. Jython is a Java-based implementation of the Python [47] programming language. The syntax is much simpler than Java and allows greater flexibility. Since it is an interpreted language all changes to the code are immediately available for execution. The greatest benefit in using Jython scripts is that it allows a programmer to update the control-code of a case without having to recompile, redeploy, and restart the server every time there is an update to the code.

Finally, all standard Java libraries and custom Java code may be used within a recipe. Java Native Interface (JNI) code may also be used, so this allows binding other outside languages such as C code through linked libraries.

DefaultRecipeImpl

Any cases that are created through the system's **Create a New Case** interface extend `com.incogen.vibe.server.webaccess.recipes.DefaultRecipeImpl` and will be called *RecipeImpl.py*. This filename indicates that the code represents a class that implements a complete set of *Recipe* functions. Developers should typically extend this class when implementing their own cases.

Figure 4 demonstrates the function call structure relating the most relevant Web-Access Actions to the functions within *Recipe*, *AbstractRecipe*, and *DefaultRecipeImpl*. The arrows indicate that the function at the tail calls the function at the head. Numbers in the vicinity of an arrow indicate that there was more than one function called from the parent function and the number indicates the order in which these children functions are called. All functions within the *Recipe/AbstractRecipe* and *DefaultRecipeImpl* boxes may be overridden from a *RecipeImpl.py* file.

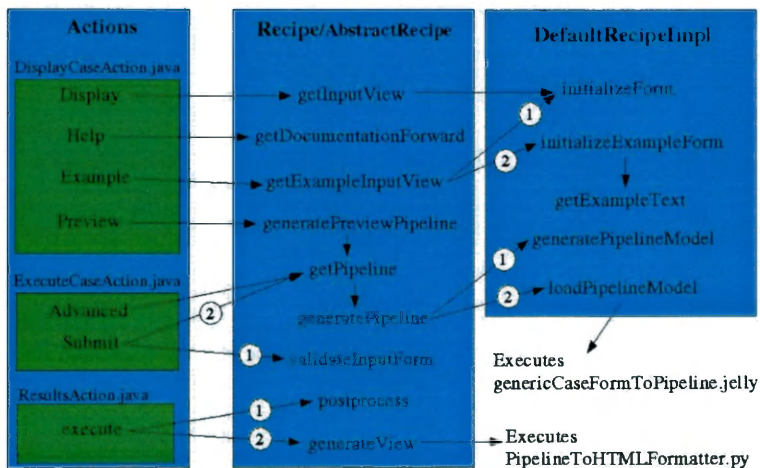


Figure 4. Action-to-Recipe Function Call-Structure.

In particular, developers may want to override the following functions of

DefaultRecipeImpl:

- **initializeForm** - Override this function to initialize the minimal input form for this case with some parameters. It may be desirable to call `DefaultRecipeImpl::generatePipelineModel` and set it to the 'pipeline' class field to determine which parameters are available with a module at runtime.
- **generatePipelineModel** - This is a good place to programmatically define a pipeline if its generation *does not* depend on any input parameters provided by the user. Also, whatever pipeline is implemented for this function will be provided as a preview schematic of the execution.
- **loadPipelineModel** - This is a good place to programmatically define a pipeline if its generation *does* depend on any input parameters provided by the user. The developer may also want to call `DefaultRecipeImpl.loadPipelineModel(self, pipelineModel, genericCaseForm)` at the end of the code for this function to take advantage of any parameter mappings in the Jelly script.
- **generateView** - Override this function to change the report view of the case.
- **postprocess** - This method gives the ***Recipe*** developer the opportunity to process the pipeline after its execution has been completed. Typically, this may be when there is a time-consuming step that is required for the generation of a report, and we'd like to do it after our pipeline has completed execution. This will be called exactly once after the analysis pipeline has completed.

Deploying a Recipe

An XML element containing the path to the recipe must be included in the *VIBE_SERVER/scripts/recipes.xml* file (see **Modifying the layout of the case hierarchy** below). The path to the recipe file must be indicated in a similar manner to a Java fully-qualified classname, *e.g.*, *recipes.default.location.ExampleCase.RecipeImpl*. The class residing within the indicated file will be instantiated with the name given by the file. Additionally, all Jython files must end with the .py file extension and they must be placed below the *VIBE_SERVER/scripts* directory.

2.1.8. Recipe Support

To support the easy implementation of a variety of cases a series of recipe support libraries were developed. The classes that make up these libraries are implemented in either Java or Jython.

RecipeUtils

The class *com.incogen.vibe.webaccess.server.recipes.RecipeUtils* is a set of utility functions to be used by recipe developers. Among these functions are convenience functions for setting and getting parameters from modules. The function lists are displayed in Appendix C.

ModuleFactory

Additional support is provided by *com.incogen.vibe.util.ModuleFactory*. This class easily generates a module to be included in a pipeline from its appropriate module-descriptor and can be used to dynamically build the pipeline in a *RecipeImpl.py* file. This may be useful if the input for the case can take multiple types of input (*i.e.*, nucleotide or

amino acid) and uses different flags and algorithms in the pipeline (e.g., blastn as opposed to blastp). This also provides a mechanism for generating pipelines from the Recipes without requiring the VIBE client. Module descriptors can be created according to the VIBE SDK documentation.

ModulePlacers

A class that implements the *com.incogen.vibe.util.ModulePlacer* interface can layout a pipeline in two dimensions. Since most of the pipelines generated through Web Access do not have x-y coordinates when they are created, they all fall on top of each other in the top-left corner (x=0, y=0) when they are imported into the VIBE client. Therefore, before the pipeline is executed within Web Access a *ModulePlacer::placeModules* function is called. Currently, the default module placer is a *FanModulePlacer* that lays the modules out from the top left corner of the workspace to the bottom right in a fan pattern. Alternative placers are available, such as the *CircularModulePlacer*. The placer that is used for the system may be configured from the server's configuration file with the “_modulePlacer” parameter.

2.1.9. System Configuration

Determining the current state of the server

The state of the server may be observed at any time using the Server Configuration page. This is located in the top menu bar of the Web Access page. Administration permissions are required to view the page.

This page indicates the version of the server, the number of jobs queued, the number of jobs running, the amount of memory used and free, and the uptime of the server. Additionally, the page indicates the particular jobs that are running and gives the

administrator the ability to stop jobs. The configuration parameters for the server and their associated values are listed along with module-specific macros. Links to the most recent updates to the log file for the server and another one to a Wiki page for editing the server configuration file are available. Finally, the page also indicates all the modules available to the system and their associated server mapping.

Specifying the location for Modules to be Remotely Executed

The *keyword-servers.properties* file is available to easily modify the module-to-server mappings. This will determine on which physical machine the module is executed on. The file simply relates a given module's keyword on the left side of the equals sign with a server on the right-hand side. After this file has been updated you will see the changes in the **Server Configuration** page immediately. For example, to specify that the *blastp* module should use the VIBE server located at

http://vibewa.incogen.com/webaccess-server, the following line in the *keyword-servers.properties file* is inserted:

```
blastp=http://vibewa.incogen.com/webaccess-server
```

Modifying the layout of the case hierarchy

The *recipes.xml* file can be edited to easily modify how the case hierarchy is presented to users. It is a simple XML file that has the top-level element, *recipes-descriptor*. Within that element there can be any one of three types of elements:

- **category** - This defines a recipe's grouping as it is presented in the case hierarchy. Categories may also contain other categories. For example, one of the categories in Figure 5 that the Structural category contains is Nucleotide. The following attribute must be defined within this element:

- **name** - (required) - The name that will be used for the category in the case-hierarchy.
- **recipe** - This defines a particular location for a case implementation within the case-hierarchy. The same element may appear in different categories to cross-list the case within the hierarchy. The **Translate a new nucleotide sequence into an amino acid sequence** case in Figure 5 is how a recipe is depicted in the case hierarchy. The following attributes may be defined within this element:
 - **recipeClassname** - (required) - The definition of the class that implements the *Recipe* interface.
 - **excludeFromParent** – (not required) – This value may be set to true or false. By default, it is set to false. True indicates that this recipe should not be included in the parent cookbook list of recipes (if there is one). This would typically be desirable if the input module of the analysis pipeline for this recipe does not match that of the cookbook's input.
 - **status** - (required) - This can be one of three valid values indicating the implementation status of the case: complete, incomplete, or partial. A developer can use this to convey to end users whether the case is ready to use.
- **cookbook** – This element associates child case implementations so they can be executed concurrently. This element should encapsulate all child recipes and categories that are included in the cookbook. The **What general information is known from a Nucleotide sequence** case in Figure 5 is how a cookbook is

presented in the case hierarchy. The following attributes must be defined within this element:

- **recipeClassname** – (required) - The definition of the class that implements the *Cookbook* interface
- **status** - (required) - This can be one of the three valid values indicating the implementation status of the case: complete, incomplete, or partial.

Editing the genericCaseFormToPipeline.jelly file

Jakarta Jelly [48] is an open-source library for turning XML into executable code. The *genericCaseFormToPipeline.jelly* is available to easily modify how the input and parameter values from **Input.jsp** files are set in the *Pipeline* before it is executed. This only affects recipe implementations that extend *DefaultRecipeImpl* and do *not* override the *loadPipelineModel* function. Each module within the Pipeline is matched within the XML file and executes certain actions depending on the elements defined within the XML file. The top-level element of the XML file is *jelly*. Within the *jelly* element there can be zero or more *module* elements with the following attributes:

- **keyword** – (required) - The keyword that uniquely distinguishes this module from other modules.
- **exactMatch** – (not required) - This may be set to true or false but defaults to false. If true, the keyword will be matched explicitly with any module keywords defined in the pipeline that is being searched. If false, it will consider a match to be any occurrence of the keyword string in the keyword to the search pipeline.

Within the module element there can be one of three types of elements. Each of these

elements can appear zero or more times within the module element:

- **addInputToModuleOutputDescriptor** – This places the data specified by the *formValueName* into a file and sets its location in the output descriptor of the parent module. The following attributes may be defined within this element:
 - **formValueName** – (required)- Specifies the name of the input value that comes from the **Input.jsp** file.
 - **className** – (not required) - If this is provided, it should define a class that may be used to call a static method before the data is saved to file.
 - **method** – (not required) - The static method that is part of the class defined in the *className* attribute. This method will be called with the value that corresponds to the *formValueName* name before it is copied to a file.
 - **fileExtension** – (not required) - The file extension of the file that the input data is copied to. If this is not provided, the default is .xml.
- **addInputToModuleInputDescriptor** - This places the data specified by the *formValueName* into a file and sets its location in the input descriptor of the parent module. The following attributes may be defined within this element:
 - **formValueName** – (required) - Specifies the name of the input value that comes from the **Input.jsp** file.
 - **className** – (not required) - If this is provided, it should define a class that may be used to call a static method before the data is saved to file.
 - **method** – (not required) - The static method that is part of the class

defined in the *className* attribute. This method will be called with the value that corresponds to the *formValueName* name before it is copied to a file.

- **fileExtension** – (not required) - The file extension of the file that the input data is copied to. If this is not provided, the default is .xml.
- **setModuleParameterValue** - This sets a module parameter within the pipeline file that is used to execute the analysis pipeline with a parameter from the input form. The following attributes may be defined within this element:
 - **name** – (required) - The name of the parameter as it is defined within the module-descriptor of the respective module.
 - **formValueName** – (required) - Specifies the name of the input value that comes from the **Input.jsp** file.
 - **list** – (not required) - If the value is one of many possible values, the name of the entire list of possible values should be defined here.

As part of the *DefaultRecipeImpl*'s implementation of *loadPipelineModel*, the *genericCaseFormToPipeline.jelly* file is first read in. The function then iterates through each of the modules within the recipe's pipeline. For each module, it checks to see if it is defined within the XML file. If it is, it will follow the directives defined by the elements contained within the module element. For example, given the following block of Jelly script:

```
<module keyword="Sequence">
  <addInputToModuleOutputDescriptor name="inputTArea"
    className="com.incogen.vibe.server.webaccess.util.PipelineModelLoaderUtils"
    method="convertFastaToXML"/>
  <addInputToModuleOutputDescriptor name="inputFile"
    className="com.incogen.vibe.server.webaccess.util.PipelineModelLoaderUtils"
    method="convertFastaFileToXML"/>
</module>
```

```

</module>
<module keyword="blast">
  <setModuleParameterValue name="EXPECTATION" formValueName="expectationValue"/>
  <setModuleParameterValue name="WORD SIZE" formValueName="wordSize"
    list="wordSizeList"/>
  <setModuleParameterValue name="TARGET SET" formValueName="dbList"/>
</module>

```

a pipeline that goes from *Sequence_nt* -> *blastn* would be matched for both modules.

The data from the **Input.jsp** file would be sent to a file and the location set to the output descriptor of the *Sequence_nt* module after being converted to XML using the static method call. Notice, that there are two *addInputToModuleOutputDescriptor* elements to compensate for the two mechanisms of providing input: the text area (*inputTArea*), and the file upload (*inputFile*). For the *blastn* module, if there was an *expectationValue* input element in the **Input.jsp** file it would be set to the value of the *EXPECTATION* parameter. For the *WORD SIZE* parameter, we can select one value of a list of values contained in *wordSizeList*. This will also be set to the parameter value so that we still have those selections editable on the Advanced case submission page. Appendix B demonstrates the associated **Input.jsp** file for this snip-it of Jelly code.

The *com.incogen.vibe.server.webaccess.util.PipelineModelLoaderUtils* used in the above example is a class that provides some simple utility functions for parsing input and placing that data into a file. Right now, the only supported types are raw binary data (*PipelineModelLoaderUtils::convertBinaryFormFileToStream*) and Fasta input (*PipelineModelLoaderUtils::convertFastaToXML*). Additional functions could be easily added or additional classes may be used for these purposes.

Editing how reports are generated by default

The *PipelineToHTMLFormatter.py* is a Jython script that is used as the default

implementation for all recipes that extend *DefaultRecipeImpl* if the *generateView* function is not implemented. This is the code that generates reports for cases that have been newly created through the **Create a New Case** form.

Additional System Configuration

From the **Edit System Configuration** page the administrator may access Web Access configuration files like *recipes.xml*, *genericCaseFormToPipeline.jelly*, *PipelineToHTMLFormatter.py*, *JythonWebViewHandler.py*, and *keyword-servers.properties*. Resources such as Java Archive files (JARs), module-descriptors, and module documentation may be added and removed from the server from this page. The definition of these module-descriptors is described in the VIBE SDK documentation.

Automatic Recipe Generation from Exported VIBE Pipeline

VPML files that have been exported from the VIBE client may be used as a base for building a specific case. This is accomplished by using the **Create a New Case** form. This is discussed in Chapter 2 in the Developer Tutorial.

CHAPTER 3

CASE IMPLEMENTATIONS

3.1. Case Requirements; Case Implementations

The second stage of the project involved determining several cases in which biological researchers are interested that can be addressed by providing a data analysis pipeline/data-mining solution. The final cases that were compiled may be found in Appendix A in the hierarchical structure. This structure is intended to show how the user of the system encounters them. Once the cases were determined, the approaches for answering the cases were examined for similarities to determine which portions might be generalized. Where possible, the system was arranged in such a way that the user can configure the search interactively. In places where this was not possible, emphasis was placed on the flexibility and extensibility of the code for future developers.

Initially, the cases to be implemented were drawn directly from current capabilities of the VIBE system. The majority of the modules are designed for sequence analysis. This includes cases for generating nucleotide sequences from sequencer trace files, generating consensus sequences, and various similarity search algorithms. Other cases were drawn from talks with researchers, such as Kristina Gleason, Dr. Dasha Malyarenko, and Dr. Joan Burnside.

Since developing a host of new modules was beyond the scope of this project,

careful selection of the few modules to be added was required. High-value module targets were the first to be implemented. The two principal modules that fall into this category were a Matlab [49] module and an R [50] module. These were considered high value because they add a great deal of functionality that may be utilized by case developers. There are large amounts of specialized scripts available through resources such as the Matlab Central File Exchange [51] and the Comprehensive R Archive Network (CRAN) [52] that may be incorporated into VIBE now. Many other researchers have custom scripts that can be added to the system with only slight modifications.

Additionally, modules that were immediately useful to researchers were a high priority. Kristina Gleason required a TESS [53] module for a research paper. This module submits requests containing nucleotide sequences to the TESS server at the University of Pennsylvania where it searches the TRANSFAC [54] database. This finds possible transcription factor binding sites along the query sequence. This was partly motivated by an interest in using the FastaViewer (part of the VIBE client) to visualize these potential binding sites.

