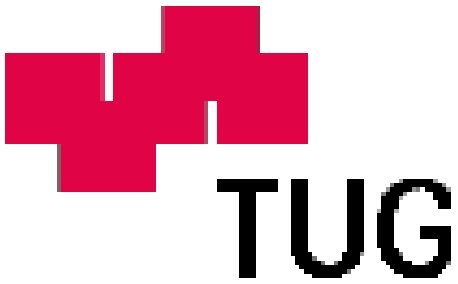
Achuthan Premanand

StocksDB: Design and Development of a Database Application for the Management of Biological Stocks

Master Thesis



Institute of Genomics and Bioinformatics
Graz University of Technology
Petersgasse 14, 8010 Graz
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Zlatko Trajanoski

Supervisor:
Dipl.-Ing. Gerhard Thallinger

Evaluator:
Univ.-Prof. Dipl.-Ing. Dr.techn. Zlatko Trajanoski

Graz, November, 2004

Abstract

Genomic studies ask for a number of molecular biology approaches that rely on different cloning procedures, PCR-based detection mechanisms and immunochemical methods. Additionally the use of different model organisms and tissues deepens the informational content in these projects. The extensive information of all components applied in an interdisciplinary research approach and the resulting data need to be organized and stored in a comprehensive way.

Organized data storage is inevitable for any laboratory which is involved in active research. So, a database exclusively for storing the information about the biological stocks used in the Institute of Genomics and Bioinformatics was planned and subsequently implemented. The ultimate aim of this database is to facilitate a concerted and powerful information use that allows easy access to all possible information sets as well as sharing and exchange of the same by a variety of different users.

The database was implemented based on three-tier architecture with a database backend (MySQL), an application server (JBoss) as the middle tier and a Web Interface as the client tier using the J2EE platform. The design and implementation builds upon an UML model. The code generator AndroMDA was used to convert the UML model into Java source code. The presentation layer was implemented using the Struts framework. Currently, the application facilitates storage of primer and plasmid related data.

Keywords: Cloning, PCR, J2EE, UML, LIMS, AndroMDA, Struts, Stocks, Primer, Plasmid

Contents

List of Figures

List of Tables

1. Introduction

Laboratories are dynamic environments in a constant state of change. Automation in the laboratories has created a demand for similar automation of information management with faster turnaround of data and increased access to information resources. This demand can be met with the help of LIMS. In the generic sense, the LIMS is how a laboratory tracks and manages its information resources, particularly the data that represents the laboratory's product. LIMS are applications that store who, what, where, when, why, etc. information about samples that the laboratory processes. A LIMS is more than software and in recent times it has become the workhorse of the laboratory.

The success of any LIMS implementation depends on the ability of the laboratory and system to anticipate and respond to its change. The most successful LIMS implementations have been those for which functional requirements have been defined for not only the current state of the laboratory, but also the future, improved state [7]. By making lab requirements a part of the selection process, a more flexible and scalable LIMS should be developed that can be easily modified as demands change in the laboratory. A LIMS brings many advantages to a laboratory by quickly providing information to the researchers in the laboratory.

The goal of any LIMS implementation is to improve the efficiency of the laboratory. And the most efficient LIMS implementations have been those that had a high degree of acceptance of the system by all departments (especially laboratory staff). The only way to achieve this is to have the laboratory drive the implementation process with the support of management, researchers, IT and other staffs.

1.2 The need for a database application to store stocks data

Biological sciences are data-intensive. Many disciplines involve either large-scale field investigations or extensive laboratory analyses. A large amount of data is accumulated from these activities [47]. The amount of biological data increases exponentially due to the availability of high efficiency laboratory equipment and technology. The avalanche of incoming data requires us to find efficient ways to store, manage, use, and share these data.

Many biologists use spreadsheets (such as Excel and Lotus) for data storage and analysis. Spreadsheets are easy to learn and convenient to use in many situations. A spreadsheet lacks many features for storage and management of large amount of data, especially, for sharing data through the Web. A spreadsheet is primarily used and maintained by a single person (computer). This is a limitation in today's scientific environments where cooperation and sharing (search and use) data is a key for success. It is either impossible or error prone to query, search, or upgrade spreadsheets remotely. The work can become even more challenging if the query, search or upgrading is across multiple spreadsheets. Under these situations, a database is a better choice, especially when it is intended to share data across network and/or integrate multiple data sets. In a non technical term, a database can be defined as a collection of data. Modern database can be described as a collection of data managed by a Database Management System. A DBMS is a software system for creating, manipulating, and managing data.

1.3 Current and future work scenarios

The current workflow scenario in the laboratory of the Institute for Genomics and Bioinformatics is outlined below,

1. A researcher orders an entity or stocks. (primers, plasmids etc.,) for his/her experiments.
2. He/She receives the entity.
3. Store the entity in a physical storage place.
4. Make an entry into his/her personal lab book or in a spread sheet in his/her own computer.

   Subsequently, from now on, all the associated information about the entity will be entered into his/her lab book. This can include movement of the entities from one physical location to the other, usage of the samples, quality evaluation and reordering the samples.

In this scenario, the data is not stored in one location. The data is spread into many places (lab books) and is stored as per the style and convenience of the individual. Moreover, the data is not available to all the persons at the same time. In short, the data is not uniformly shared and there is no data integrity. So a need for a database to store this information has raised and subsequently implemented.

The future workflow will be,

1. A researcher orders an entity or stocks for his/her experiments.
2. He/She receives the entity.
3. Store the entity in a physical storage place.
4. Make an entry into the StocksDB database.
5. The data is available to all the authorized users.

If the entities are moved, used, evaluated or reordered, all the information will be updated. The primary benefit of the system being, the data is now centrally available to be shared by all the researchers in the group.

