# AN OBJECT-ORIENTED FRAMEWORK TO ORGANIZE GENOMIC DATA

A Dissertation

by

NING WEI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2008

Major Subject: Computer Science

# AN OBJECT-ORIENTED FRAMEWORK TO ORGANIZE GENOMIC DATA

A Dissertation

by

NING WEI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

| | |
|---|---|
| Co-Chairs of Committee, | David Adelson |
| | Michael Thon |
| Committee Members, | Christine Elsik |
| | Richard Furuta |
| | Sing-Hoi Sze |
| Head of Department, | Valerie E. Taylor |

May 2008

Major Subject: Computer Science

## ABSTRACT

An Object-Oriented Framework to Organize Genomic Data. (May 2008)

Ning Wei, B.E., Tsinghua University; M.S., University of Oklahoma

Co-Chairs of Advisory Committee: Dr. David Adelson
Dr. Michael Thon

Bioinformatics resources should provide simple and flexible support for genomics research. A huge amount of gene mapping data, micro-array expression data, expressed sequence tags (EST), BAC sequence data and genome sequence data are already, or will soon be available for a number of livestock species. These species will have different requirements compared to typical biomedical model organisms and will need an informatics framework to deal with the data. In term of exploring complex-intertwined genomic data, the way to organize them will be addressed in this study. Therefore, we investigated two issues in this study: one is an independent informatics framework including both back end and front end; another is how an informatics framework simplifies the user interface to explore data. We have developed a fundamental informatics framework that makes it easy to organize and manipulate the complex relations between genomic data, and allow for query results to be presented via a user friendly web interface. A genome object-oriented framework (GOOF) was proposed with object-oriented Java technology and is independent of any database system. This framework seamlessly links the database system and web presentation components. The data models of GOOF collect the data relationships in order to provide users with access to relations across different types of data, meaning that users avoid constructing queries within the interface layer. Moreover, the module-based interface provided by GOOF could allow different users to access data in different interfaces and ways. In another words, GOOF not only gives a whole solution to informatics infrastructure, but also simplifies the organization of data modeling and presentation. In order to be a fast development solution, GOOF provides an automatic code engine by using meta-programming facilities in Java, which could allow users to generate a large amount of routine program codes. Moreover, the pre-built data layer in GOOF connecting with Chado simplifies the process to manage genomic data in the Chado schema. In summary, we studied the way to model genomic data into an informatics framework, a one-stop approach, to organize the data and addressed how GOOF constructs a bioinformatics infrastructure for users to access genomic data.

## ACKNOWLEDGEMENTS

First of all, I profoundly thank my father, Xuehua Wei, and my mother, Xiaoping Di, who have been a tower of strength all throughout my education. This work is entirely dedicated to both of them.

I would like to express my sincere and heartfelt gratitude to my committee chairs, Dr. Adelson and Dr. Thon, for guiding my research efforts. They guided me on various ideas in computational biology and the GOOF. I am very grateful to them for their encouragement, kindness, and support during my research work.

I would also like to thank my committee members, Dr. Elsik, Dr. Furuta, and Dr. Sze, for their guidance and support throughout the course of this research. Thanks also go to my friends and colleagues Jonathan and Hanni for making my time at Texas A&M University a great experience. Exchanging research ideas and discussing problems with them have enriched my knowledge and brought me important ideas for my research work. It was a pleasant experience to work with them.

# TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

# 1 INTRODUCTION

## 1.1 The Problem Statement

An informatics system should provide simple and flexible support for complex data management. Large amounts of genomic data are already routinely generated by high throughput biological technologies. There is a need for an informatics framework to deal with the ever increasing volume data. In order to explore and analyze complex and intertwined genomic data, one has to understand how to organize data efficiently. Several previous studies proposed different solutions for comprehensive informatics support of genome data management. We will address some current issues that impinge on the available informatics solutions. First of all, some informatics systems are only back-end systems, such as Chado [1] and the Genomics Unified Schema (GUS). Only GMOD [2] releases the web front-end system (Turnkey), which is the web end interface of Chado. However, the back end and the GUI are not distributed as a single package and are not yet fully integrated. This is because there is still a gap between the back-end schema Chado and Turnkey. Moreover, it is difficult to customize the Turnkey interface to fit the back-end schema, since every group does not deploy the whole Chado schema. A major part of GMOD is an integrated solution for genome data management and a web interface, such as Gbrowse. However, Gbrowse is not expandable and has only

This dissertation follows the style of *BMC Bioinformatics*.

limited functions, which are used for visualizing genome features, even though the GFF3 data format supports more alternatives for data presentation. Deploying Gbrowse is a relatively easy solution, but users have to adopt the Gbrowse fixed back-end schema and one unified Gbrowse view interface. Other informatics systems, which integrate both back and front ends, only target certain types of genome data. For instance, the Longhorn Array Database (LAD) [3] and Stanford Microarray Database (SMD) [4] only manage microarray data. In other words, currently available systems do not provide a single approach to organize genomic data. Furthermore, current informatics systems do not support flexible user interfaces. They leave the problem of exploring data relationships to users by providing query interfaces. However, by definition exploring genomic data requires users to search new information before understanding the complex relationships in the data. In particular, where one type of data links across many different related types of genomic data, it is difficult for users to construct query interfaces which can be quite complex. These problems are due to the lack of a flexible informatics system structure.

Many bioinformatics systems were individually constructed for either a certain set of function requirements or a particular type of genomic data. The current methodology to approach genomic data management problem does not provide a generic solution. It is neither efficient nor productive. For most bioinformatics system projects, system developers have to design a database schema for their particular genomic data, construct data mining procedures, build a friendly user interface and then connect each component into one system. However, developers might have to re-engineer the current system when facing with new update data or new required functions. This approach increases the time and cost for biology and biomedical researchers to study and discover knowledge from raw genomic data.

Another big challenge is how computer scientists model a system on a large amount of unstructured data. Even though there are new methodologies in programming language, data modeling and informatics research areas, computer science researchers did not propose a comprehensive from the back end to the front end informatics approach for unstructured data because their research goals were to provide the flexibility and general approach on each individual component, such as database, data transaction and interface. Most methodologies for data transactions, user interface construction were proposed and employed on the pre-structured data set. That leaves developers or end users with the difficult task of modeling their data set. Since there is no systematic approach to model unstructured data, any change in the data model could require re-engineering a higher level structure, such as data transaction and interface. Moreover, it is difficult for computer science researchers to design a systematic method of modeling data because most types of data in an informatics system do not share common characteristics to build an integrated approach. This is why this problem space is sparsely populated in computer science. However, in any bioinformatics system, one important characteristic of genomic data is that each genomic feature can be mapped on a reference genome. This means there is a foundation on top of which one could build a systematic approach to model unstructured data and then propose an integrated bioinformatics framework. As far as we know, there is no study that proposes a methodology to construct a bioinformatics system by making use of that characteristic of genomic data. In another aspect of constructing an informatics system, computer science researchers focus more on how to efficiently construct an interface in an approach, but do not include user management issues in their approaches. That is because they do not consider that an interface component should be integrated as an essential part of an informatics system. A bioinformatics system manages not only genomic data, but also users. How to integrate genomic data and user management is another challenge for a bioinformatics framework. In this study, we will present how we model the unstructured biological data and propose a systematic bioinformatics framework, which will solve a generic bioinformatics system construction problem. Therefore, there is a strong need to develop an easy to implement and flexible informatics framework, which integrates both data modeling and presentation as a way to simplify genome data management and user interface construction.

In order to solve the above research issues, we conducted a methodology research to reach the following goals.

- Develop a generic bioinformatics framework.
- Define an easy approach to describe and expand data schema.
- Be independent on any platform and database.
- Develop an interface to connect Chado schema.
- Reduce the cost of re-engineering a bioinformatics system.
- User driven interface.

This study aims to address this pressing and significant need by proposing an integrated informatics framework for genomic data.

## 1.2 Introduction of an Object-Oriented Framework

A standard way to deal with the above issues is to use object-oriented technology to develop a complex and flexible system that can easily describe genomic data without any knowledge of relational databases [5]. Even when faced with different organisms and diverse types of genomic data, object-oriented data modeling simplifies the process of data modeling and presentation in order to provide users with a straightforward way to explore genomic data.

An object based framework should link the database system and the web presentation components [6]. Object-oriented data modeling collects the data relationships and provides users with access to relationships across different types of data. This is a critical part of an object-oriented based framework and simplifies the process of querying data.  A module-based interface driven by an object-oriented structure could allow different users to access data using different interfaces and in different ways [7]. Object-oriented entities also provide a meta-programming environment, which could allow users to rapidly build a large informatics application.

At present there is no pure object-oriented genome informatics framework available. Furthermore, there is no integrated genome informatics system either. The goal of this study is to provide a flexible and expandable informatics framework to organize genomic data. Because of the underlying object-oriented architecture and the application to genomics we will call it the Genome Object-Oriented Framework (GOOF).

GOOF not only provides a whole genomic informatics framework, but also has several advanced features, including the ability to manage multiple data sources, a meta-programming utility and the built-in connection with Chado. In order to demonstrate the flexibility and capabilities of GOOF, we have built EquineBase and OMIMBase to manage Equine genome data and OMIM data [8] [9] [10]. EquineBase and OMIMBase are the first comprehensive informatics applications to manage Equine genome data and OMIM data. In addition, the development of EquineBase and OMIMBase provide an opportunity to evaluate and improve the structure and functions of GOOF. This dissertation will present the issues and research priorities relevant to a genome data management informatics framework.

## 1.3 Dissertation Organization

The rest of this dissertation is organized as follows. In section 2, we review current methodologies to build an informatics framework, including object-oriented technology, database systems and meta-programming. We will also examine several current successful informatics frameworks for genome data management and discuss the reasons why we have chosen to build an object-oriented framework. Section 3 proposes a genome object-oriented framework, GOOF, which is a generic and flexible framework for solving the issues we discussed above. We will describe the details of the methodology and structure of GOOF, how to use GOOF to build a customized informatics system and the front end user interface for a GOOF application. In order to demonstrate the potential of GOOF, section 4 and 5 will present the schemas and implementation of EquineBase and OMIMBase. In section 6, we will further evaluate the usefulness of GOOF after studying EquineBase and OMIMBase. Finally, section 7 will present our conclusions and the future directions of our genome informatics framework.

# 2 BACKGROUND

## 2.1 The Object-Oriented Structure

The real world consists of different types of objects and events. Object-oriented design is the best fit for human beings to translate any situation into a problem which can be solved by a computer [7] [6]. Object-oriented design is a programming paradigm that began in the late 60's as software programs became more and more complex. The idea behind the approach was to build software systems by modeling them based on the real-world objects that they were trying to represent. For example, banking systems would likely contain customer objects, account objects, etc. Today, object-oriented design has been widely adopted around the world. When done properly, the approach leads to simpler, concrete, robust, flexible and modular software. When done badly, the results can be disastrous. At the heart of great design are a set of principles and patterns. Design principles form the foundation of good object-oriented design and design patterns provide general repeatable solutions for common software problems [11]. To build a bioinformatics infrastructure, we need to select a good design patterns to address those issues in the section 1.

An object-oriented programming language (also called an OO language) is one that allows or encourages, to some degree, object-oriented programming techniques such as encapsulation, inheritance, modularity, and polymorphism. It is the foundation of OO design. To address a flexible bioinformatics system, we need to study different OO languages to build the good foundation. Pure object-oriented computer languages, such as Java, C#, C++, Smalltalk and Ada 2005, are the way to design and implement such an OO solution. Smalltalk and Ada 2005 do not have a solid library to build database connection and web interface. C++ is not flexible across different platforms, because C++ requires different compilers and supporting libraries on different platforms. C# is intended to be a simple, modern, general-purpose, object-oriented programming language, which developed by Microsoft as part of the .NET framework with a particular emphasis on simplification. C# comes with several comprehensive libraries for database, web application development. However, it is more stable and robust on Windows system. And most scientific applications run on Mac and Linux systems, C# could bring more problems for various platforms. Java is an object-oriented programming language, which allows the same program to be executed on multiple operating systems. One characteristic of Java, platform independence, is that programs written in the Java language must run similarly on any supported hardware/operating-system platform. One should be able to write a program once, compile it once, and run it anywhere. That is one of several reasons why we decided to build a generic genome informatics system with Java. In [12] [13], J2EE was derived from Java to develop any object-oriented system.

Biological technologies, such as DNA sequencing, proteomic mass-spectrometry, protein arrays and microarray, generate too much data for biologists to digest and manage.  In order to assist biologists with the significant amount of genomic data, many individual informatics systems have been implemented in various research communities. FlyBase [14] [15] [16] and WormBase [17] [18] [19] [20] are the leading informatics systems driven by their own genome databases. However, the efficient and productive way to discover data is still a hot research topic. FlyBase and WormBase are still being developed and are testing how informatics systems and data modeling provide the best way to manage data. Both of them, however, are not object-oriented systems. Recently, FlyBase released their database schema, Chado, as a generic organism schema. DictyBase [21] [22] adopted Chado, but used a system-specific approach to manage data. More recently, different groups started to address informatics systems for genomic data. Stanford Microarray Database [4] [23] and Longhorn database system [3] emphasized microarray data, which only gave a solution for generic microarray data management. WormBase published their GBrowse [2] for presenting genome features as an interactive web application. SynView [24] provides the additional genome comparison function on top of Gbrowse. Both of them provide only one simple function, which limits their applicability. FlyBase and WormBase are not readily adaptable and thus not likely to be adapted by other organism databases. The most comprehensive component, Chado only provides the back-end database schema and is relatively inflexible because it must be implemented on PostgreSQL. After studying different available informatics systems, we discovered that there were no methodology studies that addressed an approach to construct a generic bioinformatics system. Until now, it is still a big question that how to provide an efficient way to develop an informatics system which should provide the back end database and front end interface support.

Several problems can arise when applications contain a mixture of data access code, data transaction code, and presentation code. Such applications are difficult to maintain, because interdependencies between all of the components cause strong ripple effects whenever a change is made anywhere. High coupling makes classes difficult or impossible to reuse because they depend on so many other classes. Adding new data views often requires re-implementing or cutting and pasting data transaction code, which then requires maintenance in multiple places. Data access code suffers from the same problem, being cut and pasted among data transaction methods.

There are several approaches to design an OO system [5, 25]. The Model-View-Controller [26] [27] design pattern solves complex data access problems by decoupling data access, query logic, and data presentation and user interaction. Model is the model represents data and the query rules that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the model. View is the view renders the contents of a model. It accesses biological data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. This can be achieved by using a push model, where the view registers itself with the model for change notifications, or a pull model, where the view is responsible for calling the model when it needs to retrieve the most current data. Controller is the controller translates interactions with the view into actions to be performed by the model. In a stand-alone GUI or Web client, user interactions could be button clicks or menu selections, and in a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating data transaction processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view. In order to build GOOF to be flexible for any organism, we take advantage of object-oriented design and pattern. Since a bioinformatics system, we need to solve the complicated relations of different types of data, and a straight forward interface is also critical for our end users because most of them are equipped with good computer knowledge. A web interface is a target interface for users. Then, the MVC design pattern will be the natural choice for GOOF. However, how to solve the complexity of data access and data presentation will introduce another design concept, the middleware approach.

## 2.2 The Middleware Approach

In [28] and [12], the authors discussed different Java frameworks for a comprehensive development system. The Model-View-Controller framework (MVC) is the best and most flexible way to build an informatics system driven by a database [27, 29, 30]. In every application, we always have data model and user view components. How does MVC reduce the complexity of an application, simplify the engineering structure and improve the reusable components? The middleware approach [28] is introduced. The critical part of MVC design is the controller, which is the middleware. Middleware is computer software that connects software components or applications. The software consists of a set of enabling services that allow multiple processes running on one or more components of an application even cross different platforms. This technology evolved to provide for interoperability in support of the move to client/server architecture. It is used most often to support complex, distributed applications. It includes web servers, application servers, content management systems, and similar tools that support application development and delivery. Middleware is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture. In software engineering, the middleware approach attempts to aid system designer in the separation of concerns, which are cross different functions and work flow in an application, as an advance in modularization. Separation of concerns entails breaking down a program into distinct parts that overlap in functionality as little as possible. All programming methodologies, including procedural programming and object-oriented programming, support some separation and encapsulation of concerns (or any area of interest or focus) into single entities. For example, procedures, packages, classes, and methods all help programmers encapsulate concerns into single entities. But some concerns defy these forms of encapsulation. For a bioinformatics system, we concern the independency of database system, a generic framework and a flexible interface. How to integrate three MVC components together is the question that how we employ the middleware approach to build interfaces of these three components and link them together as one framework.

GOOF is a high level application framework. Three layers (model, view and control) in GOOF are built using three different Java frameworks, which maintain low-level API to communicate with the database and the web application server [31]. Hibernate [32] is the data object layer foundation. Since a GOOF application is the web front end one, WebWork [33] is a simplicity and flexibility choice of the interface construction. Spring [34] is the controller framework in GOOF, which handles the data transaction management.

Hibernate [35] is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves Object-Relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions. The primary feature of Hibernate is mapping from Java classes to database tables and from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. Hibernate generates the SQL calls and relieves the developer from manual result set handling and object conversion, keeping the application portable to all SQL databases, with database portability delivered at very little performance overhead. Hibernate provides transparent persistence for Plain Old Java Objects (POJOs). The portability of Hibernate provides the flexibility of GOOF on any database.

WebWork [36] is a Java web-application development framework built specifically with developer productivity and code simplicity in mind. In a standard Java EE web application, the client will typically submit information to the server via a web form. The information is then either handed over to a Java Servlet which processes it, interacts with a database and produces an HTML-formatted response, or it is given to a JavaServer Pages (JSP) document. Both approaches are often considered inadequate for large projects because they mix application logic with presentation and make maintenance difficult. WebWork distinguishes itself by having understood those existing limitations and working to eliminate them. WebWork has been designed and implemented with a specific set of features, which are very suitable for the bioinformatics research community. The interface layout design never has to touch Java code, the web action and the data layer. It provides the central resource control for the web interface, which also brings the reuse components on web pages. Since GOOF targets a solution from database to web application, WebWork is naturally chosen to build the view layer.

Spring [13] provides the full J2EE application API, which is used to link WebWork and Hibernate together. Spring, as the controller component, brings an abstraction mechanism to the Java platform. Its abstraction is capable of working with local (inside the same layer) and global transactions (cross different layers). We use a modular XML to define data access object interface, data transaction manager interface and web action definition. The data flow and messages cross three components are linked together by the XML mechanism. After reviewing the current research topics on genome informatics systems, we will discuss how GOOF is built on top of three low-level frameworks in the next section.

## 2.3 Meta-Programming

Previous empirical studies have shown 50% - 90% rates of repetitions that recurred in newly developed programs [37]. Code generation is the approach in high-performance computing and high-assurance embedded programming [38]. Because a generic program in an earlier stage can generate a specialized program in a later stage and assure it safe, the programmer need not trade off abstraction and assurance for efficiency in time and space. Code-generating programs are sometimes called meta-programs; writing such programs is called meta-programming [37]. Writing programs that write code has numerous applications. Code-generating programs allow developers to abbreviate such statements and save a lot of typing, which also prevents a lot of mistakes because there is less chance of mistyping [39]. Meta-programming is the writing of computer programs that write or manipulate other programs (or themselves) as their data or that do part of the work during compile time that is otherwise done at run time. In many cases, this allows programmers to get more done in the same amount of time as they would take to write all the code manually.

The components of meta-programming include textual macro languages, specialized code generators [40]. Textual macro languages develop and use small, domain-specific languages that are easier to write and maintain than writing them in the target language. This is best written as a macro for several reasons: A function call would take way too much overhead for a simple operation. Programs would have to pass the variables' addresses to the function rather than the variables' values. Different parts of programs have a different function for each type of data objects. Specialized code generators are usually the generic textual-substitution programs, which are more focus on automating specific aspects of a program [41]. In a bioinformatics application, we have many similar types of data objects, which could be manipulated by the same way. We can define a template by using a textual macro language to generate over reusable operations in application. A specialized code generator could automate the code generation process by a textual substitution in the template [42].

In order to solve a large amount of data objects and reusable functions in a bioinformatics framework by the meta-programming approach, we need to have a design architecture which could take advantage of meta-programming. According to the above discussions of OO design and MVC structure, what is the best data structure to present genome data? Model-driven architecture (MDA) is the answer [43] [44]. MDA supports model-driven engineering of software systems and provides a set of guidelines for structuring specifications expressed as models. MDA requires that design is done in an architecture and technology neutral way. For a bioinformatics application, it is easy to illustrate genome data in a natural way. GOOF not only takes use of OO structure to model genome data, but also composes the data transaction, web action and even the interface layout in a model-driven architecture. As a well design framework, GOOF would employ meta-programming to reduce the work of a bioinformatics application development work.

With a generic capable structure, how to achieve high productivity is one of the research issues in GOOF. We can discuss it further how a meta-programming utility will be developed in GOOF. What is the best approach to design structures that represent the similarity of patterns in a generic way? One of the best practices is the Object-Oriented structure of GOOF, which provides a basic mechanism to reuse generic structures and encourage organizing software as standard architectures in order to reuse common components. The templates [45] in many programming languages, such as STL of C++ [46] and Generics of Java [47] [48], were studied to provide and reuse common components in software [49]. Aspect-oriented programming (AOP) [50], such as AspectJ, is a methodology that advocated decomposing software by aspects of functionality that may affect other functional units [51]. AOP is more a theoretical definition than a practical guide. Meta-programming tools do not tie all of AOP, but provide a mechanism to engineer common components, which were studied in the field of software engineering. XDoclet [52] is an extended JavaDoc Doclet engine. It's a generic Java tool that permits the creation of custom JavaDoc tags, and based on those tags generate source code or other files [53]. It enables AOP for java. In short, this means that it can add more significance to programs by adding meta data (attributes) to Java codes. XDoclet employs the approach which is dynamic execution of string expressions that contain programming commands. XDoclet is a meta-programming language which is used to compose the macro templates for data model, data transaction, web action and interface layout. We develop a specific code generator by using Ant build script. Through the tags of XDoclet, Ant can generate the program codes and other programming files according to the templates. The code generator would automatically write the codes and files according to the fix Model structure in GOOF. In the overall structure of GOOF section, we will discuss how important it is to follow the file structure and the naming rules of GOOF. In summary, the templates and JavaDoc with XDoclets in GOOF can improve productivity.

## 2.4 User Interfaces

Another problem that many existing systems face is the re-engineering development of the user interfaces [54]. The primary reason for this is that they have focused on database management and function and have ignored the characteristics of the data and users. The user interface or Human Machine Interface is the aggregate of means by which users interact with the system - a particular machine, computer program or other complex tools. The user interface provides means of: Input, allowing the users to manipulate a system; Output, allowing the system to produce the effects of the users' manipulation. The major users of bioinformatics applications do not know how to construct complicated queries to explore the data. They prefer to pull data straight out of databases rather than build the logical queries to navigate the data structures in databases. Another problem is that when dealing with new genomic data, users can not know the potential unknown relationships. Therefore, a means of presenting the data relationships without user interactions could be a valuable aspect of an informatics system. To manage a large amount of data, especially genome data, one research issue for the user interface is how to construct and manage the data access interface. When discussing meta-programming, we will employ Model-Driven Architecture in the whole framework. Java Server Page (JSP) [55] is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request [26]. This technology allows Java code and certain pre-defined actions to be embedded into static content. The JSP syntax adds additional XML-like tags, called JSP actions, to be used to invoke built-in functionality. Additionally, the technology allows for the creation of JSP tag libraries that act as extensions to the standard HTML or XML tags. Tag libraries provide a platform independent way of extending the capabilities of a Web server. JSP can embed a web action inside, which allows application to separate the interface construction from the data access programming. JSP and WebWork are the foundation of the user interface. In order to manage a large amount of data and improve the usability of an interface, the modular structure of JSP can provide a bioinformatics framework with a flexible template to construct a user interface. In the next section, we will discuss how the OO structure and the modular design of JSP easily solve this problem.

# 3 GENOME OBJECT-ORIENTED FRAMEWORK

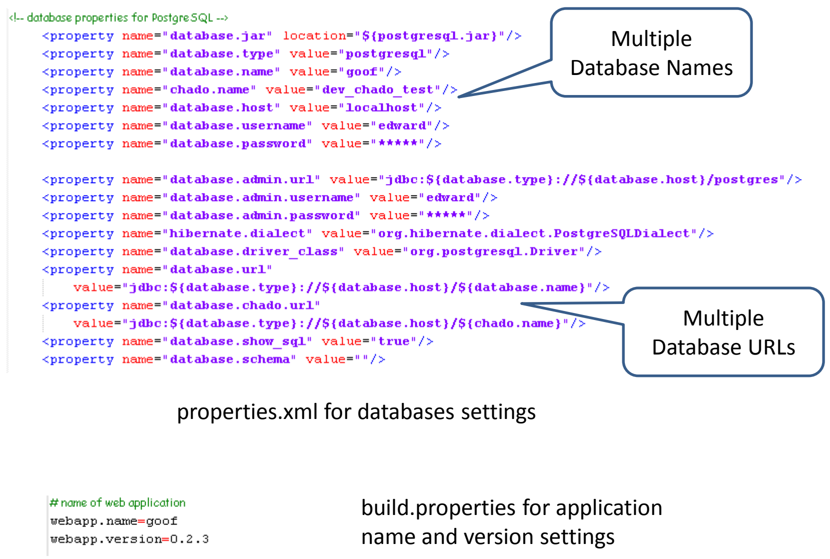
## 3.1 The Overall Structure

This section will discuss the object-oriented data modules, which are employed in GOOF. The goal of GOOF is to give a rapid and simple solution for genome data informatics systems. Therefore, GOOF should not require any low level APIs. When we proposed GOOF, another consideration was the stability, scalability and performance. That is why we built GOOF using robust Java object-oriented frameworks.

In order to achieve flexibility, GOOF needs to be independent of any back-end database. The object-oriented structure of GOOF allows developers to design the data of their system as objects. GOOF maps data objects and relations to any major database system because of the object-oriented structure. In other words, there is no need to use database specific SQL. GOOF employs the Java object-oriented Model-View-Controller design as we discussed in the background section. Hibernate is the data model framework for data object and relation mapping modules, which builds the link to the GOOF-based application and the back-end database. Spring and WebWork together structure the data transaction to control data flow within the framework and communicate with users via the web presentation layer. WebWork is the web presentation view framework to control user actions and data flow through on the interface level. The whole GOOF framework uses XML-based files for development and application deployment to manage the different modules and the application development. The XML-based configuration file is module-based and easy to be configured to be a pipeline for many development tasks.

For the easy maintenance and expandable features of GOOF, strict object naming rules are defined as the part of the document. Since GOOF comes with a utility tool, including several templates and automatic code generators, the unique naming rules are critical to the functions of this utility tool. In modeling the data objects, the naming rules could also ensure the consistency of data objects in the whole application. For the same purpose, the strict file structure is fixed. After expanding the application of GOOF, one unified XML-based configuration serves the whole application, which requires the same file structure as well. Another important component of GOOF is the automatic code generator, which provides developers with the out-of-box application to build simple data objects and relations.