The following sections contain information on how each of the selected cases was implemented using the framework.

3.2. Structural Information

3.2.1. Nucleotide Information

What general information is known from a Nucleotide sequence?

This case allows the user to select any of several specific nucleotide analyses to be executed. Therefore, it is an unusual case because the output from each specific case is combined into a portal view summarizing all the results. The specific analyses for

nucleotide information follow.

Transcription Factors: Find possible Transcription Factor binding sites within the Nucleotide sequence.

This case submits the input sequence to a transcription factor binding site prediction algorithm and generates a report summarizing all the results. Right now, it uses the TESS [53] algorithm. This relies on the TRANSFAC [54] database. The generated report simply returns a link to the TESS results at the University of Pennsylvania website. The TESS results are located on a remote server and not associated with this project.

Genes: Can genes within this sequence be found in silico?

This case submits the input sequence to the gene prediction tools Genscan [12] and GrailEXP [11]. The user has the ability to choose one of the tools or both concurrently. The generated report then returns the predicted sequences from Genscan. It also returns the GrailEXP report that is obtained directly from GrailEXP.

Genes: Does my sequence contain a known curated gene?

This takes one or more nucleotide sequences as input and searches using *blastn* against nt (non-redundant nucleotide databases) and retrieves the RefSeq [55] records.

Hidden Markov Models (HMM): Perform an HMM search using nucleotide data to construct our HMM

This case implements the example given in the Introduction of this paper. That is, it will accept nucleotide data as input and *blastx* a database. It will then pass the hits and query sequence to a *clustalw* module which is used to construct an HMM in the *hmm_build* module. Finally, it will search an amino acid database with the constructed HMM using the *hmm_search* tool. This will potentially detect similar sequences that are not detected with the less-sensitive *blastn* search.

MSA: Generate a Multiple Sequence Alignment from a series of input sequences

This case accepts nucleotide or amino acid input sequences and passes the input into the multiple sequence alignment program, *clustalw* [5]. The results are then reported along with a corresponding dendrogram demonstrating how closely the sequences appear to be phylogenetically.

MSA: Generate a Multiple Sequence Alignment from a series of traces

This case accepts a zip file containing trace files from an ABI DNA sequencer [56]. It then passes these files through phred [57][58] which converts the sequences into nucleotide sequences. Then it functions like the previous MSA case.

Sequence Manipulation: Convert a nucleotide sequence into another frame

An input nucleotide sequence is accepted and passed through the SEALS FaFrame [59] algorithm. This converts the sequence into the frame that the user selects on the basic input form.

Sequence Manipulation: Generate a consensus sequence from a set of nucleotide sequences

This case takes a set of input sequences and passes it through the Seals Agree [59] algorithm which creates a consensus sequence from those sequences.

Sequence Manipulation: Translate a nucleotide sequence into an amino acid sequence

This case takes an input sequence and translates it into the corresponding amino acid sequence using the SEALS Wimklein [59] program. The user may select the genetic code of an organism to use for the translation.

Sequence Tagged Sites: Test sequence for presence of STSs.

This case attempts to discover STSs using NCBI's e-PCR [13] tool. The

electronic PCR (e-PCR) tool is specially designed to find regions within a nucleotide sequence that matches with PCR primers. The tool came with several flat files that contain STS data that are used in aligning the sequences and function as databases. The raw results of the analysis are presented in the report for this case.

Sequence Tagged Sites: Search the STS database for a nucleotide sequence

This case accepts one or more nucleotide sequences as input and searches using blastn against the STS database. This will therefore return a list of known STSs that may have regions similar to the input sequence.

Trace: Determine the nucleotide sequence from trace information.

This case uses phred [58][57] to retrieve quality and sequence information from sequencing trace files.

Trace: Assemble shotgun sequencing data.

This case uses phrap [60] to assemble contigs. The input can be one of a couple forms. The first is as a series of nucleotide sequences. It then assembles the sequences into a contig (or contiguous consensus sequence) using phrap. The case also accepts a zip file containing the ABI trace files. It then pipes the input through phred to retrieve the nucleotide sequences and then uses phrap as with the other type of input. The contig is returned in the report along with the nucleotides used in phrap.

3.2.2. Amino Acid Information

3.2.3. Mass Spectrometry

Most of these cases rely on Matlab [49] scripts. These scripts were developed in large part by Dasha Malyarenko. The tool, Grace [61], was used to generate the SVG plots for the cases that required two dimensional graphs.

Raw Signal Spectrum: Smoothing: Apply Gaussian smoothing to Spectra

An input spectrum of values from a mass spectrometer is passed to a Matlab [49] script called *gapodize.m*. The user has the opportunity to select whether they would like to plot the original signal and the smoothed signal. The report returns a link to the new values along with the input values in matrix format. This allows users to import the data into their favorite spreadsheet or statistical tool for further analysis.

Raw Signal Spectrum: Smoothing: Apply Moving Average smoothing to Spectra

This case is similar to the previous case except the Matlab script that is used is called *mavFilter.m*. It uses a moving average function to smooth the spectra. The generated report is essentially the same as in the previous case.

Variable: Selection/Dimension Reduction: Determine Principal Components

This case calculates the principal components of an input data matrix using the script *mypca.m*. The data matrix must be in a form where columns are patients and rows are variables. It then plots the first two principal components.

Variable: Selection/Dimension Reduction: Calculate Discriminant Coordinates

This case calculates the discriminant coordinates of an input data matrix using the script *dc3classall.m*. The data matrix must be in a form where columns are patients and rows are variables. The plot that is generated currently expects there to be three groups and labels them Normal, ATL, and HAM.

Classification & Cross-validation: Determine Leave-1-Out Cross-validation Error

This case calculates the principal components of an input data matrix using the script *classr13.m*. The generated report simply returns the error associated with the calculation.

Classification & Cross-validation: Determine Random Permutation Cross-validation Error

This case calculates the principal components of an input data matrix using the script *cc3classrp.m*. The generated report simply returns the error associated with the calculation.

Classification & Cross-validation: Measurement-Out Cross-validation Errors

This case calculates the principal components of an input data matrix using the script *measldar13cv.m*. The generated report plots the cross-validation error on the ordinate and the variable that was taken out on the abscissa.

3.2.4. Organism Information

Homology: What organisms have a sequence with a strong likelihood of homology?

This accepts either an NT or AA sequence and will blastx or blastp it as appropriate against a set of amino acid databases defined by the user. Then it filters out the lowest expectation values based on a threshold provided by the user. After that, it passes all the hits that passed the filter to *clustalw*. It also has the option of passing the query sequence's description line to an Entrez search and can pass these hits on to *clustalw* as well. The user is presented a report containing a dendrogram from *clustalw* and links out to the specific Genbank records for each hit. This is also cross-listed under both Nucleotide Information and Amino Acid Information.

3.2.5. Statistical

Many of the statistical cases rely on scripts written within the R [50] environment. The output data from these modules will typically be in StatDataML [62][63].

Matrix: Generate a biplot from an input data matrix

This case calculates the principal components of the input data matrix and then generates a biplot. This biplot plots the data as well as the variables on the same plot, with the variables are plotted as arrows and the objects as numbers.

Matrix: Generate a screeplot from an input data matrix

This case calculates the principal components of the input data matrix and then generates a screeplot. The screeplot is a bar graph that plots the variances of each principal component.

Dissimilarity: Determine a 2D representation for an input dissimilarity matrix

This case adds the transpose of the input matrix to the input matrix and use that as an input matrix to an R script. The user may select one or two multiple dimension scaling techniques to generate the 2D representation. The first is classical MDS. The second is `sstress.r` [64]. The report shows the graphical representation of the data in two dimensions and also the calculated values in StatDataML format.

Clustering: Perform agglomerative clustering on an input dissimilarity matrix

This case uses the *agnes* function as part of R's *clustering* package to calculate an agglomerative cluster for the input. The user may select one of three types of methods for constructing the clusters: single-linkage, complete-linkage, and average-linkage. The output consists of a dendrogram depicting the relationship between each of the input rows.

Correlation: Display correlation matrix in one dimension

This case accepts a correlation matrix and generates a polar plot displaying how closely correlated each value is to its related value on a polar plot. The closer the values

appear, the higher their correlation. The original S-Plus [65] script utilized by this case was developed by Dr. Trosset [66].

Correlation: Display correlation matrix in one dimension and separated by groups

This case accepts a correlation matrix and generates a polar plot displaying how closely correlated each value is to its related value on a polar plot. The closer the values appear, the higher their correlation. The values are also plotted on different shells about the origin to indicate different grouping. This R implementation was developed for a class project from Dr. Trosset's graduate course [66] and relied on an S-Plus script that he developed.

CHAPTER 4

A TOUR OF THE SYSTEM

In order to demonstrate the usefulness of VIBE WA it is instructive to present the system in a tutorial format. This will both demonstrate in a hands-on manner how the system would be encountered by users and also shows how the various technologies work together in a cohesive manner.

The system is intended to be used by two primary audiences: the end-users, which for this project are typically biologists, and developers, who may frequently be bioinformaticists. The reader is encouraged to experience the system from <http://vibewa.incogen.com/webaccess-server/index.html> through a web browser. These descriptions will provide an overview of the system's features and show how they are beneficial.

4.1. End-User Tutorial

Introduction

The first examples demonstrate much of the core functionality of VIBE Web Access and are intended to give new end-users a quick introduction to the system. To properly view the graphs and plots requires the installation of an SVG browser plug-in provided free-of-charge by Adobe [42]. The system will prompt you when it is needed if

it is not installed.

This section will first demonstrate how two cases may be executed through VIBE WA. The first is a straightforward case that is intended to show how cases may be executed through the system. The second case demonstrates the transparent access to some of the more advanced features. Finally, a third example shows how multiple cases may be concurrently executed through the system.

4.1.1. Example 1: Translate a nucleotide sequence into an amino acid sequence.

This example demonstrates a very simple situation that biologists may frequently encounter and how to solve it by using VIBE Web Access. That is, converting a nucleotide sequence into an amino acid sequence using a particular genetic code.

1. Click in the left panel on the **Case Hierarchy** link.
2. A tree representing various categories of biological problems is presented in a hierarchical fashion.
3. Click on Structural, then Nucleotide, and then Sequence Manipulation (see Figure 5).

Submit a Question

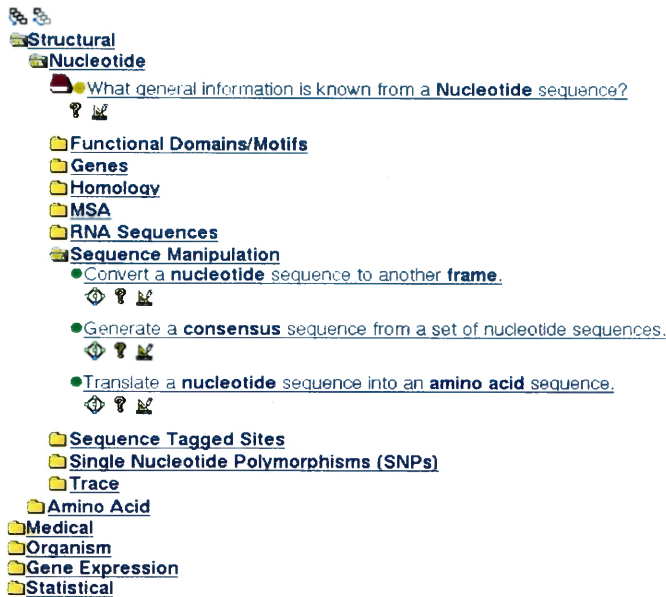





Figure 5. Problem Selection.

4. Each problem description represents a different case that is available to examine.

Notice that there are three icons immediately below each case.

- **Preview** -  - Clicking this icon presents a preview of the analyses that will be executed.
- **Help** -  - Clicking this icon retrieves documentation for what the analysis does.
- **Configuration** -  - Clicking this icon brings up a webpage that allows developers to edit the implementation of the case if they have the proper permissions.

5. Click the **Translate a nucleotide sequence into an amino acid sequence** link.

6. You are then presented a minimal input view for the case (see Figure 6). This allows users to provide the analyses with multiple forms of input and a subset of

parameters. These parameters will typically be the most interesting ones to modify when investigating this case.

Translate a nucleotide sequence into an amino acid sequence.

Pipeline Input

FASTA Text:	
Or input the FASTA file:	Browse...


Pipeline Parameters

Code	Standard	▼
------	----------	---

Submit | Advanced | Reset



Figure 6. Minimal Input for Case.

7. The same 3 icons from the previous page are located to the right of an icon of a vertical green arrow, . Clicking the vertical green arrow icon loads example input for this case into the appropriate fields.
8. Change the value in the drop-down box to the right of *Code* to **Bacterial**.
9. Then click **Submit** to begin executing this case.
10. After that, you will see a page similar to Figure 7 (with a different Job ID). This allows users to modify how the output of the analysis will be presented.
 - Case-Specific View - This view is dynamically generated based on the output of the analyses, and it is intended to be formatted in the most intuitive way to answer the question.

- **Printer-Friendly Report** – This option returns the same report as Case-Specific View without the header and footer of the website, so that the page is easier to print-out.
- **Graphic** - This generates an SVG document schematically representing the flow of the analysis. Green bordered modules will indicate completed modules, red borders indicate failed modules, and blue indicates modules that were not executed.
- **VPML** - This is a VIBE template file (*.vpml). The template file may be saved and opened from the VIBE client so the pipeline may be further examined. No data is preserved in this file, just the structure of the pipeline.
- **Archive** - This retrieves a VIBE archive file (*.zva). This file may be saved and loaded from the VIBE client like the VPML output. The data is also packaged into the archive file so that it may be viewed and explored through the VIBE client.



For now, leave "Case-Specific View" selected and press **Submit**.

Format Results

Submit | Reset

Job ID:	Case_946496
Display Type:	Question-specific View Printer-Friendly Report Graphic VPML Archive

Previously Executed Cases

Case Description	Date Started	Job ID	Preview	Edit
Translate a nucleotide sequence into an amino acid sequence.	Fri Feb 13 20:31:26 EST 2004	Case 946496		

Submit | Reset

Figure 7. Output Type Selection.

text format.

4.1.2. Example 2: What organisms have a sequence with strong likelihood of homology?

In this example, we will examine some of the more advanced features of the execution process in VIBE Web Access.

1. Click in the left panel on the **Case Hierarchy** link.
2. Click Organism, then Homology, and then **What organisms have a sequence with strong likelihood of homology?** (see Figure 9).

Submit a Question

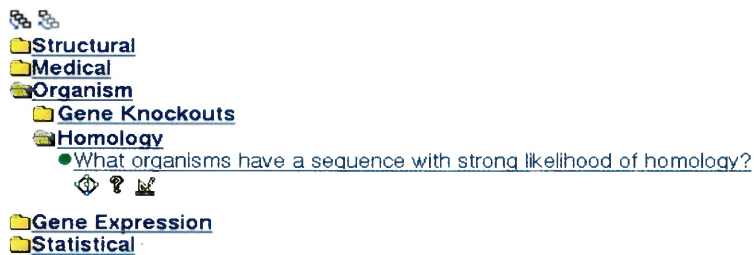


Figure 9. Location of the Homology Case.


This case may also be reached from two alternative locations (see Figure 10). For many cases it may be useful to the user to have cases cross-listed in different locations, since the same case may be used for various purposes. This one is cross-listed since it may take either nucleotide or amino acid sequences as input.

Submit a Question

The screenshot shows a web interface for submitting a question. At the top, there are two small icons. Below them is a main menu with the following items:

- Structural**
 - Nucleotide**
 - What general information is known from a **Nucleotide** sequence? (with a question mark icon)
 - Functional Domains/Motifs
 - Genes
 - Homology
 - What organisms have a sequence with strong likelihood of homology? (with a question mark icon)
 - MSA
 - RNA Sequences
 - Sequence Manipulation
 - Sequence Tagged Sites
 - Single Nucleotide Polymorphisms (SNPs)
 - Trace
 - Amino Acid**
 - What general information is known from an **Amino Acid** sequence? (with a question mark icon)
 - Functional Domains/Motifs
 - Homology
 - What organisms have a sequence with strong likelihood of homology? (with a question mark icon)
 - Macromolecular Structures
 - Mass Spectrometry
 - Metabolic Pathways
 - MSA
 - Receptors
 - Sequence Manipulation
 - Medical
 - Organism
 - Gene Expression
 - Statistical

Figure 10. Alternative Links to the Homology Case.