1.4 The need for a web application

The Internet has fundamentally changed the way we conduct our research or do our business. Even 10 years ago or so, it was not possible to order a book or access a sequence database in another continent using the computer in our office. All these become possible due to the progress in information technology. It is especially important for biologists to realize that information technology has brought us enormous opportunities in sharing our data. This is possible because of the advancement in programming techniques and tools. Now a database can be easily integrated with the Web. This is a real revolution in terms of information sharing and exchanging. It brings us enormous opportunity and convenience for sharing biological data.

The Web is a client-server type of network, using a series of protocols collectively called TCP/IP (Transmission Control Protocol/Internet Protocol), for communication. Client is any computer or program requesting a service or file. Server is the computer or program accepting the request and provides the requested service. We can use client-server architecture to link our biological database with any users in any place. This makes it possible that we can sit in the office querying the sequence data in GenBank, or conducting literature search using the computer instead of searching the index page of a journal.

Though, currently StocksDB stores only our laboratory information, in future other laboratories in our university or outside our university might also use the database to store their data or simply benefit from accessing the data. As outlined above, as the data has to be shared and used by many people, locally or externally, it was decided to make StocksDB a web application.

1.4 Overview of our laboratory and the techniques relevant to StocksDB

Our laboratory, the Institute for Genomics and Bioinformatics, Christian Doppler Laboratory for Genomics and Bioinformatics, Graz University of Technology, with approximately 30 staff members and students is primarily involved in Microarray research and services**.** Additional activities are providing bioinformatics education and services for the scientific community, development of specialized databases and tools for computational analysis of genomic data, and consulting for software engineering.

1.4.1 Microarray Technology

Microarray is a technique that enables parallel assessment of the relative expression of thousands of mRNAs in response to different experimental conditions or in different tissues.

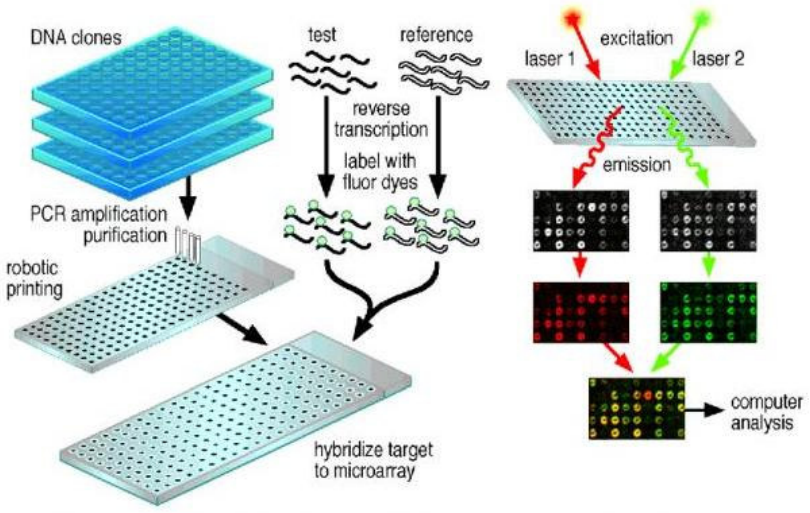


Figure 1.1 The different stages of Microarray Workflow are shown: Microarray Production, Sample Preparation, Hybridization, Image Analysis, Data Analysis [24]

It consists of large numbers of cDNAs or oligonucleotides spotted onto a glass microscope slide. RNA isolated from two populations of cells, one control and one altered by experimental treatment or disease, is labeled with two different fluorochromes before being hybridized to the microarray. After a standard hybridization reaction, a scanner records the intensity of the two fluorochromes. The data after normalization, which means to adjust microarray data for effects

which arise from variation in the technology rather than from biological differences between the RNA samples or between the printed probes [14], can be analyzed using special software that enables clustering of genes that have similar expression patterns.

However, there is little known about the accuracy of microarrays in identifying regulated transcripts or about the relationships of the relative changes in mRNA levels obtained using microarrays to the actual relative levels of these mRNAs in the samples assayed [39]. The usual way to validate the Microarray results would be to cross check the results with quantitative RT-PCR.

1.4.2 PCR (Polymerase Chain Reaction)

Polymerase chain reaction (PCR) is a common method of creating copies of specific fragments of DNA. PCR has found applications in almost every imaginable facet of molecular biology like sequencing, expressing, mutating etc.,

The first step for PCR would be to synthesize "**primers**". Primer is a small oligonucleotide (anywhere from 6 nucleotides long) used to prime DNA synthesis. The DNA polymerases are only able to extend a pre-existing strand along a template; they are not able to take a naked single strand and produce a complementary copy of it de-novo. A primer which sticks to the template is therefore used to initiate the replication. Primers are used in **pairs** - one primer is directed against one strand in the DNA sequence to be amplified, the other against the complementary strand as shown in Figure 1.2.

PCR is carried out using a machine known as a thermocycler. It heats and cools reactions in an accurate and programmable manner. The reaction is set to contain the target or template cDNA, two oligonucleotide primers, DNA polymerase, and deoxynucleotide triphosphates. At the beginning of the reaction, the temperature is increased. The double-stranded DNA template denatures into single strands. Then the temperature is lowered. As the temperature decreases, the primers bind to, or anneal, to complementary sequences in the template DNA. After annealing, the temperature is increased to optimize the activity of the DNA polymerase, which extends the primers, based on the template DNA sequence. After a set amount of time, the polymerase can synthesis approximately 1000 bases of DNA per minute, the temperature is raised again to approximately 94 °C to denature all of the strands in the reaction. When the temperature is returned to the annealing temperature, the cycle begins again. Each cycle leads to a doubling of the number of target DNA.