The default database connection is configured with PostgreSQL as the back-end database, but this can be changed to any major database by reconfiguring the framework. In the properties.xml, the multiple database connections are configured as figure 1. The development control script, called build.xml, generates the database environment variables which are used in the framework. In this example, there are two database connections defined. Both of them are located on the same physical machine and one database system. If an application is designed to manage multiple database systems on different physical machines or different database systems, for instance, MySQL and Postgresql, then multiple database driver libraries and host names should be given in this file. GOOF calls those variables and builds the database connections for the data model layer.



```
<!-- database properties for PostgreSQL -->
    <property name="database.jar"  location="${postgresql.jar}"/>
    <property name="database.type"  value="postgresql"/>
    <property name="database.name"  value="goof"/>
    <property name="chado.name"  value="dev_chado_test"/>
    <property name="database.host"  value="localhost"/>
    <property name="database.username"  value="edward"/>
    <property name="database.password"  value="*****"/>

    <property name="database.admin.url"  value="jdbc:${database.type}://${database.host}/postgres"/>
    <property name="database.admin.username"  value="edward"/>
    <property name="database.admin.password"  value="*****"/>
    <property name="hibernate.dialect"  value="org.hibernate.dialect.PostgreSQLDialect"/>
    <property name="database.driver_class"  value="org.postgresql.Driver"/>
    <property name="database.url"
        value="jdbc:${database.type}://${database.host}/${database.name}"/>
    <property name="database.chado.url"
        value="jdbc:${database.type}://${database.host}/${chado.name}"/>
    <property name="database.show_sql"  value="true"/>
    <property name="database.schema"  value=""/>
```

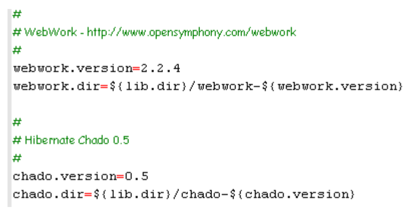
properties.xml for databases settings

```
# name of web application
webapp.name=goof
webapp.version=0.2.3
```

build.properties for application name and version settings

Figure 1  GOOF database settings

In GOOF, the definitions of objects, methods and variables follow the unified naming rules. The file structure of GOOF follows the data-web-service structure according to the MVC design pattern. The important rule for the central Ant build is that developers must follow the fixed structure. If not, Ant could report a build error. For any GOOF application, there are three basic folders. Applications are developed according to this structure. The Ant development tool wraps the web application packages compatibly with the Tomcat server.



```
#
# WebWork - http://www.opensymphony.com/webwork
#
webwork.version=2.2.4
webwork.dir=${lib.dir}/webwork-${webwork.version}

#
# Hibernate Chado 0.5
#
chado.version=0.5
chado.dir=${lib.dir}/chado-${chado.version}
```

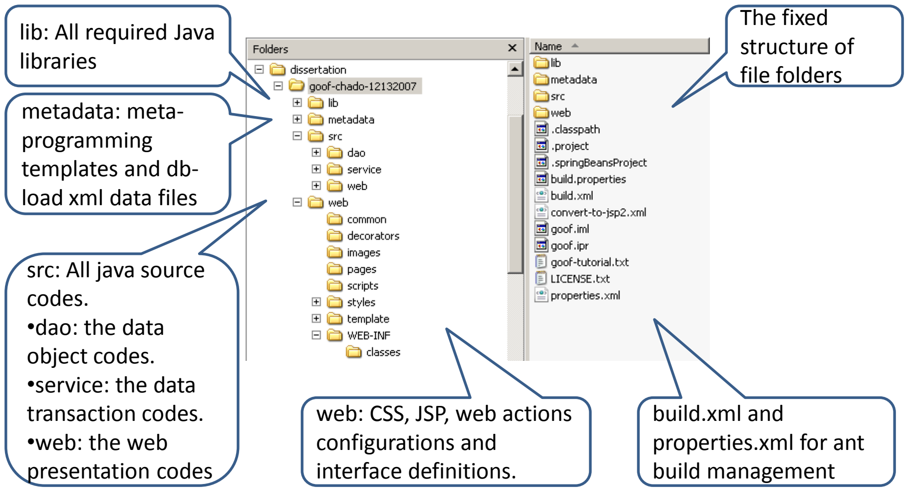
lib.properties in the GOOF's lib folder is part of build.xml and has all of information and paths of java libraries.

Figure 2  Java libraries configuration

- "lib": all necessary java libraries are located in this folder. All of version information and paths are in lib.properties of figure 2. When building the web application package, Ant includes them into the package.
- "src": all the java source codes are located in this folder. Following the MVC design, "dao", "service", and "web" contain the three separate layers.
  - "dao": the folder contains the Java code for the data model layer and data mapping files. The data model definitions and Hibernate data operations are also in this folder.

    - “service”: the folder contains the Java code for the data transaction layer. The interfaces in between the data model layer and the web presentations are defined as data managers.
    - “web”: the folder contains the Java code for the web presentation layer. The interfaces to the data transaction manager are defined here. The implementations of web actions are built in the web folder, which link the actions to the Java Server Page (JSP).
- “web”: all JSPs, web interface components definitions and the web application configurations are located in this folder. The “pages” and “WEB-INF” folders follow the Tomcat server application rules.
    - “pages”: All JSPs for the web interface are in this folder. These are the templates used to build the interfaces and generate the actual values of data objects requested by users.
    - “WEB-INF”: “classes” is where all Java binary files are. The Ant development tool compiles the Java source codes and wraps the binary class files into this folder, when building a web application package. All of the web application configurations are defined in “WEB-INF”.

Figure 3 shows the structure of folders in the GOOF package. The GOOF package is a bare bone application, which in the case is actually a ready-to-go application with a simple template for demonstration. First of all, the database section of property.xml should be changed to the proper value. In the build.xml and build.properties, the web application name should be changed from “goof” to an actual web application name. Then please refer the development management section for how to start a goof application.


lib: All required Java libraries
metadata: meta-programming templates and db-load xml data files
src: All java source codes.
•dao: the data object codes.
•service: the data transaction codes.
•web: the web presentation codes
Folders
dissertation
goof-chado-12132007
lib
metadata
src
dao
service
web
web
common
decorators
images
pages
scripts
styles
template
WEB-INF
classes
Name
lib
metadata
src
web
.classpath
.project
.springBeansProject
build.properties
build.xml
convert-to-jsp2.xml
goof.iml
goof.ipr
goof-tutorial.txt
LICENSE.txt
properties.xml
The fixed structure of file folders
web: CSS, JSP, web actions configurations and interface definitions.
build.xml and properties.xml for ant build management

The structure of folders in GOOF

Figure 3  The structure of folders in GOOF

In GOOF, the user management module, including users and roles, is independent of the genome data schema, which provides users with different levels of accesses to data objects on the view level. The independent user management module allows GOOF to integrate any existing data schema without losing role-based data access control. It is easy to take an existing user information database for data access control as well. The security access control configuration can be reconfigured to link to the existing user information database. In this way, the data schema can simply reflect the genome data without any worrying about user access control. To build user access control at the data model level would detract from the flexibility and generic structure of GOOF. This is why we designed a separated schema for user management and left the user access control within the view level. This allows role-based data access to be added on top of the web presentation layer. It controls not only data access, but also the different levels of the data query utilities in the interface. We will show how this works in the user interface section.

GOOF is developed as a single informatics back-to-front framework. Every development stage and database configuration is managed by one Ant configuration file. Therefore, one setup in the Ant build file will allow any GOOF application to run out of box. This will be discussed further in section 3.3. Once the overall structure is well understood, the MVS design and developing a GOOF application should be clear.

## 3.2 The MVC Design

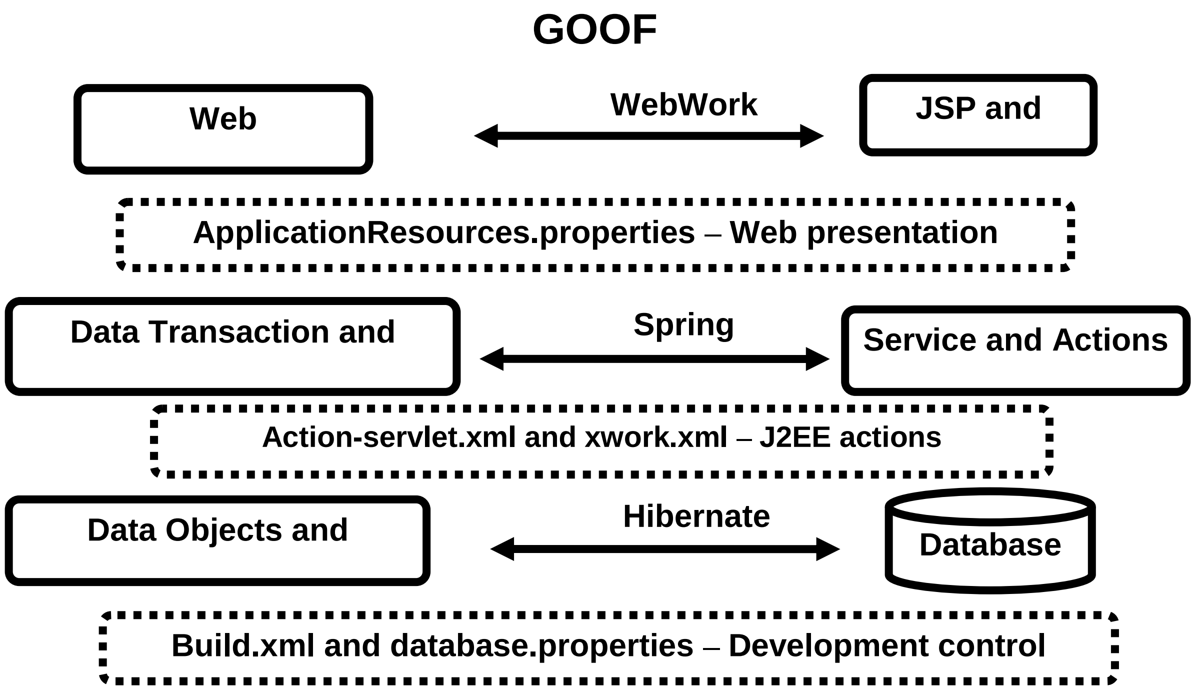

GOOF
Web
WebWork
JSP and
ApplicationResources.properties – Web presentation
Data Transaction and
Spring
Service and Actions
Action-servlet.xml and xwork.xml – J2EE actions
Data Objects and
Hibernate
Database
Build.xml and database.properties – Development control

Figure 4 The MVC structure of GOOF

In a complex informatics application that presents a large amount of data to users, we always have data (model) and user interface (view) concerns, because any change to the user interface will affect data handling and any change to the data schema can require rebuilding the user interface. The Model-View-Controller (MVC) solves this problem by decoupling data access and data transaction from data presentation and user interaction. The solution is an intermediate component: the controller. Therefore, the controller is the middle interface between the data objects and the data view (user interface). As long as the controller interface is the same, the data objects and the data view can be reconstructed according to local implementation needs without interrupting each other. GOOF follows the MVC design pattern to keep each layer independent. Figure 4 shows that the structure of GOOF is built on the layers of web presentation, data transaction (control), and data object and relations, which correspond with the three Java frameworks.

- The model layer: Hibernate is an entity and persistent service to connect actual database and objects as the data model layer of GOOF.
- The control layer: Spring is an independent API set to build data flows from the data layer to the view layer. It is the base API of the data transaction layer of GOOF
- The view layer: WebWork is an API set to operate objects and manage user actions on the web presentation. JSP and WebWork are the web presentation layer of GOOF.

GOOF is designed to manage genome data. Facing with the complexity of genome data, genomics and biomedical researchers usually do not have the knowledge and resources to handle the design work of their bioinformatics infrastructure. The MVC structure of GOOF helps to reduce the complexity of the architecture to increase flexibility and reuse the same components by decoupling models and views. Many bioinformatics system use a persistent storage mechanism (a database) to store data. GOOF does not specifically link any database system as the data access layer because the database is underneath or encapsulated by the data model layer. On the view layer, users usually interact with a GOOF application by invoking an action, which is delivered to the controller layer. A controller handles the input event from the user interface, often via a registered web action. Then, the controller accesses the data model, possibly updating it in the appropriate way, depending on the user's request. A web view uses the web interface components in the web presentation layer and the data objects indirectly to generate an appropriate user interface. Finally, the web view gets the requested data from the data model layer. At the same time, the data model has no direct knowledge of the view. The user interface waits for further user interactions, which begins another new cycle. Although the three layers of GOOF are separate in the MVC structure, they are actually well connected as a life cycle for data flow invoked by users. In the following sections, we will discuss how to design and develop a GOOF application.

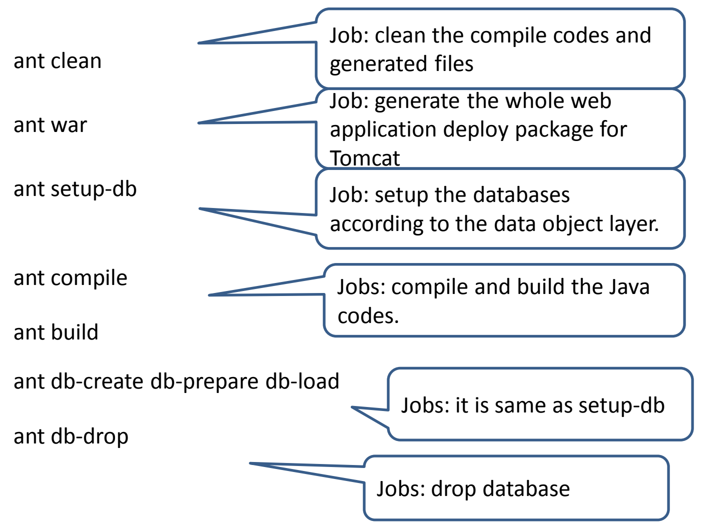
## 3.3 Managing Application Development

We envisioned GOOF as an out-of-the-box genome data informatics framework, a tool to simplify the complexity of managing application development. In order to keep GOOF consistent and easy to deploy, Apache Ant [56] is used as the automatic building tool to manage the whole project, from setting up the database, compiling, building and deploying to the web application server. Several Ant commands are shown here in figure 5. They control not only application development but also database operation. The basic process of a development using GOOF follows three major steps.

1. “Ant setup-db” prepares the database table and schema, create the database in the target database system and load data into the database if

the raw data in XML format is available.

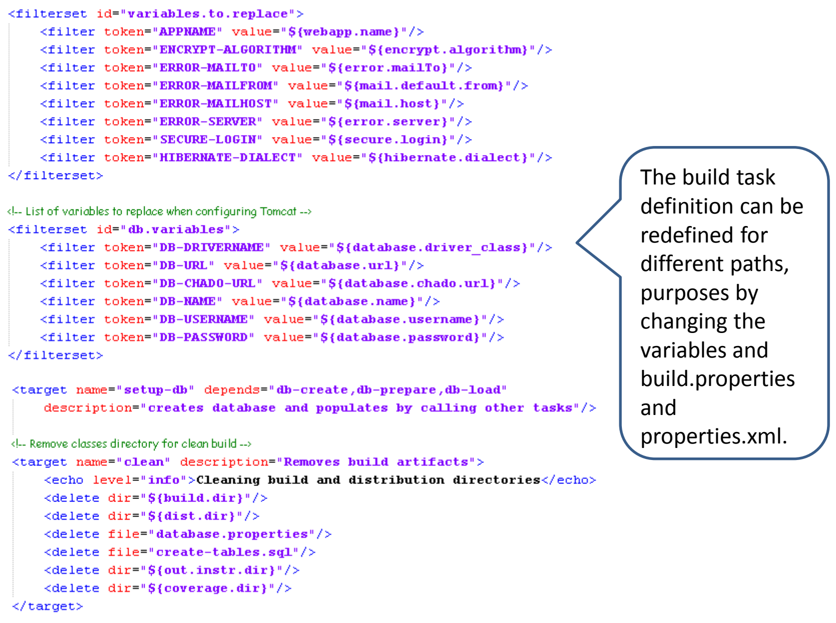
2. "Ant compile" or "Ant build" builds the binaries from Java source code. It is also useful for debugging Java source code.
3. "Ant war" generates the web application package, the war file, for deploying the application via the Tomcat server. It wraps up the binary files, the web components resource file, JSP and all of the web application configuration files in the war package.


ant clean
Job: clean the compile codes and generated files
ant war
Job: generate the whole web application deploy package for Tomcat
ant setup-db
Job: setup the databases according to the data object layer.
ant compile
ant build
Jobs: compile and build the Java codes.
ant db-create db-prepare db-load
Jobs: it is same as setup-db
ant db-drop
Jobs: drop database

Ant controls the different development stages in GOOF.

Figure 5 The Apache Ant build reference

In the overall structure section, we discussed how to setup the general application name and database connections. After that, GOOF is ready to run. The default settings are good for most cases. Users can change the variables and redefine build tasks in build.xml, properties.xml and build.properties, which are shown in figure 6.



```
<filterset id="variables.to.replace">
    <filter token="APPNAME" value="${webapp.name}"/>
    <filter token="ENCRYPT-ALGORITHM" value="${encrypt.algorithm}"/>
    <filter token="ERROR-MAILTO" value="${error.mailTo}"/>
    <filter token="ERROR-MAILFROM" value="${mail.default.from}"/>
    <filter token="ERROR-MAILHOST" value="${mail.host}"/>
    <filter token="ERROR-SERVER" value="${error.server}"/>
    <filter token="SECURE-LOGIN" value="${secure.login}"/>
    <filter token="HIBERNATE-DIALECT" value="${hibernate.dialect}"/>
</filterset>

<!-- List of variables to replace when configuring Tomcat -->
<filterset id="db.variables">
    <filter token="DB-DRIVERNAME" value="${database.driver_class}"/>
    <filter token="DB-URL" value="${database.url}"/>
    <filter token="DB-CHADO-URL" value="${database.chado.url}"/>
    <filter token="DB-NAME" value="${database.name}"/>
    <filter token="DB-USERNAME" value="${database.username}"/>
    <filter token="DB-PASSWORD" value="${database.password}"/>
</filterset>

<target name="setup-db" depends="db-create,db-prepare,db-load"
    description="creates database and populates by calling other tasks"/>

<!-- Remove classes directory for clean build -->
<target name="clean" description="Removes build artifacts">
    <echo level="info">Cleaning build and distribution directories</echo>
    <delete dir="${build.dir}"/>
    <delete dir="${dist.dir}"/>
    <delete file="database.properties"/>
    <delete file="create-tables.sql"/>
    <delete dir="${out.instr.dir}"/>
    <delete dir="${coverage.dir}"/>
</target>
```

The build task definition can be redefined for different paths, purposes by changing the variables and build.properties and properties.xml.

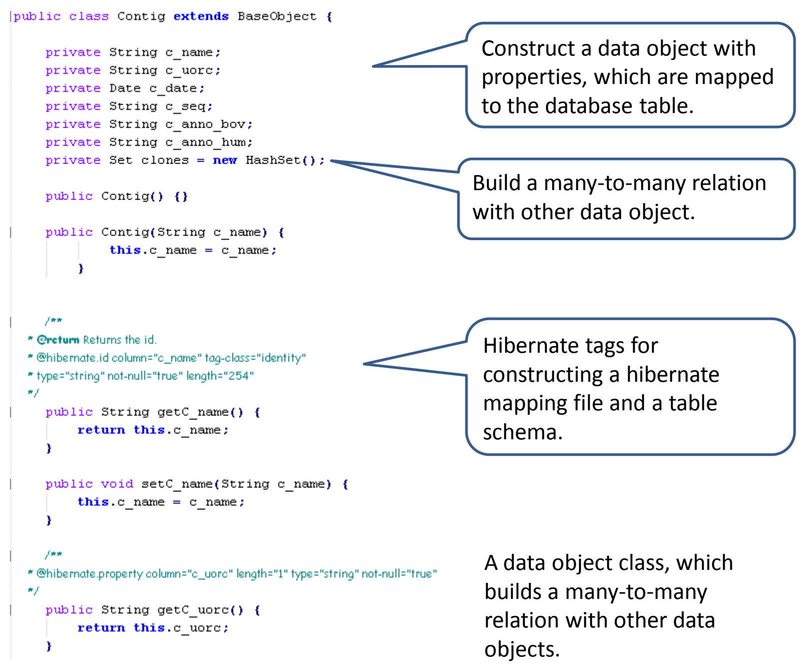
The ant build.xml definition

Figure 6 The Ant build task definitions

As an independent framework, one of the primary aims of GOOF is to solve development portability problem. GOOF is built using Java, which is naturally independent of any platform. Ant can execute tasks on both Windows and UNIX-like system. This is another important reason why we developed GOOF using Apache Ant.

## 3.4 The Data Model Layer

In the data model layer, there are two basic generic data objects in GOOF: the single data object and the many-to-many data object. The single data object is a generic data object. In order to simplify data transaction modules and presentation interfaces, the many-to-many data object is used to manipulate the relationships of data objects. Modeling the relations of data as properties of data objects is the key to avoiding complexity when building queries in the data transaction layer of GOOF.

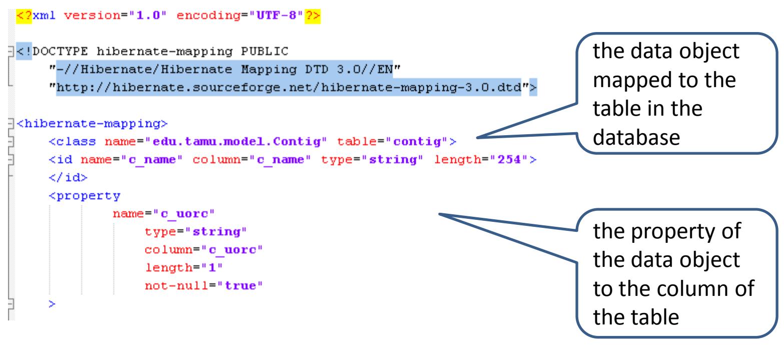


```
public class Contig extends BaseObject {

    private String c_name;
    private String c_uorc;
    private Date c_date;
    private String c_seq;
    private String c_anno_bov;
    private String c_anno_hum;
    private Set clones = new HashSet();

    public Contig() {}

    public Contig(String c_name) {
            this.c_name = c_name;
        }


    /**
  * @return Returns the id.
  * @hibernate.id column="c_name" tag-class="identity"
  * type="string" not-null="true" length="254"
  */
    public String getC_name() {
        return this.c_name;
    }

    public void setC_name(String c_name) {
        this.c_name = c_name;
    }

    /**
  * @hibernate.property column="c_uorc" length="1" type="string" not-null="true"
  */
    public String getC_uorc() {
        return this.c_uorc;
    }
```

Construct a data object with properties, which are mapped to the database table.
Build a many-to-many relation with other data object.
Hibernate tags for constructing a hibernate mapping file and a table schema.
A data object class, which builds a many-to-many relation with other data objects.

Figure 7 The data object "Contig" example

In figure 7, a “Contig” data object has several properties, such as name, sequence, annotation, date and so on. As long as you can describe a data object in an application, it is easy to construct a data object in the data model layer. From the database point view, a table in the database is a data object in the data model layer and each column of a table is a property of a data object. Some additional attributes of data object models in Java are a function of Hibernate tags that can be used by the meta-programming utility to automatically generate a Hibernate persistent mapping file for the data objects. It can also convert Java data type to the data types of the linked database.



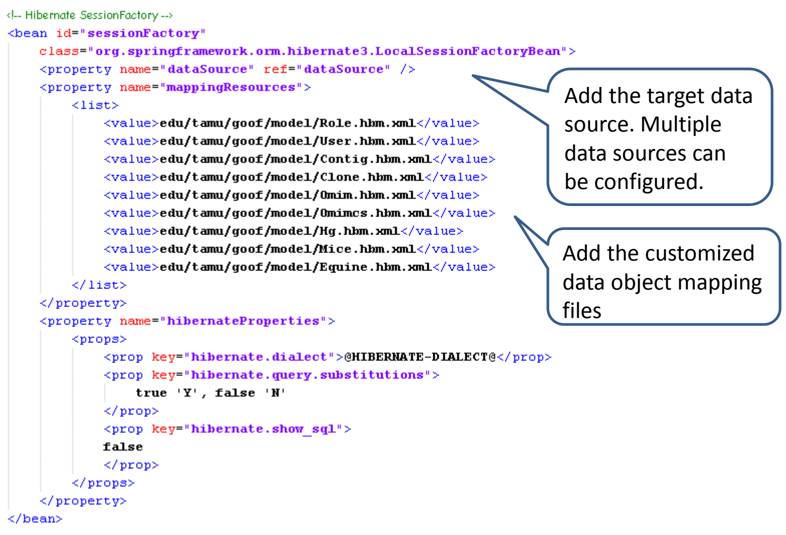
```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="edu.tamu.model.Contig" table="contig">
    <id name="c_name" column="c_name" type="string" length="254">
    </id>
    <property
            name="c_uorc"
                type="string"
                column="c_uorc"
                length="1"
                not-null="true"
    >
```

the data object mapped to the table in the database
the property of the data object to the column of the table

The Hibernate persistent data object mapping file

Figure 8    The Hibernate persistent mapping file example

The fundamental way that data object is described in the data model layer differs from a table in the relational database. A relationship between data objects can be defined as a property, just like other properties of those data objects. In figure 8, the example shows that every “Contig” has several “Clones”. In the data model, there is only a many-to-many relationship. One-to-many or one-to-one is just a simplified many-to-many relationship. Users can add Hibernate tags to restrict the relationships to one of these three types to control those relationships in the database.



```
<!-- Hibernate SessionFactory -->
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mappingResources">
        <list>
            <value>edu/tamu/goof/model/Role.hbm.xml</value>
            <value>edu/tamu/goof/model/User.hbm.xml</value>
            <value>edu/tamu/goof/model/Contig.hbm.xml</value>
            <value>edu/tamu/goof/model/Clone.hbm.xml</value>
            <value>edu/tamu/goof/model/Omim.hbm.xml</value>
            <value>edu/tamu/goof/model/Omimcs.hbm.xml</value>
            <value>edu/tamu/goof/model/Hg.hbm.xml</value>
            <value>edu/tamu/goof/model/Mice.hbm.xml</value>
            <value>edu/tamu/goof/model/Equine.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">@HIBERNATE-DIALECT@</prop>
            <prop key="hibernate.query.substitutions">
                true 'Y', false 'N'
            </prop>
            <prop key="hibernate.show_sql">
            false
            </prop>
        </props>
    </property>
</bean>
```

Add the target data source. Multiple data sources can be configured.
Add the customized data object mapping files

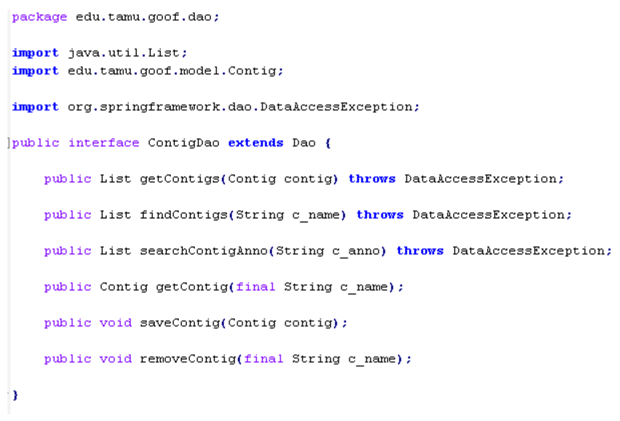
The hibernate data objects mapping configuration

Figure 9 The Hibernate data objects mapping configuration

Based on the data models and the data object relationship mapping files, we can configure the framework to connect the data objects to the target data source. That is the critical step that actually links the application to the back end database. All of these mapping files are the “mappingResources” of a data source showed in figure 9.

Since there are many genomic data types modeled in the Chado schema, the Chado connection built-in GOOF would provide the most general templates for translating genomic data into the object-oriented structure. The flexible structure of the data model layer allows any application to connect to multiple database sources and model various schemas into GOOF. Therefore, the built-in connection with Chado in GOOF is an advantage to many existing genomic data systems. We will discuss GOOF and Chado in the section 3.8. Therefore, GOOF not only has a simple data object template, but also provides a comprehensive data object template (Chado) to model genomic data into object-oriented data objects.

Once the data model has been built, the interface to the data transaction layer must be constructed. Since the MVC structure is composed of three separate layers and the data flow in the whole framework is a life cycle, every layer has an interface to communicate with the next layer, but still executes its own internal methods unknown to the adjacent layers. In the MVC structure of GOOF, the data model layer has an interface to the data transaction layer. This is basically the interface that allows the data transaction manager to approach the data object in defined ways. In figure 10, we show several interfaces of “Contig” to the data transaction layer.