- Clicking the Preview icon () displays the directed graph (Figure 11) depicting a representative flow for the implementation of this case. In this case, the input sequence is piped into an Entrez search and blastx. These results will then be joined into one file and piped into a multiple-sequence alignment program (ClustalW).
- Clicking on an oval will display the documentation for the module represented by the oval. If you click on the blastx oval you will be taken to the blastx documentation (see Figure 12).

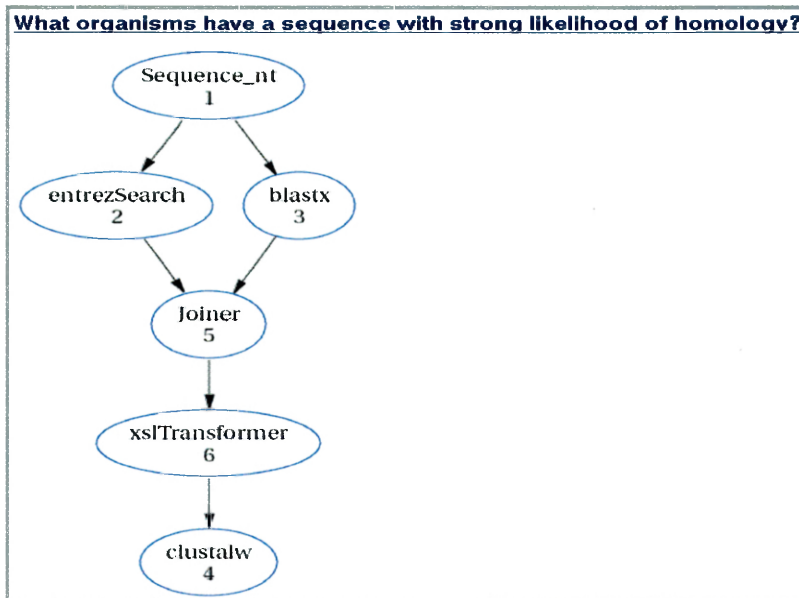


Figure 11. Representative View of Pipeline for the Homology Case.

Blastx Use the BLAST algorithm to compare the six-frame conceptual translation products of a nucleotide query sequence. (Basic Local Alignment Search Tool) programs have been designed for speed to find high scoring local alignments. BLAST is therefore able to detect relationships among sequences which share only isolated regions of similarity (Altschul et al. 1990).

[TARGET SET] The target database(s) to search. Double-click the module or the value in the parameter table to access a database.

[EXPECTATION] The statistically significant expectation value. If the statistical significance ascribed to a match is great, leading to a few chance matches being reported.

[EXTEND PENALTY] The extend penalty for each character skipped in a gap including the first. Input 0 indicates default.

[GAPPED ALIGNMENT] Specification on whether gapped alignment is to be performed or not.

- T if gapped alignment is to be performed
- F if gapped alignment is not to be performed.

[OPEN PENALTY] The open penalty for gap insertion. DeCypher uses this number as negative regardless of the sign specified.

[MATRIX] The scoring matrix file to use. These matrices are ASCII files and can have CRLF or LF line breaks. Use the name of the matrix file. The full path must be specified.

[MAX ALIGNMENTS] The maximum number of alignments to output. Fewer alignments may be shown due to use of the top-scoring alignments.

[NUM HITS] The maximum number of best hits from a region to keep.

[EXTENSION DROPOFF] Dropoff (X) for blast extensions in bits.

[TARGET GIS] If given, the search of the database will be restricted to the list of GIS.

[REGION] Location of a query sequence.

[QUERY FILTER] The specification on whether the query filter is to be used. The BLAST process filters your query sequence to eliminate potentially confounding matches from the BLAST reports. OPTIONS:

- T if the query filter is to be used
- F if the query filter is not to be used

[PROCESSORS] The number of processors to use.

Figure 12. Representative Module Documentation.

5. Now click back twice on your browser and you will be taken back to the input form (Figure 13).
6. Click the vertical green arrow icon to load example input data.

What organisms have a sequence with strong likelihood of homology?

Pipeline Input

(Note: Either Nucleotide or Amino Acid data are valid input)

FASTA Text:	
Or input the FASTA file:	Browse...

Pipeline Parameters

Search Entrez with input sequence(s)	<input checked="" type="checkbox"/>
Expectation Value	0.1
Word Size	3
Target Databases	<div style="border: 1px solid black; padding: 2px;"> D_melanogaster E_coli_refseq </div>

[Submit](#) [Advanced](#) [Reset](#)



Figure 13. Minimal Input View.

7. This input form allows the user to specify that the sequence should be used to *Search Entrez* when the checkbox is checked. Also, multiple databases to BLAST against may be selected by holding down on the Ctrl-key on your keyboard and clicking on the respective database names next to the *Target Databases* label. For this example, both *D_Melanogaster* and *E_coli_refseq* are assumed to be used. This will instruct the blastx to compare against these two specific organisms for homology.
8. Now click on the **Advanced** button. This displays the directed graph showing the actual pipeline of modules to be executed (Figure 14) as well as the full set of input options which can be modified for each of the operations. Clicking the modules in the graph will quickly take you to the parameters for that module.

Clicking the name of the module will take you to the documentation for that module similar to what was shown in Figure 12.

What organisms have a sequence with strong likelihood of homology?

Submit | Reset

Sequence nt Module (1):
 QUERY TYPE

entrezSearch Module (2):
 RunTime Standard

clustalw Module (3):

ALGORITHM	CW
DIAGONALS	5
EXTEND PENALTY	0.05
GAP DISTANCE	8
NO HYDRO GAPS	FALSE <input type="text"/>
NO RESIDUE GAPS	FALSE <input type="text"/>
HYDRO RESIDUES	GPSNDQEKR
KTUPLE	2
MATRIX	Default <input type="text"/>
OPEN PENALTY	10
OUTPUT ORDER	ALIGNED <input type="text"/>
PAIR GAP	3
SEQUENCE_NUMBERS	OFF <input type="text"/>
TOP DIAGONALS	5
TRANSITION WEIGHT	0.5
RunTime	Standard <input type="text"/>

Joiner Module (4):
 RunTime Standard

blastx Module (5):
 ALGORITHM

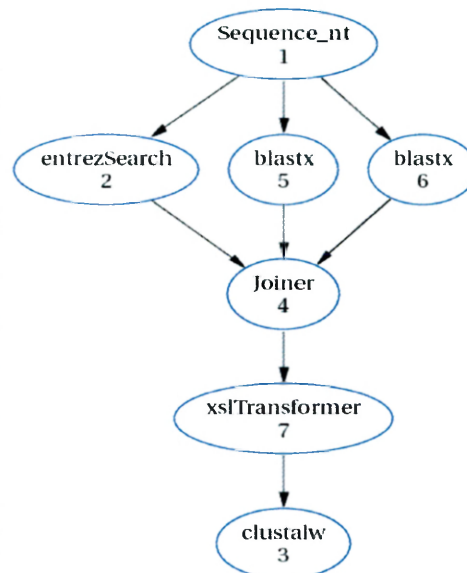




Figure 14. Advanced Parameter Modification and Actual Flow of Pipeline.

Since we kept the *Search Entrez* checkbox selected, we have an *entrezSearch* module in our pipeline. We also have two *blastx* modules, one for each of the selected target databases. If we had unselected the *Search Entrez* box we would have only the selected *blastx* modules. If we did not have a target database selected, we would have our input sequences piped directly into *ClustalW*.

9. After exploring the Advanced Parameter View, you can submit the job with the refined parameters using the **Submit** button and explore the data as you did in Example 1.

4.1.3. Example 3: Executing Several Cases Concurrently

Some of the cases in the case hierarchy have a  (cookbook) icon next to them. This indicates that the case is a special collection of cases that all operate on the same type of input. This allows users to execute any combination of the cases below it in the hierarchy concurrently.

1. Select the **What general information is known from a Nucleotide sequence?** case.
2. Add nucleotide data to the input text area. A  (load example data) icon is located immediately below the Input Text Area to load example sequences.
3. You may select any combination of the cases by selecting the checkbox to the left of each case you would like to execute. For this demonstration, select the **Can genes within this sequence be found in silico?** and **Convert a nucleotide sequence to another frame.** cases.
4. Now press the **Submit** button to execute the cases. When executing the collection of cases you will not be prompted to select an output style; the system will automatically use the case-specific view. The execution may take a moment, and the results will be presented in a portal view that summarizes the results using the specific view for each case.

This section demonstrated how a typical biological researcher may interact with the VIBE Web Access system. In Example 1, we saw the use of the case hierarchy,

minimal input form, how data is loaded, the submission of a case for execution, and how the output from a case execution may be viewed. Example 2 showed the analysis preview, advanced input view, and module details. Example 3 then showed how multiple cases could be concurrently executed on the same input data.

4.2. Developer Tutorial Introduction

These examples demonstrate how to integrate new cases and their respective implementations into VIBE Web Access. These examples will also typically require special permissions to use many of their features. The local VIBE system administrator can provide the appropriate usernames and passwords. For the demo system associated with this project the username is “vibe” and the password is “vibe”.

4.2.1. Part 1: Implementing a simple case

This example demonstrates a very simple situation that biologists may frequently encounter and how to solve it by creating an executable case using VIBE Web Access: converting a nucleotide sequence into an amino acid sequence for a particular genetic code. An implementation of this resides under the description **Translate a nucleotide sequence into an amino acid sequence** and may be used as a reference.

There are two general ways that analysis pipelines may be used through VIBE Web Access to define cases. The first method is statically, where the description of the analysis pipeline is stored in a template file. This may typically be created through the VIBE client. The second method uses a dynamic approach where the analysis pipeline is

defined programmatically. The following two sections demonstrate each method to implement the same case.

Statically-defined Analysis Pipelines

1. For users with a VIBE client installed, start the application. Following the VIBE client documentation for how to construct a pipeline, construct a pipeline in the form indicated in Figure 15. Any of the parameters on the modules may be modified from their defaults as well.

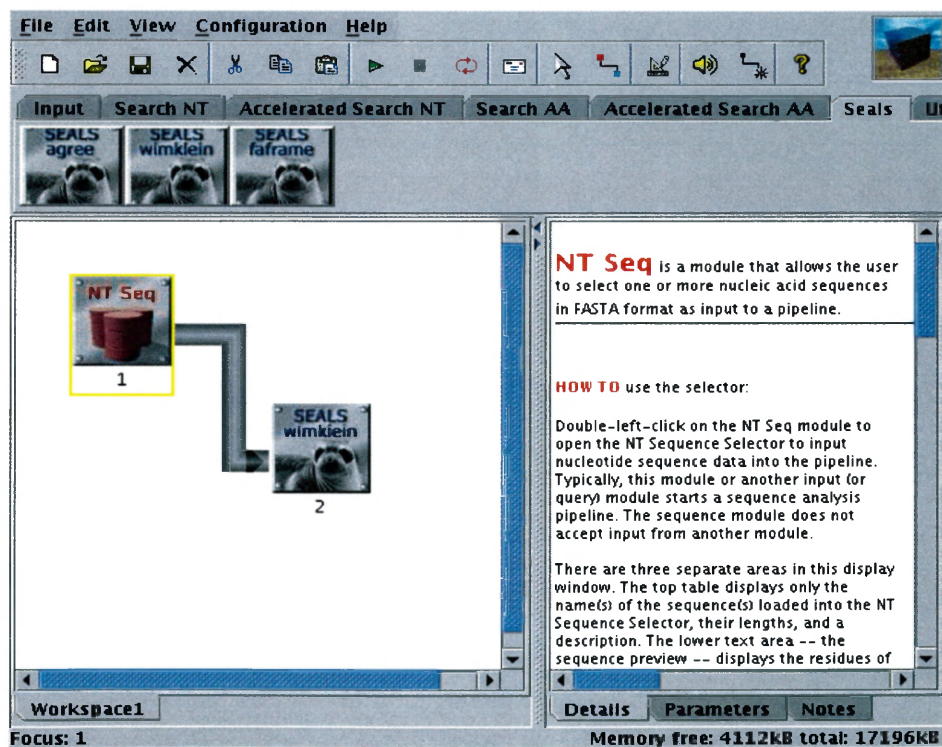


Figure 15. Construct a pipeline template in the VIBE client.

2. Click **File|Save as** and save the pipeline template to your hard-drive under any filename you like (the default is probably *Workspace1.vpml*). You can now exit the VIBE client.

3. Within your web-browser, click in the left panel on the **Create a New Case** link.

A form that looks like Figure 16 will be presented to you. Fill in the form as indicated in the figure. If you saved a template file to your hard-drive as suggested in Step 2 then browse to the file using the **Browse** button. If not, you can leave this field empty. Leave the *Create as a Cookbook* checkbox un-selected (see Part 5 for the significance of this option.) Then click **Create** to create the case.

Create a New Case	
Case Keyword (required):	TranslateNTSequence
Path to Case Implementation (required):	default/location
Case Description (optional):	Translate a nucleotide sequence into an amino acid sequence
Pipeline file (optional):	<input type="text"/> <input type="button" value="Browse..."/>
<input type="button" value="Create"/> <input type="button" value="Reset"/>	

[Edit Page](#)

Figure 16. Create a New Case form.

4. You will be sent to the case hierarchy webpage. Notice that a new category at the bottom has been created with a name generated based on the time the case was created. If you expand the new category you will see that a new case has been added with the description defined in our **Create a New Case** form (see Figure 17).

Submit a Question

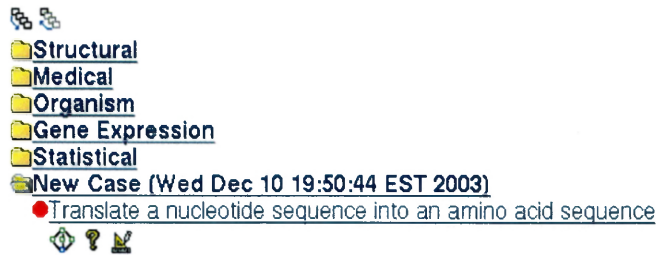


Figure 17. Case Hierarchy with new Case category.

5. If you defined the pipeline template file properly when creating this case then it will execute now. Select the description of the case from the case hierarchy and you will be presented with Figure 18. Additionally, the preview of the case will work by default and a case help page is provided. The text of the help file may be modified by selecting the **Edit Page** link at the bottom of the help page. Finally, example nucleotide (NT) sequence data is provided by the system since it knows that the source of the pipeline is a Sequence_nt module.

Translate a nucleotide sequence into an amino acid sequence

Pipeline Input

Input: Text:	
Or input a file:	<input type="text"/> Browse...

Pipeline Parameters

[Submit](#) | [Advanced](#) | [Reset](#)




[Edit Page](#)

Figure 18. Minimal Input form.

Dynamically-defined Analysis Pipelines

If you did not construct a pipeline template file you can still create a case implementation, but the analysis pipeline must be defined programmatically. The last few steps of this part of the tutorial describe how this is accomplished. The following steps also require that you created the case without defining the pipeline file. If you did define it, create a new case that does not define the *Pipeline file* in the form.

6. Select the  (edit implementation) icon next to our new case. A page similar to Figure 19 will be presented that lists the minimal necessary files for implementing a case. For a new case, these files contain default implementations. This allows you to modify case-related support files. If you skipped defining the template pipeline in Steps 1-2, then `template.vpml` will not appear in this list of files.

Case Implementation Summary

[Translate a nucleotide sequence into an amino acid sequence](#)

Location: `/scripts/recipes/default/location/TranslateNTSequence`

File	Description	Required	Size (bytes)	Edit	Remove
Help.jsp	Documentation file	yes	550	Edit	Remove
Input.jsp	Basic input view of the case	yes	1682	Edit	Remove
RecipeImpl.py	Case-specific implementation file	yes	965	Edit	Remove
__init__.py	Python package file	no	0	Edit	Remove
description.txt	Description of this case	yes	59	Edit	Remove
template.vpml	Default pipeline that is loaded by this recipe	no	2721	Edit	Remove

Upload Browse... Reset



Figure 19. Case Implementation Summary.

Select the **Edit** link next to the `RecipeImpl.py` file and you will be presented with a Wiki-style [67] interface shown in Figure 20. This constitutes the implementation of this case written in the Python programming language. You may notice that this class extends `DefaultRecipeImpl` and therefore implements

the *Recipe* interface. All case implementations must implement *Recipe*, and it is usually best to extend the *DefaultRecipeImpl* class.