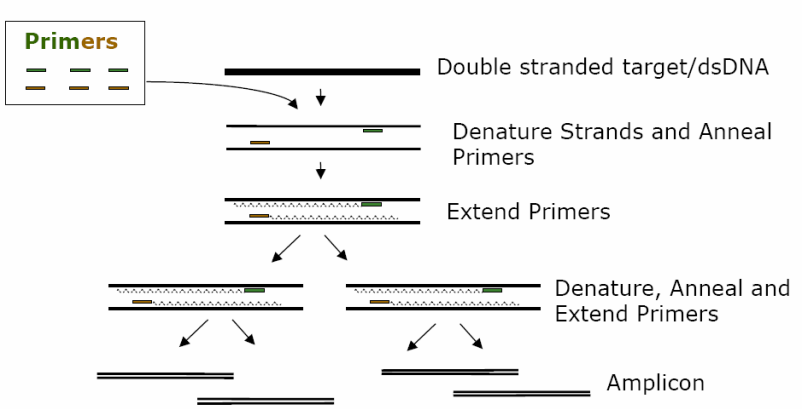


Figure 1.2 Thermal Cycling-Principle of Traditional PCR. Here the PCR specifically targets and amplifies a single sequence from a complex mixture of DNA [40]

The resulting amplified DNA can be isolated from the reaction mix by a process known as gel electrophoresis.

To analyze individual amplified DNA molecules, each originally copied from a specific template DNA molecule, they must be cloned. This is done by sub cloning pieces of DNA into a vector.

1.4.3 Cloning & Plasmids

"Vector" is an agent that can carry a DNA fragment into a host cell. If it is used for reproducing the DNA fragment, it is called a "**cloning vector**". If it is used for expressing certain gene in the DNA fragment, it is called an "**expression vector**".

Commonly used vectors include **plasmid**, Lambda phage, cosmid and yeast artificial chromosome (YAC).

Plasmids

Plasmids are circular, double-stranded DNA molecules that exist in bacteria and in the nuclei of some eukaryotic cells [30]. They can replicate independently of the host cell. The size of plasmids ranges from a few kb to near 100 kb.

There are some minimum requirements for plasmids that are useful for recombination techniques:

1. Origin of replication (ORI): They must be able to replicate themselves or they are of no practical use as a vector.

2. Selectable marker: They must have a marker so as to select the cells that have the plasmids of interest.

3. Restriction enzyme sites in non-essential regions: To prevent cutting the plasmid in necessary regions such as the ORI.

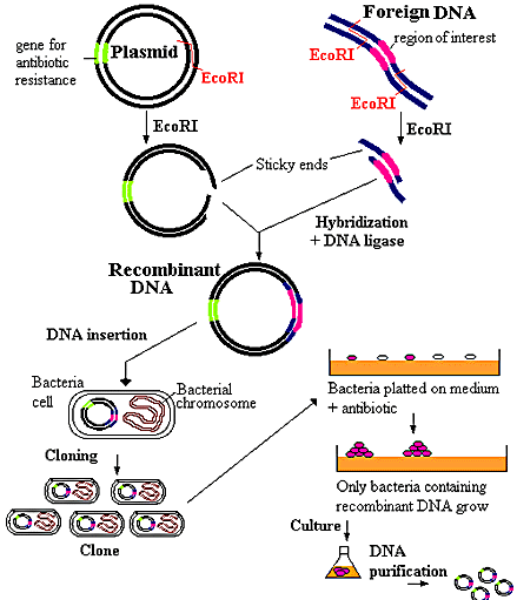


Figure 1.3 Process by which a plasmid is used to import recombinant DNA into a host cell for cloning [31]

The plasmid carrying genes for **antibiotic resistance**, and a DNA strand, which contains the gene of interest, are both cut with the same restriction endonuclease. The plasmid is opened up and the gene is freed from its parent DNA strand. They have complementary "sticky ends." The opened plasmid and the freed gene are mixed with DNA ligase, which reforms the two pieces as recombinant DNA. Plasmids and copies of the DNA fragment produce quantities of recombinant DNA.

This recombinant DNA is allowed to transform into a bacterial culture of any preferred **strain**, which is then exposed to antibiotics. All the cells except those which have been encoded by the plasmid DNA recombinant are killed, leaving a cell culture containing the desired recombinant DNA. DNA cloning allows a copy of any specific part of a DNA (or RNA) sequence to be selected among many others and produced in an unlimited amount.

PCR has completely revolutionized the detection of RNA and DNA. Traditional PCR has advanced from detection at the end-point of the reaction to detection while the reaction is occurring.

1.4.4 Real-Time PCR

The fluorescence-based real-time reverse transcription polymerase chain reaction (RT-PCR) is widely used for the quantification of steady-state mRNA levels [4]. It's an important step for the validation of expression data generated by microarray analysis or other genomics techniques. This has been facilitated by the development of instruments that measure the amount of PCR product produced at each step of the reaction or in "real time". RT-PCR is also the technique of choice for analyzing extremely low abundance mRNA derived from cells or tissues.

RT-PCR analysis detects specific nucleic acid amplification products as they accumulate in real-time. Real-Time chemistries allow for the detection of PCR amplification during the early phases of the reaction [32].

1.4.5 Real Time PCR Chemistries

There are several different techniques available that detect amplified product with reportedly similar sensitivity [45]. All of these methods use fluorescent dyes and combine the processes of amplification and detection of an RNA target to permit the monitoring of PCR reactions in real-time. The simplest method uses a fluorescent dye that specifically binds to double-stranded-DNA. The other methods rely on the hybridization of fluorescence-labeled probes to the correct amplicon.

DNA-binding dyes (SYBR Green System)

This method involves detection of the binding of a fluorescent dye (SYBR Green) to DNA. This is the simplest and cheapest chemistry. The unbound dye exhibits little fluorescence in solution, but during elongation increasing amounts of dye bind to the minor-grove of the nascent double-stranded DNA. When monitored in real-time, this results in an increase in the fluorescence signal during the polymerization step. Since the signal intensity decreases during the denaturation step, the fluorescence measurements are performed at the end of the elongation step of every PCR cycle [37].