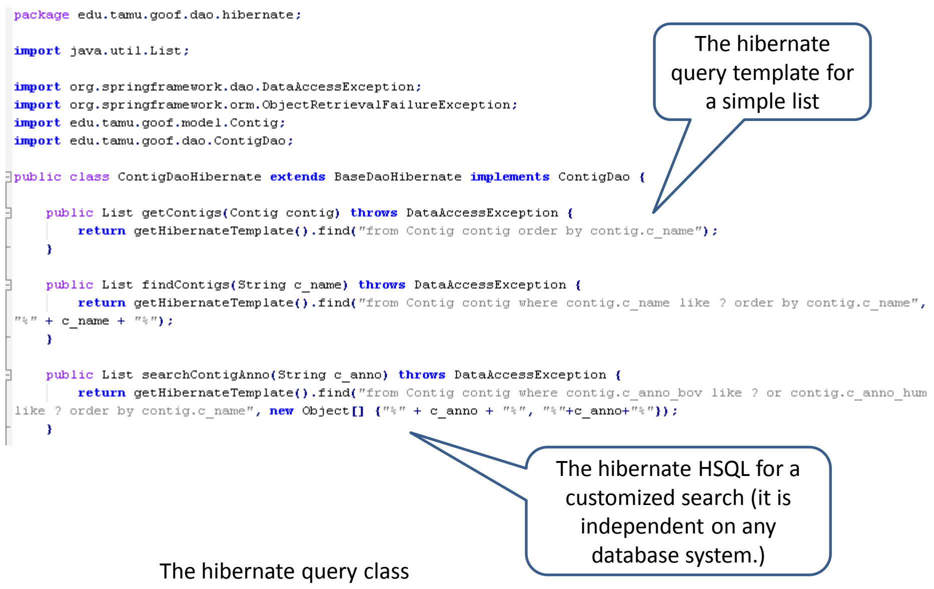


```
package edu.tamu.goof.dao;

import java.util.List;
import edu.tamu.goof.model.Contig;

import org.springframework.dao.DataAccessException;

public interface ContigDao extends Dao {

    public List getContigs(Contig contig) throws DataAccessException;

    public List findContigs(String c_name) throws DataAccessException;

    public List searchContigAnno(String c_anno) throws DataAccessException;

    public Contig getContig(final String c_name);

    public void saveContig(Contig contig);

    public void removeContig(final String c_name);

}
```

The interface to the data transaction layer
from the data object model layer

Figure 10    Example of the interface of a data object

GOOF has an interface to build an SQL query provided by Hibernate. There are several templates available. The module for listing data items on a particular data object is a simple search without any query criteria, which is built on the single-field query model. The module for editing, deleting and updating data objects is currently used in the user management module. It might be deprecated in the view module for genome data objects, unless submitting data through the web interface is required in future. To query key words in a single field or across fields is a basic version of all other complex queries in the framework. Although many relationships are pre-defined into data models, GOOF still has the flexibility to construct arbitrarily complicated query modules. While every query module requires an extra interface module, the data model layer provides an advantage in terms of simple and rapid solutions to deal with the complex relationships between data objects.



```
package edu.tamu.goof.dao.hibernate;

import java.util.List;

import org.springframework.dao.DataAccessException;
import org.springframework.orm.ObjectRetrievalFailureException;
import edu.tamu.goof.model.Contig;
import edu.tamu.goof.dao.ContigDao;

public class ContigDaoHibernate extends BaseDaoHibernate implements ContigDao {

    public List getContigs(Contig contig) throws DataAccessException {
        return getHibernateTemplate().find("from Contig contig order by contig.c_name");
    }

    public List findContigs(String c_name) throws DataAccessException {
        return getHibernateTemplate().find("from Contig contig where contig.c_name like ? order by contig.c_name",
"%" + c_name + "%");
    }

    public List searchContigAnno(String c_anno) throws DataAccessException {
        return getHibernateTemplate().find("from Contig contig where contig.c_anno_bov like ? or contig.c_anno_hum
like ? order by contig.c_name", new Object[] {"%" + c_anno + "%", "%"+c_anno+"%"});
    }
```

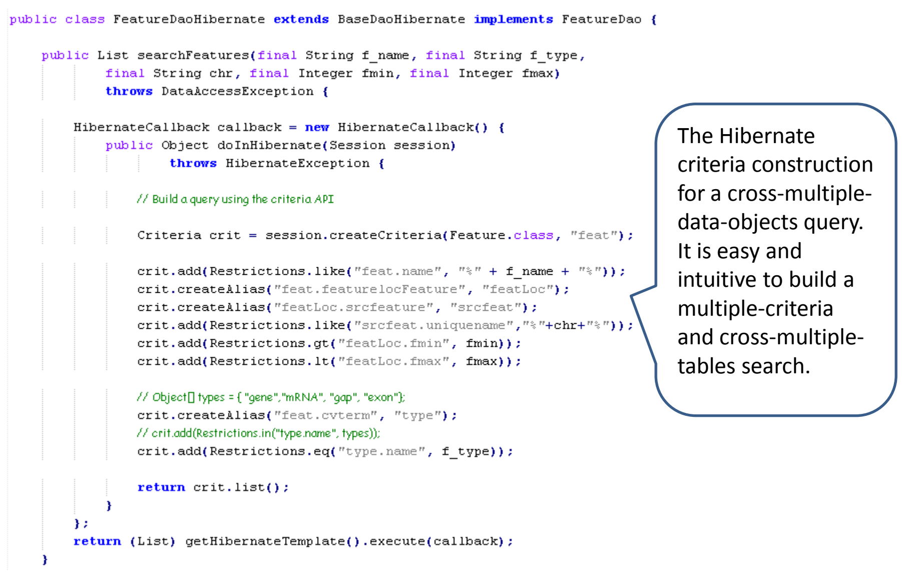
The hibernate query template for a simple list
The hibernate HSQL for a customized search (it is independent on any database system.)
The hibernate query class

Figure 11    Example of a Hibernate query based on a template

We listed several queries which are built from the templates using Hibernate SQL in figure 11. The structure and syntax of those queries are similar to those of standard SQL and independent of any database. Additional SQL functions can be also added to the existing functions. For the simple queries, the template is simple and efficient.

In addition, GOOF can operate data objects without resorting to SQL. Every table and its relationships are presented as data objects in GOOF. Since the relationships between tables in the database are already built into data objects in GOOF, adding criteria to data objects effectively constructs a query, removing the need to explicitly construct a complex query. This simplifies the data operations on the database and avoids direct database query. This is how object-oriented data modeling allows data operations to be independent of the database. However, complicated relationships between data objects could make this type of query construction very difficult. Therefore, constructing a query by using criteria simplifies the construction process.

In constructing criteria based searches, the criteria can be added as logic conditions to the properties of a data object. We can just add “equal”, “like”, “include” and other criteria to the properties of a single data object or multiple connected objects. For instance, we can navigate the complexity of relationships in the Chado schema. In the example, we construct the query to list features of a certain type and name in a range on a target chromosome. This involves four data objects and four their properties. Since the relationships of data objects are also defined as properties, we do not have to build a search with multiple joins. We simply add criteria to a property of the target data object. In figure 12, the example involves five joint criteria cross four data objects. It allows us to constructing a very long SQL based query. Since GOOF is independent of any database system, it does not have to deal the different versions of SQL implemented in various database systems.



```java
public class FeatureDaoHibernate extends BaseDaoHibernate implements FeatureDao {

    public List searchFeatures(final String f_name, final String f_type,
            final String chr, final Integer fmin, final Integer fmax)
            throws DataAccessException {

        HibernateCallback callback = new HibernateCallback() {
            public Object doInHibernate(Session session)
                    throws HibernateException {

                // Build a query using the criteria API

                Criteria crit = session.createCriteria(Feature.class, "feat");

                crit.add(Restrictions.like("feat.name", "%" + f_name + "%"));
                crit.createAlias("feat.featurelocFeature", "featLoc");
                crit.createAlias("featLoc.srcfeature", "srcfeat");
                crit.add(Restrictions.like("srcfeat.uniquename","%"+chr+"%"));
                crit.add(Restrictions.gt("featLoc.fmin", fmin));
                crit.add(Restrictions.lt("featLoc.fmax", fmax));

                // Object[] types = { "gene","mRNA", "gap", "exon"};
                crit.createAlias("feat.cvterm", "type");
                // crit.add(Restrictions.in("type.name", types));
                crit.add(Restrictions.eq("type.name", f_type));

                return crit.list();
            }
        };
        return (List) getHibernateTemplate().execute(callback);
    }
```

The Hibernate criteria construction for a cross-multiple-data-objects query. It is easy and intuitive to build a multiple-criteria and cross-multiple-tables search.

The Hibernate criteria search construction for Chado (No SQL query needed)

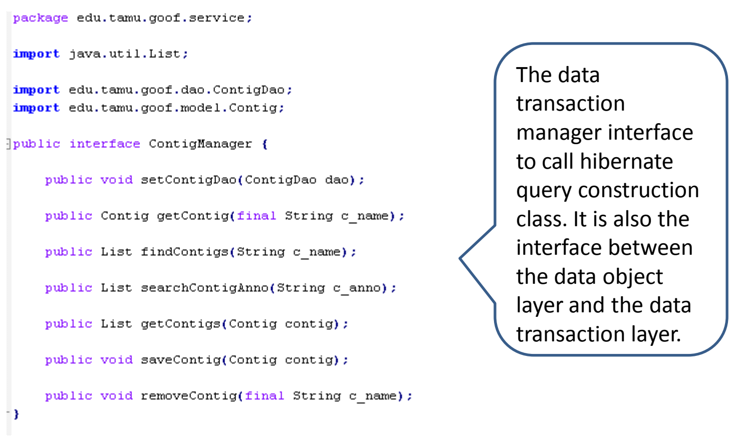
Figure 12 Example of a Hibernate criteria query

The data model layer not only defines the data model mapped to the database, but also provides a way to access those data. It defines an interface through which the data transaction layer acquires the data. It effectively hides the complexity of the data model from the other layers. Any change or reengineering of the data objects would not affect the other layers as long the interface between layers is constant.

## 3.5 The Data Transaction Layer

The data transaction layer acts as the controller module to handle data transactions between data object and data view, but keeps them independent from each other. In another words, it acts a bridge to manage the data flow from the data model layer to the web presentation layer. The data transaction layer delivers the data objects requested by users on the web presentation layer.

A design goal of GOOF is to simplify the interface and reduce data transaction complexity for the view module. In this sense, the presentation layer presents data objects from the data transaction layer and does not need to operate on data directly within the data modeling layer. The principal advantage of this approach is that if any data model changes or any query structure is modified, the presentation layer only operates on the pre-processed data objects and does not change its view layout or interface as a result of changes on the lower level.



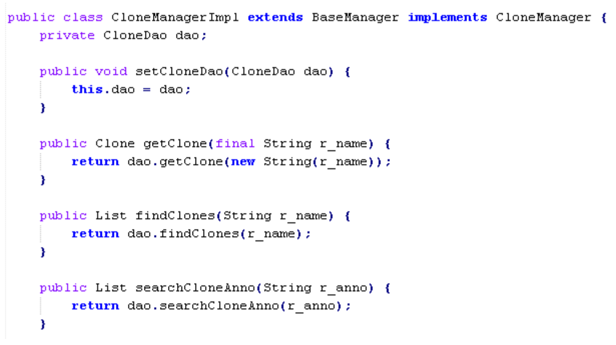
```
package edu.tamu.goof.service;

import java.util.List;

import edu.tamu.goof.dao.ContigDao;
import edu.tamu.goof.model.Contig;

public interface ContigManager {

    public void setContigDao(ContigDao dao);

    public Contig getContig(final String c_name);

    public List findContigs(String c_name);

    public List searchContigAnno(String c_anno);

    public List getContigs(Contig contig);

    public void saveContig(Contig contig);

    public void removeContig(final String c_name);
}
```

The data transaction manager interface to call hibernate query construction class. It is also the interface between the data object layer and the data transaction layer.

The data transaction manager interface

Figure 13 Example of a data transaction manager interface

First of all, a data transaction manager should be established for the target data objects. The transaction manager manages and builds several interfaces for data transaction between the target data objects. Any operation on a data object invoked by users via the web presentation layer is handled by the data transaction manager. The data transaction interfaces are also used to link the data model layer, which does not directly interact with any data object requests. Therefore, any alternation in the data model layer would not break the data flow at the level of the web presentation layer, as long as the interface is unchanged. In figure 13, each interface of the data object “Contig” corresponds to one data operation of “Contig” in the data model layer.



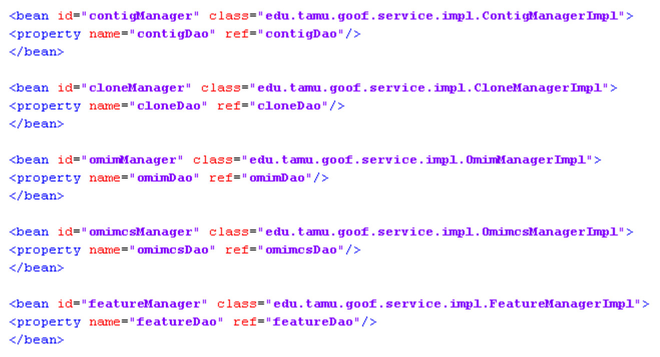
```
public class CloneManagerImpl extends BaseManager implements CloneManager {
    private CloneDao dao;

    public void setCloneDao(CloneDao dao) {
        this.dao = dao;
    }

    public Clone getClone(final String r_name) {
        return dao.getClone(new String(r_name));
    }

    public List findClones(String r_name) {
        return dao.findClones(r_name);
    }

    public List searchCloneAnno(String r_anno) {
        return dao.searchCloneAnno(r_anno);
    }
```

The implementation of a data transaction manager. It basically delivers the variables from the web presentation layer to the Hibernate query java class. But it does not know the construction of the query class.

Figure 14    Example of the implementation of a data transaction manager

The next step is to establish how the data transaction manager communicates with the appropriate data objects. The data transaction manager also transfers the variables from the web presentation layer to the data model layer across the data interface. The data transaction manager communicates with the data object layer for the requested data. The data transaction manager can be changed if any updated data object structures are updated or if any data object query interface is altered. As long as the interface of a data transaction manager is unchanged, web actions do not have to be altered to get data from the data model layer.

In figure 14, the example demonstrates another important point, namely that name of an interface is the same as that of the corresponding query class in the data model layer. This naming rule is not required, but helps to establish the clean variable name space in the framework. In the overall structure section, strict naming rules are helpful when developing and updating a GOOF application. In the meta-programming section, we will discuss how the template generates codes and why it is critical to follow naming rules.



```xml
<bean id="contigManager" class="edu.tamu.goof.service.impl.ContigManagerImpl">
<property name="contigDao" ref="contigDao"/>
</bean>

<bean id="cloneManager" class="edu.tamu.goof.service.impl.CloneManagerImpl">
<property name="cloneDao" ref="cloneDao"/>
</bean>

<bean id="omimManager" class="edu.tamu.goof.service.impl.OmimManagerImpl">
<property name="omimDao" ref="omimDao"/>
</bean>

<bean id="omimcsManager" class="edu.tamu.goof.service.impl.OmimcsManagerImpl">
<property name="omimcsDao" ref="omimcsDao"/>
</bean>

<bean id="featureManager" class="edu.tamu.goof.service.impl.FeatureManagerImpl">
<property name="featureDao" ref="featureDao"/>
</bean>
```

The data transaction manager definitions in applicationContext-service.xml. It links the data transaction manager to the implementation java class and the target managed data object.

Figure 15 Configuration of the data transaction manager

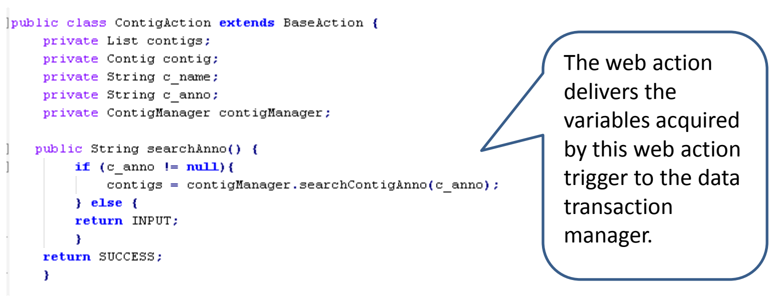
Figure 15 shows how an application defines the data transaction manager to manage a data object. Once data transaction managers have been implemented, they are defined in applicationContext-service.xml file. This allows the framework to access the data objects being managed. The web component passes those Java beans (Java class object) to the variables of JSP. This will be discussed in detail in the next section.

As shown above, the data transaction layer shows how data transaction manager builds the interface between the data model and the web presentation layer. More accurately, the data transaction manger does not do anything, but rather manages the data flow between the data models and the web view. This is the middleware approach. The importance of the data transaction layer is that the data models and the web actions can be easily defined, reused and made to interoperate each other.

## 3.6 The Web Presentation Layer

Users do not want to understand any of the data management, data flow or data mining procedures that comprise a bioinformatics data management system, they just want to transform data into knowledge. The web presentation layer is the interface through which users interact with the whole data management application. The data model and transaction layers are more abstract data construction components in GOOF. The web presentation layer not only acts as another abstract data flow component, but also consists of the user view module. There are three basic components that make up the web presentation layer: web actions, view modules and web interface resource definitions.

- Web actions: these are Java classes, which define the actions invoked by users. They provide the interface to the view module (JSP) and manage the data flow from the view module to the data transaction layer.
- View modules: these are composed of JSP pages, which describe the layout of the user interface. In the view module, all the components, including web actions, menu, titles and interface elements, are defined as variables.
- Web interface resource definitions: in order to manage and maintain the interface, every component of the view module is defined in the web interface resource file.  Every component can easily be reused in anywhere in the interface. It provides a central web interface management tool to reduce the complexity of interface design.



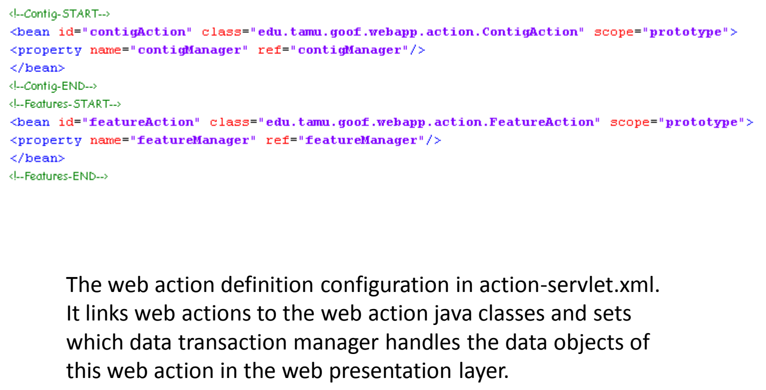
```
public class ContigAction extends BaseAction {
    private List contigs;
    private Contig contig;
    private String c_name;
    private String c_anno;
    private ContigManager contigManager;

   public String searchAnno() {
        if (c_anno != null){
            contigs = contigManager.searchContigAnno(c_anno);
        } else {
        return INPUT;
        }
    return SUCCESS;
    }
```

The web action delivers the variables acquired by this web action trigger to the data transaction manager.

The web action java class in the web presentation layer. It instructs the web action to which data transaction should be called in the data transaction manager. The data transaction manager handles which Hibernate query action should be called.

Figure 16  Web action

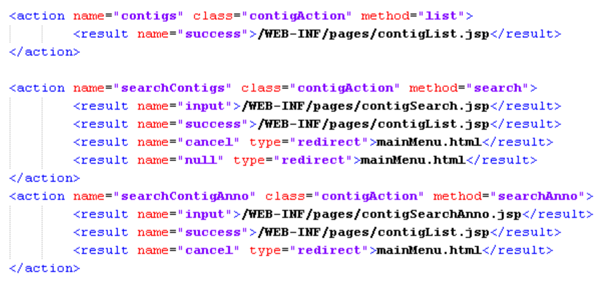
In figure 16, the web action of the data object “Contig” is defined. Each web action class consists of several web actions which are invoked in the same view module. Each web action is connected to a data transaction manager for the data requested by this action and delivers the variables which are needed by that data transaction manager to perform the query. A web action can also instruct a view module via a view interface what content should be delivered to users as a result of that web action. In this example, if it is a failed input from the view module, the web action returns an “INPUT” message to the view module. If the requested data were successfully retrieved from the data transaction manager, it returns a “SUCCESS” message.



```xml
<!--Contig-START-->
<bean id="contigAction" class="edu.tamu.goof.webapp.action.ContigAction" scope="prototype">
<property name="contigManager" ref="contigManager"/>
</bean>
<!--Contig-END-->
<!--Features-START-->
<bean id="featureAction" class="edu.tamu.goof.webapp.action.FeatureAction" scope="prototype">
<property name="featureManager" ref="featureManager"/>
</bean>
<!--Features-END-->
```

The web action definition configuration in action-servlet.xml. It links web actions to the web action java classes and sets which data transaction manager handles the data objects of this web action in the web presentation layer.

Figure 17 Web action configuration

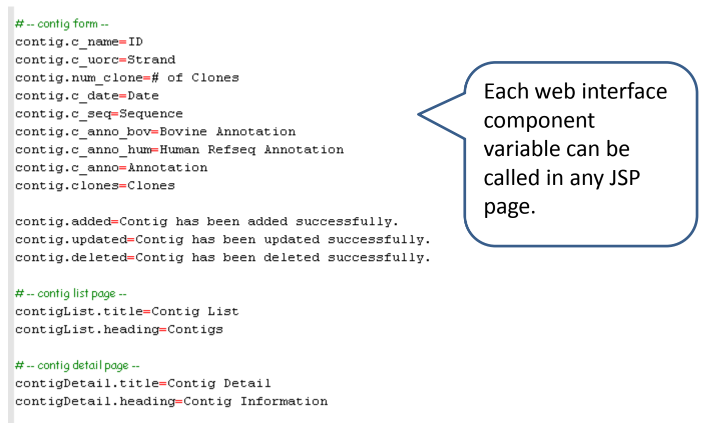


```
<action name="contigs" class="contigAction" method="list">
        <result name="success">/WEB-INF/pages/contigList.jsp</result>
</action>

<action name="searchContigs" class="contigAction" method="search">
        <result name="input">/WEB-INF/pages/contigSearch.jsp</result>
        <result name="success">/WEB-INF/pages/contigList.jsp</result>
        <result name="cancel" type="redirect">mainMenu.html</result>
        <result name="null" type="redirect">mainMenu.html</result>
</action>
<action name="searchContigAnno" class="contigAction" method="searchAnno">
        <result name="input">/WEB-INF/pages/contigSearchAnno.jsp</result>
        <result name="success">/WEB-INF/pages/contigList.jsp</result>
        <result name="cancel" type="redirect">mainMenu.html</result>
</action>
```

The web action directed to which JSP pages are defined in xwork.xml of the web presentation layer. The defined results of each action are corresponded to the definitions in the web action java classes.

Figure 18 Web action and view model configuration

The view module consists of two parts. One is the web action definition, which specifies how the JSPs handle the results of web actions in figure 17. Another is the JSP, which defines the layout of the user interface. As the above example shows in figure 18, each web action of a JSP is linked to the web action generated by Java class in the xwork.xml file. The messages returned from the web action Java class instruct the corresponding JSP to deliver the data objects or messages requested.



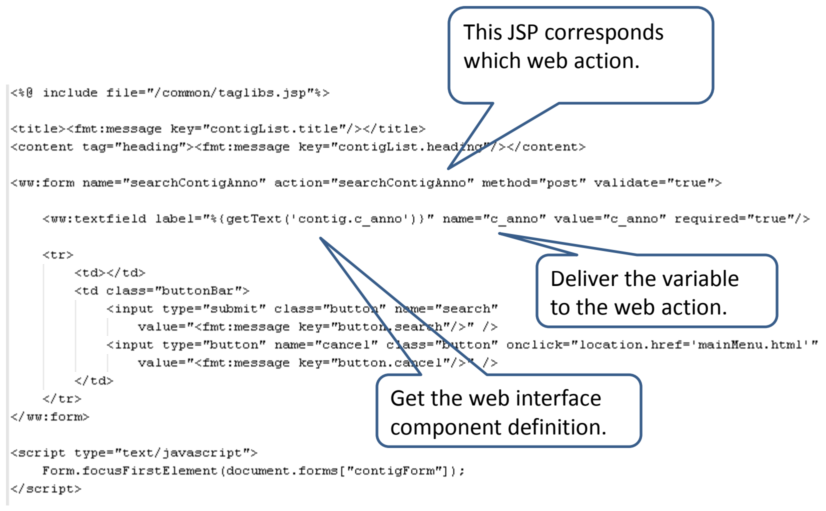
```
# -- contig form --
contig.c_name=ID
contig.c_uorc=Strand
contig.num_clone=# of Clones
contig.c_date=Date
contig.c_seq=Sequence
contig.c_anno_bov=Bovine Annotation
contig.c_anno_hum=Human Refseq Annotation
contig.c_anno=Annotation
contig.clones=Clones

contig.added=Contig has been added successfully.
contig.updated=Contig has been updated successfully.
contig.deleted=Contig has been deleted successfully.

# -- contig list page --
contigList.title=Contig List
contigList.heading=Contigs

# -- contig detail page --
contigDetail.title=Contig Detail
contigDetail.heading=Contig Information
```

Each web interface component variable can be called in any JSP page.

The definitions of web interface components in applicationResources.properties of the web presentation layer.

Figure 19 Web interface component definition

The logical extension of object-oriented design is to use a central application resource file to define the names of menus, readable names of objects, and the content of the web interface in GOOF. One Cascading Style Sheet (CSS) is used to define fonts, colors and layout. The central resource definition in figure 19 and CSS file also allow the interface to be designed as object-oriented models. The view module (JSP) presents the data objects to the web interface and transfers queries invoked by web actions to the data transaction manager. We have three basic templates for the view module: the search view, the list view and the detail view. Every view module defines the layout and function of the interface. The layout consists of web interface components, which are defined as variables and can be reused any view module. The values of the web interface components are defined in the applicationResource.properties. This approach to interface design can significantly reduce the large amount of time required to construct many view modules and also improves the re-use of the interface components.



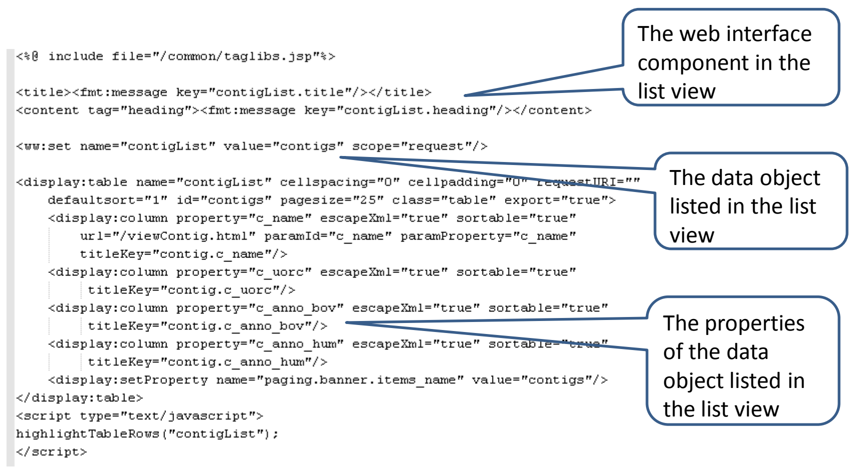
```
<%@ include file="/common/taglibs.jsp"%>

<title><fmt:message key="contigList.title"/></title>
<content tag="heading"><fmt:message key="contigList.heading"/></content>

<ww:form name="searchContigAnno" action="searchContigAnno" method="post" validate="true">

    <ww:textfield label="%{getText('contig.c_anno')}" name="c_anno" value="c_anno" required="true"/>

    <tr>
        <td></td>
        <td class="buttonBar">
            <input type="submit" class="button" name="search"
                value="<fmt:message key="button.search"/>" />
            <input type="button" name="cancel" class="button" onclick="location.href='mainMenu.html'"
                value="<fmt:message key="button.cancel"/>" />
        </td>
    </tr>
</ww:form>

<script type="text/javascript">
    Form.focusFirstElement(document.forms["contigForm"]);
</script>
```

This JSP corresponds which web action.
Deliver the variable to the web action.
Get the web interface component definition.

The search view JSP of a data object in the web presentation layer

Figure 20    JSP search view

The search view in figure 20 is the interface to pass a query to the data transaction layer. The search view not only defines which web action responses the user requests, but also specifies the value variables defined in the web action based on user input. The comprehensive search view allows users to add criteria to relationships within data objects to create a complex query. The drop down list, check box and text input field can be used for the more complicated query construction. The search view can be used to design a data mining interface to invoke a data mining procedure in the data transaction layer. Well thought out good design can hide the complexity of the logic flow from the user. The above example shows the simple text input field in a search view.