7. Immediately below the lines,

```
def generatePipelineModel(self):  
    pipeline = Pipeline()  
    pipeline.setName('Template Pipeline')
```

enter the following code:

```
    sequenceModule = ModuleFactory.newModule('Sequence_nt')  
    pipeline.add(sequenceModule)  
  
    sealsModule = ModuleFactory.newModule('SealsWimklein')  
    pipeline.add(sealsModule)  
    pipeline.addConnection(sequenceModule, sealsModule)
```

This accomplishes the same thing we did from the VIBE client but programmatically. See the VIBE Web Access Javadocs accessible through the left panel for API details.

Programmer's Note: Python code is sensitive to white-space and indentation, so it is important to have the above code lined up vertically with the previous lines of code.

/scripts/recipes/default/location/TranslateNTSequence/RecipeImpl.py

Update | Reset

Update to Previous File

No backups for this file were found.

File Text:

```

_logger = LogFactory.getLog('TemplateRecipe.RecipeImpl')

class RecipeImpl(DefaultRecipeImpl):
    pipelineModel = None

    def __init__(self):
        _logger.debug("Instantiating TemplateRecipe")

    def initializeForm(self, session, caseform):
        _logger.debug("Initializing Form")
        pass

    def generatePipelineModel(self):
        pipeline = Pipeline()
        pipeline.setName('Template Pipeline')

        # Remove the comment symbols in the following block and the import statements
        #   at the top of this file to determine what VPML is being generated
        #   for the pipeline
        #output = BufferedWriter(FileWriter('testingPipeline.xml'))
        #output.write(XMLUtils.ElementToString(pipeline.toXML()))
        #output.flush()
        #output.close()

        return pipeline

```

Update | Reset

Instructions for Updating Scripts

Valid [JSP](#) may be used to render the input forms, while valid [Jython](#) may be used to tweak the Recipes.

[Edit Page](#)

Figure 20. Editing the Case Implementation file.

After finishing editing the file, select the **Update** button to commit your changes.

You will then be taken back to the case implementation summary page.

- Assuming your *RecipeImpl* is correct, you will be able to execute the case, retrieve a preview, etc., just as Step 5 described.

4.2.2. Part 2: Allowing users to modify parameters from the minimal input view

Now that we have made it possible to execute our analysis pipeline, we may be

interested in highlighting a few key parameters that the user may wish to change. This will emphasize the parameters that the developer deems to be the most relevant to this particular case. However, any available parameters for modules that make up the analysis pipeline may be modified from the Advanced page with no need for developer intervention.

1. Navigate to the **Case Implementation Summary** for the case in Part 1 of the developer's tutorial. This will correspond to Figure 19 above.
2. Select **Edit** beside the **RecipeImpl.py** file. Then copy over the

```
def initializeForm(self, session, genericCaseForm):
    _logger.debug("Initializing Form")
    pass
```

lines with the following (make sure that the lines are properly indented):

```
def initializeForm(self, session, genericCaseForm):
    _logger.debug("Initializing Form")
    pipeline = self.generatePipelineModel()
    modules = pipeline.getModules()
    codeList = self.getCodeList(modules)
    genericCaseForm.setParameterValue("codeList", codeList)

def getCodeList(self, modules):
    arrayList = ArrayList()
    for module in modules:
        if module.getKeyword().find('Wimklein') >= 0:
            codes = RecipeUtils.getParameterValueAsList(module, 'CODE')
            for code in codes:
                arrayList.add(LabelValueBean(code, code))
    return arrayList
```

This code extracts the list of genetic codes from the SealsWimklein module and passes it to the genericCaseForm object (an instance of *GenericCaseForm*). The genericCaseForm maintains the values of form fields in the HTML page. Note the following significant features of this block of code:

- The *RecipeUtils* class is a utility class with convenience methods provided by the Web Access API that allows users to easily set and get parameters

from Modules. It is part of the *com.incogen.vibe.server.webaccess.recipes* package. This should be loaded by default for any cases created using the **Create a New Case** form.

- *ArrayList* is a standard Java Collections class that is part of the *java.util* package.
- *LabelValueBean* is a class provided by the Jakarta Struts [43] package that VIBE WA uses.

3. Click **Update** to commit your changes.
4. Now navigate back to the **Case Implementation Summary** for the case.
5. Between the `<!-- START PARAMETERS -->` and `<!-- END PARAMETERS -->` tags in the corresponding **Input.jsp** file place the following block of code (see Part 5 for the significance of these tags):

```
<table border="1" width="75%" cellpadding="2" cellspacing="2"
  style="text-align: left; margin-left: 0px; margin-right: auto;">
  <tbody>
    <tr>
      <td style="text-align: right;">Code</td>
      <td>
        <html:select property="parameterValue(code)" size="1">
          <html:optionsCollection property="parameterValu(codeList)"/>
        </html:select>
      </td>
    </tr>
  </tbody>
</table>
```

This alters the minimal input view that is provided to users. In this case, we simply want to allow the users to change the genetic code that will be used for the translation operation. The *parameterValue(codeList)* property loads the list of genetic codes with the name *codeList* that we set to *genericCaseForm* in our *initializeForm* method. (For more information, see the Jakarta Struts

documentation describing access to mapped properties. Also, see the Struts-HTML tag library documentation.)

Likewise, the *parameterValue(code)* property indicates that we are keying into the parameterValues Map of our *GenericCaseForm* using *code* as the key.

When the form is submitted, the selected genetic code will be placed in an instance of *GenericCaseForm* and sent to the

DefaultRecipeImpl::loadPipelineModel method. One of the responsibilities of the developers is to make sure that the parameters and input from the **Input.jsp** make it into the pipeline to be executed. This is made considerably easier by the **genericCaseFormToPipeline.jelly** script. If you are using Modules that have already been deployed to this web application you should not typically need to alter this Jelly script.

6. Click **Update** to commit your changes. Now your basic input view should look and function the same as **Translate a nucleotide sequence into an amino acid sequence**.

4.2.3. Part 3: More Advanced Case Implementations

We may frequently be interested in providing cases that can dynamically build analysis pipelines based on user-defined parameters. This requires more coding, but gives the user much more flexibility than a static analysis pipeline as described in Section 4.2.1. We will implement a case that allows the user to specify one or two gene-finding algorithms on a given input data set.

1. Using the **Create a New Case** form, create a case with the keyword, *GeneSearch* and the following description:

Can **genes** within this **sequence** be found *in silico*?

2. Then, go to the implementation summary for the new case and edit the **RecipeImpl.py** file.

Preparing the Input Form

3. This will generally follow the structure of Part 2 of this tutorial. Overwrite the method:

```
def initializeForm(self, session, genericCaseForm):
    _logger.debug("Initializing Form")
    pass
```

with the following code:

```
def initializeForm(self, session, genericCaseForm):
    _logger.debug("Initializing Form")
    genericCaseForm.setParameterValue('useGenscan', Boolean(1))
    genericCaseForm.setParameterValue('useGrailEXP', Boolean(1))
```

This will initialize the checkboxes to be selected on the form. (The checkboxes are added in the **Modify the Input View** section below.) By default, these checkboxes would be un-selected. Alternatively, we could explicitly set them to be un-selected similar to the above except we would use a Boolean(0) value.

Dynamically generate the Analysis Pipeline

4. Now we want to configure the way the pipeline will be generated before it is executed. Overwrite the following:

```
def generatePipelineModel(self):
    pipeline = Pipeline()
    pipeline.setName('Template Pipeline')
```

```

# Remove the comment symbols in the following block
#   and the import statements at the top of this file to
#   determine what VPML is being generated for the pipeline
#output = BufferedWriter(FileWriter('testingPipeline.xml'))
#output.write(XMLUtils.ElementToString(pipeline.toXML()))
#output.flush()
#output.close()

return pipeline

with:

def loadPipelineModel(self, pipeline, genericCaseForm):
    # This is to verify that we don't get multiple pipelines being
    #   built
    pipeline.removeAllElements()

    sequenceModule = ModuleFactory.newModule('Sequence_nt')
    pipeline.add(sequenceModule)

    # Finish constructing the rest of the pipeline
    useGenscan = genericCaseForm.getBooleanParameterValue('useGenscan')
    _logger.debug("useGenscan: " + str(useGenscan))
    if useGenscan == 1:
        genscanModule = ModuleFactory.newModule('genscan')
        pipeline.add(genscanModule)
        pipeline.addConnection(sequenceModule, genscanModule)
    useGrailEXP = genericCaseForm.getBooleanParameterValue('useGrailEXP')
    _logger.debug("useGrailEXP: " + str(useGrailEXP))
    if useGrailEXP == 1:
        grailEXPModule = ModuleFactory.newModule('grailEXP')
        pipeline.add(grailEXPModule)
        pipeline.addConnection(sequenceModule, grailEXPModule)

    # This uses the genericCaseFormToPipeline.jelly script to map input
    #   parameters to pipeline parameters.
    DefaultRecipeImpl.loadPipelineModel(self, pipeline, genericCaseForm)

# Generates a representative pipeline that shows what this case can
#   do.
def generatePreviewPipeline(self, request):
    pipeline = Pipeline()
    pipeline.setName('Gene-Finding Pipeline')
    sequenceModule = ModuleFactory.newModule('Sequence_nt')
    _logger.debug("sequenceModule: " + str(sequenceModule))

    pipeline.add(sequenceModule)

    genscanModule = ModuleFactory.newModule('genscan')
    pipeline.add(genscanModule)
    pipeline.addConnection(sequenceModule, genscanModule)

    grailEXPModule = ModuleFactory.newModule('grailEXP')
    pipeline.add(grailEXPModule)
    pipeline.addConnection(sequenceModule, grailEXPModule)

    return pipeline

# We're going to actually build the pipeline in the loadPipelineModel
#   method after we've received input from the user indicating which
#   modules to include in the pipeline.
def generatePipelineModel(self):
    pipeline = Pipeline()

```

```

pipeline.setName('Gene-Finding Pipeline')

# This is so the system can figure out what our pipeline's input will
# be and will load the correct example data.
sequenceModule = ModuleFactory.newModule('Sequence_nt')
pipeline.add(sequenceModule)
return pipeline

```

Here we've implemented three functions, two of which are new to this series of tutorials. (see the *DefaultRecipeImpl* Javadocs in Appendix C for documentation on each function.)

Modify the Input View

5. Now, go to the implementation summary for the new case and Edit the **Input.jsp** file.
6. Between the `<!-- START PARAMETERS -->` and `<!-- END PARAMETERS -->` tags in the corresponding **Input.jsp** file place the following block of code (see Part 5 for the significance of these tags):

```

<table border="1" width="75%" cellpadding="2" cellspacing="2"
  style="text-align: left; margin-left: 0px; margin-right:auto;">
  <tbody>
    <tr>
      <td style="text-align: right;">Use Genscan</td>
      <td><html:checkbox property="parameterValue(useGenscan)"/>
      </td>
    </tr>
    <tr>
      <td style="text-align: right;">Use GrailEXP</td>
      <td><html:checkbox property="parameterValue(useGrailEXP)"/>
      </td>
    </tr>
  </tbody>
</table>

```

7. Finally, after making all the above changes, go to the case hierarchy, select the new case, and load the example data. This will load a Fasta nucleotide sequence since we provided a *Sequence_nt* module to the pipeline in the *generatePipelineModel* function. Both checkboxes will be selected. You can

check that the pipeline is being generated correctly by selecting the **Advanced** button. You will get a pipeline graph that looks like the one in the preview page. Now click the case's description at the top of the Advanced page. Then, select load example data again. This time de-select one of the checkboxes and click the **Advanced** button. You will now see the *Sequence_nt* module connected only to the selected module.

8. The case will now execute correctly, and except for the additional basic parameters will function just like the implementation of **Can genes within this sequence be found *in silico*?** that the system provides.

4.2.4. Part 4: Post-Processing of Analysis Pipelines

Post-processing a pipeline after it has completed is where the developer has the opportunity to substantially alter the presentation results for a case. By default, after the pipeline has completed, the pipeline is traversed and each module is called with a particular conversion utility that generates HTML from the XML data. This is defined in the file, *PipelineToHTMLFormatter* and is accessible from the **System Configuration Summary** page (see Section 2.1.9). You can modify this file and it will apply to all implementations that extend *DefaultRecipeImpl* but do not implement the *generateView* method themselves.

Alternatively, you are free to override the *DefaultRecipeImpl::generateView* (*pipeline, writer*) method in your *RecipeImpl* and implement a different case-specific view. This may be as simple as printing a simple statement yes or no if you are searching

for a particular value or threshold in a data file. Or it can be as complicated as you like. Potentially, this could be a complex data-mining step that integrates data from several modules in the pipeline.

Two examples of how to change the output reports include writing the HTML programmatically or using the Element Construction Set (ECS) library.

Changing the Output View

1. Select the case implementation that you created in Part 3 and edit the **RecipeImpl.py** file. At the bottom of your file, lined up with the previous function, place the following code:
- ```
def generateView(self, pipeline, writer):
 writer.write("this is where the results should be")
```
2. Then select the **Update** button.
  3. Execute the case, and select the Case-specific output view. You will see something like Figure 21. The printer-friendly link will work as well.




---

this is where the results should be

**Figure 21. Updated Case-Specific Results.**

When testing your output views you may not want to execute the pipelines every time you view them. It would be better to execute the case once, copy the job id, and just use the **Retrieve Results** form for subsequent views. Changes to the viewing implementation will be reflected in each **Retrieve Results** submission. Other previously executed cases are also listed in this page and those results may be accessed as well.

## Using the Element Construction Set (ECS)

The Element Construction Set (ECS) is a Jakarta library that aids in generating various markup. It has objects for most HTML elements and simplifies the construction of HTML and XML documents.

4. The following code fragment works very similarly to the

*PipelineToHTMLFormatter* except it only prints data for the modules that are in the case created in Part 3. It uses the ECS to construct the HTML document that will ultimately be sent to the browser.

```
def generateView(self, pipeline, writer):
 _logger.debug("Generating View : "+str(pipeline.getName()))

 # Create the ECS document that is used to generate the report
 document = org.apache.ecs.Document()

 # This puts the link to our case at the top of the report
 recipeClassname = str(self.getRecipeClassname())
 recipeDescriptionLink = WebAccessLinkUtils.getRecipeDescriptionLink
(recipeClassname)
 recipeLinkHeader = org.apache.ecs.html.H2(recipeDescriptionLink)
 document = document.appendBody(recipeLinkHeader)

 # Retrieve the modules from the analysis pipeline that was executed
 modules = pipeline.getModules()

 document = document.appendBody(org.apache.ecs.html.H2("Modules:"))
 ulElement = org.apache.ecs.xhtml.ul()
 _logger.debug('modules size: ' + str(modules.size()))

 # Iterate over the list of modules so we can generate the list of
 # modules that are included in the report
 for module in modules:
 ilElement = org.apache.ecs.xhtml.li()
 name = module.getKeyword() + " - " + module.getName()
 moduleLink = org.apache.ecs.xhtml.a()
 moduleLink.setHref('#' + name)
 moduleLink.addElement(name)
 ilElement.addElement(moduleLink)
 ulElement.addElement(ilElement)
 document.appendBody(ulElement)

 # Iterate over the list of modules and generate a report snippet for
 # each module
 for module in modules:
 name = module.getKeyword() + " - " + module.getName()
 moduleAnchor = org.apache.ecs.xhtml.a()
 moduleAnchor.setName(name)
```



```

moduleAnchor.addElement(name)
moduleAnchorHeader = org.apache.ecs.html.H3(moduleAnchor)
document = document.appendBody(moduleAnchorHeader)

Iterate through all input locations
inputLocations = RecipeUtils.getInputLocationsForModule(module)
if inputLocations.size() > 0:
 for inputLocation in inputLocations:
 a = org.apache.ecs.xhtml.a(inputLocation, "Full Input")
 document = document.appendBody(a)
 document = document.appendBody(org.apache.ecs.xhtml.br())

Iterate through all output locations
outputLocations = RecipeUtils.getOutputLocationsForModule(module)
if outputLocations.size() > 0:
 for outputLocation in outputLocations:
 a = org.apache.ecs.xhtml.a(outputLocation, "Full Output")
 document = document.appendBody(a)

 if _logger.isDebugEnabled():
 _logger.debug("outputLocation: " + str(outputLocation))

 # Generate specific output chunk for each module
 if module.getKeyword().find('genscan') >= 0:
 document.appendBody(org.apache.ecs.html.H4('Genscan Results'))
 document = self.generateHtmlSequences(document, outputLocation)
 elif module.getKeyword().find('grail') >= 0:
 grailResultsHeader = org.apache.ecs.html.H4('Grail Results')
 document.appendBody(grailResultsHeader)
 document = self.getPreformattedText(document, outputLocation)
 elif module.getKeyword().find('Sequence') >= 0:
 document = self.generateHtmlSequences(document, outputLocation)
 elif module.getKeyword().find('Wimklein') >= 0:
 document = self.generateHtmlSequences(document, outputLocation)

Writer out the document
writer.write(document.toString())

def getPreformattedText(self, document, outputLocation):
 _logger.debug('document: ' + str(document))
 document.appendBody('<pre>')
 _logger.debug('outputLocation: ' + str(outputLocation))
 document.appendBody(IUtils.read(URL(outputLocation)))
 document.appendBody('</pre>')
 return document

def generateHtmlSequences(self, document, outputLocation):
 document = document.appendBody(org.apache.ecs.html.H4('Sequence(s)'))
 document = document.appendBody(org.apache.ecs.xhtml.hr())
 sequenceData = SequenceModuleData()
 sequenceData.load(URL(outputLocation))
 formattedData = sequenceData.getFormattedData()
 document.appendBody("<pre>")
 document.appendBody(formattedData)
 document.appendBody("</pre>")
 return document

```

You will also need to import a few additional classes. Add the following code at the top of your **RecipeImpl.py** file among the other import statements.

```
from com.incogen.vibe.model import SequenceModuleData
```

```

from com.incogen.commons.io import IOUtils
from java.net import URL
import org.apache.ecs

```

5. After updating the **RecipeImpl.py** file, try to view the results of your case. If you implemented the above functions as part of the case created in **Part 3** they will look the same as the default view. Feel free to modify the above code to customize the view in whatever manner you like. Modifications will only affect this case.