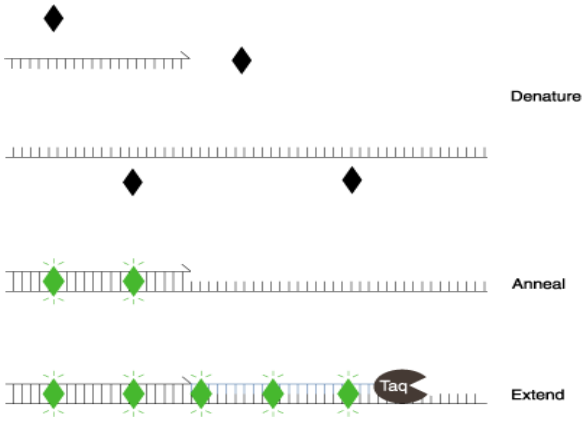


Figure 1.4 Schematic representation of real-time PCR with the SYBR Green I dye. SYBR Green I dye (black diamonds) becomes fluorescent (green diamonds) upon binding to double-stranded DNA, providing a direct method for quantitating PCR products in real time [37].

TaqMan System

The Taqman assay utilizes the 5'-nuclease activity of the DNA polymerase [15] to hydrolyze a hybridization probe bound to its target amplicon.

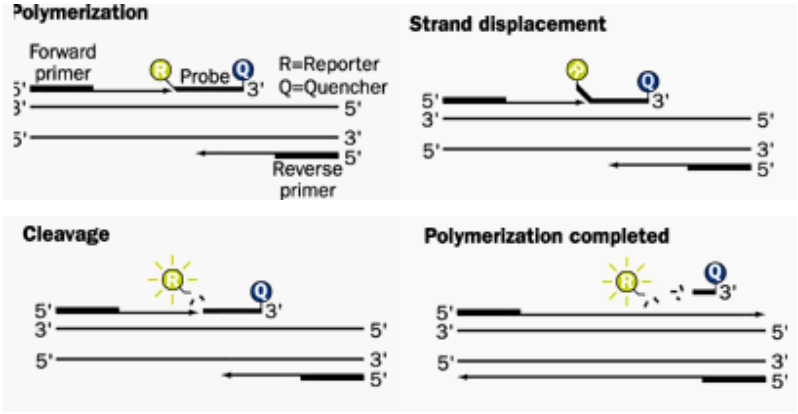


Figure 1.5 TaqMan System [48]

It continuously measures PCR product accumulation using a dual-labeled fluorogenic oligonucleotide probe called a TaqMan® probe. This probe is composed of a short (approx 20-25 bases) oligodeoxynucleotide that is labeled with a reporter dye on the 5' terminus and a quenching dye on the 3' terminus. This oligonucleotide probe sequence is complimentary to an internal target sequence present in the PCR amplicon. When the probe is intact, energy transfer occurs

between the two fluorophors and emission from the reporter is quenched by the quencher. During the extension phase of PCR, the probe is cleaved by 5' nuclease activity of Taq polymerase, thereby releasing the reporter from the oligonucleotide-quencher and producing an increase in reporter emission intensity [32].

LUX (Light Upon eXtension) System

The LUX™ effect presents a novel fluorescent detection mechanism for real-time analysis. LUX™ Primers are oligonucleotides labeled with a single fluorophore, custom-synthesized according to the DNA/RNA of interest. Typically 20-30 bases in length, they are designed with the fluorophore close to the 3´ end in a hairpin structure. This configuration, an advancement from the dual-labeled probe format, intrinsically renders fluorescence quenching capability so that a separate quenching moiety is not needed [21].

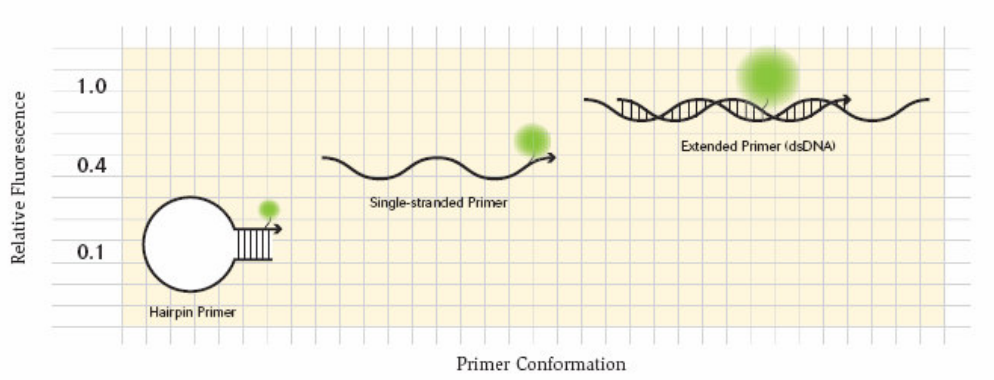


Figure 1.6 LUX System. The increase in Fluorescence Intensity under hairpin, single-stranded and extended primer conformations are shown [21]

When the primer is incorporated into the double-stranded PCR product, the fluorophore is dequenched, resulting in a significant increase in fluorescent signal. This signal increase is the basis for the LUX™ detection platform.

From the above overview, please note the keywords which are important from the StocksDB point of view: Forward Primer, Reverse Primer, Primer pairs, SybrGreen, TaqMan, LUX, Cloning, Plasmids, antibiotic resistance, and strain. StocksDB stores information about all these entities.