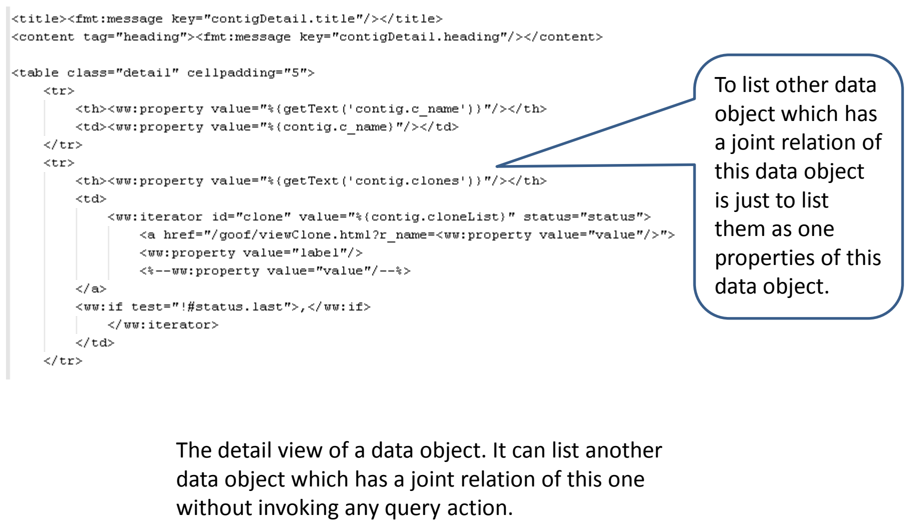
```
<%@ include file="/common/taglibs.jsp"%>

<title><fmt:message key="contigList.title"/></title>
<content tag="heading"><fmt:message key="contigList.heading"/></content>

<ww:set name="contigList" value="contigs" scope="request"/>

<display:table name="contigList" cellspacing="0" cellpadding="0" requestURI=""
    defaultsort="1" id="contigs" pagesize="25" class="table" export="true">
    <display:column property="c_name" escapeXml="true" sortable="true"
        url="/viewContig.html" paramId="c_name" paramProperty="c_name"
        titleKey="contig.c_name"/>
    <display:column property="c_uorc" escapeXml="true" sortable="true"
         titleKey="contig.c_uorc"/>
    <display:column property="c_anno_bov" escapeXml="true" sortable="true"
         titleKey="contig.c_anno_bov"/>
    <display:column property="c_anno_hum" escapeXml="true" sortable="true"
         titleKey="contig.c_anno_hum"/>
    <display:setProperty name="paging.banner.items_name" value="contigs"/>
</display:table>
<script type="text/javascript">
highlightTableRows("contigList");
</script>
```

The web interface component in the list view
The data object listed in the list view
The properties of the data object listed in the list view

The list view of a data object composed in JSP. It consists of data objects, properties of them and web interface components.

Figure 21 JSP list view

The data list view in figure 21 lists the data objects that match the query criteria. The data list view provides a link to the detail view of each data object in table format and provides the options of exporting data as XML, EXCEL and CSV formats. Once the list web action gets the list of data objects from the data transaction manager, it returns a "SUCCESS" message. In the web action definition configuration, this message is directed to the list view module. The list view receives the data object and links to the detail view of each individual data object.



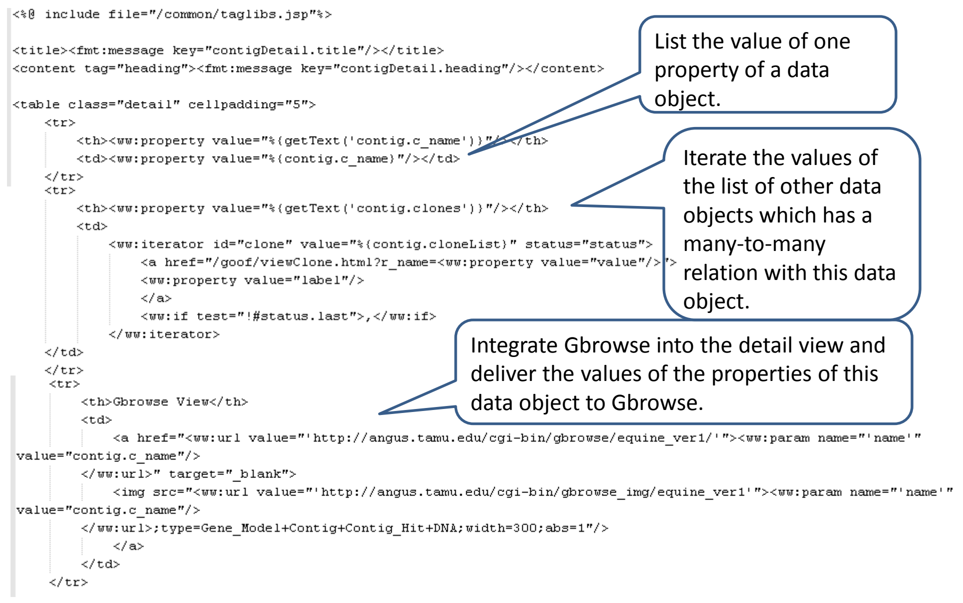
```
<title><fmt:message key="contigDetail.title"/></title>
<content tag="heading"><fmt:message key="contigDetail.heading"/></content>

<table class="detail" cellpadding="5">
    <tr>
        <th><ww:property value="%{getText('contig.c_name')}"/></th>
        <td><ww:property value="%{contig.c_name}"/></td>
    </tr>
    <tr>
        <th><ww:property value="%{getText('contig.clones')}"/></th>
        <td>
            <ww:iterator id="clone" value="%{contig.cloneList}" status="status">
                <a href="/goof/viewClone.html?r_name=<ww:property value="value"/>">
                <ww:property value="label"/>
                <%--ww:property value="value"/--%>
        </a>
        <ww:if test="!#status.last">,</ww:if>
            </ww:iterator>
        </td>
    </tr>
```

To list other data object which has a joint relation of this data object is just to list them as one properties of this data object.

The detail view of a data object. It can list another data object which has a joint relation of this one without invoking any query action.

Figure 22    JSP detail view

In figure 22, the cross data object view is shown as a detail view. The cross data object view is built to list data objects with many-to-many relationships of others. An object-oriented data model can model the relationships with other data objects as normal properties of a data object. The beauty of this approach is seen in the detail view of a data object, where list the other data objects as properties, which have many-to-many relationships with this object. We do not need to construct another query to get the related data object. Therefore, the object-oriented way of constructing the data model can significantly reduce query construction and simplify the detail view module.



```
<%@ include file="/common/taglibs.jsp"%>

<title><fmt:message key="contigDetail.title"/></title>
<content tag="heading"><fmt:message key="contigDetail.heading"/></content>

<table class="detail" cellpadding="5">
    <tr>
        <th><ww:property value="%{getText('contig.c_name')}"/></th>
        <td><ww:property value="%{contig.c_name}"/></td>
    </tr>
    <tr>
        <th><ww:property value="%{getText('contig.clones')}"/></th>
        <td>
            <ww:iterator id="clone" value="%{contig.cloneList}" status="status">
                <a href="/goof/viewClone.html?r_name=<ww:property value="value"/>">
                <ww:property value="label"/>
                </a>
                <ww:if test="!#status.last">,</ww:if>
            </ww:iterator>
    </td>
    </tr>
     <tr>
        <th>Gbrowse View</th>
        <td>
            <a href="<ww:url value="'http://angus.tamu.edu/cgi-bin/gbrowse/equine_ver1/'"><ww:param name="'name'"
value="contig.c_name"/>
        </ww:url>" target="_blank">
            <img src="<ww:url value="'http://angus.tamu.edu/cgi-bin/gbrowse_img/equine_ver1'"><ww:param name="'name'"
value="contig.c_name"/>
        </ww:url>;type=Gene_Model+Contig+Contig_Hit+DNA;width=300;abs=1"/>
            </a>
        </td>
     </tr>
```

List the value of one property of a data object.
Iterate the values of the list of other data objects which has a many-to-many relation with this data object.
Integrate Gbrowse into the detail view and deliver the values of the properties of this data object to Gbrowse.

The detail view JSP of a data object in the web presentation layer

Figure 23 Gbrowse detail view JSP

Modular design allows GOOF to integrate Gbrowse as a visualization module. The presentation layer manages to deliver information from data objects to Gbrowse and bring back the visual presentation of data into the view module. The above example from figure 23 shows how the data detail view module deliver a property of the data object to Gbrowse. The Gbrowse view is then embedded into this detail view module. This approach can deliver data objects to other web data services and construct an interface to exchange data.

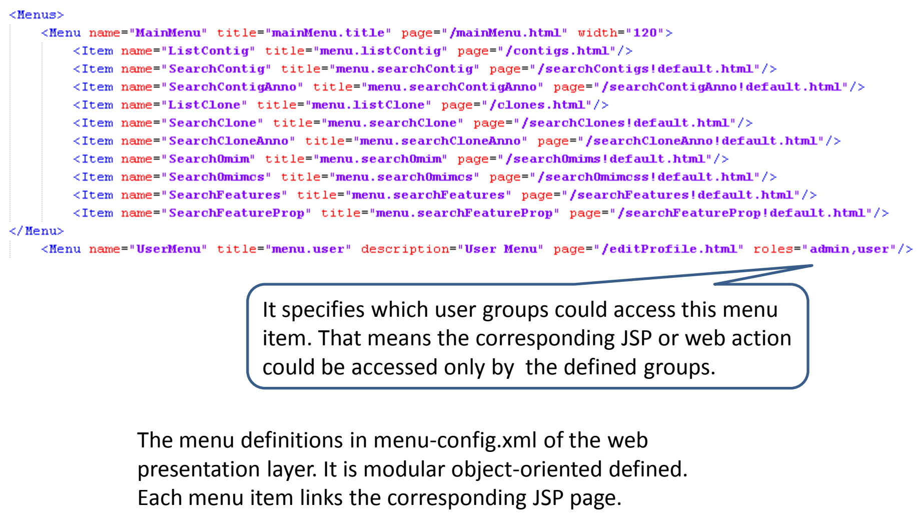
The presentation layer provides GOOF with the functionality to simplify interfaces and a straightforward way to explore data.

## 3.7 Managing User Interfaces

GOOF is proposed as a genome data informatics system, which not only manage data but also provides a user friendly interface. Simplification of the user interface is a critical research subject in GOOF. There are two questions that we need to address for this issue.

- How does GOOF employ an object-oriented data modeling to simplify the query interface?
- How does GOOF provide a user adapted interface based on different data access requirement?

To answer the first question, we need to review the data model layer in GOOF. Users can list many-to-many data objects without queries in the list view. In order to simplify this query process, GOOF deals with this issue in the data model layer. The object-oriented data modeling approach builds relationships as properties of data objects, while the relational database needs to construct another table to present the same relationship. This means that the data module allows users to explore data through the web interface. In the data transaction and web presentation layers, we do not need to construct an extra module to retrieve the relationships from the database. This reduces the complexity of the relationships between as far as users are concerned. In a genome data informatics system, many data relationships are pre-defined by biological explanations, this is a special characteristic of genome data informatics systems and makes this approach feasible. This method of constructing data objects also provides the foundation to query data by using simple criteria and avoids the construction of complicated SQL queries. The data model layer hides the data query complexity from the data transaction and web presentation layer. This simplifies the construction of the user interface for data queries.



```
<Menus>
    <Menu name="MainMenu" title="mainMenu.title" page="/mainMenu.html" width="120">
        <Item name="ListContig" title="menu.listContig" page="/contigs.html"/>
        <Item name="SearchContig" title="menu.searchContig" page="/searchContigs!default.html"/>
        <Item name="SearchContigAnno" title="menu.searchContigAnno" page="/searchContigAnno!default.html"/>
        <Item name="ListClone" title="menu.listClone" page="/clones.html"/>
        <Item name="SearchClone" title="menu.searchClone" page="/searchClones!default.html"/>
        <Item name="SearchCloneAnno" title="menu.searchCloneAnno" page="/searchCloneAnno!default.html"/>
        <Item name="SearchOmim" title="menu.searchOmim" page="/searchOmims!default.html"/>
        <Item name="SearchOmimcs" title="menu.searchOmimcs" page="/searchOmimcss!default.html"/>
        <Item name="SearchFeatures" title="menu.searchFeatures" page="/searchFeatures!default.html"/>
        <Item name="SearchFeatureProp" title="menu.searchFeatureProp" page="/searchFeatureProp!default.html"/>
</Menu>
    <Menu name="UserMenu" title="menu.user" description="User Menu" page="/editProfile.html" roles="admin,user"/>
```

It specifies which user groups could access this menu item. That means the corresponding JSP or web action could be accessed only by the defined groups.

The menu definitions in menu-config.xml of the web presentation layer. It is modular object-oriented defined. Each menu item links the corresponding JSP page.

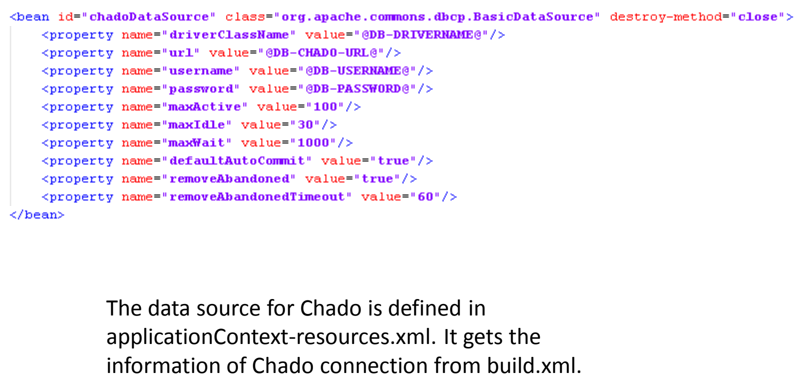
Figure 24 Menu definition

For the second question, we need to look at the user management module in the web presentation layer. The user management module not only manages the user information, but also data access. This module controls the user interface components as well. The dynamic menus driven by user information are implemented in the user management module. Since the web interface components are designed as different objects, the interface and menus components can be specific for user groups, which are independent of the genomic data schema in database. That means the interface can be tailored to the needs of a particular user group without changing any existing data schema. Different groups can have different user interfaces and have different levels of data access. This modular approach gives GOOF flexibility without redundant development. In the example, each menu item is one component in the view module. In figure 24, we define which groups can access this menu item in the "roles" property.

Overall, the object-oriented approach for constructing the user interface is one of many contributions that GOOF brings to the bioinformatics research community.

## 3.8 GOOF and Chado

Chado is a relational database schema that underlies many GMOD installations. It is capable of representing many of the general classes of data frequently encountered in modern biology such as sequences, sequence comparisons, phenotypes, genotypes, ontology, publications, and phylogeny. It has been designed to handle complex representations of biological knowledge and is one of the most sophisticated relational schemas currently available in molecular biology. Since Chado underlies many molecular biology databases and there are several standardized file formats which present biological data and are defined according to the Chado schema, it is critical for GOOF to connect to the Chado schema. The object-oriented structure of GOOF enables the data transaction and presentation layers to manipulate data objects without knowing about the data sources as long as the data objects are well defined in the data model layer. In figure 25, the data manager called by the data transaction layer is defined in GOOF to link to the target data source.

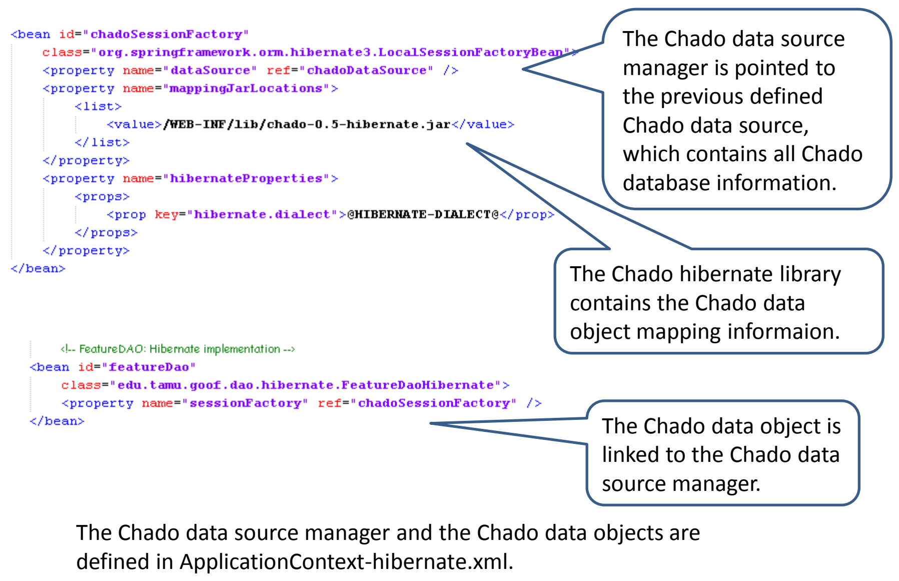


```
<bean id="chadoDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="@DB-DRIVERNAME@"/>
    <property name="url" value="@DB-CHADO-URL@"/>
    <property name="username" value="@DB-USERNAME@"/>
    <property name="password" value="@DB-PASSWORD@"/>
    <property name="maxActive" value="100"/>
    <property name="maxIdle" value="30"/>
    <property name="maxWait" value="1000"/>
    <property name="defaultAutoCommit" value="true"/>
    <property name="removeAbandoned" value="true"/>
    <property name="removeAbandonedTimeout" value="60"/>
</bean>
```

The data source for Chado is defined in applicationContext-resources.xml. It gets the information of Chado connection from build.xml.

Figure 25  Data source definition of Chado

In the section of the overall GOOF structure, we showed how GOOF configures multiple database connections. Once Chado database information is provided, the data source manager needs to be configured to know which data source is for Chado. The data source deification retrieves the variables for Chado database connection from property.xml. Users only need to fill in their own Chado database configuration in property.xml and then GOOF builds the Chado connection into the application. The data source consists of the raw Chado database connection information. After that, the actual Chado data objects will be mapped to the Chado database by the Chado data source manager. In figure 26, we show how the data source manager refers to the actual data source. After that, the Chado data objects used in the application need to be configured for the Chado data source manager. Due to the complexity of the Chado schema, the criteria approach to build Chado queries is recommended and shown in the section of describing the data model layer.



```xml
<bean id="chadoSessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="chadoDataSource" />
    <property name="mappingJarLocations">
        <list>
            <value>/WEB-INF/lib/chado-0.5-hibernate.jar</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">@HIBERNATE-DIALECT@</prop>
        </props>
    </property>
</bean>
```

```xml
    <!-- FeatureDAO: Hibernate implementation -->
<bean id="featureDao"
    class="edu.tamu.goof.dao.hibernate.FeatureDaoHibernate">
    <property name="sessionFactory" ref="chadoSessionFactory" />
</bean>
```

The Chado data source manager and the Chado data objects are defined in ApplicationContext-hibernate.xml.

The Chado data source manager is pointed to the previous defined Chado data source, which contains all Chado database information.
The Chado hibernate library contains the Chado data object mapping informaion.
The Chado data object is linked to the Chado data source manager.

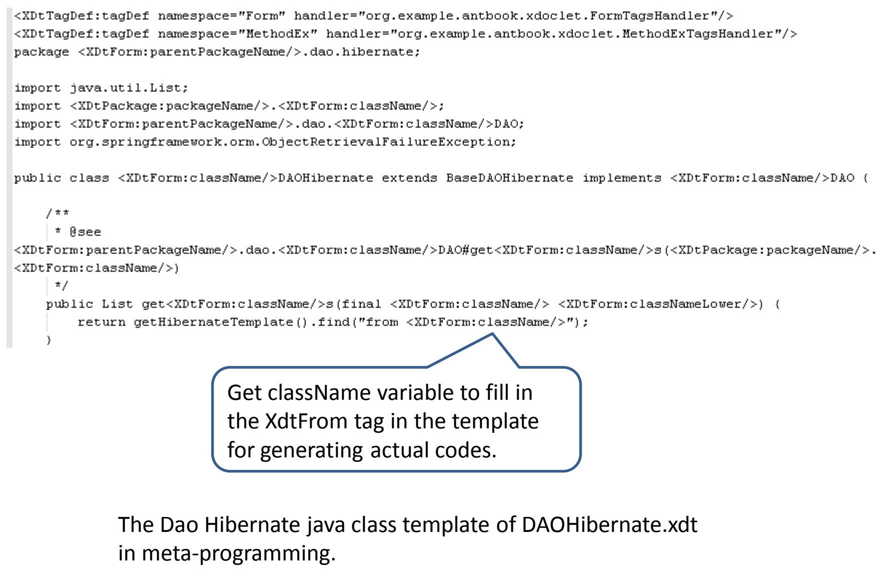
Figure 26    Chado data source manager

In order to remain flexible and independent, GOOF maintains a separated data source schema from Chado, which manages user and database security for the whole system. Another advantage of building GOOF in this way is that many bioinformatics systems already have their existing Chado database. It will allow GOOF to be deployed in many current genome data database systems. Since the connection with Chado in

GOOF is already configured, users can develop their Chado transaction and view models in accord with their needs. Since many GMOD tools are driven by Chado, GOOF can take advantage of Chado to integrate other GMOD tools, such as Gbrowse.

## 3.9 Meta-programming Utility

Program generation is one of GOOF’s important contributions. A database-driven application on top of GOOF contains many reusable components [57]. GOOF can cut the development cost and ease development tasks by using meta-programming. In order to increase the automation of the informatics development task, GOOF employs XDoclet, which generates extra Java classes and applications configuration files according to the conventions of GOOF. The meta-programming utility consists of templates and Java programs annotated with JavaDoc comments to describe what functional components should be used for program generation. The Java templates for classes driven by models in the data object layer can generate classes of data transaction and web presentation layers. The parameterization of meta-programming can unify the similar patterns in the templates. The templates then read the similar generic patterns in the Java class of data object models to generate other Java classes. For this to work properly, meta-component tags in the templates have to be parameterized in fairly restrictive and arbitrary ways. Otherwise, meta-programming might fail to guarantee that generated programs perform correctly. This brings up tradeoffs which we had to consider when developing the framework. The design of meta-programming templates is associated with two several tradeoffs: safety properties and expressive power. The safety properties refer to the integrity of generated programs. In GOOF, the template represents the generated program as abstract syntax trees in a structured meta-programming tool - Xdoclet. This statically ensures that any generated code is correct. The meta-programming tools of GOOF only provide simple and basic templates in order to ensure safety properties. Moreover, there is a tradeoff of flexibility for efficiency between the automation of meta-programming and the generated programs. The Java classes generated by the templates certainly follow the conventions of the function definitions in GOOF. In other words, they carry out their functions in the same way but with different data objects. However, they can still be modified to suit the desires of GOOF developers. The usefulness of meta-programming in GOOF depends on the design goals of the systems. If developers construct functions for different data objects in inconsistent fashion, meta-programming will only generate the basic codes of Java classes, not the whole target codes. In most cases, the majority of applications for managing genome data management present a significant amount of different data in similar ways. In this case, meta-programming could improve the efficiency of development tasks. Another concern is expressive power, which means what meta-programming language can define in the templates and JavaDoc. The meta-programming tools of GOOF can generate extra configuration files for development tasks. However, this brings up the issue discussed above, when safety properties are the first priority, the JavaDoc with certain XDoclet tags in Java programs can generate only part of the basic configuration files. Developers can use them directly in the pre-constructed configuration files provided with GOOF. This aspect of meta-programming leaves the tedious job of application configuration to GOOF. The XDoclet tags in Java programs can dynamically generate configuration files for the whole application when it is compiled. Since GOOF consists of three MVC layers, there are many tedious configuration files that must be integrated together so that the data objects communicated across these three layers. The tags and conventions of comments in Java code could be read at the compile time in order to generate extra code, such as Hibernate mapping files, by the meta-programming tool of GOOF. This function of meta-programming is beneficial to developers of GOOF, because all of these configuration files are mechanical.



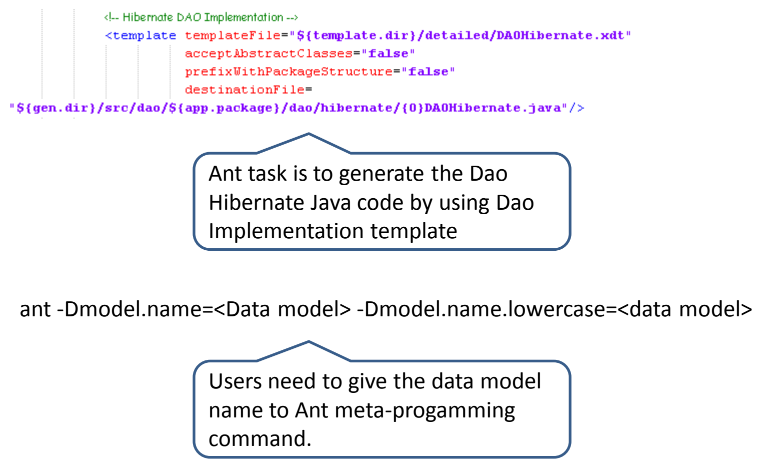
```
<XDtTagDef:tagDef namespace="Form" handler="org.example.antbook.xdoclet.FormTagsHandler"/>
<XDtTagDef:tagDef namespace="MethodEx" handler="org.example.antbook.xdoclet.MethodExTagsHandler"/>
package <XDtForm:parentPackageName/>.dao.hibernate;

import java.util.List;
import <XDtPackage:packageName/>.<XDtForm:className/>;
import <XDtForm:parentPackageName/>.dao.<XDtForm:className/>DAO;
import org.springframework.orm.ObjectRetrievalFailureException;

public class <XDtForm:className/>DAOHibernate extends BaseDAOHibernate implements <XDtForm:className/>DAO {

    /**
     * @see
<XDtForm:parentPackageName/>.dao.<XDtForm:className/>DAO#get<XDtForm:className/>s(<XDtPackage:packageName/>.
<XDtForm:className/>)
     */
    public List get<XDtForm:className/>s(final <XDtForm:className/> <XDtForm:classNameLower/>) {
        return getHibernateTemplate().find("from <XDtForm:className/>");
    }
```

Get className variable to fill in the XdtFrom tag in the template for generating actual codes.

The Dao Hibernate java class template of DAOHibernate.xdt in meta-programming.

Figure 27    Dao Hibernation template for meta-programming

In figure 27, the Dao Hibernate template is illustrated. Templates generate code according to the data model Java class that users define in the data model layer. Users need to give the name of the data model to the Ant meta-programming utility command. As we mentioned above, naming rule are important in GOOF: the file name of the data model Java code must be the same as the name of the data model. The templates use the name of that data model and the variables in the Java code as variables to replace the templates tags. In order to generate code for several templates, there is an Ant build file for this task shown in figure 28.



```
<!-- Hibernate DAO Implementation -->
<template templateFile="${template.dir}/detailed/DAOHibernate.xdt"
          acceptAbstractClasses="false"
          prefixWithPackageStructure="false"
          destinationFile=
"${gen.dir}/src/dao/${app.package}/dao/hibernate/{0}DAOHibernate.java"/>
```

```
ant -Dmodel.name=<Data model> -Dmodel.name.lowercase=<data model>
```

Ant task is to generate the Dao Hibernate Java code by using Dao Implementation template
Users need to give the data model name to Ant meta-progamming command.

Figure 28 Meta-programming Ant build task definition

GOOF has templates for generic data models, transaction models and view models to extend the framework. The templates can be used as automatic code generators, providing a rapid solution to deploy GOOF-based applications.

## 3.10 Summary

GOOF is a Model-Control-View object-oriented genome informatics framework. GOOF provides a built-in connection with Chado and GMOD that has advantages over other genome data informatics systems.

GOOF is released a whole package, including the source code. GOOF can run on any system, UNIX/Linux/OS X/Windows. Here are the software requirements.

- A server side database supported by Hibernate. MySQL and PostgreSQL connections are built into the GOOF. For use with Chado, PostgreSQL is recommended.
- A Java SDK. JDK 5 has been tested. At present, JDK 5 is recommended. All other required Java libraries are included in GOOF.
- Apache Tomcat server. Tomcat 5 has been tested and is recommended. Other Java application servers have not been tested.
- Apache Ant. Ant is the central tool to manage the different development stages in GOOF.

The primary features and contributions of GOOF are listed below.

- The structure of GOOF is built on object-oriented Java, which has the advantages of allowing the framework to incorporate modular design pattern. GOOF is the first object-oriented generic bioinformatics framework. Genomic data can be described in Java by OO design pattern, which simplifies the database schema design and data transaction design.

- GOOF is the first systematic approach to model unstructured biological data. The data transaction and presentation layers are built based on the foundation of the OO data model layer. Therefore, the higher layers can use all kinds of different biological data as an object without worrying the unstructured data. Computer scientists could model unstructured data in other areas by referring GOOF's approach.
- The MVC structure allows GOOF to be independent of database system and makes it easy to manage data object work flow and maintain applications that may be updated as data schema changes. Each single data object integrates the three layers and make GOOF the first integrated from back end to front end framework.
- GOOF can manage multiple database sources, providing the flexibility to adopt any existing database or build a unified web application on top of multiple databases.
- The built-in connection with Chado in GOOF is of benefit to the whole GMOD community, which allows a GOOF application to manage a Chado-based database and integrate other GMOD tools together.
- The meta-programming utility of GOOF allows the rapid generation of large amounts of routine program codes. The templates allow GOOF developers to rapidly deploy similar data objects into a production environment. Developers can construct reusable components by using templates and reduce routine code programming.