#### 4.2.5. Part 5: Implementing a Cookbook

Cases may be grouped together to allow users to execute them together. To be consistent with the Recipe metaphor these groupings are called Cookbooks. After the implementations have been constructed using the previous sections of the developer tutorial, you can create a case that allows the user to execute them in parallel.

1. Create a Cookbook from the **Create a New Case** page. This can be accomplished by filling in the fields with variables similar to those provided in Figure 22. Make sure to select the **Create as a Cookbook** checkbox. Then click the **Create** button.

**Create a New Case**

|                                         |                                                                                            |
|-----------------------------------------|--------------------------------------------------------------------------------------------|
| Case Keyword (required):                | FindGenes                                                                                  |
| Path to Case Implementation (required): | default/location                                                                           |
| Case Description (optional):            | Find new or known genes from curated database or <code>&lt;i&gt;in silico&lt;/i&gt;</code> |
| Pipeline file (optional):               | <input type="text"/> Browse...                                                             |
| Create as a Cookbook                    | <input checked="" type="checkbox"/>                                                        |

Create | Reset

**Figure 22. Create a New Cookbook.**

2. Like the normal case implementations this will place a new category in the case hierarchy. Now we want to define the recipes that make up the cookbook. To do this, click the **Edit System Configuration** link. You will be presented with a view like Figure 23.

**Web Access System Configuration Summary**

| File                            | Description               | Required | Size (bytes) | Edit                 |
|---------------------------------|---------------------------|----------|--------------|----------------------|
| genericCaseFormToPipeline.jelly | None Available            | no       | 2212         | <a href="#">Edit</a> |
| recipes.xml                     | Case hierarchy descriptor | yes      | 9353         | <a href="#">Edit</a> |
| web.xml                         | Web app configuration     | yes      | 24641        | <a href="#">Edit</a> |
| PipelineToHTMLFormatter.py      | None Available            | no       | 6226         | <a href="#">Edit</a> |

**Figure 23. System Configuration Summary.**

3. We want to edit the descriptor that defines how the case hierarchy is presented and internally structured. This is manipulated by editing the *recipes.xml* file (see Section 2.1.9 for more details). So, click the **Edit** link next to this file. You will go to a Wiki-style interface like Figure 24.

## /WEB-INF/recipes.xml

Update | Reset

**Update to Previous File**

Tue Dec 16 16:52:48 EST 2003 | Reload

**File Text:**

```
<?xml version="1.0" ?>
<recipes-descriptor>
 <category name="Structural">
 <category name="Nucleotide">
 <cookbook status="partial"
recipeClassname="recipes.structural_information.nucleotide_information.general_information.CookbookImpl" />
 <category name="Functional Domains/Motifs">
 <category name="Motifs">
 <recipe status="incomplete"
recipeClassname="recipes.structural_information.nucleotide_information.functional_domains_motifs_motifs.RecipeImpl" />
 </category>
 <category name="Transcription Start Sites (TSS)">
 <recipe status="incomplete"
recipeClassname="recipes.structural_information.nucleotide_information.functional_domains_motifs_transcription_start_sites.RecipeImpl" />
 </category>
 <category name="Transcription Factors">
 <recipe status="incomplete"
recipeClassname="recipes.structural_information.nucleotide_information.functional_domains_motifs_transcription_factors.RecipeImpl" />
 </category>
 <category name="Genes">
 <recipe status="complete"
recipeClassname="recipes.structural_information.nucleotide_information.genes.GeneSearchInSilico.RecipeImpl" />
 <recipe status="complete"
recipeClassname="recipes.structural_information.nucleotide_information.genes.SequenceContainKnownCuratedGene.RecipeImpl" />
 </category>
 </category>
 </category>
</recipes-descriptor>
```

Update | Reset

**Figure 24. Case Hierarchy Descriptor file.**

4. Find the **category** element whose name attribute contains the value "Genes". You should find two child elements with the following syntax.

```
<recipe status="complete"
 recipeClassname="recipes.structural_information.nucleotide_information.genes.GeneSearchInSilico.RecipeImpl"/>
<recipe status="complete"
 recipeClassname="recipes.structural_information.nucleotide_information.genes.SequenceContainKnownCuratedGene.RecipeImpl"/>
```

Copy this code, and scroll down to the bottom of the file. You will find the new **category** with the time-stamp as its name and a child element that looks like the following:

```
<cookbook
recipeClassname="recipes.default.location.FindGenes.CookbookImpl"
status="partial"/>
```

Add a closing `cookbook` tag and replace the trailing `/>` with just `>`. Then paste the two `recipe` elements between the opening and closing `cookbook` tags. This includes these two recipes in the cookbook, and will allow you to execute the two cases that correspond to the recipes with the same input and in parallel. You can also change the status of this case's implementation by changing the value of the `status` attribute in the `cookbook` element from "partial" to "complete". The block should then look like the following (indentation is not important here):

```
<cookbook status="complete"
 recipeClassname="recipes.default.location.FindGenes.CookbookImpl">
 <recipe status="complete"
 recipeClassname="recipes.structural_information.nucleotide_informat
ion.genes.GeneSearchInSilico.RecipeImpl"/>
 <recipe status="complete"
 recipeClassname="recipes.structural_information.nucleotide_informat
ion.genes.SequenceContainKnownCuratedGene.RecipeImpl"/>
</cookbook>
```

Then click the **Update** button.

5. You will see that the two gene cases have now been included immediately below our new case in the case hierarchy. Additionally, if you click on the cookbook's description you will get an area for providing input data and the respective case parameters available from each of the gene cases. Finally, there is a checkbox next to each case that allows you to select which cases to execute when this form is submitted. By default, the parameters for each of the child recipes are drawn from between the `<!-- START PARAMETERS -->` and `<!-- END PARAMETERS -->` tags in the corresponding **Input.jsp** files.
6. Provide some input data, press the **Select All** link, and click the **Submit** button to execute the cases. After a few moments, you will be returned to a portal view of the outputs from each case.

This section demonstrated how a developer may interact with VIBE Web Access to create new cases. Part 1 showed how to implement a case using either statically or dynamically-defined analysis pipelines. The second part showed how the minimal input view for a case may be changed to cater to the requirements for a given case. Part 3 showed how a case with more complex logic can be defined within the system. Part 4 displayed how the case-specific view may be changed to depict only the results most pertinent to the case. The final part showed how cases can be grouped to construct “cookbooks” of cases that accept the same input.

## CHAPTER 5

### CONCLUSIONS

The principal framework and case implementation goals of this project were completed. The framework of the project required the development of an extensible infrastructure for creating and modifying cases that rely on analysis pipelines. The case implementation-side required the enumeration of cases relevant to researchers and their inclusion into the system. The following sections discuss the outcome of this project in more detail.

#### 5.1. Framework Conclusions

Initially, a mechanism for executing VIBE pipelines on a remote server without the need of a VIBE client was completed. This allows users to interact with their pipelines and cases from any location that has access to the server simply by using a web browser. Emphasis was placed on the ability for users to transparently observe and modify pipelines and case implementations without relying on knowledge of the details of the pipelines to execute them.

*Recipe* implementations (*RecipeImpl.py* files) were developed to act as an agent for the users. They provide a central location for manipulating how a case is interacted with. The *Recipes* also mediate the interaction with the VIBE Server by generating VIBE pipelines. It may accept input parameters and generate an appropriate VIBE pipeline for

the given parameters. This agent also has the ability of generating custom reports that only display the results pertinent to the case. This also allows the cases to easily be shared between groups at different physical locations that may be attempting to solve similar problems. This provides the ability of building a repository of problem solutions (or decision-support) that is tailored to a research group's goals.

A set of library functions was developed to ease the generation of reports, these include the *RecipeUtils* class of functions for easily pulling the raw data out of a given module, and the *PipelineToHTMLFormatter.py* provides default views for most cases. Many of the sequence analysis modules have formatters that will generate reports for the specific data type in the form of *ModuleData* classes. Additionally, views may be customized using the Apache ECS library that is provided within the system or by generating HTML programmatically.

A class that allows the easy generation of modules from within a recipe is provided in the form of a *ModuleFactory*. This will load the module and contact the server to get any updated information. For example, the module will retrieve a current list of databases for database searching tools. The *ModuleFactory* also resolves the location of the remote service by accessing the *keyword-servers.properties* file. The *Pipeline* class makes it easy to dynamically create analysis pipelines from the modules created using the *ModuleFactory*.

All these pieces combine to form the framework that allows the easy integration of new and novel cases. This enables researchers and developers to more easily manipulate their data.



## **5.2. Case Conclusions**

The cases that were implemented as part of this project will help researchers by providing decision-support in the form of generated reports. This allows researchers to leverage previously determined techniques for solving problems without having to replicate the generation of these techniques repeatedly. The cases are intended to emphasize the power and flexibility of the system.

The sequence analysis cases demonstrate many cases that are frequently created from within the VIBE client. Several of the statistical cases are meant to demonstrate the interoperability with the R [50] program. Additional cases that also demonstrate the interoperability with Matlab [49] are available with an emphasis on cases relevant to mass-spectrometry researchers.

The ability to edit the scripts and resources associated with VIBE WA added considerable flexibility and extensibility to this system since it prevents the need for long code recompilations and redeployment when building case implementations. This level of flexibility was also made available within the rest of the VIBE framework. This allows developers to implement VIBE services and modules within Jython and could augment the published VIBE SDK.

## **5.3. Future Work**

### **5.3.1. Framework Improvements**

Several immediate improvements to the framework were discovered in the development of this project. The interface could be improved by including support for

more recent web technologies such as Java Server Faces (JSF) [68] and Velocity templates [69]. These are alternatives or enhancements over the more traditional Java Server Pages that were utilized for the majority of this project. In particular, this could potentially simplify the development of minimal case input forms. The inclusion of these technologies may also attract developers that are not already familiar with JSP.

Another enhancement that was investigated as part of this project was the integration of Struts with Cocoon. This is well beyond the scope of this project, but it could improve the way that the UI is generated so that header and footer files do not need to explicitly be appended to each JSP file as they currently are. This would simplify the *Input.jsp* files.

A significant functional enhancement would be the ability to make Remote Procedure Calls (RPC) to invoke VIBE services. This would enable the VIBE WA code that executes the graphviz [40] program to take advantage of the VIBE server's advanced command-line generation functionality. It would also make it simple to include a graphviz VIBE module for generating directed or undirected graphs from within pipelines.

A significant performance enhancement would be to reduce the overhead associated with creating a VIBE module. Since VIBE modules have been used in the past for only client-side work there is a certain amount of overhead in creating them that is unnecessary in a strictly server-side architecture.

When creating cases from pipeline templates built from the VIBE client, it would be desirable to initialize the help documentation with the contents of the notes associated with the pipelines.

When the VIBE server supports the ability to manage users it will be feasible to associate the VIBE WA users with their data. A feature that would naturally extend from this is to present the user's analysis results and custom-cases on separate webpages to prevent others from accessing them. These may be presented as “My Results” and “My Cases” webpages.

Finally, the ability to import case implementations from other Web Access servers could be useful. This would complement the export case implementation functionality that is already present and allow easier collaboration among researchers at other sites, so that they may modify and extend the cases of their colleagues.

### **5.3.2. Improvements involving the VIBE Client**

There are several improvements to the VIBE client that naturally extend from the research performed for this project. The first improvement would be in improving the interoperability between the VIBE client and VIBE Web Access. A simple dialog box that allows the user to specify the VIBE Web-Access server to deploy the pipeline to and where the pipeline (and case) will be located in the case hierarchy could be added. This would prevent developers from needing to save the pipeline template locally and explicitly uploading it to the Web Access server.

Another potentially useful feature would be to allow the VIBE client to point to a Web Access server's data repository. This might allow the user to select any previously executed pipelines to further explore within the VIBE client without having to save the pipeline locally and then importing the file. This would allow users to take advantage of the more interactive data viewers available through the VIBE client.

Since the *PipelineExecutionService* was developed as part of this project, an

entire pipeline may potentially be represented by a single module within the VIBE client. This may be useful if it was desirable to treat a complicated pipeline as a black-box. The ability to start a pipeline executing and then close the client down could also be easily added.

### 5.3.3. Case Improvements

As discussed in Chapter 1, one area of further improvement would be the creation of a software-based reasoner on top of the current framework. The general requirements of a system such as this would be an input form that takes information describing a given problem. Then, a mechanism for evaluating similar cases within the case repository would have to be created. This would be very challenging. The reasoner may consider the input data types and the documentation files (*Help.jsp*) to find similar cases. Then, it may suggest a subset of the case repository to be used for further analysis.

A limitless number of cases could be added to the system assuming that there are appropriate VIBE SDK Module implementations. Some particularly interesting ones may be the following:

- Xppaut module – An xppaut module would allow users to solve systems of ordinary differential equations (ODEs) as well as take advantage of xppaut's other features.
- Octave module – The Octave program [70] is a free software clone of Matlab that could potentially be easily integrated to offer cases that are analogous to the Matlab cases. This may be useful if a site that requires the Matlab cases does not have a Matlab license. Additionally, it appears that Octave supports MPI [71] for parallel computation across Unix clusters. This may be useful for

very time-consuming computations since to do this within Matlab requires a separate software license for each node of the cluster.

- Additional bioinformatic tools that could be added to the system include: Glimmer [72][73], MEME/MAST [74], RepeatMasker [75], several KEGG searches [76], and a multitude of others.

Finally, additional support libraries could be constructed to aid in the generation of reports for common data types. A series of XSL stylesheets that convert StatDataML to HTML reports for different data types would be useful for depicting various views of statistical results from both the Matlab and R modules.

## APPENDIX A

### Case Hierarchy

The following hierarchical structure depicts cases that are available for exploratory analysis. In general, the cases listed have been implemented as part of this project.

However, a few headings (*e.g.*, Cluster Analysis) are included as a place-holder for cases that could be implemented in future work.

#### Structural Information

##### Nucleotide Information

What general information is known from a Nucleotide sequence?

##### Functional Domains/Motifs

##### Transcription Factors

Find possible Transcription Factor binding sites within the Nucleotide sequence.

##### Genes

Can genes within this sequence be found *in silico*?

Does my sequence contain a known curated gene?

##### Homology

What organisms have a sequence with a strong likelihood of homology?

##### Hidden Markov Models (HMM)

Perform an HMM search using nucleotide data to construct our HMM

##### MSA

Generate a Multiple Sequence Alignment from a series of input sequences

Generate a Multiple Sequence Alignment from a series of traces

##### Sequence Manipulation

Convert a nucleotide sequence into another frame

Generate a consensus sequence from a set of nucleotides

Translate a nucleotide sequence into an amino acid sequence

##### Sequence Tagged Sites

Test sequence for presence of STSs.

Search the STS database for a nucleotide sequence

##### Trace

Assemble shotgun sequencing data.

Determine the nucleotide sequence from trace information.

Generate a Multiple Sequence Alignment from a series of traces

#### Amino Acid Information

##### Homology

What organisms have a sequence with a strong likelihood of homology?