2. Goals and Objectives

The overall goal of this thesis is to design and develop a database to store all relevant information about the biological stocks (like primers, plasmids etc.,) used at our institute, as well as to provide a web-based application and user interface to access this database. Moreover, from the development point of view, the application should be developed with Model Driven Architecture using AndroMDA, J2EE and Struts framework. These technologies will be covered in detail in the next chapter. In general, the application has to meet the following objectives.

Easy database access: Users should be able to store the data and also browse through the stored data easily through a user-friendly interface.

Understandable presentation of information: The data should be presented to the end user in a simple and useful manner.

Efficient search features: There should be features to search the database efficiently. There should be flexible interface to query the database for both novice and advanced users.

Minimal need for manual input of data: The interface should reduce the manual typing of the information as and when possible, which helps to reduce the errors and increase the data uniformity.

Extensibility: It should be possible to integrate external applications easily. It should be flexible enough to export some data from StocksDB to other databases like MARS (Microarray analysis and retrieval system), another database application developed and used at our institute.

User access control: There should be a flexible user and access control mechanism that grants different levels of rights to different users, based on the defined criteria.

Support for the overall experimental process: Ultimately, the researchers should benefit from the system in such a way that it should hasten and also help to improve their day-to-day research activities.

3. Methods

3.1 Techniques for integrating a database with the Web

There are many programming technologies that can integrate a database with the Web such as CGI, ASP, JSP, and PHP. CGI, Common Gateway Interface, used to be the most common technology for the task. It has become less and less popular these days due to its slow execution, slow programming, and declining technical support [47]. Many programming languages, such as Perl, C, C++, can be used in CGI.

Currently, there are two prominent server-side platforms namely Sun Microsystems' Java 2 Platform, Enterprise Edition (J2EE), which includes the Enterprise JavaBeans (EJB) server-side component architecture. The second is Microsoft's Windows .NET platform. The discussion will be restricted to the J2EE framework as **StocksDB** was developed with it.

3.2 Distributed architectures and J2EE

In the past, two-tier applications -- also known as client/server applications -- were commonplace. Figure 3.1 illustrates the typical two-tier architecture. In some cases, the only service provided by the server was that of a database server. In those situations, the client was then responsible for data access, applying business logic, converting the results into a format suitable for display, displaying the intended interface to the user, and accepting user input. The client/server architecture is generally easy to deploy at first, but is difficult to upgrade or enhance. It also makes reuse of business and presentation logic difficult, if not impossible. Finally, and perhaps most important in the era of the Web, two-tier applications typically do not prove very scalable and are therefore not well suited to the Internet [36].

Sun designed J2EE in part to address the deficiencies of two-tier architectures. As such, J2EE defines a set of standards to ease the development of *n*-tier enterprise applications. It defines a set of standardized, modular components; provides a complete set of services to those components; and handles many details of application behavior, such as security and multithreading, automatically [36].

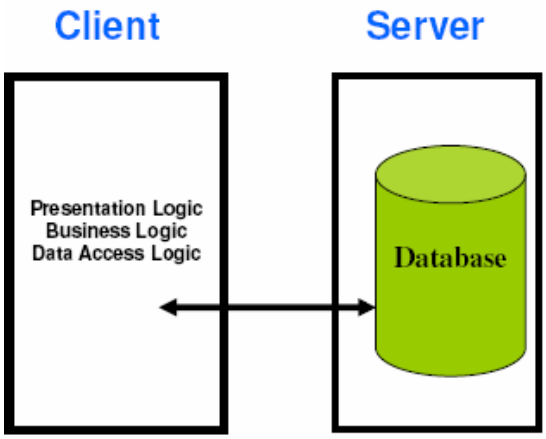


Figure 3.1 Two-tier architecture is shown. Fat client and thin server can be observed

Using J2EE to develop *n*-tier applications involves breaking apart the different layers in the two-tier architecture into multiple tiers. An *n*-tier application could provide separate layers for each of the following services:

1. **Presentation:** In a typical Web application, a browser running on the client machine handles presentation.

2. **Dynamically generated presentation:** Although a browser could handle some dynamically generated presentation, for the widest support of different browsers much of the action should be done on the Web server using JSPs, servlets, or XML (Extensible Markup Language) and XSL (Extensible Stylesheet Language).

3. **Business logic:** Business logic is best implemented in Session EJBs (described later).

4. **Data access:** Data access is best implemented in Entity EJBs (described later) and using JDBC.

5. **Backend system integration:** Integration with backend systems may use a variety of technologies. The best choice will depend upon the exact nature of the backend system.

Why should there be so many layers? Well, the layered approach makes for a more scalable enterprise application. It allows each layer to focus on a specific role, for example, allowing a Web server to serve Web pages, an application server to serve applications, and a database server to serve databases. Because it is built on top of the Java 2 Platform, Standard Edition (J2SE), J2EE provides all the same advantages and features of J2SE. These include "Write Once, Run Anywhere" portability, JDBC for database access, CORBA technology for interaction with

existing enterprise resources, and a proven security model. Building on this base, J2EE then adds support for Enterprise JavaBean (EJB) components, Java servlets, JavaServer Pages (JSPs), and XML technology.