- The modular design and the object-oriented structure of GOOF provide a user driven interface and manages role driven data access in the web presentation layer.
- The single unified development management file provides central control informatics application development. This is facilitated by GOOF's the modular design.
- GOOF's object-oriented presentation layer provides a central mechanism to define web interface components. This approach manages user interface components and data access together while simplifying the data model schema. This could be used in other interface frameworks to simplifying interface components without complicating the data structure.
- Since GOOF is Java-base, users are free to migrate their bioinformatics infrastructures to different operating systems.

# 4 EQUINEBASE

## 4.1 Introduction

Equine genomics has been catapulted from a field with a few thousand DNA sequences to one with a full genome sequence within the last two years. As large amounts of Equine genomic sequence became available, horse genome researchers have been forced to find a way to integrate that sequence data with all the other available equine annotations. Without integration, application of the genome sequence to horse genomic research will remain slow and inefficient. In order to facilitate this overall integration, an easy access informatics system for managing and visualizing data is required. This makes GOOF a near perfect solution for building an Equine genome data system. As a demonstration of GOOF's potential, we developed EquineBase using GOOF. Since two versions of the equine genome assemble, we elected to setup up two different database systems. This decision was based on the difference in the degree of annotation available for the two assemblies. The first version had GenBank [58] annotations and other sources of annotated information. For the second version, we had ESTs and Repeats sequence data. The Chado schema was a good choice to build the back end database system, because the annotated sequence data was released in the standard format and we want to take use of other GMOD tools to analyze and visualize the data.

## 4.2 Structure and Content

Since GOOF is a flexible and object-oriented informatics framework, it is easy to integrate multiple database sources and GMOD tools with a single web access interface to provide flexible and powerful search utilities. EquineBase consists of three schemas, which are all implemented as PostgreSQL databases. The GOOF default schema manages the users and groups, which is a part of the security user management schema for EquineBase. It also administers privileges for data access and the role-driven interface for each user group. The other two Chado schemas store the two versions of the equine sequence assembly data and their annotated information. We manage equine sequence data version 1 and 2 in two schemas to reduce the single database load and balance the performance of the databases. In figure 29, we show the overall structure of EquineBase.

EquineBase mainly provides an interface for data mining on Equine genomic sequence data. Although Gbrowse can present data on the web, it has a very limited

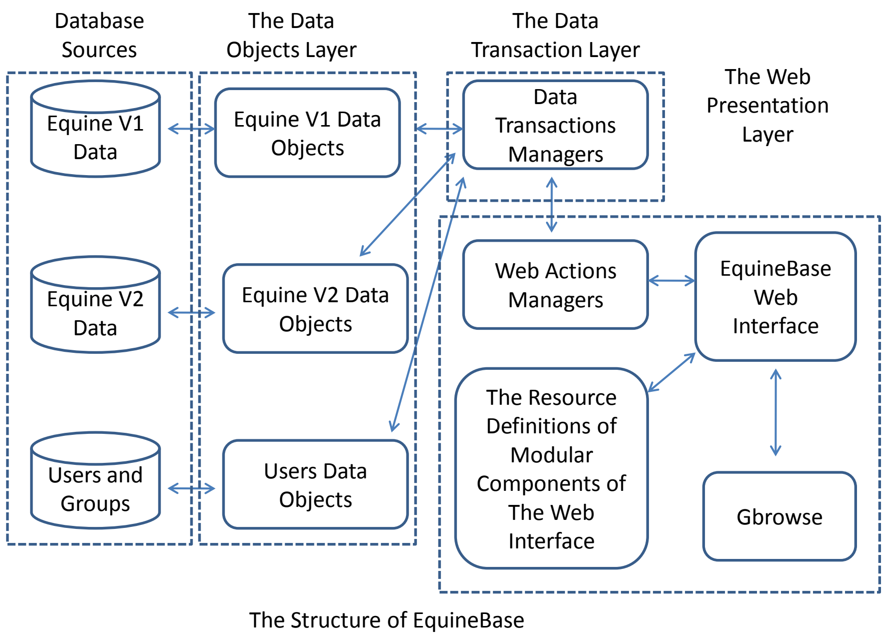
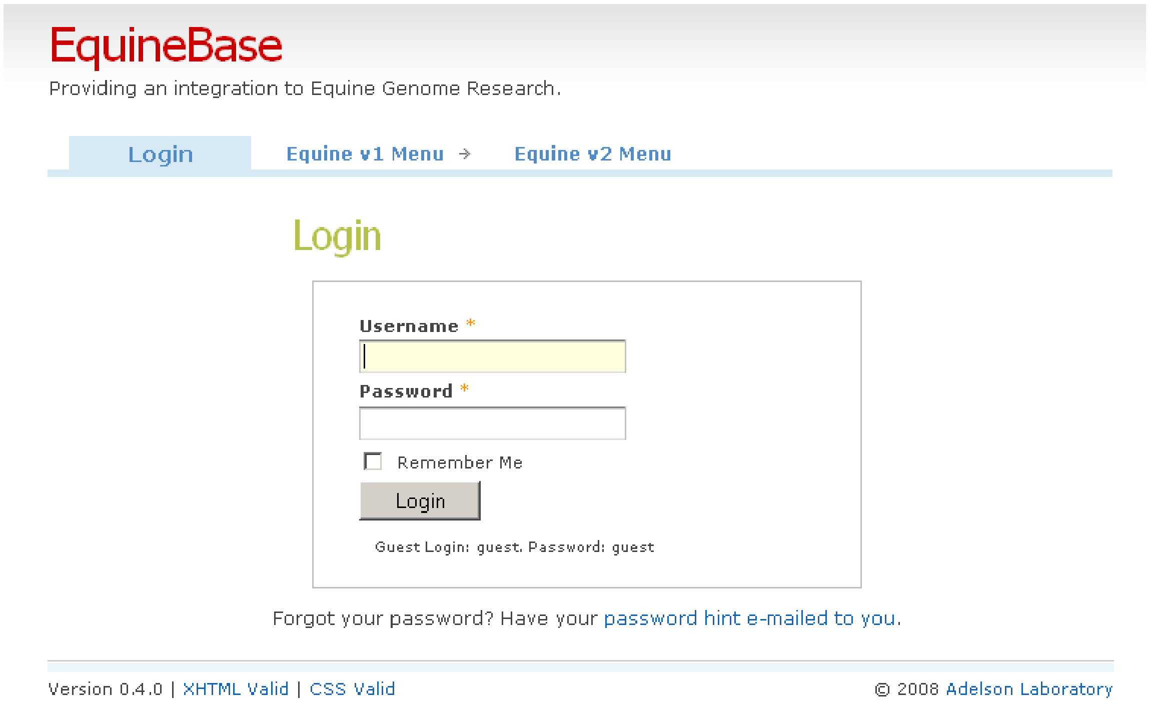


Figure 29 The structure of EquineBase

search and data mining tools. The Gbrowse visualization of the data is not powerful enough to discover all the inherent relationships in the data. In contrast, the data transaction layer of GOOF gives EquineBase many ways to search the data. The templates and MVC structure of GOOF made it easy to build EquineBase.

Figure 30 shows the login interface, which contains the security check managed by the user management module. Figure 31 the detail view of a feature, which is the standard detail view based on the template view module of GOOF. The detail view is integrated with a Gbrowse view, which visualizes the feature on the reference genome and the relationships with other features to help researchers discover more biological associations. There is a link on the Gbrowse view to the full Gbrowse view, which shows more annotated information for each feature.


EquineBase

Providing an integration to Equine Genome Research.

Login | Equine v1 Menu → | Equine v2 Menu

Login

Username *

Password *

☐ Remember Me

Login

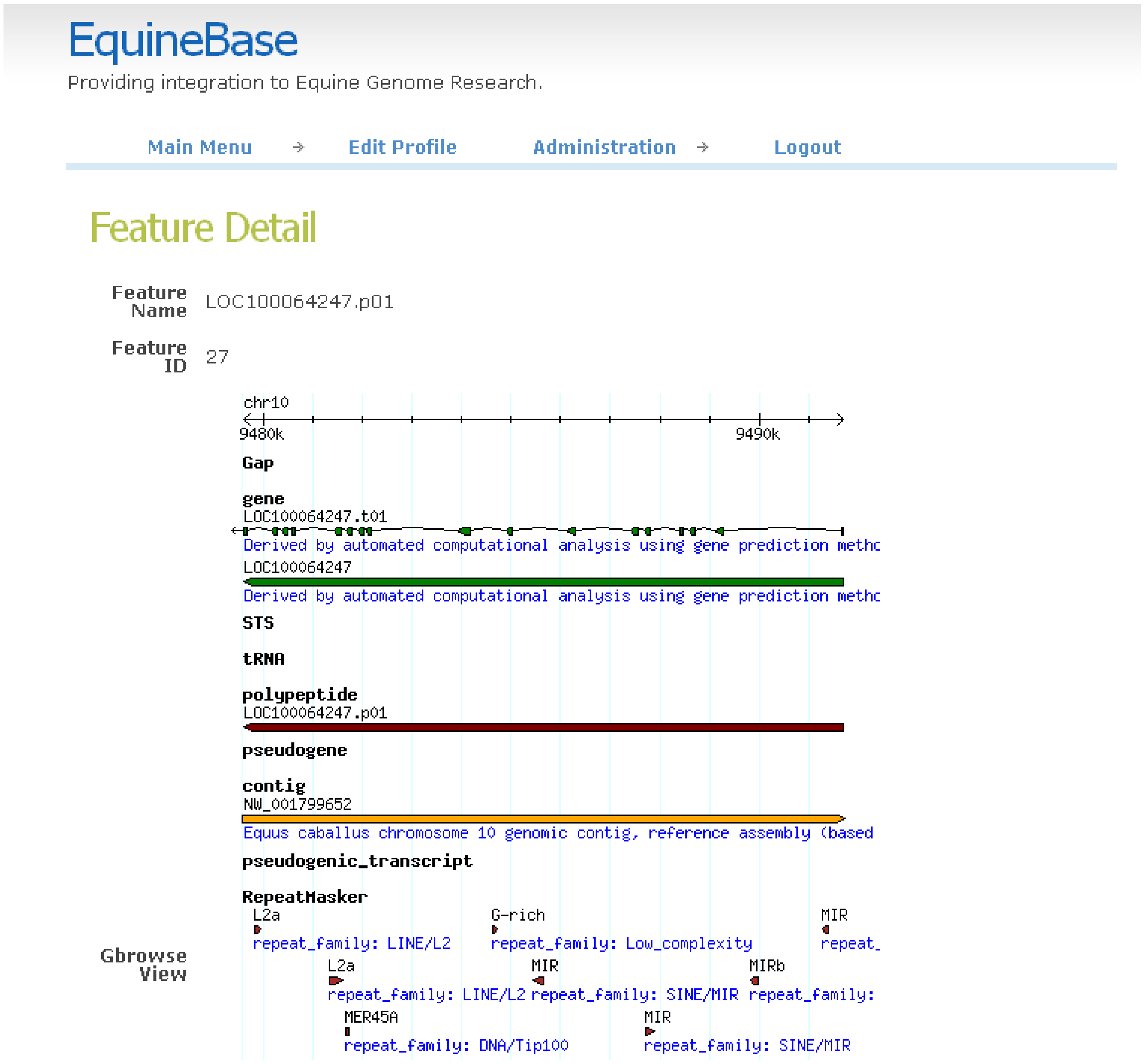
Guest Login: guest. Password: guest

Forgot your password? Have your password hint e-mailed to you.

Version 0.4.0 | XHTML Valid | CSS Valid

© 2008 Adelson Laboratory

Figure 30 EquineBase login interface



Feature Name LOC100064247.p01

Feature ID 27

```
LOC100064247.t01
Derived by automated computational analysis using gene prediction methc
LOC100064247
Derived by automated computational analysis using gene prediction methc
LOC100064247.p01
NW_001799652
Equus caballus chromosome 10 genomic contig, reference assembly (based
L2a                              G-rich                                    MIR
repeat_family: LINE/L2           repeat_family: Low_complexity             repeat_
          L2a                    MIR                        MIRb
          repeat_family: LINE/L2 repeat_family: SINE/MIR repeat_family:
            MER45A                                 MIR
            repeat_family: DNA/Tip100              repeat_family: SINE/MIR
```

Figure 31 EquineBase data detail view

## 4.3 The Search Schema

Initially, we developed the EquineBase schema, including gene annotation, microarray probes, gene mapping, and EST components. After reviewing the Chado schema, we decided to transfer our existing EquineBase schema to Chado. Chado provides a generic organism model and more comprehensive components. This makes it easy to integrate other GMOD tools into EquineBase. Other utility tools in Chado help us handle the raw sequence data in other standard file formats, such as GenBank and GFF3. Even though Chado has more features than necessary for EquineBase, the migration would be a single database and data object mapping and subsequently provide the standard mapping interface of GOOF for other organisms.

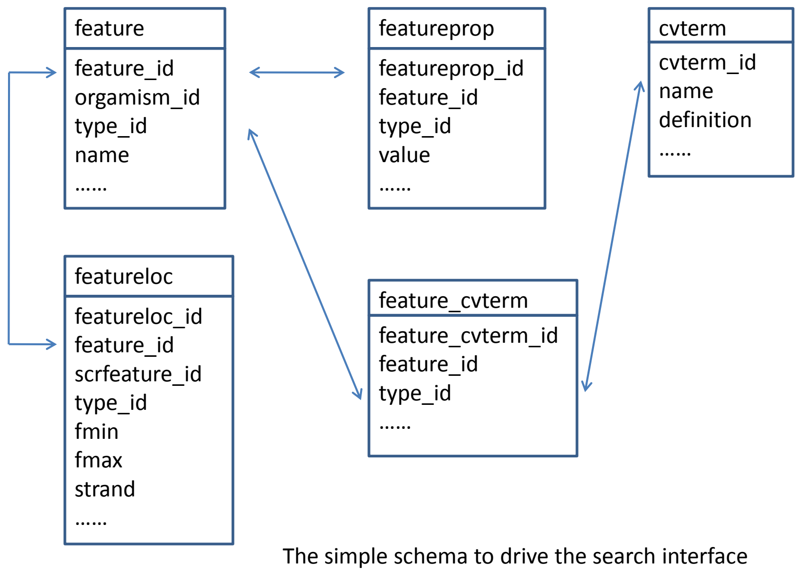
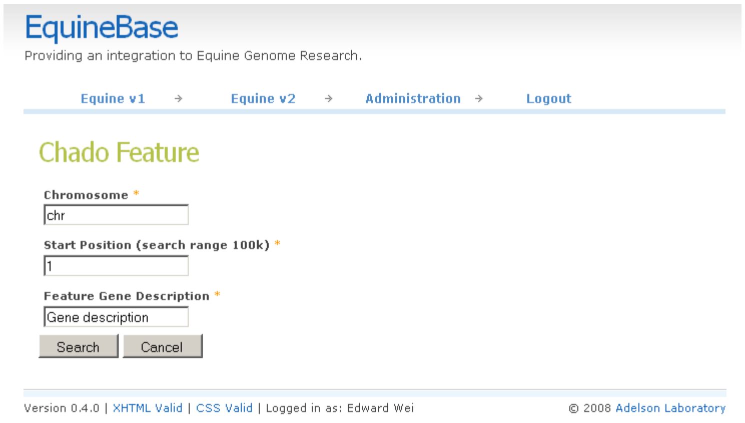


Figure 32 EquineBase basic search schema

The schema to drive the search interface is shown in figure 32 and is a simplified version of the Chado sequence schema. Each feature, which could be EST, gene, mRNA, SNP, repeat or any genomic data object, is listed according to its location on the reference genome. The features are grouped into different tracks or types of genomic data objects aligning on the reference genome. Each object can have an unlimited number of properties. Each property has its own unique properties (tags) and a value. This schema provides search strategies to allow researchers to carry out data mining of the raw genomic data.

- Search equine data features by matching name on a particular track across difference reference genomes.
- Search equine data features across different tracks within a certain distance on a reference genome.
- Search equine data features with matched values of properties cross different tracks.

Figure 33 shows the interface to explore genes based on the matched gene description. This simple search function involves the feature, featureprop, cvterm tables. Cvterm is set to “Gene” track for the features. Featureprop is set to the value which users request, and the link from featureprop to cvterm is set to gene description, the “product” tag. Since the complexity of Chado could confuse users, EquineBase provides a simple interface and functions which Gbrowse does not have.


EquineBase
Providing an integration to Equine Genome Research.
Equine v1
Equine v2
Administration
Logout

Chado Feature

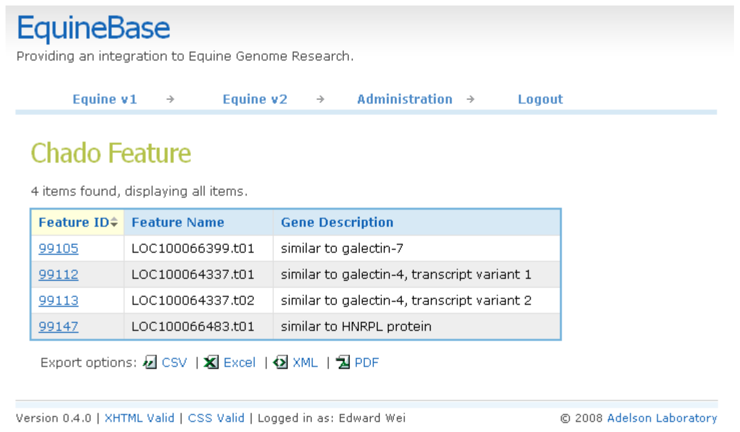
|  |  |
|---|---|
| Chromosome * | chr |
| Start Position (search range 100k) * | 1 |
| Feature Gene Description * | Gene description |

Search
Cancel
Version 0.4.0 | XHTML Valid | CSS Valid | Logged in as: Edward Wei
© 2008 Adelson Laboratory

The search interface to explore gene description in EquineBase

Figure 33 EquineBase search interface

In figure 34, we show the results of a gene description search. Since the property values of features are stored in the featureprop table and the property name is stored in the cvterm table, the search results are actually built using three data objects. In the web presentation layer, the web action binds them together as one data object in order to simplify the web interface layout design and fit the data into the web data list template.



Chado Feature

4 items found, displaying all items.

| Feature ID | Feature Name | Gene Description |
|---|---|---|
| 99105 | LOC100066399.t01 | similar to galectin-7 |
| 99112 | LOC100064337.t01 | similar to galectin-4, transcript variant 1 |
| 99113 | LOC100064337.t02 | similar to galectin-4, transcript variant 2 |
| 99147 | LOC100066483.t01 | similar to HNRPL protein |

Export options: CSV | Excel | XML | PDF

Version 0.4.0 | XHTML Valid | CSS Valid | Logged in as: Edward Wei © 2008 Adelson Laboratory

the search results of gene description on EquineBase

Figure 34 The results of a gene description search in EquineBase

## 4.4 Summary

EquineBase is an integrated online resource for managing equine genomic data and provides a data mining and visualization tool. For equine genomic sequence data to be fully and efficiently explored by animal and biomedical researchers, EquineBase will keep its data updated and add more data mining utilities according to researchers' needs. Since part of database schema of EquineBase was adopted from Chado, it is easy to communicate with remote equine genomic data sources and update the local data source. Because of the object-oriented structure of GOOF, EquineBase is flexible and can add more function modules in the future. In addition, EquineBase also contributes to the development of GOOF as a part of this research study. GOOF provides EquineBase with a systematic approach to model unstructured Equine genome data and a framework to build a data mining interface. Therefore, EquineBase would help enriching the knowledge in the horse research community.

# 5 OMIMBASE

## 5.1 Introduction

OMIM, Online Mendelian Inheritance in Man, is a database of human genes and genetic disorders. It would help researchers studying horse genetic disease to be able to use the knowledge in OMIM in an equine context. In order to do this, one must map OMIM onto the equine reference genome. Since the first version of equine assembly is fully annotated, we have taken advantage of this and fact that human gene annotation exists for many of the entries in OMIM. This information has been used to link the equine genome data and the OMIM data where possible. An application module to support searches of these linked data has been created and is called the Online Mendelian Inheritance in Man (OMIM) Base. OMIMBase is the first resource to provide animal and biomedical researchers with an integrated Horse and OMIM knowledge database. OMIMBase not only integrates both databases, but also builds a data mining utility to discover biomedical knowledge.

## 5.2 Structure and Content

First of all, the OMIM data in NCBI is a standalone flat data file. There is no good facility to search OMIM data, even though NCBI gives the relationships linking OMIM and the Human genome sequence. In order to build the connection from the clinical study portion of OMIM to the Equine genome, OMIM information was processed into a clean format from the NCBI OMIM flat file. OMIM data was then mapped onto the equine genome sequences by BLAST [59]. Finally, we modeled the

OMIM data into Chado. OMIMBase data is displayed as another track on the equine reference genome.

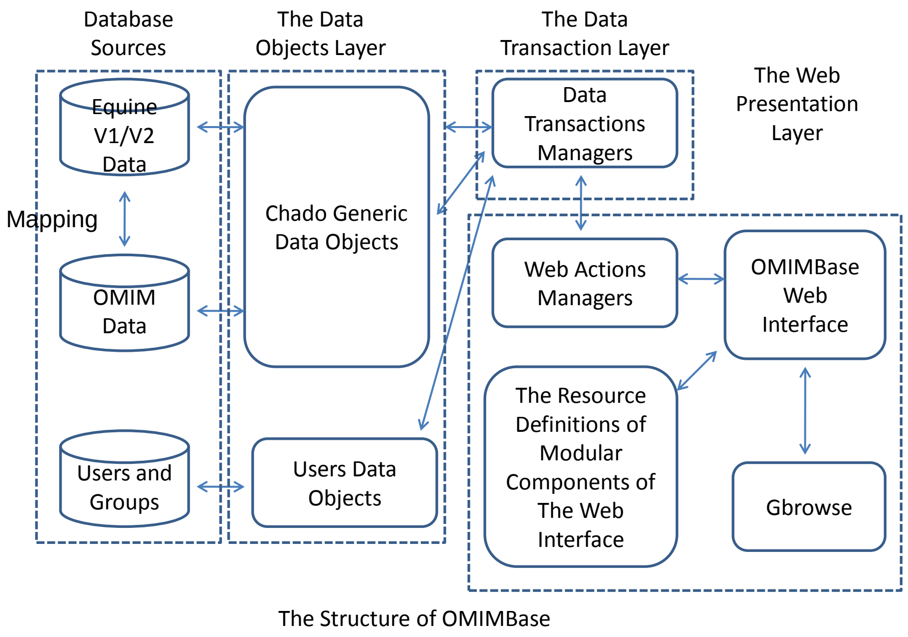
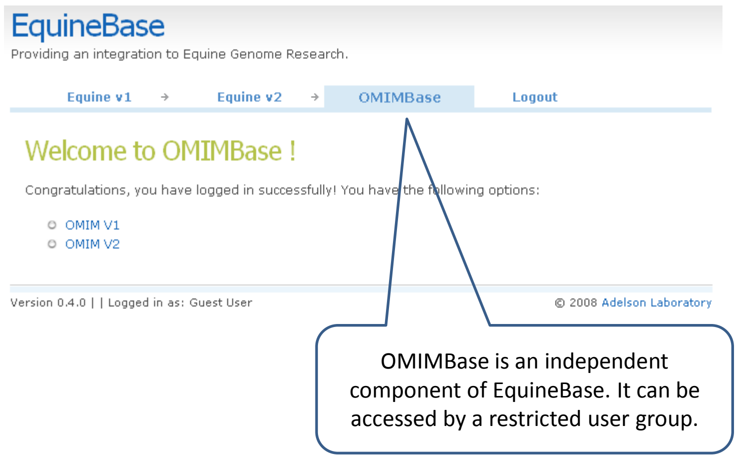

Database Sources
The Data Objects Layer
The Data Transaction Layer
The Web Presentation Layer
Equine V1/V2 Data
Mapping
OMIM Data
Users and Groups
Chado Generic Data Objects
Users Data Objects
Data Transactions Managers
Web Actions Managers
OMIMBase Web Interface
The Resource Definitions of Modular Components of The Web Interface
Gbrowse
The Structure of OMIMBase

Figure 35 The structure of OMIMBase

In figure 35, the overall structure of OMIMBase is the same as EquineBase, because both of them are built using GOOF. They are both modeled using Chado generic data object model. The relationships linking OMIM data and equine genes are built on the equine genome sequence. In other words, they are connected by the same range of locations on the equine genome sequence. We built the key clinical study data of OMIM into our own in-house database. Therefore, OMIMBase can provide researchers with better search functions to discover relationships between human and equine genetic diseases.


EquineBase
Providing an integration to Equine Genome Research.
Equine v1
Equine v2
OMIMBase
Logout
Welcome to OMIMBase !
Congratulations, you have logged in successfully! You have the following options:
OMIM V1
OMIM V2
Version 0.4.0 | | Logged in as: Guest User
© 2008 Adelson Laboratory
OMIMBase is an independent component of EquineBase. It can be accessed by a restricted user group.

OMIMBase as a component of EquineBase

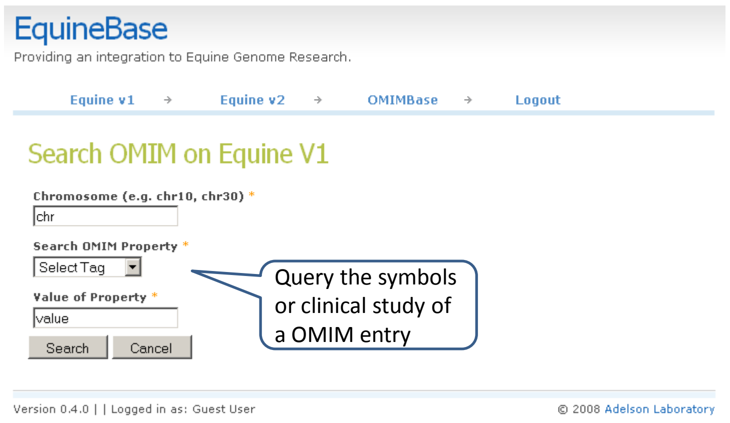
Figure 36 OMIMBase menu in EquineBase

Figure 36 shows that OMIMBase was built as an independent component of EquineBase. The search interface of OMIMBase is a separate menu, which can be accessed by a restricted user group. Since there are two versions of the equine genome assembly, we mapped OMIM data to both of them. We built two data transaction managers for OMIMBase. One is for the first version of the equine genome assembly. The other is for the second version. The search interfaces for OMIM on both of equine genome assemblies can be accessed via the OMIMBase menu. In this fashion, OMIMBase data can be easily updated without affecting EquineBase data, even though they share the same data structure in the database. This latter feature is needed because NCBI updates the OMIM database much more often than genome sequences. The goal of OMIMBase is to provide comparative genomic analysis in order to study genetic diseases in horse and human, so we do not need to keep the OMIM descriptions updated. To take advantages of NCBI's frequent updates of disease and literature reference database, OMIMBase offers a link to NCBI for the detailed clinical study report and reference.

## 5.3 The Search Schema

The schema to drive the search interface in OMIMBase is also based on Chado. However, the search strategy is different from that of EquineBase because of different user requirements. The search results show a list of OMIM entries retrieved, and each entry is linked to a Gbrowse view that shows equine gene products in the same region.

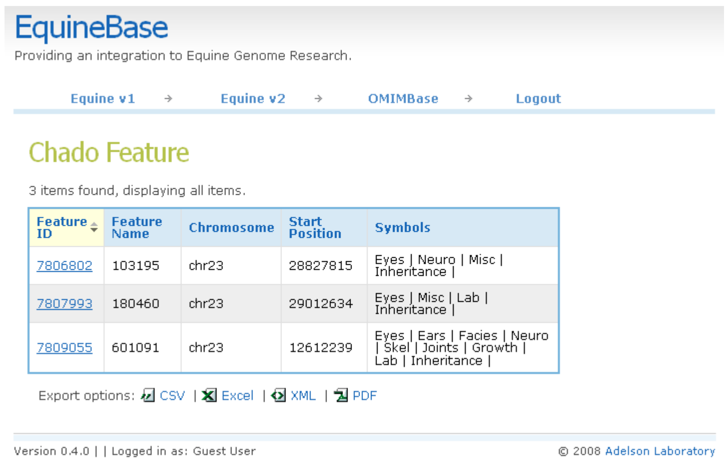
- Search OMIM within a limited region of the certain equine genome.
- Search OMIM according to the clinical study data across the genomes.