##### Mass Spectrometry

##### Raw Signal Spectrum

##### Filtering

##### Smoothing

Apply Gaussian smoothing to spectra

Apply Moving Average smoothing to spectra

##### Database Searching

- Find Amino Acid sequences from mass spectrometry data.
- Peak Detection
- Data Matrix
  - Variable Selection/Dimension Reduction
    - Determine Principal Components
    - Calculate Discriminant Coordinates
    - Classification & Cross-validation
      - Determine Leave-1-Out Cross-validation Error
      - Determine Random Permutation Cross-validation Error
      - Measure-Out Cross-validation Errors
- MSA
  - Generate a Multiple Sequence Alignment from a series of input sequences
- Sequence Manipulation
  - Generate a consensus sequence from a set of nucleotides
- Organism Information
  - Homology
    - What organisms have a sequence with a strong likelihood of homology?
- Gene Expression Information
  - EST Analysis
  - Microarray Analysis
    - Cluster Analysis
    - Classification
- Statistical
  - Matrix
    - Calculate the covariance matrix and respective eigenvalues of an input matrix
    - Generate a biplot from an input data matrix
    - Generate a screeplot from an input data matrix
  - Dissimilarity
    - Determine a 2D representation of an input dissimilarity matrix
  - Clustering
    - Perform agglomerative clustering on an input dissimilarity matrix
  - Correlation
    - Display correlation matrix in 1 dimension
    - Display correlation matrix in 1 dimension and separated by groups
- Variable Selection/Dimension Reduction
  - Determine Principal Components
  - Calculate Discriminant Coordinates
  - Classification & Cross-validation
    - Determine Leave-1-Out Cross-validation Error
    - Determine Random Permutation Cross-validation Error
    - Measure-Out Cross-validation Errors

## APPENDIX B

### Example Input.jsp

The following code demonstrates an example *Input.jsp* file and the way it appears through a web browser (see Figure 25). This also shows how the *GenericCaseForm* is used to set values where we can access them from a *RecipeImpl*.

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/incogen.tld" prefix="incogen" %>

<%@include file="/header.html" %>

<html:errors/>

<h2><incogen:recipeTitle/></h2>
<html:form action="/executeCase" enctype="multipart/form-data" >
 <h3><bean:message key="case.input.heading"/></h3>
 <h4>(Note: Either Nucleotide or Amino Acid data are valid input)</h4>
 <table border="1" width="100%" cellpadding="2" cellspacing="2"
 style="text-align: left; margin-left: 0px; margin-right: auto;">
 <tbody>
 <tr>
 <td style="text-align: right;">FASTA Text: </td>
 <td><html:textarea cols="80" rows="10" name="case.input"
 property="inputValue(inputTArea)"/>
 </td>
 </tr>
 <tr>
 <td style="text-align: right;">Or input the FASTA file:</td>
 <td><html:file size="80" name="case.input"
 property="inputValue(inputFile)"/>
 </td>
 </tr>
 </tbody>
 </table>
 <h3><bean:message key="case.parameters.heading"/></h3>
 <!-- START PARAMETERS -->
 <table border="1" width="100%" cellpadding="2" cellspacing="2"
 style="text-align: left; margin-left: 0px; margin-right: auto;">
 <tbody>
 <tr>
 <td style="text-align: right;">Search Entrez with input sequence
 </td>
 <td>
 <html:checkbox property="parameterValue(searchEntrez)"/>
 </td>
 </tr>
 </tbody>
 </table>
```



```

</tr>
<tr>
 <td style="text-align: right;">Expectation Value</td>
 <td>
 <html:text property="parameterValue(expectationValue)"
 size="3"/>
 </td>
</tr>
<tr>
 <td style="text-align: right;">Word Size</td>
 <td>
 <html:select property="parameterValue(wordSize)" size="1">
 <html:optionsCollection
 property="parameterValue(wordSizeList)"/>
 </html:select>
 </td>
</tr>
<tr>
 <td style="text-align: right;">Target Databases</td>
 <td>
 <html:select property="parameterValue(selectedDbList)"
 multiple="true" size="5">
 <html:optionsCollection property="parameterValue(dbList)"/>
 </html:select>
 </td>
</tr>
</tbody>
</table>
<!-- END PARAMETERS -->

<html:submit property="method" value="Submit"/>
<html:submit property="method" value="Advanced"/>
<html:reset/>
</html:form>
<incogen:caseLinkBar/>
<%@include file="/footer.html" %>

```

The first line indicates that the file should be interpreted as a Java Server Page.

The lines containing “taglib” indicate the location of tag library descriptors. This defines the tags that we can use within this file that will be recognized by the JSP processor.

After that is a line that defines the *header.html* file. This includes the content from the file at the beginning of the processed file. There is a similar *footer.html* file defined at the tail of this file. The `html:errors` line that follows is a Struts tag that will inline any errors detected when this form is submitted. After that is an HTML element (`h2`) that surrounds an `incogen:recipeTitle`. This places the case's description in a

header block and was implemented for this project.

The `html:form` element contains all the variables that will be submitted to the server for this case execution. The values of interest in this example are `inputValue(inputTArea)`, `inputValue(inputFile)`, `parameterValue(searchEntrez)`, `parameterValue(expectationValue)`, `parameterValue(wordSizeList)`, and `parameterValue(dbList)`. These values were initialized in the *DefaultRecipeImpl::initializeForm* function (not shown).

When the web form is submitted, Struts will assign the values `inputValue(inputTArea)`, `inputValue(inputFile)`, `parameterValue(searchEntrez)`, `parameterValue(expectationValue)`, `parameterValue(wordSize)`, and `parameterValue(selectedDbList)` to the *GenericCaseForm* with the values from the web form. These values can then be used in the *DefaultRecipeImpl::loadPipelineModel* method to alter the pipeline that will actually be executed for this case. The `<!-- START PARAMETERS -->` and `<!-- END PARAMETERS -->` are comments that are used by the default cookbook `Input.jsp` to pull out the important parameters from child cases.

**What organisms have a sequence with strong likelihood of homology?**

**Pipeline Input**

(Note: Either Nucleotide or Amino Acid data are valid input)

FASTA Text:	
Or input the FASTA file:	<input type="button" value="Browse..."/>

**Pipeline Parameters**

Search Entrez with Input sequence	<input checked="" type="checkbox"/>
Expectation Value	0.1
Word Size	3
Target Databases	<input type="checkbox"/> D_melanogaster <input type="checkbox"/> E_coli_refseq



**Figure 25. Appendix B's Input.jsp as it appears through a web browser.**

## APPENDIX C

### Selected Javadocs

The following documentation was generated using the Javadoc [77] tool on the respective classes. All the classes that are part of the Web Access framework have documentation and this is accessible from the Web Access Javadocs link.

#### **com.incogen.vibe.server.webaccess.recipes** **Interface Recipe**

**All Known Subinterfaces:**

[Cookbook](#)

**All Known Implementing Classes:**

[AbstractRecipe](#), [DefaultCookbookImpl](#)

public interface **Recipe**

This is the basic interface for all case implementations in the VIBE Web Access framework. Developers should typically extend the DefaultRecipeImpl when implementing their own cases.

**Author:**

J. A. Hayes

Field Summary	
static java.lang.String	<a href="#">DESCRIPTION_FILE</a> The name of the recipe's description file, usually 'description.txt'.
static java.lang.String	<a href="#">HELP_JSP_FILE</a> The name of the recipe's help file, usually 'Help.jsp'.
static java.lang.String	<a href="#">INIT_PY_FILE</a> The name of the Jython package file, usually '__init__.py'.
static java.lang.String	<a href="#">INPUT_JSP_FILE</a> The name of the recipe's basic input file, usually 'Input.jsp'.
static java.lang.String	<a href="#">POST_PROCESSED_ATTR</a>
static java.lang.String	<a href="#">RECIPE_FILE</a> The name of the recipe's Jython implementation file, usually 'RecipeImpl.py'.

static java.lang.String	<a href="#">RECIPE_FILENAME</a> The name of the recipe's Jython implementation file w/o the file extension, usually 'RecipeImpl'.
static java.lang.String	<a href="#">TEMPLATE_VPML_FILE</a> The name of the recipe's template pipeline file (if there is one), usually 'template.vpml'.

Method Summary	
com.inco gen.vibe. pipeline. Pipeline	<a href="#">generatePipeline</a> (javax.servlet.http.HttpServletRequest request) Generates a new pipeline from scratch.
com.inco gen.vibe. pipeline. Pipeline	<a href="#">generatePreviewPipeline</a> (javax.servlet.http.HttpServletRequest request) Should return a new instance of a pipeline capturing the most general structure of the case's pipeline.
void	<a href="#">generateView</a> (com.incogen.vibe.pipeline.Pipeline pipeline, java.io.Writer writer) By default, should generate a generic report based on Module->Report in a portal view.
java.lan g.String	<a href="#">getAttribute</a> (java.lang.String name) Retrieves an arbitrary value that has been previously associated with this execution of the case.
java.uti l.Map	<a href="#">getAttributeMap</a> () Retrieves the map of attribute names and values associated with this case.
ActionFo rward	<a href="#">getDocumentationForward</a> () Make a default implementation search for the help in the same directory as the recipe.
ActionFo rward	<a href="#">getExampleInputView</a> (ActionMapping mapping, javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Returns the Input form with example input initialized to it.
ActionFo rward	<a href="#">getInputView</a> (ActionMapping mapping, javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) By default, should return the input jsp in the same directory as the recipe.
com.inco gen.vibe. pipeline. Pipeline	<a href="#">getPipeline</a> (javax.servlet.http.HttpServletRequest request) Returns a previously set pipeline (that may have had default parameter changes).
java.lan g.String	<a href="#">getRecipeClassname</a> () Gets the classname of the Jython recipe.
java.lan g.String	<a href="#">getRecipeDescription</a> () This retrieves the text that will appear in the Web Access Case list.
java.lan g.String	<a href="#">getRecipeFilepath</a> () Returns the path to the Recipe's Jython file relative to the webapp's context.
java.lan g.String	<a href="#">getRecipeKeyword</a> () Retrieves the unique keyword associated with this recipe.
boolean	<a href="#">hasPostProcessed</a> () Returns true if postprocess has been called on this run of the case.

void	<a href="#">postprocess</a> (com.incogen.vibe.pipeline.Pipeline pipeline) This method gives the Recipe developer the opportunity to process the pipeline after its execution has been completed.
java.lang.String	<a href="#">removeAttribute</a> (java.lang.String name) Removes a previously set attribute that was associated with this execution run.
void	<a href="#">setAttribute</a> (java.lang.String name, java.lang.String value) Associates an arbitrary attribute name with it's value.
void	<a href="#">setAttributeMap</a> (java.util.Map attributeMap) Sets the map of attribute names and values associated with this case.
void	<a href="#">setPostProcessed</a> (boolean tf) Sets whether or not postprocess has been called on this run of the case.
void	<a href="#">setRecipeClassname</a> (java.lang.String classname) Sets the classname of the Jython recipe.
void	<a href="#">setRecipeDescription</a> (java.lang.String description) This sets the text that will appear in the Web Access Case hierarchy.
void	<a href="#">setRecipeFilepath</a> (java.lang.String filepath) Sets the path to the Recipe's Jython file relative to the webapp's context.
void	<a href="#">setRecipeKeyword</a> (java.lang.String recipeKeyword) Sets the unique keyword associated with this recipe.
void	<a href="#">ValidateInputForm</a> (com.incogen.vibe.server.webaccess.struts.GenericCaseForm genericCaseForm, ActionErrors errors) This method gives the user the opportunity to determine if the input values and parameter values are valid for this case.

#### Field Detail

##### DESCRIPTION\_FILE

```
public static final java.lang.String DESCRIPTION_FILE
```

The name of the recipe's description file, usually 'description.txt'.

**See Also:**

[Constant Field Values](#)

##### HELP\_JSP\_FILE

```
public static final java.lang.String HELP_JSP_FILE
```

The name of the recipe's help file, usually 'Help.jsp'.

**See Also:**

[Constant Field Values](#)

**INIT\_PY\_FILE**

```
public static final java.lang.String INIT_PY_FILE
```

The name of the Jython package file, usually '\_\_init\_\_.py'.

**See Also:**

[Constant Field Values](#)

---

**INPUT\_JSP\_FILE**

```
public static final java.lang.String INPUT_JSP_FILE
```

The name of the recipe's basic input file, usually 'Input.jsp'.

**See Also:**

[Constant Field Values](#)

---

**POST\_PROCESSED\_ATTR**

```
public static final java.lang.String POST_PROCESSED_ATTR
```

**See Also:**

[Constant Field Values](#)

---

**RECIPE\_FILE**

```
public static final java.lang.String RECIPE_FILE
```

The name of the recipe's Jython implementation file, usually 'RecipeImpl.py'.

**See Also:**

[Constant Field Values](#)

---

**RECIPE\_FILENAME**

```
public static final java.lang.String RECIPE_FILENAME
```

The name of the recipe's Jython implementation file w/o the file extension, usually 'RecipeImpl'.

**See Also:**

[Constant Field Values](#)

---

**TEMPLATE\_VPML\_FILE**

```
public static final java.lang.String TEMPLATE_VPML_FILE
```

The name of the recipe's template pipeline file (if there is one), usually 'template.vpml'.

**See Also:**

[Constant Field Values](#)

**Method Detail****generatePipeline**

```
public com.incogen.vibe.pipeline.Pipeline generatePipeline
(javax.servlet.http.HttpServletRequest request)
```

Generates a new pipeline from scratch. By default, this should parse the pipeline from the filesystem in some specified location and return that.

**generatePreviewPipeline**

```
public com.incogen.vibe.pipeline.Pipeline generatePreviewPipeline
(javax.servlet.http.HttpServletRequest request)
```

Should return a new instance of a pipeline capturing the most general structure of the case's pipeline. By default, this returns the same Pipeline from Recipe::generatePipeline.

**generateView**

```
public void generateView(com.incogen.vibe.pipeline.Pipeline pipeline,
java.io.Writer writer)
```

By default, should generate a generic report based on Module->Report in a portal view. Note, this will be called each time the Format Results page is used to generate a Case-Specific report. So, it may be desirable for time-consuming portions of this code to be moved into the implementation of the postprocess method.

**Parameters:**

`pipeline` - Pipeline - The executed pipeline for this case.

`writer` - Writer - The writer that writes to this case's specific output report. Send either HTML or raw text to this.

**getAttribute**

```
public java.lang.String getAttribute(java.lang.String name)
```

Retrieves an arbitrary value that has been previously associated with this execution of the case. For example, it might be useful to set an attribute in DefaultRecipeImpl::loadPipelineModel that affects the view generated in DefaultRecipeImpl::generateView.



---

**getAttributeMap**

```
public java.util.Map getAttributeMap()
```

Retrieves the map of attribute names and values associated with this case.

---

**getDocumentationForward**

```
public ActionForward getDocumentationForward()
```

Make a default implementation search for the help in the same directory as the recipe.

---

**getExampleInputView**

```
public ActionForward getExampleInputView(ActionMapping mapping,
 javax.servlet.http.HttpServletRequest request,
 javax.servlet.http.HttpServletResponse response)
```

Returns the Input form with example input initialized to it.

---

**getInputView**

```
public ActionForward getInputView(ActionMapping mapping,
 javax.servlet.http.HttpServletRequest request,
 javax.servlet.http.HttpServletResponse response)
```

By default, should return the input.jsp in the same directory as the recipe.

---

**getPipeline**

```
public com.incogen.vibe.pipeline.Pipeline getPipeline
(javax.servlet.http.HttpServletRequest request)
```

Returns a previously set pipeline (that may have had default parameter changes). If there is none, it should create a new instance using `generatePipeline` and set it to the pipeline attribute for the recipe. By default, returns the same Pipeline as `Recipe::generatePipeline`.

---

**getRecipeClassname**

```
public java.lang.String getRecipeClassname()
```

Gets the classname of the Jython recipe. For example, "recipes.template\_recipe.TemplateRecipe.RecipeImpl". Therefore, this should be a standard java fully-qualified classname. Initially, this is set by the Web-Access framework.

---

**getRecipeDescription**

```
public java.lang.String getRecipeDescription()
```

This retrieves the text that will appear in the Web Access Case list.

**getRecipeFilepath**

```
public java.lang.String getRecipeFilepath()
```

Returns the path to the Recipe's Jython file relative to the webapp's context. For example, for `recipes.template_recipe.TemplateRecipe.RecipeImpl` you'll get returned: `/scripts/recipes/template_recipe/TemplateRecipe` This is where the framework expects to find the input jsps, help jsps, recipe script file, and any `template.vpml` files.

**getRecipeKeyword**