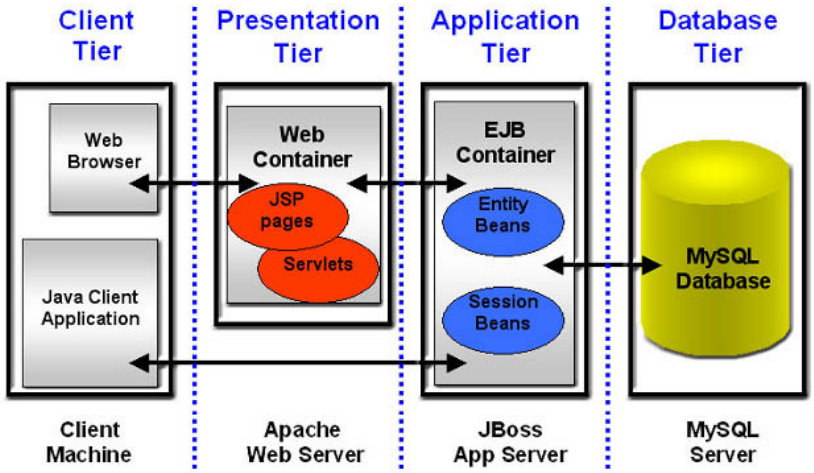


Figure 3.2 A n-tier architecture with client, presentation, application and database tier is shown [12]

The J2EE platform

The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered Web-based applications. In J2EE there are components and services. Components include servlets, JSP and EJB. J2EE services (to service components) are RMI, JDBC and JNDI. These components may be accessed through Web based client (browsers) as well as pure Java clients

Around 13 core technologies (components and services) make up J2EE: JDBC, JNDI, EJBs, RMI, JSP, Java servlets, XML, JMS, Java IDL, JTS, JTA, JavaMail, and JAF, perhaps the most commonly used J2EE technologies include JDBC, JNDI, EJB, JSPs, and servlets.

The J2EE technologies

In the following sections, each of the technologies making up J2EE will be briefly described, and also it will be shown how JBoss Server supports them in a distributed application. Figure 3.3 illustrates where each of the J2EE technologies are most commonly used within a distributed application.

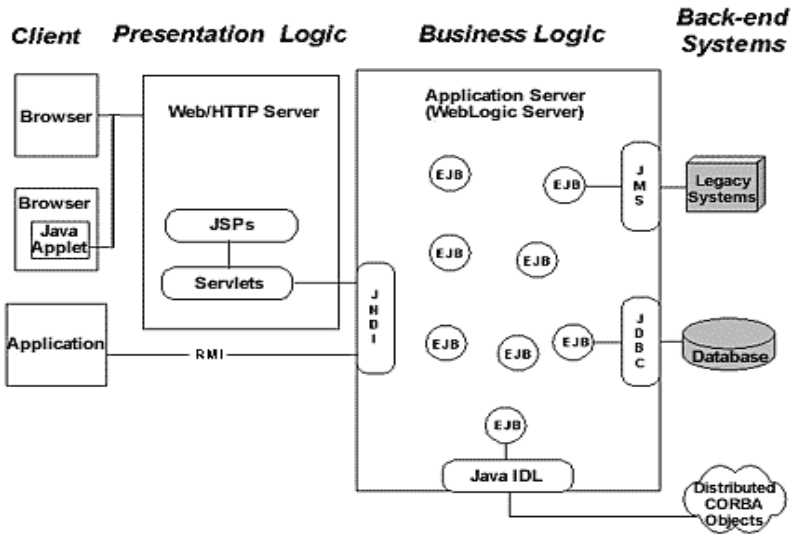


Figure 3.3 A sample n-tier Architecture with all the components of each layer is shown [36]

3.2.1 Java Database Connectivity (JDBC)

The JDBC API accesses a variety of databases in a uniform way. Like ODBC, JDBC hides proprietary database issues from the developer. Because it's built on Java, JDBC also is able to provide platform-independent access to databases.

JDBC defines four fundamentally different types of drivers.

Type 1: JDBC-ODBC Bridge

Type 2: JDBC-native Driver Bridge

Type 3: JDBC-network Bridge

Type 4: Pure Java driver

Please refer to [16] for further details.

3.2.2 Java Naming and Directory Interface (JNDI)

The JNDI API is used to access naming and directory services. As such, it provides a consistent model for accessing and manipulating such enterprise-wide resources as DNS, local filesystems, or objects in an application server. In JNDI, every node in a directory structure is called a

context. Every JNDI name is relative to a context; there is no notion of an absolute name. An application can obtain its first context using the `InitialContext` class:

```
Context ctx = new InitialContext();
```

From this initial context, the application can traverse the directory tree to locate the desired resources or objects. For example, assume that an EJB is deployed within JBoss Server and bound the home interface to the name myApp.myEJB. A client of this EJB, after obtaining an initial context, could then locate the home interface using:

```
MyEJBHome home = ctx.lookup( "myApp.myEJB" );
```

Once a reference to the acquired object is obtained, in this case, the home interface of the EJB -- it is then possible to invoke methods on it.

3.2.3 Enterprise Java Beans (EJB)

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In **StocksDB**, for example, the enterprise beans might implement the business logic in methods such as calculateTm() or fetchSequence(). By invoking these methods, remote clients can access the services provided by the application.

The EJBs specification defines three fundamental types of bean:

Stateless session beans: These provide a single-use service, do not maintain any state, do not survive server crashes, and are relatively short lived. For example, a stateless session bean may be used to perform temperature conversion.

Stateful session bean: These provide a conversational interaction with the client and, as such, store state on the behalf of the client. An online shopping cart is a classic example of a stateful session bean. Stateful session beans do not survive server crashes, are also relatively short lived, and each instance can be used only by a single thread.

Entity beans: These provide a representation of persistent data, typically stored in a database, and can therefore survive a server crash. Multiple clients can use EJBs that represent the same data. An example of an entity EJB: a Primer entity bean.

There is another type of bean called Message-Driven bean, which is not used in StocksDB.

In spite of their differences, all EJBs have much in common. They all possess a home interface that defines how a client can create and destroy the EJB; a remote interface that defines the methods a client can invoke on the bean; and a bean class that implements the main business logic.

Some Definitions

Bean Class (Enterprise Bean Instance)

The Enterprise bean class implements the business logic for the bean. This is where most of the coding is done in an EJB application. The bean class is not accessed directly by the client, instead the remote or local interfaces of the bean is used to call these methods.