EquineBase
Providing an integration to Equine Genome Research.
Equine v1
Equine v2
OMIMBase
Logout
Search OMIM on Equine V1
Chromosome (e.g. chr10, chr30) *
chr
Search OMIM Property *
Select Tag
Value of Property *
value
Search
Cancel
Query the symbols or clinical study of a OMIM entry
Version 0.4.0 | | Logged in as: Guest User
© 2008 Adelson Laboratory

A OMIMBase query interface to search the symbol or clinical study of a OMIM in a certain Equine genome.

Figure 37 OMIMBase search interface

Figure 37, for instance, shows an interface for searching OMIM data on an equine reference genome. On this interface, OMIM data can be searched based on either the clinical study or the gene symbols. Both of search functions were built using the Chado feature property search template, which is also used by several search functions in EquineBase. This is benefit from the generic data structure and the flexible MVC structure of GOOF.



Chado Feature

3 items found, displaying all items.

| Feature ID | Feature Name | Chromosome | Start Position | Symbols |
|---|---|---|---|---|
| 7806802 | 103195 | chr23 | 28827815 | Eyes \| Neuro \| Misc \| Inheritance \| |
| 7807993 | 180460 | chr23 | 29012634 | Eyes \| Misc \| Lab \| Inheritance \| |
| 7809055 | 601091 | chr23 | 12612239 | Eyes \| Ears \| Facies \| Neuro \| Skel \| Joints \| Growth \| Lab \| Inheritance \| |

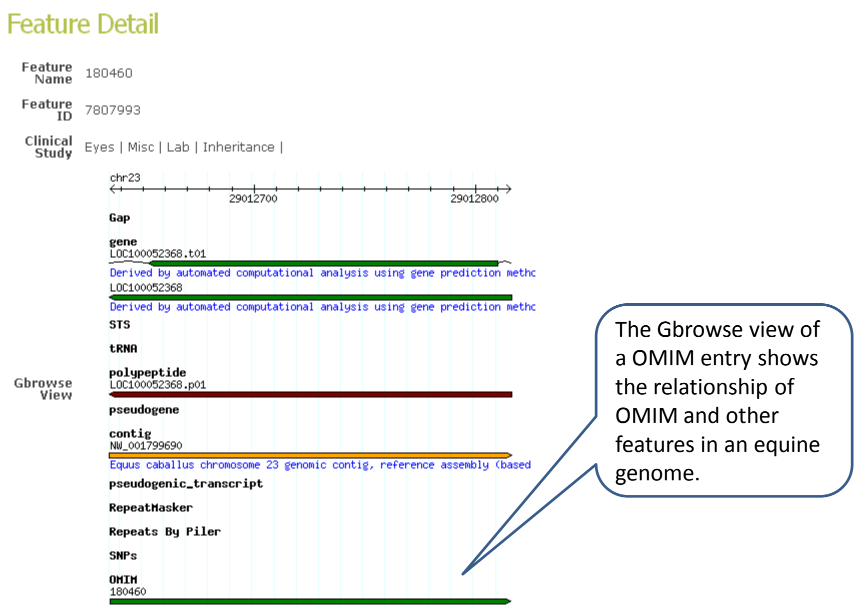
Export options: CSV | Excel | XML | PDF

Version 0.4.0 | | Logged in as: Guest User © 2008 Adelson Laboratory

OMIMBase search results using the data object list template

Figure 38 OMIMBase search results list

Figure 38 shows an OMIM search result example, which is built on the data object list template. Users can sort the results based on feature name, start position, or symbols to quickly identify the most interesting data. Figure 39 shows the OMIM detail view, which shows only a brief description of clinical study information. For the further details of a clinical study report and references, users can click the link on the Gbrowse view to NCBI OMIM database. One of many contributions by OMIMBase is that the search functions for OMIM not only provide an interface to search for human genetic disorders in OMIM, but also present the otherwise cryptic relationships between OMIM and features on the equine genome, such as genes, SNPs, repeats etc.. That cryptic knowledge is not directly presented in the database, but can be discovered by using OMIMBase.



Feature Detail

| | |
|---|---|
| Feature Name | 180460 |
| Feature ID | 7807993 |
| Clinical Study | Eyes \| Misc \| Lab \| Inheritance \| |

Gbrowse View

chr23
29012700
29012800
Gap
gene
LOC100052368.t01
Derived by automated computational analysis using gene prediction metho
LOC100052368
Derived by automated computational analysis using gene prediction metho
STS
tRNA
polypeptide
LOC100052368.p01
pseudogene
contig
NW_001799690
Equus caballus chromosome 23 genomic contig, reference assembly (based
pseudogenic_transcript
RepeatMasker
Repeats By Piler
SNPs
OMIM
180460

The Gbrowse view of a OMIM entry shows the relationship of OMIM and other features in an equine genome.

OMIMBase the detail view of a OMIM entry

Figure 39 OMIMBase OMIM detail view

## 5.4 Summary

OMIMBase is developed as an independent component of EquineBase, which enhances the scientific importance of equine genome data and OMIM. As the first online resource that links OMIM data and the equine genome, OMIMBase provides animal researchers and biomedical scientists with a data mining tool and an integrated knowledge database of OMIM and horse. OMIMBase will add more data and organize the OMIM clinical study data in response to the new OMIM data and annotations of equine genome. OMIMBase will keep updated with NCBI OMIM and will have more data mining functions. The main difficulty at the current development stage of OMIMBase is that the clinical study data in NCBI OMIM is in free text format. One big contribution of OMIMBase is to model unstructured OMIM and Equine genome data and provide a data mining interface. That accelerates the Equine genetics research.

OMIMBase, as a GOOF application, demonstrated the flexibility of the OO structure of GOOF. The user driven interface can control OMIMBase as an independent application for certain user group. In other words, a GOOF application can add on another GOOF application as a controllable module, which would not affect the original structure of the application.

# 6 EVALUATION

After the general release of GOOF, EquineBase and OMIMBASE, user feedback has led to improvements on them and helped us evaluate whether or not GOOF, EquineBase and OMIMBase reached our original expectations. Critical evaluation criteria for a genome informatics framework are critical to evaluate the applicability and flexibility [60] [61]. Only a successful application will demonstrate how GOOF contributes a bioinformatics system development. We will first discuss the improvements to GOOF based on the development of EquineBase and OMIMBase. Then, we will compare the features of GOOF with those of other bioinformatics system. We will discuss two parts of evaluations on this whole research work: one is GOOF developers' feedback and another is EquineBase and OMIMBase's user survey. Since GOOF was proposed as a generic bioinformatics, we have conducted a survey to different levels of bioinformatics developers to evaluate whether GOOF solves the research issues discussed in the introduction section. Since the user interface is one part of this research study, we have evaluated the GOOF interface through user feedback of EquineBase and OMIMBase.

## 6.1 Improvements

Since applicability and user feedback are part of this research study, a demonstration was required. Two independent applications, EquineBase and OMIMBase, show the applicability and interface issues of GOOF.

GOOF was initially proposed as an informatics framework more for the developers of a genome informatics infrastructure. However, GOOF targets genome data management. When we started to build EquineBase and OMIMBase, we concluded that we should incorporate characteristics of genome data and existing bioinformatics system into GOOF. The object-oriented structure of GOOF is flexible enough to be expanded and integrate other modules. Here are the improvements of GOOF which we incorporated from the development of EquineBase and OMIMBase.

- GOOF expands to a multiple database driven front end development framework. The user and security schema is independent of any data schema. Data access is shifted to the presentation layer and is controlled by the interface modules based on the user schema.
- GOOF sets up a separated data source manager to link the data model layer and the target data database. Therefore, data objects in the framework can ignore their data sources and make development independent of the database system.
- GOOF improves the data model layer and is built with the Chado connection. This makes it easy to build a GOOF application on a Chado schema database and integrate other GMOD tools.
- GOOF adds the criteria search module into the data transaction template. This makes it easy to build a multiple joint query on a complicated database schema, such as Chado.

## 6.2 Comparison of Different Frameworks

The major difference of GOOF from other java frameworks is that GOOF interlinks the data model, data transactions and presentation layer together. In order to provide a simple and rapid solution, GOOF uses a single data object across three MVC layers at the expense of losing some functions found in other frameworks. Moreover, any informatics infrastructure create using GOOF must follow strict rules. With other bioinformatics system, GOOF provides a generic genome data management framework. The object-oriented structure of GOOF leads a flexible and simple solution compared to other frameworks.

GOOF is an integration of three MVC layers. Hibernate, iBase, Pure JDBC are several data access Java frameworks. They are all flexible, but a large amount of work is required to configure them and connect the database. Spring, Struct, Tapestry and WebWork are frameworks for data transaction and data presentation development. They can provide many functions for the interface. None of them are integrated together because that was not their original goal. GOOF links them together and many configurations are available within GOOF. Database connections, data sources, data transaction managers and web components are defined. Since the goal of GOOF is to provide a rapid method for implementing a bioinformatics infrastructure, the data model and data transaction targets the characteristics of genome data. The connection of GOOF with Chado is a significant contribution. The templates provided by GOOF are developed for the genome data model. In summary, GOOF is an integration framework built using other basic Java framework.

Until now, the most mature bioinformatics system to make use of Chado was Gbrowse. But Gbrowse is only a data visualization tool for genome data, and it lacks a comprehensive data query interface and user data access management. Another web front end is Turnkey, which lacks many recent developments due to its inflexible structure. LAD and SMD are developed as gene expression data systems, which do not manage general genome data. Moreover, users of LAD and SMD have to use a specific schema for gene expression data. Chado is a very comprehensive genome database schema, including mapping, sequence, library, phenotype, organism modules and so on. The flexible object-oriented structure of GOOF has been adapted to Chado and can add any customized data schema from existing bioinformatics system. The modular design of the interface allows a GOOF application to be scaled to a large application. Since GOOF can manage multiple data schemas, a separated user schema reduces the complexity of the data schema but retains data access control in the presentation layer. This mechanism gives GOOF an advantage over other bioinformatics systems in term of user access control, since others systems either build user data access into the data schema or do not have it.

## 6.3 Evaluation of User Feedback

For the developers to choose GOOF, the most important criterion is why they would pick GOOF over other frameworks to construct new systems. Since GOOF only targets certain applications, it will face only particular groups of developers; biology researchers who need their own informatics system to manage genome data. The development time, lines of codes and reduction of maintenance are the major evaluation matrices. If the database is based on Chado, they do not need to develop a data model for GOOF. They only need to customize the data transaction query and the web interface according to their needs. If they want to develop an individual data schema, they only need to describe a data object in the data model layer and then GOOF will automatically build the data schema and the target database. The templates provided by GOOF help developers significantly reduce the amount of time needed to develop a data query transaction. The modular design of the interface and the centralized management of web interface components reduces both the development time and the amount of redundant code programming. The MVC structure reduces the maintenance of a GOOF application, because any change in any layer will not interrupt others as long as the interfaces are the same.

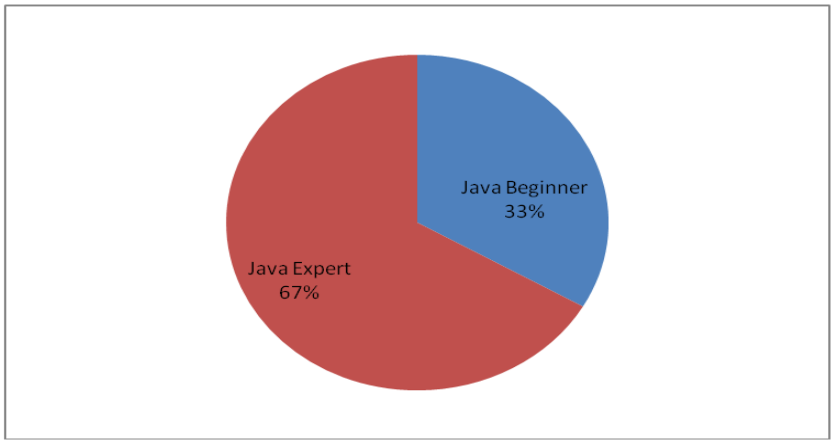
A list of questions was created towards evaluating the goals and features of GOOF. We conducted the survey with different types of developers and wanted to see how developers view GOOF as a generic bioinformatics framework. There were three participants: one is a bioinformatics developer of Bovine Genome Consortium, who works on a similar type of data and is a PHP developer with a little experience on Java; the other is a Java web application but not a bioinformatics system developer; the other is a C/C++ database application developer with few experience on Java and interface. We address the following questions and purposes in the survey.

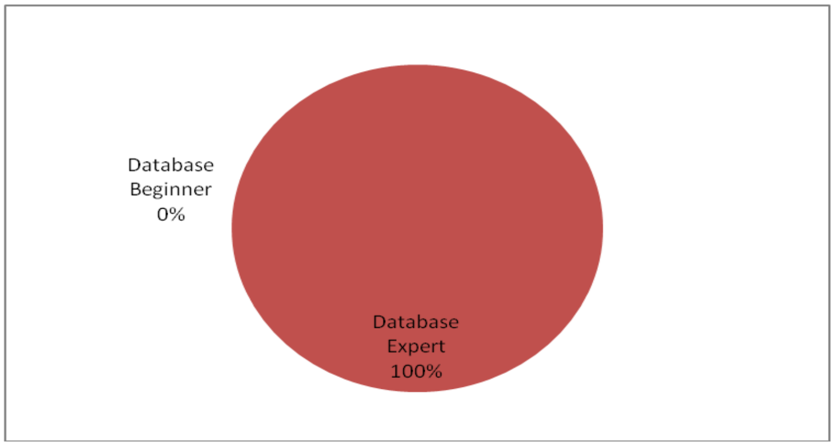
- Is the MVC structure of GOOF generic and flexible enough to construct a new project? Does the integrated structure of GOOF help the development procedure? Does the data model layer make it easy to construct a database application?
- Does GOOF reduce the programming lines of work for a bioinformatics/web

database project? What is the percentage of reduced code that you estimate by comparing with other approaches? Please compare it to different programming languages and frameworks that you have used before.

- Does GOOF reduce the time of development work since it is a new framework for a developer? How about the learning curve?
- Does GOOF help to update a bioinformatics system when adding new functions and interface?
- What are your suggestions for GOOF? What are the advantages and shortcomings of GOOF? What is the wish list of features of GOOF?

Three of them completed the survey and gave us some additional feedback. Figure 40 shows the profiles of GOOF developers in the survey.





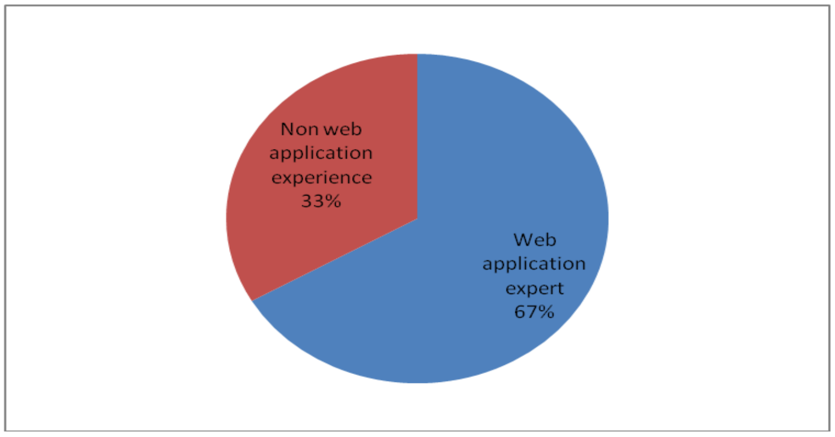


Figure 40 Developers' profiles in GOOF survey

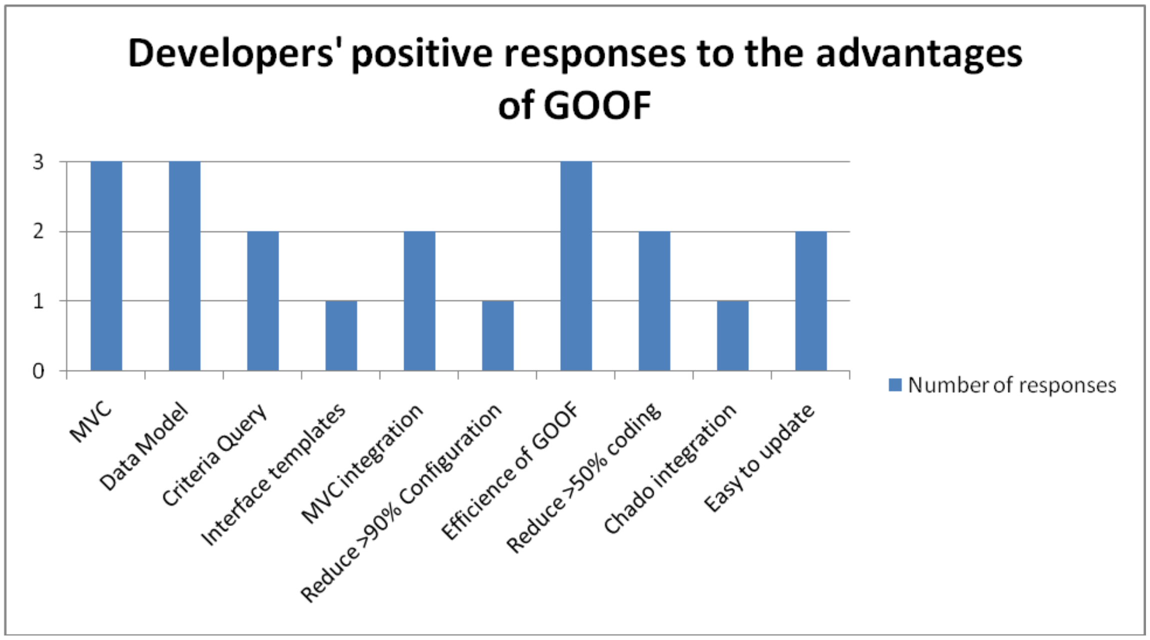


Figure 41 Developers' positive responses to the advantages of GOOF in the survey

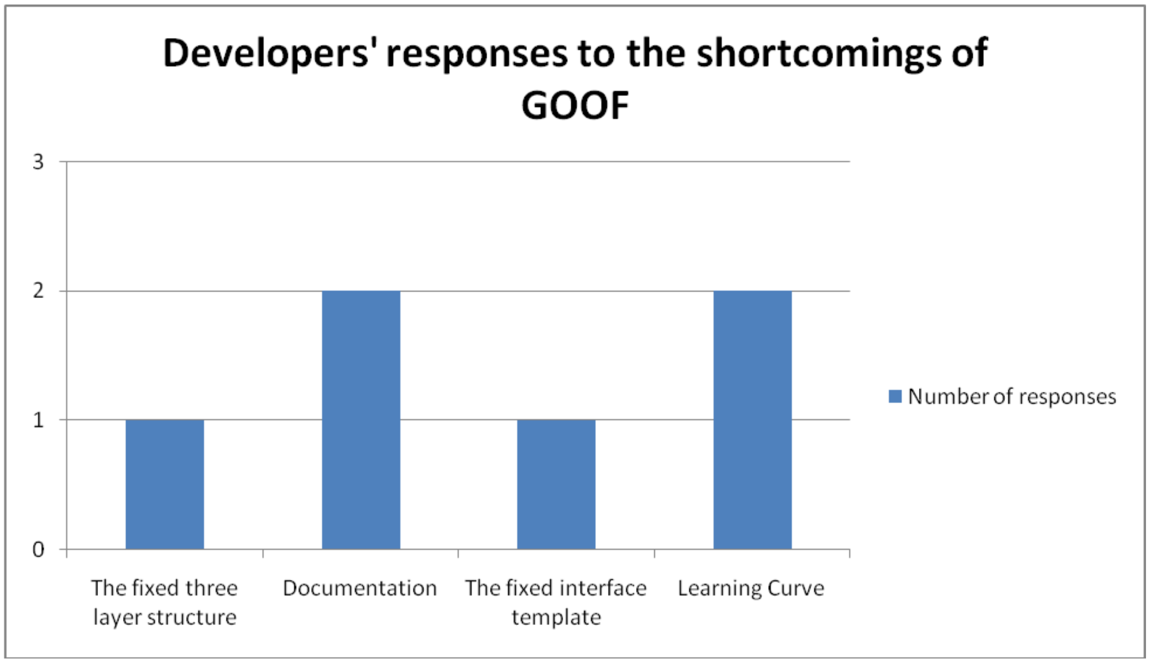


Figure 42 Developers' responses to the shortcomings of GOOF in the survey

Figure 41 shows the summary of developers' responses to the advantages of GOOF and figure 42 shows the summary of developers' responses to the shortcomings in the survey. We summarized several aspects from a development point of view on GOOF.

- For developers who are not familiar with object-oriented concept, once they understand the structure of GOOF, MVC layers and the data model approach are efficient and effective. The data model layer is flexible to build a database application fast on either a new database schema or an existing one by comparing to PHP or C/C++.
- Three independent layers make it individually to construct each component in a GOOF application. The centralized resource definitions in the interface layer keep it easy to manage and develop interface comparing with PHP and C/C++ approaches.
- By comparing the number of files and cumulative lines in the PHP approach on a bioinformatics project, the object-oriented approach presented by GOOF reduces the code by approximately fifty percent. With updated data and functions in a bioinformatics web application, it could reduce the code further.
- The data model layer and templates in GOOF significantly reduce the time on developing a bioinformatics system, but are not flexible and suitable on other web applications.
- For non-Java developers, there is a learning curve. However, the clarity of

MVC design pattern helps developers understand GOOF and apply it on a bioinformatics system construction.

- Making modifications and updates has been streamlined, because all levels of the GOOF structure are independent of one another. Applications developed by using GOOF reduce the cost of the future maintenance and migration.

For end users of a GOOF application, GOOF simplifies the construction of user interfaces by using OO structure and models. This modular approach could allow GOOF to evaluate different groups and then construct a best fit interface. The user survey helped us understand how certain groups of users feel about the functionality of interfaces. Two major measurements of usability are the time which users take to reach their target data, and expert opinions on how the interface is easy to use. The evaluation results will feed back to the future development on the interface layer of GOOF and EquineBase and OMIMBase.

We conducted a question survey with total five people of Maxwell H. Gluck Equine Research Center and Texas A&M University College of Veterinary Medicine on EquineBase. We also made an observation on the user experience of EquineBase with one person at Texas A&M University. They are all genetic or biomedical researchers who have experience with many different bioinformatics web applications. Only one of them has some knowledge of databases. The survey questions aimed at determining whether or not the interface is appealing and easy to understand its function. The observation was conducted to see how fast the user learns the interface and their responses. Figure 43 shows the profiles of users in the survey. Here are the questions in the survey.

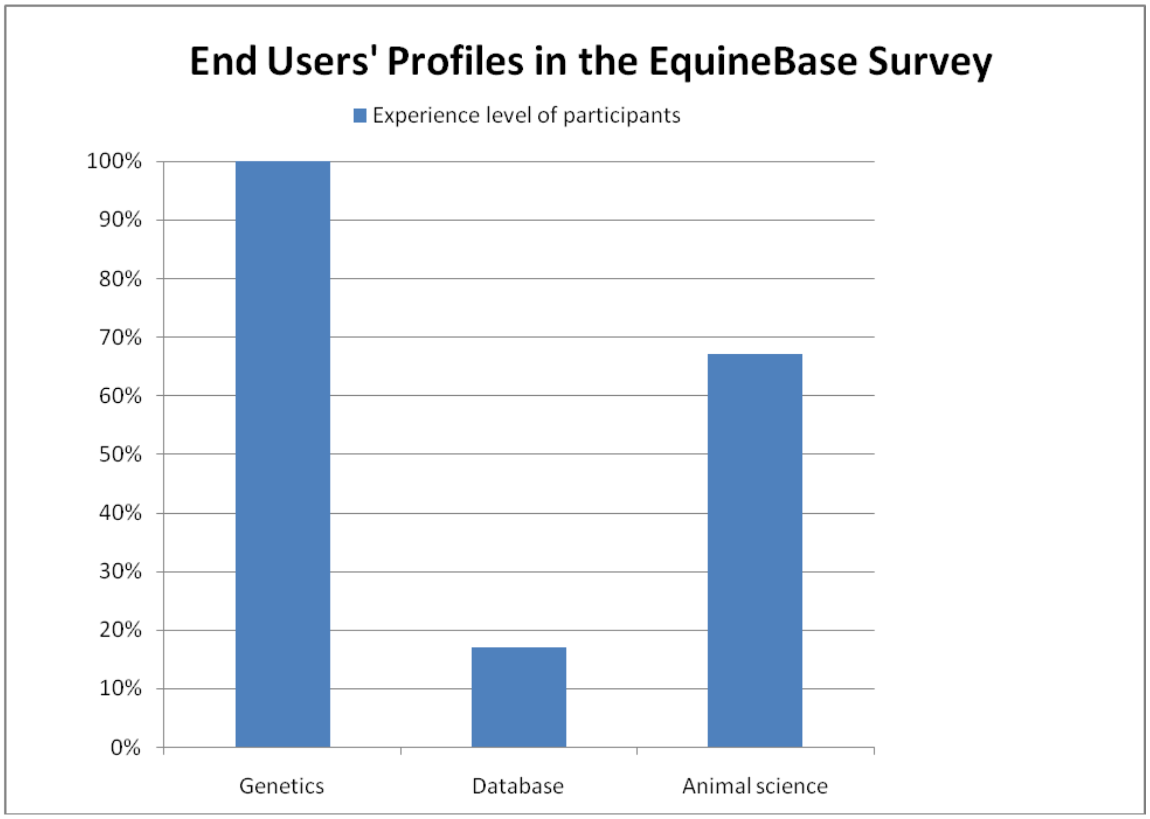


Figure 43 Users' profiles in EquineBase survey

- Does the search interface in EquineBase help you find the target data fast and easily?
- Is easy it to navigate the interface and find the function that you need? Do you like the layout of menus?

- What are your favorite functions of EquineBase? Do you have any suggestion on improving the interface?

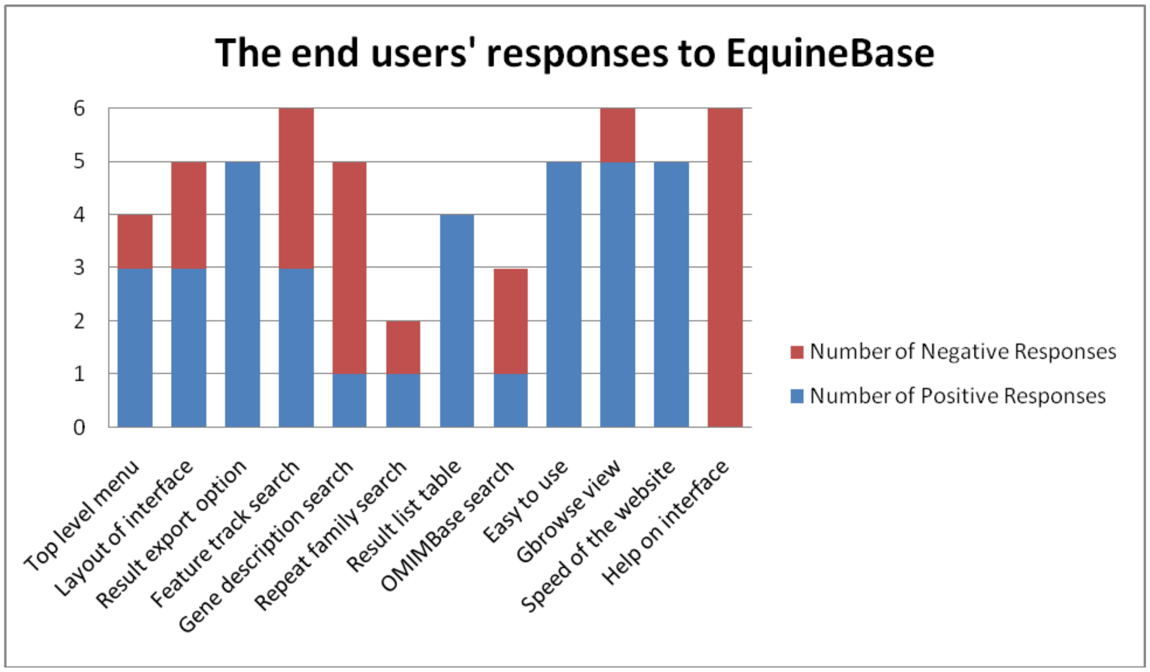


Figure 44 The summary of the user survey

Figure 44 shows the summary of end users' responses to the features of EquineBase in the survey. We summarized several aspects of the usability of EquineBase and our improvements.

- Equinebase has the search functions which can go deep into the data, but are not provided by other bioinformatics tools and equine database. That means we achieved the goals of EquineBase.