```
public java.lang.String getRecipeKeyword()
```

Retrieves the unique keyword associated with this recipe.

**hasPostProcessed**

```
public boolean hasPostProcessed()
```

Returns true if `postprocess` has been called on this run of the case.

**postprocess**

```
public void postprocess(com.incogen.vibe.pipeline.Pipeline pipeline)
```

This method gives the Recipe developer the opportunity to process the pipeline after its execution has been completed. Typically, this may be when there is a time-consuming step that is required for the generation of a report, and we'd like to go ahead and do it after our pipeline has completed execution. This will be called once after the analysis pipeline has completed.

**removeAttribute**

```
public java.lang.String removeAttribute(java.lang.String name)
```

Removes a previously set attribute that was associated with this execution run.

**setAttribute**

```
public void setAttribute(java.lang.String name,
 java.lang.String value)
```

Associates an arbitrary attribute name with its value. For example, it might be useful to set an attribute in `DefaultRecipeImpl::loadPipelineModel` that affects the view generated in `DefaultRecipeImpl::generateView`.

**setAttributeMap**

```
public void setAttributeMap(java.util.Map attributeMap)
```

Sets the map of attribute names and values associated with this case.

**setPostProcessed**

```
public void setPostProcessed(boolean tf)
```

Sets whether or not `postprocess` has been called on this run of the case.

---

**setRecipeClassname**

```
public void setRecipeClassname(java.lang.String classname)
```

Sets the classname of the Jython recipe. For example, `"recipes.template_recipe.TemplateRecipe.RecipeImpl"`. Therefore, this should be a standard java fully-qualified classname. Initially, this is set by the Web-Access framework.

**setRecipeDescription**

```
public void setRecipeDescription(java.lang.String description)
```

This sets the text that will appear in the Web Access Case hierarchy.

**setRecipeFilepath**

```
public void setRecipeFilepath(java.lang.String filepath)
```

Sets the path to the Recipe's Jython file relative to the webapp's context. For example, for `recipes.template_recipe.TemplateRecipe.RecipeImpl` you should define: `/scripts/recipes/template_recipe/TemplateRecipe` This is where the framework expects to find the input jsps, help jsps, recipe script file, and any `template.vpml` files.

**setRecipeKeyword**

```
public void setRecipeKeyword(java.lang.String recipeKeyword)
```

Sets the unique keyword associated with this recipe.

**validateInputForm**

```
public void validateInputForm
(com.incogen.vibe.server.webaccess.struts.GenericCaseForm genericCaseForm,
ActionErrors errors)
```

This method gives the user the opportunity to determine if the input values and parameter values are valid for this case. If they're not, an appropriate error should be added to the ActionErrors argument.

**com.incogen.vibe.server.webaccess.recipes****Class RecipeUtils**

```
java.lang.Object
|
+--com.incogen.vibe.server.webaccess.recipes.RecipeUtils
```

```
public class RecipeUtils
extends java.lang.Object
```

A set of utility functions to be used by Recipe developers.

**Author:**

J. A. Hayes

Method Summary	
static java.lang.String	<a href="#">getAbsoluteRecipeFilepath</a> (java.lang.String recipeClassname) Returns the full path to the Recipe's Jython file.
static java.util.List	<a href="#">getInputLocationsForModule</a> (com.incogen.vibe.openapi.Module module) Returns all the input locations for the given module.
static java.util.List	<a href="#">getOutputLocationsForModule</a> (com.incogen.vibe.openapi.Module module) Returns all the output locations for the given module.
static java.util.List	<a href="#">getParameterNames</a> (com.incogen.vibe.openapi.Module module) Convenience method for getting all the parameter names of a VIBE Module.
static java.lang.String	<a href="#">getParameterValue</a> (com.incogen.vibe.openapi.Module module, java.lang.String name) Convenience method for getting a parameter value from a VIBE Module.

static java.util.List	<a href="#">getParameterValueAsList</a> (com.incogen.vibe.openapi.Module module, java.lang.String name) Convenience method for getting a parameter value from a VIBE Module.
static com.incogen.vibe.gui.ParameterSelectionList	<a href="#">getParameterValueAsSelectionList</a> (com.incogen.vibe.openapi.Module module, java.lang.String name) Convenience method for getting a parameter value from a VIBE Module.
static java.lang.String	<a href="#">getRecipeDescription</a> (java.lang.String filepath) Retrieves any text from the description file (Recipe.DESCRPTION_FILE) in the directory specified as argument.
static java.lang.String	<a href="#">getRecipeDescriptionByClassname</a> (java.lang.String recipeClassname) Retrieves the recipe description associated with the argument.
static java.lang.String	<a href="#">getRecipeFilepath</a> (java.lang.String recipeClassname) Returns the path to the Recipe's Jython file relative to the webapp's context.
static java.lang.String	<a href="#">getRecipeKeyword</a> (java.lang.String recipeClassname) Retrieves the keyword (the directory name) of the recipe defined by the argument.
static boolean	<a href="#">haveInputData</a> (java.util.Map inputMap) Checks the input Map for any data in a very primitive way.
static void	<a href="#">setInputLocationsForModule</a> (com.incogen.vibe.openapi.Module module, java.util.List inputLocations) Sets all the input locations for the given module using the types of the previous input descriptors.
static void	<a href="#">setOutputLocationsForModule</a> (com.incogen.vibe.openapi.Module module, java.util.List outputLocations) Sets all the output locations for the given module using the types of the previous input descriptors.
static void	<a href="#">setParameterValue</a> (com.incogen.vibe.openapi.Module module, java.lang.String name, java.util.ArrayList value) Convenience method for setting a parameter value on a VIBE Module.
static void	<a href="#">setParameterValue</a> (com.incogen.vibe.openapi.Module module, java.lang.String name, LabelValueBean value, java.util.List list) Convenience method for setting a parameter value on a VIBE Module.
static void	<a href="#">setParameterValue</a> (com.incogen.vibe.openapi.Module module, java.lang.String name, java.lang.String value) Convenience method for setting a parameter value on a VIBE Module.
static void	<a href="#">setParameterValueAsSelectionList</a> (com.incogen.vibe.openapi.Module module, java.lang.String name, java.util.ArrayList value) Convenience method for setting a parameter value on a VIBE Module.
static void	<a href="#">setParameterValueAsSelectionList</a> (com.incogen.vibe.openapi.Module module, java.lang.String name, java.lang.String value) Convenience method for setting a parameter value on a VIBE Module.

#### Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,
wait, wait, wait
```

### Method Detail

#### **getAbsoluteRecipeFilepath**

```
public static final java.lang.String getAbsoluteRecipeFilepath(java.lang.String
recipeClassname)
```

Returns the full path to the Recipe's Jython file. For example, for `recipes.template_recipe.TemplateRecipe.RecipeImpl` you'll get returned: `/usr/local/vibe/jakarta-tomcat-4.1.27/webapps/webaccess-server/scripts/recipes/template_recipe/TemplateRecipe` This is where the framework expects to find the input jsps, help jsps, recipe script file, and any `template.vpml` files.

#### **getInputLocationsForModule**

```
public static final java.util.List getInputLocationsForModule
(com.incogen.vibe.openapi.Module module)
```

Returns all the input locations for the given module.

**Returns:**

List - A list of String objects representing input locations

#### **getOutputLocationsForModule**

```
public static final java.util.List getOutputLocationsForModule
(com.incogen.vibe.openapi.Module module)
```

Returns all the output locations for the given module.

**Returns:**

List - A list of String objects representing output locations

#### **getParameterNames**

```
public static final java.util.List getParameterNames
(com.incogen.vibe.openapi.Module module)
```

Convenience method for getting all the parameter names of a VIBE Module. These can be used to retrieve the parameter values from any of the `getParameterValue` methods.

#### **getParameterValue**

```
public static final java.lang.String getParameterValue
```

(com.incogen.vibe.openapi.Module module, java.lang.String name)  
 Convenience method for getting a parameter value from a VIBE Module.

#### **getParameterValueAsList**

```
public static final java.util.List getParameterValueAsList
(com.incogen.vibe.openapi.Module module, java.lang.String name)
 Convenience method for getting a parameter value from a VIBE Module.
```

#### **getParameterValueAsSelectionList**

```
public static final com.incogen.vibe.gui.ParameterSelectionList
getParameterValueAsSelectionList(com.incogen.vibe.openapi.Module module,
java.lang.Stringname)
```

Convenience method for getting a parameter value from a VIBE Module.

#### **getRecipeDescription**

```
public static java.lang.String getRecipeDescription(java.lang.String filepath)
```

Retrieves any text from the description file (Recipe.DESCRPTION\_FILE) in the directory specified as argument. Typically, it is easier to use the function `RecipeUtils::getRecipeDescriptionByClassname()`.

#### **getRecipeDescriptionByClassname**

```
public static java.lang.String getRecipeDescriptionByClassname(java.lang.String
recipeClassname)
```

Retrieves the recipe description associated with the argument.

##### **Parameters:**

recipeClassname - String - The recipeClassname in the form  
 recipes.some.location.RecipeImpl

#### **getRecipeFilepath**

```
public static final java.lang.String getRecipeFilepath(java.lang.String
recipeClassname)
```

Returns the path to the Recipe's Jython file relative to the webapp's context. For example, for `recipes.template_recipe.TemplateRecipe.RecipeImpl` you'll get returned: `/scripts/recipes/template_recipe/TemplateRecipe` This is where the framework expects to find the input jsps, help jsps, recipe script file, and any template.vpml files.

---

**getRecipeKeyword**

```
public static final java.lang.String getRecipeKeyword(java.lang.String
recipeClassname)
```

Retrieves the keyword (the directory name) of the recipe defined by the argument.

**haveInputData**

```
public static final boolean haveInputData(java.util.Map inputMap)
```

Checks the input Map for any data in a very primitive way. If there is a FormFile as a value, it will check that the file size is > 0. If there is a String it will verify that the string has length > 0. It doesn't know how to handle other data types.

---

**setInputLocationsForModule**

```
public static final void setInputLocationsForModule
(com.incogen.vibe.openapi.Module module, java.util.List inputLocations)
 Sets all the input locations for the given module using the types of the previous input descriptors.
```

---

**setOutputLocationsForModule**

```
public static final void setOutputLocationsForModule
(com.incogen.vibe.openapi.Module module, java.util.List outputLocations)
 Sets all the output locations for the given module using the types of the previous input descriptors.
```

**setParameterValue**

```
public static final void setParameterValue(com.incogen.vibe.openapi.Module
module, java.lang.String name, java.util.ArrayListvalue)
 Convenience method for setting a parameter value on a VIBE Module.
```

---

**setParameterValue**

```
public static final void setParameterValue(com.incogen.vibe.openapi.Module
module, java.lang.String name, LabelValueBean value, java.util.List list)
 Convenience method for setting a parameter value on a VIBE Module.
```

---

**setParameterValue**

```
public static final void setParameterValue(com.incogen.vibe.openapi.Module
module, java.lang.String name, java.lang.Stringvalue)
```



Convenience method for setting a parameter value on a VIBE Module.

#### **setParameterValueAsSelectionList**

```
public static final void setParameterValueAsSelectionList
(com.incogen.vibe.openapi.Module module, java.lang.String name,
java.util.ArrayList value)
```

Convenience method for setting a parameter value on a VIBE Module.

#### **setParameterValueAsSelectionList**

```
public static final void setParameterValueAsSelectionList
(com.incogen.vibe.openapi.Module module, java.lang.String name,
java.lang.String value)
```

Convenience method for setting a parameter value on a VIBE Module.

## **com.incogen.vibe.server.webaccess.recipes**

### **Class DefaultRecipeImpl**

```
java.lang.Object
|
+-- com.incogen.vibe.server.webaccess.recipes.AbstractRecipe
|
+-- com.incogen.vibe.server.webaccess.recipes.DefaultRecipeImpl
```

#### **All Implemented Interfaces:**

[Recipe](#)

#### **Direct Known Subclasses:**

[DefaultCookbookImpl](#)

```
public class DefaultRecipeImpl
extends AbstractRecipe
```

Implements a complete set of Recipe functions. Developers should typically extend this class when implementing their own cases. In particular, developers may want to override the following functions for the listed reasons:

- **initializeForm** - Override this function if you would like to initialize your basic input form for this case with some parameters. It may be desirable to call `DefaultRecipeImpl::generatePipelineModel` and set it to the 'pipeline' class field so that you can determine what parameters are available with a module at runtime.
- **generatePipelineModel** - This is a good place to programmatically define a pipeline if its generation DOES NOT depend on any input parameters provided by the user. Also, whatever pipeline is implemented for this function will be provided as a preview schematic of the execution.
- **loadPipelineModel** - This is a good place to programmatically define a pipeline if its generation DOES depend on any input parameters provided by the user. You'll probably also want to call `DefaultRecipeImpl.loadPipelineModel(self, pipelineModel, genericCaseForm)` at the end of your code for this function to take advantage of any parameter mappings in the Jelly script.
- **generateView** - Override this function if you would like to change the report view of your case. This will be called everytime we print out a case-specific report.
- **postprocess** - This method gives the Recipe developer the opportunity to process the pipeline after its

execution has been completed. Typically, this may be when there is a time-consuming step that is required for the generation of a report, and we'd like to go ahead and do it after our pipeline has completed execution. This will be called once after the analysis pipeline has completed.

**Author:**

J. A. Hayes

Field Summary	
<b>Fields inherited from class <code>com.incogen.vibe.server.webaccess.recipesAbstractRecipe</code></b>	
<a href="#">attributes</a> , <a href="#">pipeline</a> , <a href="#">recipeClassname</a> , <a href="#">recipeDescription</a> , <a href="#">recipeFilepath</a> , <a href="#">recipeKeyword</a>	
<b>Fields inherited from interface <code>com.incogen.vibe.server.webaccess.recipesRecipe</code></b>	
<a href="#">DESCRIPTION_FILE</a> , <a href="#">HELP_JSP_FILE</a> , <a href="#">INIT_PY_FILE</a> , <a href="#">INPUT_JSP_FILE</a> , <a href="#">POST_PROCESSED_ATTR</a> , <a href="#">RECIPE_FILE</a> , <a href="#">RECIPE_FILENAME</a> , <a href="#">TEMPLATE_VPML_FILE</a>	
Constructor Summary	
<a href="#">DefaultRecipeImpl()</a> Creates a new instance of DefaultRecipeImpl	
Method Summary	
com.in cogen.  vibe.pi pipeline. Pipeline	<a href="#">generatePipeline</a> ( <code>javax.servlet.http.HttpServletRequest request</code> ) Generates a new pipeline from scratch.
com.in cogen.  vibe.pi pipeline. Pipeline	<a href="#">generatePipelineModel</a> () Generates a new pipeline from scratch.
void	<a href="#">generateView</a> ( <code>com.incogen.vibe.pipeline.Pipeline pipeline</code> , <code>java.io.Writer writer</code> ) By default, generates a generic report based on Module->Report in a portal view.
Action Forward	<a href="#">getDocumentationForward</a> () The default implementation of this searches for the Recipe.HELP_JSP_FILE in the same directory as the recipe.
Action Forward	<a href="#">getExampleInputView</a> ( <code>ActionMapping mapping</code> , <code>javax.servlet.http.HttpServletRequest request</code> , <code>javax.servlet.http.HttpServletResponse response</code> ) Returns the Input form with example input initialized to it.

protected java.lang. String	<a href="#">getExampleText</a> (java.lang.String keyword) Returns the text of a file from the recipe's directory with the pattern {source_keyword}.txt.
Action Forward	<a href="#">getInputView</a> (ActionMapping mapping, javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) By default, returns the Recipe.INPUT_JSP_FILE in the same directory as the recipe after calling DefaultRecipeImpl::initializeForm.
void	<a href="#">initializeExampleForm</a> (javax.servlet.http.HttpServletRequest request, com.incogen.vibe.server.webaccess.struts.GenericCaseForm form) Attempts to find the source module for the pipeline and load the example text from a file using the getExampleText method.
void	<a href="#">initializeForm</a> (javax.servlet.http.HttpSession session, com.incogen.vibe.server.webaccess.struts.GenericCaseForm form) Allows a developer to initialize the basic input form for the recipe with any value.
void	<a href="#">loadPipelineModel</a> (com.incogen.vibe.pipeline.Pipeline pipelineModel, com.incogen.vibe.server.webaccess.struts.GenericCaseForm genericCaseForm) This function loads the pipelineModel with values from the GenericCaseForm (that came from the form submission).