Home Interface

The home interface defines the methods a client uses to create, locate, and destroy instances of an enterprise bean.

Remote Interface

The remote interface defines the business methods implemented in the bean. A client accesses these methods through the remote interface.

Local Interface and Local Home Interface

Local Interface and Local Home Interface provides support for lightweight access from enterprise beans that are local clients (existing in the same JVM). This way enterprise beans can be directly accessed without an overhead associated with remote method calls.

EJB Object and EJB Local Object

EJB Object and EJB local Object implements the remote and local interfaces of a bean that is provided by the container. EJB Local Object is new in version 2.0 of the EJB specification and can be used between enterprise beans existing in the same JVM.

EJB Home Object and EJB Local Home Object

EJB Home Object and EJB Local Home Object implement the home and local home interfaces that are provided by the container. They manage the lifecycle of the enterprise beans instance

that they are representing. The Local Home Object is new in version 2.0 of the EJB specification and can be used between enterprise beans existing in the same JVM.

Deployment Descriptor

Each bean is declared in the deployment descriptor of the EJB application, which describes how the beans should be deployed and how the container should deal with them. A deployment descriptor is written in XML and is called **ejb-jar.xml**.

Describing how to develop an EJB is beyond the scope of this chapter. However, once an EJB has been developed or purchased from a third party, it must be deployed in your application server.

Once an EJB has been deployed, a client can locate the EJB using its JNDI name. First, it must obtain a reference to the home interface. Then, using that interface, the client can invoke one of the bean's `create()` methods to obtain a handle to a bean instance running on the server. Finally, the client may use this handle to invoke methods on the bean.

Benefits of Enterprise Java Beans

For several reasons, enterprise beans simplify the development of large, distributed applications.

First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container—not the bean developer—is responsible for system-level services such as transaction management and security authorization.

Second, because the beans—and not the clients—contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant J2EE server.

3.2.4 JavaServer Pages (JSPs)

JSPs [18] are the platform-independent equivalent of Microsoft's Active Server Pages (ASPs). They were designed to help Web content developers create dynamic Web pages with relatively

little coding. Web designers who don't know how to program can use JSPs to create dynamic pages. A JavaServer Page consists of HTML code interspersed with Java code. The server processes the Java code when the page is requested by the client, returning the generated HTML page to the browser.

3.2.5 Java servlets

Servlets are Java program classes that extend the functionality of a Web server. When a Web browser makes a request that corresponds to a servlet, the Web server calls the class's service() method. This method must generate content (usually HTML or XML) to be conveyed back to the browser. Servlets and JSPs are widely used to provide a Web interface to an EJB application. JSPs typically consist mostly of HTML code interspersed with small amounts of Java code, servlets, on the other hand, are written totally in Java and produce HTML code

With Java servlets, an overview of J2EE's major technologies have been covered, but that's not the end to what J2EE has to offer. There are other technologies like RMI, Java IDL and CORBA, JTA, and XML, which are beyond the scope of this chapter. Refer [36] for a detailed explanation of these technologies.

3.2.6 Distributed architectures with JBoss Application Server

An application server provides an EJB container. The container, where EJB components live, supplies middleware services to the EJB components and manages them. Examples of EJB containers are BEA's WebLogic, Sun-Netscape-AOL alliance's iPlanet, IBM's WebSphere, Oracle's Oracle 9i, and the JBoss open source application server.

JBoss [17] is an applications server that provides an EJB container along with other low-level services such as distributed transaction management, security, resource management, persistent management, remote accessibility, multi-client support and location transparency to an enterprise system. Figure 3.4 illustrates JBoss application server that is located between a Web server and backend database system. Typically, accessing EJB components that are deployed to the JBoss application server is performed via a Web server such as Apache from a browser.

J2EE provides a framework -- a standard API -- for developing distributed architectures. The implementation of an engine to implement this framework is left up to third-party vendors. Some vendors will focus on particular components of the overall J2EE architecture. For example, Apache's Tomcat [38] provides support for JSPs and servlets. JBoss server provides a complete

implementation of the J2EE specification and makes it easy to build and deploy scalable, distributed applications.

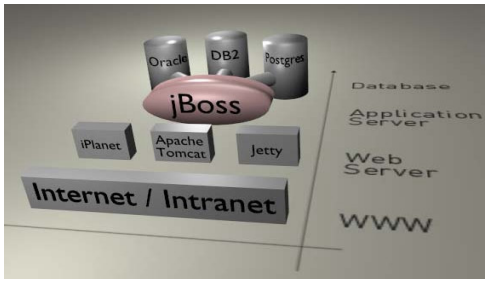


Figure 3.4 The place where the JBoss Application Server resides in an n-tier architecture is shown [3]

Deployment of components in JBoss is easy. The components have to be just bundled as per the J2EE specification by creating ejb-jar.xml and the web archive and placed in the deploy directory in the JBoss server. For **StocksDB** development JBoss 3.2.4 was used.

By now, it would apparently appear that packaging and deploying a component is simple and straightforward. But in reality, working in Java either means writing a little bit of complex code or writing a lot of grunt work code. J2EE is a prime example; implementing the persistence for a single database table takes five classes, two interfaces, and a deployment description using EJBs, and almost all of the classes are clerical work.

Of course everything has to be written, but it need not necessarily be coded manually by hand. Code-generation techniques can make building high-quality EJB code a cake-walk. The following section will cover what is code generation and how the EJBs are generated in **StocksDB**.

3.3 Code Generation

Code generation is the technique of writing and using programs that build application and system code [6]. To understand code generation, it is imperative to understand what goes in and what comes out. What goes in is the design for the code in a declarative form: "We need two tables named primer and plasmid with so and so fields." What comes out is one or more target files. It

could be Java code, deployment descriptors, SQL, documentation, or any type of controlled output.