- The drop-down option list keeps several search functions in one interface. This simplifies the user interface and reduces the time for user to explore different functions on EquineBase.
- The sortable search result list and export options of the data are very useful. It helps users identify the target data and download the result data. However, users require having more options of sorting the result data, such as the positions and the annotations of data. After the survey, we updated the search result list template and added extra fields in the result list.
- The search functions lack the flexibility of setting the range for searching a feature. We updated feature the search function template to allow setting the range for searching a feature. However, we still provide only several range parameters for searching. One concern is that we want to reduce the search time. Another is that we need to balance the functions and the performance of hardware.
- Users would like to have some hints and a user guide on the interface. We added some examples on search interfaces and will provide a comprehensive user guide.

In summary, the user surveys are critical for this study because they provide us with inside views and different prospects of GOOF and EquineBase. As a proposed bioinformatics framework targets both of developers and end users, the feedback could improve and direct the future research work. EquineBase was developed to provide a user friendly web resource for access equine genome data. The end users help EquineBase improving the usability.

# 7 CONCLUSIONS AND FUTURE WORK

## 7.1 Summary and Conclusions

While there are a number of genome informatics systems available, we have concentrated on the whole system, from the back end to the front end genomic data management solution. GOOF is novel because it is the first framework that integrates a Model-View-Controller object-oriented genome informatics framework with many pre-built utilities to help genomics and biomedical researchers build their own bioinformatics infrastructure. In this research study, we successfully built EquineBase and OMIMBase using GOOF. There is no reason why other organism specific bioinformatics systems cannot develop their own bioinformatics infrastructure rapidly by taking advantages of GOOF. The Bovine research community has plans to build the next generation of QTL viewer system based on GOOF. The built-in connection on the data object layer with Chado could provide a rapid procedure to develop any organism specific genomic bioinformatics infrastructure. Good examples of this are EquineBase and OMIMBase. GOOF is a new contribution to the informatics component of this research and EquineBase and OMIMBase are significant outputs from the biological component of this work.

Advantages and flexibility of GOOF:

- GOOF is a simple, object-oriented structure, which allows users to develop their own complex data models.
- GOOF is developed using standard and open source technology, which is reliable and easily updated.

- GOOF is a Java framework, which is easy to update and support simple migration to different platforms.
- GOOF is a comprehensive solution for the back-end, middleware and front-end framework, and is flexible enough to adapt to any existing genome database system, since it is independent of the back-end database and uses the object/relation mapping tool to connect with any major database.
- Any bioinformatics application developed with GOOF can be either simple or complex according to requirements. There are several data object templates available for users to construct their data schema. Or the pre-built connection with Chado as the back end database schema allows users to make use of the GMOD toolkit.
- The object-oriented data modeling structure in GOOF can simplify the query interface and improve users' data exploration.
- The modular design of GOOF's web interface makes it a simple task to tailor interfaces according to the needs of certain user groups.

Disadvantages of GOOF:

- GOOF is built on pure Java. GOOF cannot take advantage of BioPerl [62] to process raw biological data.
- GOOF follows strict object-oriented rules. Any GOOF–based application has to conform to the object-oriented pattern to develop bioinformatics infrastructures.

  Advantages of EquineBase and OMIMBase:

- EquineBase is the first bioinformatics resource for Equine genome data.

- OMIMBase is the first combined OMIM and Equine genome data resource for animal and biomedical researchers who are interested in studying genetic diseases of the horse.
- EquineBase and OMIMBase provide a flexible and straightforward data mining interface. Users do not have to construct queries themselves.
- EquineBase and OMIMBase integrate data mining tools (the web presentation of GOOF) and a data visualization utility (Gbrowse).
- The back bone structure of EquineBase and OMIMBase make it easy to update the bioinformatics infrastructure and process new types of data.
- The fact that the user schema is separated from the genomic data schema allows easy data access management in EquineBase and OMIMBase.

## 7.2 Directions for Future Work

In this research study, we already showed the advantages of a genome object-oriented informatics framework over other genome informatics framework. Through EquineBase and OMIMBase, we demonstrated GOOF's applicability and flexibility. However, from an end user point of view, there are several features that need to be developed in the future in order to make GOOF a more comprehensive data management and data mining toolkit.

Even though GOOF was originally developed as a web accessible genomic data management framework, some end users would like to generate a data mining report consisting of large numbers of data entries. This requirement suggests that the web access interface would not be a good solution for these types of output. Therefore, a high performance data mining interface is required in GOOF. This would not be difficult to implement within the currently data transaction layer. In the web presentation layer, the current available template could build an interface for complex data mining criteria. This, however, creates a new problem for the system as a whole. One example of such a scenario has users constructing a complex data mining process to query across all the genomes in the database. This task could use significant system resources and take several hours or even days to generate a data report. GOOF could then provide the resulting report as a link to users and notify them automatically by email. Usually, high throughput data mining such as this could require a significant hardware resource. This in turn requires a job queuing system, a back end computing cluster and a database cluster. The complexity of this kind of system is far beyond the scope of this research study. The next research task for GOOF development should include a data mining interface to generating high throughput reports in the background.

Should users develop their own data management schema from a scratch by using the data object template of GOOF, a utility to transfer raw data files into XML data entry format would be a useful future. Currently, the central development management utility in GOOF can build a schema and load data into the target database without directly operating on the database system. However, users would still need to develop their own parsers to convert the raw data into XML format.

The meta-programming utility in GOOF currently only generates standard query data transaction codes using the single data object definition. The tag for the relationships of multiple data objects should be developed to allow construction of the data transaction codes from the meta-programming template.

EquineBase and OMIMBase will continue to be developed according the needs of the research community and in response to the generation of new data. The new data will in turn stimulate the development of more data mining procedures. As more users come online and submit queries a front end proxy cache system might be needed to improve the performance of EquineBase and OMIMBase. In addition, links to other data source systems should be developed in the future. In other words, EquineBase and OMIMBase should allow remote data query from other bioinformatics system across the network. As more data mining utilities and more annotated data become available, EquineBase and OMIMBase should significantly speed up the equine and biomedical research in the future.

## REFERENCES

1. Mungall CJ, Emmert DB: **A Chado case study: an ontology-based modular schema for representing genome-associated biological information**. *Bioinformatics* 2007, **23**(13):i337-346.
2. Stein LD, Mungall C, Shu S, Caudy M, Mangone M, Day A, Nickerson E, Stajich JE, Harris TW, Arva A *et al*: **The generic genome browser: a building block for a model organism system database**. *Genome Res* 2002, **12**(10):1599-1610.
3. Killion PJ, Sherlock G, Iyer VR: **The Longhorn Array Database (LAD): an open-source, MIAME compliant implementation of the Stanford Microarray Database (SMD)**. *BMC Bioinformatics* 2003, **4**:32-37.
4. Demeter J, Beauheim C, Gollub J, Hernandez-Boussard T, Jin H, Maier D, Matese JC, Nitzberg M, Wymore F, Zachariah ZK *et al*: **The Stanford Microarray Database: implementation of new analysis tools and open source release of software**. *Nucleic Acids Res* 2007, **35**(Database issue):D766-770.
5. Erich G, Richard H, Ralph J, John V: **Design patterns: elements of reusable object oriented software**. *European Journal of Control* 2007, **13**(5).
6. Gellersen HW, Gaedke M: **Object-oriented web application development**. *Internet Computing, IEEE*, Washington, D.C., 1999, **3**(1):60-68.

7. Fayad ME, Schmidt DC, Johnson RE: **Building application frameworks: object-oriented foundations of framework design**: John Wiley & Sons Inc., New York, NY 1999.

8. McKusick VA: **Mendelian Inheritance in Man and its online version, OMIM**. *Am J Hum Genet* 2007, **80**(4):588-604.

9. Hamosh A, Scott AF, Amberger JS, Bocchini CA, McKusick VA: **Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders**. *Nucleic Acids Res* 2005, **33**(Database issue):D514-517.

10. Amladi S: **Online Mendelian Inheritance in Man 'OMIM'**. *Indian J Dermatol Venereol Leprol* 2003, **69**(6):423-424.

11. Larmann C: **Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process**: Prentice Hall PTR, Indianapolis, IN 2002.

12. Johnson R: **J2EE development frameworks**. *Computer* 2005, **38**(1):107-110.

13. Arthur J, Azadegan S: **Spring framework for rapid open source J2EE Web application development: a case study**. *Proceedings of the 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* and *1st ACIS International Workshop on Self-Assembling Wireless Networks*, Towson, MD 2005:90-95.

14. Wilson RJ, Goodman JL, Strelets VB: **FlyBase: integration and improvements to query tools**. *Nucleic Acids Res* 2008, **36**(Database issue):D588-593.

15. Drysdale RA, Crosby MA: **FlyBase: genes and gene models**. *Nucleic Acids Res* 2005, **33**(Database issue):D390-395.

16. Crosby MA, Goodman JL, Strelets VB, Zhang P, Gelbart WM: **FlyBase: genomes by the dozen**. *Nucleic Acids Res* 2007, **35**(Database issue):D486-491.

17. Rogers A, Antoshechkin I, Bieri T, Blasiar D, Bastiani C, Canaran P, Chan J, Chen WJ, Davis P, Fernandes J *et al*: **WormBase 2007**. *Nucleic Acids Res* 2008, **36**(Database issue):D612-617.

18. Bieri T, Blasiar D, Ozersky P, Antoshechkin I, Bastiani C, Canaran P, Chan J, Chen N, Chen WJ, Davis P *et al*: **WormBase: new content and better access**. *Nucleic Acids Res* 2007, **35**(Database issue):D506-510.

19. Schwarz EM, Antoshechkin I, Bastiani C, Bieri T, Blasiar D, Canaran P, Chan J, Chen N, Chen WJ, Davis P *et al*: **WormBase: better software, richer content**. *Nucleic Acids Res* 2006, **34**(Database issue):D475-478.

20. Harris TW, Stein LD: **WormBase: methods for data mining and comparative genomics**. *Methods Mol Biol* 2006, **351**:31-50.

21. Chisholm RL, Gaudet P, Just EM, Pilcher KE, Fey P, Merchant SN, Kibbe WA: **dictyBase, the model organism database for Dictyostelium discoideum**. *Nucleic Acids Res* 2006, **34**(Database issue):D423-427.

22. Fey P, Gaudet P, Pilcher KE, Franke J, Chisholm RL: **dictyBase and the Dicty Stock Center**. *Methods Mol Biol* 2006, **346**:51-74.

23. Marinelli RJ, Montgomery K, Liu CL, Shah NH, Prapong W, Nitzberg M, Zachariah ZK, Sherlock GJ, Natkunam Y, West RB *et al*: **The Stanford tissue microarray database**. *Nucleic Acids Res* 2008, **36**(Database issue):D871-877.

24. Pan X, Stein L, Brendel V: **SynBrowse: a synteny browser for comparative sequence analysis**. *Bioinformatics* 2005, **21**(17):3461-3468.

25. Garzas J, Piattini M: **An ontology for understanding and applying object-oriented design knowledge**. *International Journal of Software Engineering and Knowledge Engineering* 2007, **17**(3):407-421.

26. Sauter P, Vogler G, Specht G, Flor T: **Extending the MVC design pattern towards a task-oriented development approach for pervasive computing applications**. *Organic and Pervasive Computing Arcs* 2004, **2981**:309-321.

27. Adamko A: **UML-based modeling of data-oriented WEB applications**. *Journal of Universal Computer Science* 2006, **12**(9):1104-1117.

28. Zarras A: **A comparison framework for middleware infrastructures**. *Journal of Object Technology* 2004, **3**(5):103-123.

29. Gomez-Perez A, Gonzalez-Cabero R, Lama M: **A framework for design and composition of semantic web services**. *AAAI Spring Symposium Series*, Stanford University, Palo Alto, CA 2004.

30. Li L, Liu HW: **The least configuration files implement MVC model of customized query**. *Dynamics of Continuous Discrete and Impulsive Systems-Series a-Mathematical Analysis* 2006, **13**:1914-1921.

31. **AppFuse**. http://www.appfuse.org/ Accessed September 2007.

32. Bauer C, King G: **Hibernate in action**. Manning Co., Greenwich, CT 2005.

33. Lightbody P, Carreira J: **WebWork in action**. Manning Co., Greenwich, CT 2006.

34. Walls C, Breidenbach R: **Spring in action**. Manning Co., Greenwich, CT 2005.

35. Walnes J, ebrary Inc.: **Java open source programming with XDoclet, JUnit, WebWork, Hibernate**. *Java Open Source library*, Wiley, Indianapolis, IN 2004.

36. Ford N: **Art of Java web development: Struts, Tapestry, Commons, Velocity, JUnit, Axis, Cocoon, InternetBeans, WebWork**. Manning Co., Greenwich, CT 2003.

37. Simonyi C: **Meta-programming: a software production method**. Dissertation (Ph D), Dept of Computer Science, Stanford University 1977.

38. Batory D: **Multi-level models in model driven development, product-lines, and metaprogramming**. *IBM Systems Journal* 2006, **45**(3):527-540.

39. Abrahams D, Gurtovoy A: **C++ template metaprogramming: concepts, tools, and techniques from boost and beyond**. Addison-Wesley Co., Boston, MA 2005.

40. Jarzabek S, Shubiao L: **Eliminating redundancies with a" composition with adaptation" meta-programming technique**. *Proceedings of the 9th European Software Engineering Conference* held jointly with *11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Helsinki Finland 2003:237-246.

41. Hightower R, Lesiecki N: **Java tools for extreme programming**. *Journal of Computing and Information Technology* 2003, **11**(2):01-02.

42. Löwe W, Noga ML: **Metaprogramming applied to web component deployment**. *Electronic Notes in Theoretical Computer Science* 2002, **65**(4):106-116.

43. Atkinson C, Kuhne T: **Model-driven development: a metamodeling foundation**. *Software, IEEE,* Washington, D.C., 2003, **20**(5):36-41.

44. Pettersson M: **DML: a meta-language for compiler generation from denotational specifications**. Dept. of Computer and Information Science, Linkoeping University, 1990.

45. Garcia R, Jarvi J, Lumsdaine A, Siek J, Willcock J: **An extended comparative study of language support for generic programming**. *Journal of Functional Programming* 2007, **17**:145-205.

46. Basit HA, Rajapakse DC, Jarzabek S: **Beyond templates: a study of clones in the STL and some general implications**. *Proceedings of the 27th International Conference on Software Engineering* 2005:451-459.

47. Naftalin M, Wadler P: **Java generics and collections**. O'Reilly, Sebastopol, CA 2007.

48. Von Dincklage D, Diwan A: **Converting Java classes to use generics**. *ACM SIGPLAN Notices* 2004, **39**(10):1-14.

49. Ghosh D: **Generics in Java and C++ - a comparative model**. *ACM SIGPLAN Notices* 2004, **39**(5):40-47.

50. Lopez-Herrejon R, Batory D, Lengauer C: **A disciplined approach to aspect composition**. *Proceedings of the 2006 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation* 2006:68-77.

51. de Oliveira AA, Braga TH, de Almeida Maia M, da Silva Bigonha R: **MetaJ: an extensible environment for metaprogramming in Java**. *J Universal Computer Science* 2004, **10**(7):872–891.

52. Walls C, Richards N: **XDoclet in action**. Manning Co., Greenwich, CT 2004.

53. Tratt L: **Compile-time meta-programming in a dynamically typed OO language**. *Proceedings of the 2005 Conference on Dynamic Languages Symposium*, St. Louis, MO 2005:49-63.

54. Shneiderman B: **Designing the user interface: strategies for effective human-computer interaction**. Addison-Wesley Co., Boston, MA 1992.

55. Fields DK, Kolb MA, Bayern S: **Web development with Java server pages**. Manning Co., Greenwich, CT 2001.

56. Hatcher E, Loughran S: **JAVA development with Ant**. Manning Co., Greenwich, CT 2003.

57. Jarzabek S, Pettersson U: **Cost-effective engineering of web applications pragmatic reuse: building web application product lines**. *International Conference on Software Engineering*, Shanghai, China 2006:1053-1054.

58. Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Wheeler DL: **GenBank**. *Nucleic Acids Res* 2008, **36**(Database issue):D25-30.

59. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: **Basic local alignment search tool**. *Journal of Molecular Biology* 1990, **215**(3):403-410.

60. Schneidewind NF: **Experience report on using object-oriented design for software maintenance**. *Journal of Software Maintenance and Evolution-Research and Practice* 2007, **19**(3):183-201.

61. Papastavrou S, Chrysanthis PK, Samaras G, Pitoura E: **an evaluation of the Java-based approaches to web database access**. *International Journal of Cooperative Information Systems* 2001, **10**(4):401-422.

62. Stajich JE, Block D, Boulez K, Brenner SE, Chervitz SA, Dagdigian C, Fuellen G, Gilbert JG, Korf I, Lapp H *et al*: **The Bioperl toolkit: Perl modules for the life sciences**. *Genome Res* 2002, **12**(10):1611-1618.

# APPENDIX A: INSTALLATION OF GOOF

GOOF is released as a zip or tar.gz package. Once it is unzipped, there is a "goof" folder, which contains all of Java libraries, source codes and configuration files.

Before developing applications based on GOOF, install JDK (v5 or higher, which has been tested), ant (v1.6.5 or higher), MySql or PostgreSQL, and Tomcat (v 5.5, which has been tested). Set up the environment variables: JAVA_HOME, ANT_HOME, CATALINA_HOME for UNIX or Windows platform.

1. Build a new applications. Run "ant new", which basically renames and replace the application variables and packages. It generates a new folder with the new application name. This is the foundation of a GOOF application.

   ```
   ========================
   ant new
   ========================
   ```

```
========================
Buildfile: build.xml
Trying to override old definition of datatype resources
clean:
    [echo] Cleaning build and distribution directories
init:
new:
    [echo]
    [echo] +------------------------------------------------------------+
```

```
    [echo] |    -- Welcome to the Application Wizard! --               |
    [echo] +-----------------------------------------------------------+
    [echo]
   [input] What would you like to name your application [myapp]? [myapp]
BovineQTL
   [input] What would you like to name your database [mydb]? [mydb]
BovineQTL
   [input] What package name would you like to use [edu.tamu.goof]? [edu.tamu.goof]
edu.tamu.bovineqtl
    [echo] Creating new application named 'BovineQTL'...
    [echo] Repackaging info written to rename.log
    [echo] +-----------------------------------------------------------+
    [echo] |          -- Application created successfully! --          |
    [echo] | Now you should be able to cd to your application and run:  |
    [echo] | > ant setup war                                            |
    [echo] +-----------------------------------------------------------+
BUILD SUCCESSFUL
Total time: 22 seconds
========================
```

2. Setup the connection with the database. If you decide to use PostgreSQL or MySQL, please uncomment the corresponding sections in build.properties, which overrides the default setting in properties.xml. The default setting for database

connection is PostgreSQL. Please modify the database property (database address, user name, password and the name of database) in propoerties.xml.

```
=================
ant setup-db
=================
```

Run "ant setup-db". This creates a database according to the schema of data models (src/dao/edu/tamu/goof/dao/model) and load data into the database from /metadata/sql/sample-data.xml.

3. Setup SMTP mail server in web/WEB-INF/classes/mail.properties.
4. Setup Tomcat server. Modify the properties of tomcat in properties.xml.
5. Build application.

```
===================
ant clean
===================
```

This cleans the old codes generated by the previous building.

```
===================
ant war
===================
```

This builds the war file for the whole application, if you don't want to deploy the application directly to the Tomcat server. It creates goof.war in /dist/webapps/.

```
===================
ant deploy
```

===================

This command builds the war file and automatically deploy the application to the web application package of the Tomcat server.

===================

ant undeploy

===================

This removes applications from the Tomcat server.

==========================

ant undelpoy clean deploy

==========================

This is what I usually run if there is no change in the data model

6. Build a new data driven application. Then please refer to the GOOF section of this study for details of how to build the data model, data transaction and web actions.
7. Customize interface.

WEB-INF/classes/ApplicationResources.properties defines the content of the interface. In web/WEB-INF/styles/, there are some CSS files from the internet for default applications. In web/WEB-INF/common/, modify footer.jsp and header.jsp.

## APPENDIX B: DATA MODEL LAYER - Contig.java

```
package edu.tamu.goof.model;

import java.util.Date;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.Iterator;

import java.util.List;

import java.util.Set;


import org.apache.commons.lang.builder.ToStringBuilder;

import org.apache.commons.lang.builder.ToStringStyle;

import edu.tamu.goof.model.Clone;

/**

 * @hibernate.class table="contig"

  * @struts.form include-all="true" extends="BaseForm"

 */

public class Contig extends BaseObject {

	private String c_name;

	private String c_uorc;

	private Date c_date;

	private String c_seq;

	private String c_anno_bov;
```

```
        private String c_anno_hum;

        private Set clones = new HashSet();

        public Contig() {}

        public Contig(String c_name) {

        this.c_name = c_name;

        }

/**

 * @return Returns the id.

 * @hibernate.id column="c_name" tag-class="identity"

 * type="string" not-null="true" length="254"

 */

        public String getC_name() {

                return this.c_name;

        }

        public void setC_name(String c_name) {

                this.c_name = c_name;

        }

/**

 * @hibernate.property column="c_uorc" length="1" type="string" not-null="true"

 */

        public String getC_uorc() {

                return this.c_uorc;
        }
        public void setC_uorc(String c_uorc) {
                this.c_uorc = c_uorc;
        }
/**
 * @hibernate.property column="c_date" type="timestamp"
 */
        public Date getC_date() {
                return this.c_date;
        }
        public void setC_date(Date c_date) {
                this.c_date = c_date;
        }
/**
 * @hibernate.property column="c_seq" type="text" not-null="true"
 */
        public String getC_seq() {
                return this.c_seq;
        }

        public void setC_seq(String c_seq) {
                this.c_seq = c_seq;
	}
/**
 * @hibernate.property column="c_anno_bov" type="text" not-null="true"
 */
	public String getC_anno_bov() {
		return this.c_anno_bov;
	}

	public void setC_anno_bov(String c_anno_bov) {
		this.c_anno_bov = c_anno_bov;
	}

/**
 * @hibernate.property column="c_anno_hum" type="text" not-null="true"
 */
	public String getC_anno_hum() {
		return this.c_anno_hum;
	}

	public void setC_anno_hum(String c_anno_hum) {
		this.c_anno_hum = c_anno_hum;
	}
```

```
    /**
     * @hibernate.set table="contig_clone" cascade="save-update" lazy="false"
     * @hibernate.collection-key column="c_name"
     * @hibernate.collection-many-to-many class="edu.tamu.goof.model.Clone"
column="r_name"
     */
    public Set getClones() {
        return clones;
    }
    /**
     * Adds a clone for the contig
     * @param clone
     */
    public void addClone(Clone clone) {
        getClones().add(clone);
    }
    public void setClones(Set clones) {
        this.clones = clones;
    }

    /**
     * Convert contig clones to LabelValue objects for convenience.
     */
    public List getCloneList() {
        List contigClones = new ArrayList();
        if (this.clones != null) {
            for (Iterator it = clones.iterator(); it.hasNext();) {
                Clone clone = (Clone) it.next();
                contigClones.add(new LabelValue(clone.getR_name(),
                                              clone.getR_name()));
            }
        }
        return contigClones;
    }
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Contig)) return false;
        final Contig contig = (Contig) o;
        if (c_name != null ? !c_name.equals(contig.getC_name()) : contig.getC_name() !=
null) return false;
        return true;
    }
    public int hashCode() {
        return (c_name != null ? c_name.hashCode() : 0);
    }

    public String toString() {

        return new ToStringBuilder(this, ToStringStyle.DEFAULT_STYLE)

                .append("c_name", this.c_name).toString();

    }

}
```

## APPENDIX C: DATA MODEL LAYER - ContigDao.java

```
package edu.tamu.goof.dao;

import java.util.List;

import edu.tamu.goof.model.Contig;

import org.springframework.dao.DataAccessException;

public interface ContigDao extends Dao {

	public List getContigs(Contig contig) throws DataAccessException;

	public List findContigs(String c_name) throws DataAccessException;

	public List searchContigAnno(String c_anno) throws DataAccessException;

	public Contig getContig(final String c_name);

	public void saveContig(Contig contig);

	public void removeContig(final String c_name);


}
```

# APPENDIX D: DATA MODEL LAYER - ContigDaoHibernate.java

```
package edu.tamu.goof.dao.hibernate;
import java.util.List;
import org.springframework.dao.DataAccessException;
import org.springframework.orm.ObjectRetrievalFailureException;
import edu.tamu.goof.model.Contig;
import edu.tamu.goof.dao.ContigDao;
public class ContigDaoHibernate extends BaseDaoHibernate implements ContigDao {
        public List getContigs(Contig contig) throws DataAccessException {
                return getHibernateTemplate().find("from Contig contig order by
contig.c_name");
        }
        public List findContigs(String c_name) throws DataAccessException {
                return getHibernateTemplate().find("from Contig contig where
contig.c_name like ? order by contig.c_name", "%" + c_name + "%");
        }
        public List searchContigAnno(String c_anno) throws DataAccessException {
                return getHibernateTemplate().find("from Contig contig where
contig.c_anno_bov like ? or contig.c_anno_hum  like ? order by contig.c_name", new
Object[] {"%" + c_anno + "%", "%"+c_anno+"%"});
        }
        public Contig getContig(final String c_name) {
                Contig contig = (Contig) getHibernateTemplate().get(Contig.class,
c_name);
                if (contig == null) {
                    log.warn("uh oh, contig '" + c_name + "' not found...");
                    throw new ObjectRetrievalFailureException(Contig.class, c_name);
                }
               return contig;
        }
        public void saveContig(final Contig contig) {
                if (log.isDebugEnabled()) {
                    log.debug("contig's id: " + contig.getC_name());
                }
                getHibernateTemplate().saveOrUpdate(contig);
                getHibernateTemplate().flush();
        }
        public void removeContig(final String c_name) {
                    getHibernateTemplate().delete(getContig(c_name));
        }
}
```

## APPENDIX E: DATA MODEL LAYER - FeatureDaoHibernate.java

```
package edu.tamu.goof.dao.hibernate;

import java.util.List;

import org.springframework.dao.DataAccessException;

import org.springframework.orm.ObjectRetrievalFailureException;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import org.springframework.orm.hibernate3.HibernateTemplate;

import org.springframework.orm.hibernate3.HibernateCallback;

import org.hibernate.criterion.*;

import org.hibernate.*;

import edu.tamu.goof.dao.FeatureDao;

import fr.inra.gmod.chado.Feature;

import fr.inra.gmod.chado.FeatureImpl;

public class FeatureDaoHibernate extends BaseDaoHibernate implements FeatureDao {

	public List searchFeatures(final String f_name, final String f_type,

			final String chr, final Integer fmin, final Integer fmax)

			throws DataAccessException {

		HibernateCallback callback = new HibernateCallback() {

			public Object doInHibernate(Session session)

					throws HibernateException {

				Criteria crit = session.createCriteria(Feature.class, "feat");
```

```
                                crit.add(Restrictions.like("feat.name", "%" + f_name +
"%"));
                                crit.createAlias("feat.featurelocFeature", "featLoc");
                                crit.createAlias("featLoc.srcfeature", "srcfeat");
        crit.add(Restrictions.like("srcfeat.uniquename","%"+chr+"%"));
                                crit.add(Restrictions.gt("featLoc.fmin", fmin));
                                crit.add(Restrictions.lt("featLoc.fmax", fmax));
                                crit.createAlias("feat.cvterm", "type");
                                crit.add(Restrictions.eq("type.name", f_type));
                                return crit.list();
                        }
                };
                return (List) getHibernateTemplate().execute(callback);
        }
        public FeatureImpl getFeature(Integer featureId) throws DataAccessException {
                return (FeatureImpl) getHibernateTemplate().get(FeatureImpl.class,
                                featureId);
        }
        public List findFeatures(String f_name) throws DataAccessException {
                return getHibernateTemplate().find(
                                "from FeatureImpl f where f.name like ?", "%" + f_name +
"%");
```

```
          }

      }
```

## APPENDIX F: DATA MODEL LAYER - applicationContext-hibernate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

        <!-- Hibernate SessionFactory -->
        <bean id="sessionFactory"
                class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
                <property name="dataSource" ref="dataSource" />
                <property name="mappingResources">
                        <list>
                                <value>edu/tamu/goof/model/Role.hbm.xml</value>
                                <value>edu/tamu/goof/model/User.hbm.xml</value>
                                <value>edu/tamu/goof/model/Contig.hbm.xml</value>
                                <value>edu/tamu/goof/model/Clone.hbm.xml</value>
                                <value>edu/tamu/goof/model/Omim.hbm.xml</value>
                                <value>edu/tamu/goof/model/Omimcs.hbm.xml</value>
                                <value>edu/tamu/goof/model/Hg.hbm.xml</value>
                                <value>edu/tamu/goof/model/Mice.hbm.xml</value>
                                <value>edu/tamu/goof/model/Equine.hbm.xml</value>
```