#### Methods inherited from class com.incogen.vibe.server.webaccess.recipesAbstractRecipe

[generatePreviewPipeline](#), [getAttribute](#), [getAttributeMap](#), [getPipeline](#),  
[getRecipeClassname](#), [getRecipeDescription](#), [getRecipeFilepath](#), [getRecipeKeyword](#),  
[hasPostProcessed](#), [postprocess](#), [removeAttribute](#), [setAttribute](#), [setAttributeMap](#),  
[setPostProcessed](#), [setRecipeClassname](#), [setRecipeDescription](#), [setRecipeFilepath](#),  
[setRecipeKeyword](#), [validateInputForm](#)

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,  
wait, wait, wait

#### Constructor Detail

##### DefaultRecipeImpl

```
public DefaultRecipeImpl()
```

Creates a new instance of DefaultRecipeImpl

#### Method Detail

##### generatePipeline

```
public com.incogen.vibe.pipeline.Pipeline generatePipeline
(javax.servlet.http.HttpServletRequest request)
```

Generates a new pipeline from scratch. By default, calls `DefaultRecipeImpl::generatePipelineModel` if the recipe's pipeline field is null. Then calls `DefaultRecipeImpl::loadPipelineModel` with the pipeline field.

**Specified by:**

[generatePipeline](#) in interface [Recipe](#)

**Specified by:**

[generatePipeline](#) in class [AbstractRecipe](#)

### **generatePipelineModel**

```
public com.incogen.vibe.pipeline.Pipeline generatePipelineModel ()
```

Generates a new pipeline from scratch. By default, this parses the pipeline from the filesystem in the recipe's directory with the name whose value matches `Recipe.TEMPLATE_VPML_FILE` (probably 'template.vmpl') specified location and returns that.

DEVELOPER'S NOTE: This is a good place to programmatically define a pipeline if its generation DOES NOT depend on any input parameters provided by the user. Also, whatever pipeline is implemented for this function will be provided as a preview schematic of the execution if `generatePreviewPipeline` isn't overridden as well.

### **generateView**

```
public void generateView (com.incogen.vibe.pipeline.Pipeline pipeline,
 java.io.Writer writer)
```

By default, generates a generic report based on `Module->Report` in a portal view. Uses the script in `Recipe.DEFAULT_PIPELINE_TO_HTML_FORMATTER` to generate the view.

DEVELOPER'S NOTE: Override this function if you would like to change the report view of your case.

**Specified by:**

[generateView](#) in interface [Recipe](#)

**Specified by:**

[generateView](#) in class [AbstractRecipe](#)

**Parameters:**

`pipeline` - Pipeline - The executed pipeline for this case.

`writer` - Writer - The writer that writes to this case's specific output report. Send either HTML or raw text to this.

### **getDocumentationForward**

```
public ActionForward getDocumentationForward ()
```

The default implementation of this searches for the `Recipe.HELP_JSP_FILE` in the same directory as the recipe.

**Specified by:**

[getDocumentationForward](#) in interface [Recipe](#)

**Specified by:**

[getDocumentationForward](#) in class [AbstractRecipe](#)

### **getExampleInputView**

```
public ActionForward getExampleInputView(ActionMapping mapping,
 javax.servlet.http.HttpServletRequest
request,
 javax.servlet.http.HttpServletResponse
response)
```

Returns the Input form with example input initialized to it. This is accomplished by first calling DefaultRecipeImpl::initializeForm and then calling DefaultRecipeImpl::initializeExampleForm.

**Specified by:**

[getExampleInputView](#) in interface [Recipe](#)

**Specified by:**

[getExampleInputView](#) in class [AbstractRecipe](#)

### **getExampleText**

```
protected java.lang.String getExampleText(java.lang.String keyword)
```

Returns the text of a file from the recipe's directory with the pattern {source\_keyword}.txt. If the file cannot be found there, it attempts to return the text from a file with the same pattern in AppConfiguration.getInstance().getBasePath() + File.separator + WebAccessFileUtils.RECIPE\_INPUT\_EXAMPLES\_DIR.

### **getInputView**

```
public ActionForward getInputView(ActionMapping mapping,
 javax.servlet.http.HttpServletRequest request,
 javax.servlet.http.HttpServletResponse
response)
```

By default, returns the Recipe.INPUT\_JSP\_FILE in the same directory as the recipe after calling DefaultRecipeImpl::initializeForm.

**Specified by:**

[getInputView](#) in interface [Recipe](#)

**Specified by:**

[getInputView](#) in class [AbstractRecipe](#)

### **initializeExampleForm**

```
public void initializeExampleForm(javax.servlet.http.HttpServletRequest request,
 com.incogen.vibe.server.webaccess.struts.Gene
ricCaseForm form)
```

Attempts to find the source module for the pipeline and load the example text from a file using the `getExampleText` method. If a source module cannot be found, the method currently uses `Sequence_nt` by default. See `DefaultRecipeImpl::getExampleText` for a description of how the example text is retrieved by default.

#### **initializeForm**

```
public void initializeForm(javax.servlet.http.HttpSession session,
 com.incogen.vibe.server.webaccess.struts.GenericCase
Form form)
```

Allows a developer to initialize the basic input form for the recipe with any value.

DEVELOPER'S NOTE: Override this function if you would like to initialize your basic input form for this case with some parameters. It may be desirable to call `DefaultRecipeImpl::generatePipelineModel` and set it to the 'pipeline' class field so that you can determine what parameters are available with a module at runtime.

#### **loadPipelineModel**

```
public void loadPipelineModel(com.incogen.vibe.pipeline.Pipeline pipelineModel,
 com.incogen.vibe.server.webaccess.struts.GenericC
aseForm genericCaseForm)
```

This function loads the `pipelineModel` with values from the `GenericCaseForm` (that came from the form submission). By default, this executes the Jelly script at `'WebAccessFileUtils.FULL_CONFIGURATION_DIR + File.separator + PIPELINEMODEL_MAP_XML'`.

DEVELOPER'S NOTE: This is a good place to programmatically define a pipeline if its generation DOES depend on any input parameters provided by the user. You'll probably also want to call `DefaultRecipeImpl.loadPipelineModel(self, pipelineModel, genericCaseForm)` at the end of your code for this function to take advantage of any parameter mappings in the Jelly script.

## Bibliography

- 1: National Institute of Standards and Technology Advanced Technology Program - <http://www.atp.nist.gov>
- 2: INCOGEN - <http://www.incogen.com>
- 3: VIBE - Visual Integrated Bioinformatics Environment - <http://www.incogen.com/index.php?type=Product&param=VIBE>
- 4: Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. (1990) Basic local alignment search tool. *J. Mol. Biol.* 215:403-10. <http://www.ncbi.nih.gov/BLAST/>
- 5: Thompson JD, Higgins DG, Gibson TJ. (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.* 22:4673-80. <http://www.ebi.ac.uk/clustalw/>
- 6: Eddy SR. (2001) HMMER: Profile hidden Markov models for biological sequence analysis. <http://hmmer.wustl.edu/>
- 7: Perl - <http://www.perl.com/>
- 8: Stein, LD. (2003) Integrating Biological Databases. *Nature Reviews Genetics* 4:337-345.
- 9: GeneCards: encyclopedia for genes, proteins and diseases. Weizmann Institute of Science, Bioinformatics Unit and Genome Center (Rehovot, Israel), 1997. <http://bioinformatics.weizmann.ac.il/cards>
- 10: Genome Analysis Pipeline - <http://compbio.ornl.gov/tools/pipeline/>
- 11: GrailEXP - <http://compbio.ornl.gov/grailexp/>
- 12: Genscan - <http://genes.mit.edu/GENSCAN.html>
- 13: Schuler GD. (1997) Sequence mapping by electronic PCR. *Genome Res.* 7(5):541-50. <http://www.ncbi.nlm.nih.gov/STS/> PMID: 9149949
- 14: GeneLynx - <http://www.genelynx.org>
- 15: OmniGene - <http://omnigene.sourceforge.net/>
- 16: EJB - Enterprise Java Beans - <http://java.sun.com/products/ejb/>

- 17: JBoss - <http://www.jboss.org/>
- 18: Hoon S, Ratnapu KK, Chia J, Kumarasamy B, Juguang X, Clamp M, Stabenau A, Potter S, Clarke L, Stupka E. (2003) Biopipe: A Flexible Framework for Protocol-Based Bioinformatics Analysis. *Genome Res.* 13(8):1904-15. <http://www.biopipe.org> PMID: 12869579
- 19: Ernst P, Glatting KH, Suhai S. (2003) A task framework for the web interface W2H. *Bioinformatics* 19(2):278-82. <http://www.w2h.dkfz-heidelberg.de/>
- 20: Deutsches Krebsforschungszentrum, Heidelberg, Germany - <http://genome.dkfz-heidelberg.de/>
- 21: European Bioinformatics Institute, Hinxton, UK - <http://www.ebi.ac.uk/>
- 22: AmiGO - <http://www.godatabase.org/cgi-bin/go.cgi>
- 23: Zuyderduyn SD, Jones SJM. (2003) A knowledge discovery object model API for Java. *BMC Bioinformatics* 4(51):1-13. <http://www.biomedcentral.com/1471-2105/4/51>
- 24: Stevens RD, Robinson AJ, Goble CA. (2003) myGrid: personalised bioinformatics on the information grid. *Bioinformatics* 19(1):i302-4. <http://www.mygrid.org.uk/>
- 25: Wilkinson MD, Links M. (2002) BioMOBY: An open source biological web services proposal. *Briefings in Bioinformatics* 3(4):331-41. <http://www.biomoby.org>
- 26: The Taverna Project - <http://taverna.sourceforge.net/>
- 27: Freefluo - <http://freefluo.sourceforge.net/>
- 28: Web Services Flow Language (WSFL) Version 1.0 - <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>
- 29: Liao, S. (2003) Knowledge management technologies and applications - literature review from 1995 to 2002. *Expert Systems with Applications* 25(2):155-64.
- 30: Yan H, Jiang Y, Zheng J, Fu B, Shouzhong X, Peng C. (2004) The internet-based knowledge acquisition and management method to construct large-scale distributed medical expert systems. *Computer Methods and Programs in Biomedicine* 74:1-10.
- 31: Leake DB. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press; 1996. <http://www.cs.indiana.edu/~leake/papers/a-96-book.html>
- 32: Watson I. *Applying Case-Based Reasoning : Techniques for Enterprise Systems*. Morgan Kaufmann; 1997.
- 33: Schmidt R, Montani S, Bellazzi R, Portinale L, Gierl L. (2001) Case-Based Reasoning for medical knowledge-based systems. *International Journal of Medical Informatics* 64:355-67.
- 34: Apache Jakarta Struts - <http://jakarta.apache.org/struts/>
- 35: Apache Forrest - <http://xml.apache.org/forrest/>



- 36: Apache Cocoon - <http://cocoon.apache.org/>
- 37: Java Server Pages (JSP) - <http://java.sun.com/products/jsp/>
- 38: World Wide Web Consortium SVG Resources - <http://www.w3.org/Graphics/SVG/Overview.htm>
- 39: XSLT - XSL Transformations - <http://www.w3.org/TR/xslt>
- 40: World Wide Web Consortium (W3C) - <http://www.w3.org>
- 41: Graphviz - open source graph drawing software - <http://www.research.att.com/sw/tools/graphviz/>
- 42: Extensible Stylesheet Language Family (XSL) - <http://www.w3.org/Style/XSL/>
- 43: Adobe SVG Viewer - <http://www.adobe.com/svg/>
- 44: Hypertext Transfer Protocol - <http://www.w3.org/Protocols/>
- 45: VIBE Software Development Kit - [http://www.incogen.com/public\\_documents/vibe/sdk-dev/docs/index.html](http://www.incogen.com/public_documents/vibe/sdk-dev/docs/index.html)
- 46: Jython - <http://www.jython.org>
- 47: Python - <http://www.python.org/>
- 48: Jelly - Executable XML - <http://jakarta.apache.org/commons/jelly/>
- 49: Matlab - <http://www.mathworks.com/>
- 50: R - <http://www.r-project.org/>
- 51: Matlab Central File Exchange - <http://www.mathworks.com/matlabcentral/fileexchange/loadCategory.do>
- 52: CRAN - Comprehensive R Archive Network - <http://cran.r-project.org/>
- 53: TESS - Transcription Element search - <http://www.cbil.upenn.edu/tess/>
- 54: Matys V, Fricke E, Geffers R, GoBling E, Haubrock M, Hehl R, Hornischer K, Karas D, Kel AE, Kel-Margoulis OV, Kloos DU, Land S, Lewicki-Potapov B, Michael H, Munch R, Reuter I, Rotert S, Saxel H, Scheer M, Thiele S, Wingender E. (2003) TRANSFAC: transcriptional regulation, from patterns to profiles. *Nucleic Acids Res.* 31 (1):374-8. <http://transfac.gbf.de/TRANSFAC/> PMID: 12520026
- 55: Pruitt KD, Tatusova T, Maglott DR. (2003) NCBI Reference Sequence Project: update and current status. *Nucleic Acids Res.* 31(1):34-7. <http://www.ncbi.nih.gov/RefSeq/> PMID: 12519942
- 56: ABI DNA Sequencers - <http://www.appliedbiosystems.com/>
- 57: Ewing B, Green P. (1998) Base-calling of automated sequencer traces using phred. II. Error probabilities. *Genome Research* 8(3):186-94. <http://www.phrap.com/phred/index.htm>

- 58: Ewing B, Hillier L, Wendl MC, Green P. (1998) Base-calling of automated sequencer traces using phred. I. Accuracy assessment. *Genome Research* 8(3):175-85. <http://www.phrap.com/phred/index.htm>
- 59: Walker DR, Koonin EV. (1997) SEALS: A System for Easy Analysis of Lots of Sequences. *Intelligent Systems for Molecular Biology* 5:333-9. <http://www.ncbi.nlm.nih.gov/CBBresearch/Walker/SEALS/index.html>
- 60: phrap - <http://www.phrap.org>
- 61: Grace - <http://plasma-gate.weizmann.ac.il/Grace/>
- 62: Meyer D, Leisch F, Hothorn T, Hornik K. (2002) StatDataML: An XML format for statistical data. *Compstat 2002 - Proceedings in Computational Statistics*. 545-50. Physika Verlag, Heidelberg, Germany, 2002. ISBN 3-7908-1517-9.
- 63: Meyer D, Leisch F, Hothorn T, Hornik K. (2003) StatDataML: An XML format for statistical data. *Computational Statistics*. Forthcoming:. <http://www.omegahat.org/StatDataML/>
- 64: Trosset MW. Formulations of Multidimensional Scaling for Cluster Analysis and Classification. URL: <http://math.wm.edu/~trosset/Research/MDS/adc.ps.gz>.
- 65: S-Plus - <http://www.insightful.com/products/splus/default.asp>
- 66: Trosset MW. (2002) Visualizing Correlation. URL: <http://math.wm.edu/~trosset/Research/MDS/vc.pdf>.
- 67: Wiki - <http://wiki.org/wiki.cgi?WhatIsWiki>
- 68: Java Server Faces - <http://java.sun.com/j2ee/javaserverfaces/index.jsp>
- 69: Velocity - <http://jakarta.apache.org/velocity/>
- 70: Octave - <http://www.octave.org/>
- 71: MPI - Message Passing Interface - <http://www-unix.mcs.anl.gov/mpi/>
- 72: Salzberg S, Delcher A, Kasif S, White O. (1998) Microbial gene identification using interpolated Markov models. *Nucleic Acids Research* 26(2):544-8.
- 73: Delcher AL, Harmon D, Kasif S, White O, Salzberg SL. (1999) Improved microbial gene identification with GLIMMER. *Nucleic Acids Research* 27(23):4636-41.
- 74: MEME - Motif database search - <http://meme.sdsc.edu/meme/website/intro.html>
- 75: RepeatMasker - <http://repeatmasker.genome.washington.edu/cgi-bin/RepeatMasker>
- 76: Kanehisa M, Goto S, Kawashima S, Nakaya A. (2002) The KEGG databases at GenomeNet. *Nucleic Acids Research* 30(1):42-46. <http://www.genome.ad.jp/kegg/> PMID: 11752249
- 77: Javadoc - <http://java.sun.com/j2se/javadoc/>

## VITA

### John Ashley Hayes

The author was born in Lexington, Kentucky on April 25, 1979. He worked at INCOGEN, Inc. as a software engineer beginning in May 1999 while attending college. He then received a Bachelor of Science in Biosystems Engineering with an emphasis in Biotechnology from Clemson University in May, 2001. After completing his degree, he continued to work full-time at INCOGEN, Inc. He moved with the company to Virginia in Fall 2001. In the Fall of 2002, he entered The College of William & Mary's Applied Science graduate program.