```
                    </list>
                </property>
                <property name="hibernateProperties">
                    <props>
                        <prop key="hibernate.dialect">@HIBERNATE-
DIALECT@</prop>
                        <prop key="hibernate.query.substitutions">
                            true 'Y', false 'N'
                        </prop>
                      <prop key="hibernate.show_sql">
                      false
                      </prop>
                    </props>
                </property>
        </bean>
        <!-- ContigDda: Hibernate implementation -->
        <bean id="contigDao"
                class="edu.tamu.goof.dao.hibernate.ContigDaoHibernate">
                <property name="sessionFactory" ref="sessionFactory" />
        </bean>
</beans>
```

## APPENDIX G: DATA TRANSACTION LAYER - ContigManager.java

```
package edu.tamu.goof.service;

import java.util.List;

import edu.tamu.goof.dao.ContigDao;

import edu.tamu.goof.model.Contig;

public interface ContigManager {

    public void setContigDao(ContigDao dao);

    public Contig getContig(final String c_name);

    public List findContigs(String c_name);

    public List searchContigAnno(String c_anno);

    public List getContigs(Contig contig);

    public void saveContig(Contig contig);

    public void removeContig(final String c_name);

}
```

## APPENDIX H: DATA TRANSACTION LAYER - ContigMangerImpl.java

```
package edu.tamu.goof.service.impl;
import java.util.List;
import edu.tamu.goof.dao.ContigDao;
import edu.tamu.goof.model.Contig;
import edu.tamu.goof.service.ContigManager;
import org.springframework.dao.DataIntegrityViolationException;
public class ContigManagerImpl extends BaseManager implements ContigManager {
        private ContigDao dao;
        public void setContigDao(ContigDao dao) {
                this.dao = dao;
        }
        public Contig getContig(final String c_name) {
                return dao.getContig(new String(c_name));
        }
        public List findContigs(String c_name) {
                return dao.findContigs(c_name);
        }
        public List searchContigAnno(String c_anno) {
                return dao.searchContigAnno(c_anno);
        }
        public List getContigs(final Contig contig) {
```

```
		return dao.getContigs(contig);
	}
}
```

## APPENDIX I: DATA TRANSACTION LAYER - applicationContext-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
           http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
           http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">

    <aop:config>
        <aop:advisor id="userManagerTx" advice-ref="userManagerTxAdvice"
pointcut="execution(* *..service.UserManager.*(..))" order="0"/>
        <aop:advisor id="userManagerSecurity" advice-ref="userSecurityAdvice"
pointcut="execution(* *..service.UserManager.saveUser(..))" order="1"/>
        <aop:advisor id="managerTx" advice-ref="txAdvice" pointcut="execution(*
*..service.*Manager.*(..))" order="2"/>
    </aop:config>
    <tx:advice id="txAdvice">
```

```
        <tx:attributes>
            <tx:method name="get*" read-only="true"/>
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>
    <tx:advice id="userManagerTxAdvice">
        <tx:attributes>
            <tx:method name="save*" rollback-for="UserExistsException"/>
        </tx:attributes>
    </tx:advice>
    <bean id="lookupManager"
class="edu.tamu.goof.service.impl.LookupManagerImpl">
        <property name="lookupDao" ref="lookupDao"/>
    </bean>
    <bean id="manager" class="edu.tamu.goof.service.impl.BaseManager">
        <property name="dao" ref="dao"/>
    </bean>
    <bean id="userManager" class="edu.tamu.goof.service.impl.UserManagerImpl">
        <property name="userDao" ref="userDao"/>
    </bean>
    <bean id="userSecurityAdvice" class="edu.tamu.goof.service.UserSecurityAdvice">
        <property name="userCache" ref="userCache"/>
    </bean>


    <bean id="userCache"
class="org.acegisecurity.providers.dao.cache.EhCacheBasedUserCache">
        <property name="cache">
            <bean class="org.springframework.cache.ehcache.EhCacheFactoryBean">
                <property name="cacheManager">
                    <bean
class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
                </property>
                <property name="cacheName" value="userCache"/>
            </bean>
        </property>
    </bean>
    <bean id="roleManager" class="edu.tamu.goof.service.impl.RoleManagerImpl">
        <property name="roleDao" ref="roleDao"/>
    </bean>
    <bean id="contigManager" class="edu.tamu.goof.service.impl.ContigManagerImpl">
            <property name="contigDao" ref="contigDao"/>
    </bean>
</beans>
```

## APPENDIX J: WEB PRESENTATION LAYER - ContigAction.java

```
package edu.tamu.goof.webapp.action;

import java.io.IOException;

import java.util.ArrayList;

import java.util.LinkedHashMap;

import java.util.List;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.commons.lang.StringUtils;

import edu.tamu.goof.Constants;

import edu.tamu.goof.model.Contig;

import edu.tamu.goof.service.ContigManager;

import edu.tamu.goof.util.StringUtil;

import edu.tamu.goof.webapp.util.RequestUtil;

import com.opensymphony.webwork.ServletActionContext;

public class ContigAction extends BaseAction {

    private List contigs;

    private Contig contig;

    private String c_name;

    private String c_anno;

    private ContigManager contigManager;

    public List getContigs() {
```

```
    return contigs;
}
public void setC_name(String c_name) {
    this.c_name = c_name;
}
public void setC_anno(String c_anno) {
    this.c_anno = c_anno;
}
public void setContigManager(ContigManager contigManager) {
    this.contigManager = contigManager;
}
public Contig getContig() {
    return contig;
}
public void setContig(Contig contig) {
    this.contig = contig;
}
public String view() throws IOException {
    if (c_name != null){
            contig = contigManager.getContig(c_name);
    }
    return SUCCESS;
```

```
}
public String execute() {
   return SUCCESS;
}
public String doDefault() {
   return INPUT;
}
public String list() {
   contigs = contigManager.getContigs(new Contig());
   return SUCCESS;
}
public String search() {
   if (c_name != null){
      contigs = contigManager.findContigs(c_name);
   } else {
               return INPUT;
   }
      return SUCCESS;
}
public String searchAnno() {
   if (c_anno != null){
      contigs = contigManager.searchContigAnno(c_anno);
```

```
        } else {
                return INPUT;
        }
            return SUCCESS;
    }
}
```

## APPENDIX K: WEB PRESENTATION LAYER - ContigView.jsp

```
<%@ include file="/common/taglibs.jsp"%>


<title><fmt:message key="contigDetail.title"/></title>

<content tag="heading"><fmt:message key="contigDetail.heading"/></content>

<table class="detail" cellpadding="5">

   <tr>

         <th><ww:property value="%{getText('contig.c_name')}"/></th>

         <td><ww:property value="%{contig.c_name}"/></td>

   </tr>

   <tr>

         <th><ww:property value="%{getText('contig.c_uorc')}"/></th>

         <td><ww:property value="%{contig.c_uorc}"/></td>

   </tr>

    <tr>

      <th><ww:property value="%{getText('contig.c_date')}"/></th>

      <td><ww:property value="%{contig.c_date}"/></td>

   </tr>


   <tr>

         <th><ww:property value="%{getText('contig.c_seq')}"/></th>

      <td>
    <script type="text/javascript">

    var seq ="<ww:property value="%{contig.c_seq}"/>"

    var l_seq = seq.length

    var format_seq = ""

    for (var i = 0; i < l_seq; i++) {

            if (i+65 >l_seq) {

            format_seq = format_seq + seq.substring(i,l_seq-1)

            }

            else {

            format_seq = format_seq + seq.substring(i,i+65)+"\r\n"

            }

            i = i+65

    }

    document.write(format_seq.toUpperCase())

    </script>

    </td>

</tr>

<tr>

        <th><ww:property value="%{getText('contig.c_anno_bov')}"/></th>

        <td><ww:property value="%{contig.c_anno_bov}"/></td>

</tr>

<tr>
```

```
            <th><ww:property value="%{getText('contig.c_anno_hum')}"/></th>
            <td><ww:property value="%{contig.c_anno_hum}"/></td>
    </tr>
    <tr>
            <th><ww:property value="%{getText('contig.clones')}"/></th>
            <td>
            <ww:iterator id="clone" value="%{contig.cloneList}" status="status">
                        <a href="/goof/viewClone.html?r_name=<ww:property
value="value"/>">
                        <ww:property value="label"/>
                        <%--ww:property value="value"/--%>
                        </a>
                        <ww:if test="!#status.last">,</ww:if>
            </ww:iterator>
            </td>
    </tr>
</table>
```

## APPENDIX L: WEB PRESENTATION LAYER - ContigSearchAnno.jsp

```
<%@ include file="/common/taglibs.jsp"%>
<title><fmt:message key="contigList.title"/></title>
<content tag="heading"><fmt:message key="contigList.heading"/></content>


<ww:form name="searchContigAnno" action="searchContigAnno" method="post"
validate="true">
   <ww:textfield label="%{getText('contig.c_anno')}" name="c_anno" value="c_anno"
required="true"/>
   <tr>
      <td></td>
      <td class="buttonBar">
         <input type="submit" class="button" name="search"
            value="<fmt:message key="button.search"/>" />
         <input type="button" name="cancel" class="button"
onclick="location.href='mainMenu.html'"
            value="<fmt:message key="button.cancel"/>" />
      </td>
   </tr>
</ww:form>
```

## APPENDIX M: WEB PRESENTATION LAYER - ContigList.jsp

```
<%@ include file="/common/taglibs.jsp"%>


<title><fmt:message key="contigList.title"/></title>
<content tag="heading"><fmt:message key="contigList.heading"/></content>


<ww:set name="contigList" value="contigs" scope="request"/>


<display:table name="contigList" cellspacing="0" cellpadding="0" requestURI=""
    defaultsort="1" id="contigs" pagesize="25" class="table" export="true">
    <display:column property="c_name" escapeXml="true" sortable="true"
        url="/viewContig.html" paramId="c_name" paramProperty="c_name"
        titleKey="contig.c_name"/>
    <display:column property="c_uorc" escapeXml="true" sortable="true"
         titleKey="contig.c_uorc"/>
    <display:column property="c_anno_bov" escapeXml="true" sortable="true"
         titleKey="contig.c_anno_bov"/>
    <display:column property="c_anno_hum" escapeXml="true" sortable="true"
         titleKey="contig.c_anno_hum"/>
    <display:setProperty name="paging.banner.item_name" value="contig"/>
    <display:setProperty name="paging.banner.items_name" value="contigs"/>
</display:table>
```

## APPENDIX N: WEB PRESENTATION LAYER - xwork.xml

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork 1.1.1//EN"
   "http://www.opensymphony.com/xwork/xwork-1.1.1.dtd">
<xwork>
    <!-- Include webwork defaults -->
    <include file="webwork-default.xml"/>

    <!-- Configuration for the default package. -->
    <package name="default" extends="webwork-default">
      <interceptors>
        <!-- Interceptor to handle allowing only admins to certain actions -->
        <interceptor name="adminOnly" class="adminInterceptor"/>
        <!-- https://issues.apache.org/struts/browse/WW-1187 -->
        <interceptor-stack name="defaultStack">
          <interceptor-ref name="exception"/>
          <interceptor-ref name="alias"/>
          <interceptor-ref name="servlet-config"/>
          <interceptor-ref name="prepare"/>
          <interceptor-ref name="i18n"/>
          <interceptor-ref name="chain"/>
          <interceptor-ref name="model-driven"/>
          <interceptor-ref name="fileUpload"/>
```

```
            <interceptor-ref name="static-params"/>
            <interceptor-ref name="params"/>
            <interceptor-ref name="conversionError"/>
            <interceptor-ref name="validation">
                <param
name="excludeMethods">cancel,execute,delete,edit,list,default</param>
            </interceptor-ref>
            <interceptor-ref name="workflow">
                <param name="excludeMethods">input,back,cancel</param>
            </interceptor-ref>
        </interceptor-stack>
        <interceptor-stack name="fileUploadStack">
            <interceptor-ref name="fileUpload"/>
            <interceptor-ref name="defaultStack"/>
        </interceptor-stack>
        <interceptor-stack name="adminCheck">
            <interceptor-ref name="defaultStack"/>
            <interceptor-ref name="adminOnly"/>
        </interceptor-stack>
    </interceptors>
    <global-results>
        <result name="mainMenu" type="redirect">mainMenu.html</result>
```

```
            <result name="dataAccessFailure">/WEB-
INF/pages/dataAccessFailure.jsp</result>
        </global-results>
        <global-exception-mappings>
            <exception-mapping
exception="org.springframework.dao.DataAccessException"
                result="dataAccessFailure"/>
        </global-exception-mappings>
        <action name="activeUsers" class="com.opensymphony.xwork.ActionSupport">
            <result name="success">/WEB-INF/pages/activeUsers.jsp</result>
        </action>
        <action name="mainMenu" class="com.opensymphony.xwork.ActionSupport">
        <result name="success">/WEB-INF/pages/mainMenu.jsp</result>
        </action>
        <action name="users" class="userAction" method="list">
            <interceptor-ref name="adminCheck"/>
            <result name="success">/WEB-INF/pages/userList.jsp</result>
        </action>
        <action name="editUser" class="userAction" method="edit">
            <interceptor-ref name="adminCheck"/>
            <result name="success">/WEB-INF/pages/userForm.jsp</result>
            <result name="input">/WEB-INF/pages/userList.jsp</result>
        </action>
        <action name="editProfile" class="userAction" method="edit">
            <result name="success">/WEB-INF/pages/userForm.jsp</result>
            <result name="error">/WEB-INF/pages/mainMenu.jsp</result>
        </action>
        <action name="saveUser" class="userAction" method="save">
            <result name="cancel" type="redirect">users.html</result>
            <result name="input">/WEB-INF/pages/userForm.jsp</result>
            <result name="success" type="redirect">users.html</result>
            <result name="addAnother"
type="redirect">editUser.html?method=Add&amp;from=list</result>
        </action>
        <action name="passwordHint" class="passwordHintAction">
            <result name="success">/</result>
        </action>
        <action name="reload" class="edu.tamu.goof.webapp.action.ReloadAction">
            <interceptor-ref name="adminCheck"/>
            <result name="success">/WEB-INF/pages/mainMenu.jsp</result>
        </action>
            <!--Contig-START-->
            <action name="contigs" class="contigAction" method="list">
            <result name="success">/WEB-INF/pages/contigList.jsp</result>
            </action>
            <action name="searchContigs" class="contigAction" method="search">
            <result name="input">/WEB-INF/pages/contigSearch.jsp</result>
            <result name="success">/WEB-INF/pages/contigList.jsp</result>
            <result name="cancel" type="redirect">mainMenu.html</result>
            <result name="null" type="redirect">mainMenu.html</result>
            </action>
            <action name="searchContigAnno" class="contigAction"
method="searchAnno">
            <result name="input">/WEB-INF/pages/contigSearchAnno.jsp</result>
            <result name="success">/WEB-INF/pages/contigList.jsp</result>
            <result name="cancel" type="redirect">mainMenu.html</result>
            </action>
            <action name="viewContig" class="contigAction" method="view">
                <result name="success">/WEB-INF/pages/contigView.jsp</result>
        </action>
    <!--Contig-END-->
      </package>
</xwork>
```

## APPENDIX O: WEB PRESENTATION LAYER - action-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
  <bean id="adminInterceptor"
class="edu.tamu.goof.webapp.interceptor.UserRoleAuthorizationInterceptor">
     <property name="authorizedRoles" value="admin"/>
  </bean>
  <bean id="userAction" class="edu.tamu.goof.webapp.action.UserAction"
scope="prototype">
     <property name="userManager" ref="userManager"/>
     <property name="roleManager" ref="roleManager"/>
     <property name="mailEngine" ref="mailEngine"/>
     <property name="message" ref="mailMessage"/>
     <property name="templateName" value="accountCreated.vm"/>
  </bean>
     <bean id="passwordHintAction"
class="edu.tamu.goof.webapp.action.PasswordHintAction" scope="prototype">
     <property name="userManager" ref="userManager"/>
     <property name="mailEngine" ref="mailEngine"/>
```

```
        <property name="message" ref="mailMessage"/>
    </bean>
    <!--Contig-START-->
    <bean id="contigAction" class="edu.tamu.goof.webapp.action.ContigAction"
scope="prototype">
            <property name="contigManager" ref="contigManager"/>
    </bean>
            <!--Contig-END-->
    <!-- Add additional actions here -->
</beans>
```

## APPENDIX P: WEB PRESENTATION LAYER - menu-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<MenuConfig>
  <Displayers>
    <Displayer name="Velocity"
type="net.sf.navigator.displayer.VelocityMenuDisplayer"/>
  </Displayers>
  <Menus>
    <Menu name="MainMenu" title="mainMenu.title" page="/mainMenu.html"
width="120">
                    <Item name="ListContig" title="menu.listContig"
page="/contigs.html"/>
      <Item name="SearchContig" title="menu.searchContig"
page="/searchContigs!default.html"/>
      <Item name="SearchContigAnno" title="menu.searchContigAnno"
page="/searchContigAnno!default.html"/>
      </Menu>
    <Menu name="UserMenu" title="menu.user" description="User Menu"
page="/editProfile.html" roles="admin,user"/>
    <Menu name="AdminMenu" title="menu.admin" description="Admin Menu"
roles="admin" width="120" page="/users.html">
      <Item name="ViewUsers" title="menu.admin.users" page="/users.html"/>
```

```
            <Item name="ActiveUsers" title="mainMenu.activeUsers"
page="/activeUsers.html"/>
        </Menu>
        <Menu name="Logout" title="user.logout" page="/logout.jsp"
roles="admin,user"/>
    </Menus>
</MenuConfig>
```

## APPENDIX Q: GOOF DEVELOPERS' SURVEY

Survey questions:

- Is the MVC structure of GOOF generic and flexible to construct a new project? Does the integrated structure of GOOF help the development procedure? Does the data model layer make it easy to construct a database application?
- Does GOOF reduce the programming lines of work for a bioinformatics/web database project? What is the percentage of reduced code that you estimate by comparing with other approaches? Please compare it to different programming languages and frameworks that you have used before.
- Does GOOF reduce the time of development work since it is a new framework for a developer? How about the learning curve?
- Does GOOF help to update a bioinformatics system when adding new functions and interface?
- What are your suggestions for GOOF? What are the advantages and shortcomings of GOOF? What is the wish list of features of GOOF?

Participant #1 (Bioinformatics developer with Java beginner experience and PHP, database expert):

1. The structure of goof: is it clear to construct a new project.

Yes, once I grasped the concept, I have found the three layer model approach to be effective and efficient.

2.  Does GOOF reduce the program lines of development work for a new project? Percentage?  Compare it to the old version of the QTL viewer.

By comparing the number of files and cumulative lines in the original php version of the Bovine QTL Viewer I can say that the object oriented format presented by goof reduces the code by approximately fifty percent by ratio.  The updated Bovine QTL Viewer implements more data but still has reduced the code.

3.  Does GOOF reduce the time of development work since it is a new framework for a developer?  How about the learning curve?

As I am not a native java developer, it took probably more time than usual to begin development.  Nevertheless, once I understood the basic structure things picked up.

4.  Does GOOF help on the maintenance and update capability of a bioinformatics system?  In terms of adding new functions, and interface?

Certainly, because all levels of the goof structure are independent of one another, making modifications and updates has been streamlined.

5.  Suggestions on GOOF?  Pros and cons?  The wish list of functions in GOOF?

I think the most significant module of the Chado, sequence, should have variables/values publicly available across all files in the web layer.  This would streamline construction and allow the user to focus on design. Nevertheless, overall a robust system.

Participant #2 (Java web application developer; Java expert, database expert):

1. Is the MVC structure of GOOF generic and flexible to construct a new project? Does

the integrated structure of GOOF help the development procedure? Does the data model layer make it easy to construct a database application?

We usually build our own MVC structure. For the particular type of project, in your case, a bioinformatics system, it saves a lot of time. For non-Java developer, the data model is easy to build a new schema. For a complicated schema, the data model layer is useful. But for a simple database or several easy database queries, the direct database call is enough.

2. Does GOOF reduce the programming lines of work for a bioinformatics/web database project? What is the percentage of reduced code that you estimate by comparing with other approaches? Please compare it to different programming languages and framework that you have used before.

For a web database project, MVC is the best and most common structure. For a large database project, it is worthy to build a MVC structure. I don't think I need to write many codes for a project by using the fix structure of GOOF. 90% of configurations in MVC are done already by GOOF. That is the most headache part.

3. Does GOOF reduce the time of development work since it is a new framework for a developer? How about the learning curve?

There is no learning curve for me. But I got to know what GOOF is built for.

4. Does GOOF help to update a bioinformatics system when adding new functions and interface?

If following the templates, adding a new function or an interface is easy. But if I want something out of template, I have to understand the links in GOOF.

5. What are your suggestions for GOOF? What are the advantages and shortcomings of GOOF? What is the wish list of features of GOOF?

For your bioinformatics project, GOOF is the way to go. For a general web application, GOOF has some restrictions and limitation. The layout of interface is not flexible if we want other background or banner or different menu structure.  I usually don't deal with CSS.

Participant #3 (C/C++ and database developer, Java beginner, Database expert):

1. Is the MVC structure of GOOF generic and flexible to construct a new project? Does the integrated structure of GOOF help the development procedure? Does the data model layer make it easy to construct a database application?

The MVC is complicated at the beginning for me. But I like it separated the database connection and interface. GOOF is a really big integrated framework. The data model layer is very useful to deal with a lot of queries. I know JAVA and .Net all do it in that way.

2. Does GOOF reduce the programming lines of work for a bioinformatics/web database project? What is the percentage of reduced code that you estimate by comparing with other approaches? Please compare it to different programming languages and frameworks that you have used before.

GOOF does a lot of things already. Once the data model is built, the query part is easy.  No SQL. I like the criteria search.  In term of connecting database, and building a query, GOOF maybe save more than 40~50% of code. PHP or Perl has the database connection session issue. GOOF basically takes care of it. It's not worried any more.

3. Does GOOF reduce the time of development work since it is a new framework for a developer? How about the learning curve?

I'm not a Java developer.  There is a learning curve for me. The structure of GOOF is quite clear. The OO structure is easy to understand.

4. Does GOOF help to update a bioinformatics system when adding new functions and interface?

Yes. The MVC does help.

5. What are your suggestions for GOOF? What are the advantages and shortcomings of GOOF? What is the wish list of features of GOOF?

I would like to have a more detail documentation.  An example GOOF application should be helpful.

## APPENDIX R: EQUINEBASE END USERS' SURVEY

Survey questions:

- Does the search interface in EquineBase help you find the target data fast and easily?
- Is easy it to navigate the interface and find the function that you need? Do you like the layout of menus?
- What are your favorite functions of EquineBase? Do you have any suggestion on improving the interface?

Participant #1(equine genetics researcher):

Nice looking windows.  LOVE, that you can sort the results by name etc. That's a one-up on UCSC.  I'd like to see the address of each hit in the results window, helps me orient and pick which ones really are on the chr I'm looking for, then I could search either in order on the chr or by name.  Like the download options, definitely helps keep multiple search results neat and orderly, rather than the stacks of printed screen shots I have now!

V2 Repeat Search: Got it to work after a bit of fiddling, ie, does "start position" need commas, does "chromosome" need chr, do I need to clear the box that says "repeat family", some simple instructions on format, or an example search would be good. (Along those lines, is the 100k search window just a trial?)  What do the orange asterisks mean?  Can I not search using the repeat name, it's too tempting not to try and pull up all incidences of say... "ERE2" in a particular 100k window.  So far, in the repeat results, if I click on a hit the Gbrowse pic isn't coming up.

V1 Repeats: Ditto on the missing Gbrowse view (seems to get stuck,)

V1 Features: no luck with repeat masker track, got this error

I am using a test region of chr3, starting at 71100000.

- piler track seems to work pretty well, can't always link out by clicking the Gbrowse view, sometimes is says "landmark not recognized"
- ESTs work, though it was a bit educational trying to find one to use as an example, didn't realize just how thin they are. But it is a little awkward that the EST track doesn't appear in the Gbrowse view that comes up when I click the search hit. Made me think for a second I'd been sent to the wrong place. Could maybe show only the relevant track on that view, then I can pick other tracks once I click out to the full viewer. This Gbrowse view is much prettier, and easier than I thought, I use the link out button on the ABCC to get an idea of the neighborhood for SNPs, the window does that without having to click, and I can get more info if I need to without having to cut and paste an address.
- SNPs work great, wish I could see the position though in the search result, helps me decide which is nearest my target, or sort by quality (I know, I am demanding .
- Contig, can't get anything at all to come up on that, maybe I'm doing something wrong?
- no luck with tRNAs, can't find any on the browser anyway

- ditto w/ mRNAs but there isn't an mRNA track really, so I don't know what that's all about.
- Genes, now this is a problem, even in regions where there are genes in the browser, I can't get any to come up in the search…?
- V1 Gene Description: no luck here either, I was pretty excited about it, but I can't get any hits, anywhere, even with searches like "gene" or " ligand" in regions where I have just found genes using the browser. User error? Bummer….

Participant #2 (Genetics researcher):

1. Does the search interface in EquineBase help you find the target data fast and easily?

It is easy to understand which menu I should use to search data. Some search interface is confusing. I don't know what I should put into the search box. I tried something and it didn't come up with anything. Any example? Do you support GenBank ID search? Do you any other search options on EST or SNP track? OMIMBase is better than NCBI.

2. Is easy it to navigate the interface and find the function that you need? Do you like the layout of menus?

The layout of interface is clean. "Search Features" menu is unclear to me.

3. What are your favorite functions of EquineBase? Do you have any suggestion on improving the interface?

The search result table export function is useful. I don't have to drag down mouse to select the data. Gbrowse view is good. I like it is integrated into the search interface. The pop-up window on Gbrowse is very cool. Can you have the pop-up window in EquineBase to show some explanations on the search box?

Participant #3 (Animal genetics researcher with some knowledge of database):

The interface is clean and straightforward. Gbrowse view integration is handy. Each menu represents a group of functions. I like that different users could see different menus.

- One feature search is to search different tracks. It is clean and easy to find the target feature. It is a function which Gbrowse should have.
- Feature search is very fast. But Gbrowse view is slow. And some of Gbrowse view does not show up.
- The search result table export function is very helpful.

Suggestions and bugs:

- Different tracks should have different ranges to choose. I understand a big range search can hit the server very hard.
- It should allow to search across two tracks or more at the same time. It should be useful.
- Top menu could be crowded if you have more functions. Left-hand menu should be good.
- Do I have to fill every field when searching?
- I love to see some functions which show what kind of data or some statistics report of data in the database? For instance, how many genes annotated by March 2008?

- There is no result sometime? Or the search jumps back the menu or the search interface? Is it bug? Some warning messages would be helpful.

Participant #4 (Animal genetics researcher):

I played your website. The overall structure is good. The layout of menu is clear to understand. Gbrowse view is handy so that I don't have to jump to another website for that. There should be some example on OMIMBase search and Gene description search boxes. Sometimes, I could not find any results. Any warning message? The website is fast.

Participant #5 (Genetics researcher):

1. Does the search interface in EquineBase help you find the target data fast and easily?

The search interface is very clear. The research result list format is better than NCBI. I like the export options. The website is fast. Gbrowse view is slow. I guess that is Perl problem. It might be better to have separated menus for each track. What should I put in OMIMBase search box?

2. Is easy it to navigate the interface and find the function that you need? Do you like the layout of menus?

The layout of interface is simple. Left-hand menu could give your more space if you have more functions.

3. What are your favorite functions of EquineBase? Do you have any suggestion on improving the interface?

I like the Gbrowse view. Each database is one top level menu. Show a summary of data in your database. So I know what data I can query in your database. You should have some feedback link.

The report of Observation #1 (Animal genetics researcher):

- Top menu is easy to understand.
- It is fast to find the data.
- Fill-in box is confusing.
- The interface is friendly.
- Gbrowse view is a bonus.
- Result list table is useful.
- Export option is useful.
- Frequent questions on selecting range options.
- The performance website is good.

# VITA

| | |
|---|---|
| Name: | Ning Wei |
| Address: | 3-22, BLVG #22 |
| | Fuxingmenwai St, |
| | Beijing, 100000 |
| | P.R.China |
| Email Address: | xiaoningning@yahoo.com |
| Education: | B.E., Mechanical Engineering and Computer Science, University of Tsinghua, Beijing, China 2002 |
| | M.S., Computer Science, University of Oklahoma, 2002 |