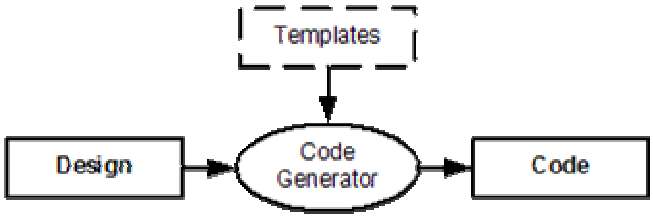


Figure 3.5 The code generator reads in the design, and then uses a set of templates to build output code that implements the design [6]

The components can change slightly between the different models, but the story remains the same. The code generator reads in the design, and then uses a set of templates to build output code that implements the design.

3.3.1 The Benefits

Before getting into specific examples of code generators for Java, the end goals should be firmly defined. One way to approach this is to think about the qualities that are expected from an optimal generator.

Quality: The output code should be at least as good as what would have been written by hand.

Consistency: The code should use consistent class, method, and argument names.

Productivity: It should faster to generate the code than to write it by hand.

Abstraction: The design should be specified in an abstract form, free of implementation details. That way the generator can be re-used again at a later date if there is a need to move to another technology platform.

What is Expected of the Generator?

The output files of a generator are called the target files. There are several generation targets within the Java enterprise application stack. Figure 3.6 shows the stack:

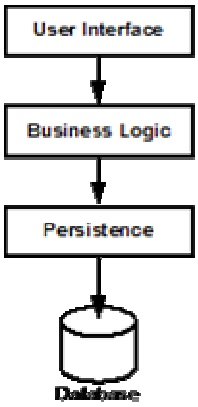


Figure 3.6 Shows the stack with different target elements [6]

All four of these elements of the stack are potential generation targets, but some are more common than others. From the bottom to the top:

Database: Given Java's object-persistence approaches to database work, there isn't much call for direct generation of SQL for database code or stored procedures.

Persistence: Database persistence code is the most common generation target in the Java environment, because it is generally redundant grunt code. Generated database-persistence code also is an excellent foundation for a solid application, because it is consistent and relatively bug-free.

Business Logic and **User Interfaces:** Only MDA and custom generators build production business logic and user interfaces. The critical factor in generating this code is building on top of a stable, predictable platform, ideally a generated persistence layer.

Given an understanding of which Java application components can be generated and what have to be looked out for, the next section gives a brief overview of the generators that build them.

3.3.2 Code-Driven Approach: XDoclet

The most popular code generator for Java is XDoclet [46]. It is easy and pragmatic, and it fits all needs. XDoclet builds database-persistence beans to match the requirements specified in special JavaDoc comments within the Java entity bean code. This is called the "code driven approach" because it uses source code as the design input source.

Given a single entity bean with some markup, XDoclet will create the session beans, interfaces, and data access object required to complete the functional set.

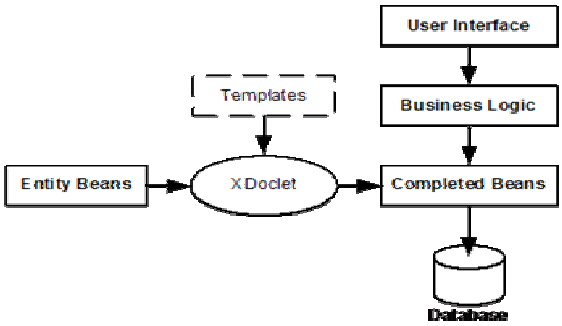


Figure 3.7 Show how XDoclet relates to the application stack[6]

XDoclet's only drawback is its level of abstraction. Because the design is described in JavaDoc tags embedded in the code, code and design are bound tightly together with implementation specifics. Given this binding, it would be difficult to use XDoclet markup to generate complete code in a different language (e.g., C#).

3.3.3 Model-Driven Approach

The alternative to the code-driven approach is to build code from an abstract model of the design. MDA is an approach to system development, which increases the power of models in that work. It is ***model-driven*** because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.

Model-Driven Architecture (MDA), is an emerging industry standard for promoting active use of models in software development. MDA provides an approach for, and enables tools to be provided for:

- specifying a system independently of the platform that supports it,
- specifying platforms,
- choosing a particular platform for the system, and
- transforming the system specification into one for a particular platform.

The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns. The central idea is simple: turn a model in UML (Unified Modeling Language) into code.

Figure 3.8 The flow of MDA generator-Platform Independent Model to output code through the Platform Specific Model [6]

The generator starts with the Platform Independent Model (PIM), created in a UML editing tool, like Poseidon or MagicDraw. (The PIM can be in an exported XML format called XMI.) A Platform-Specific Model (PSM) is then created using a transformation. Templates are applied to the PSM to create the output code.

It is easier to understand the difference between a PIM and a PSM in context. The PIM specifies the application business logic, for example, a table named primer with these five fields. The PSM is a model of the implementation on a particular platform. In the EJB world, this is the set of UML models of the entity and session beans required to implement the primer table.

Some of the more popular MDA solutions are: OptimalJ, ArcStyler, Codagen and AndroMDA. **StocksDB** was developed using AndroMDA [1]. The main reason for choosing AndroMDA is that we had already developed a very stable custom framework with AndroMDA from a previous Master's student project. Please refer the Master Thesis '**Data Integration into a Gene Expression Database**' by Thomas Truskaller [8]. The Thesis covers all about AndroMDA, templates, cartridges etc., in more detail. A very short introduction to AndroMDA is given in the next section. Before that, it would be more relevant to see about UML here.

3.4 Unified Modeling Language

The UML [41] prescribes a standard set of diagrams and notations for modeling object oriented systems, and describe the underlying semantics of what these diagrams and symbols mean [26]. UML can be used to model different kinds of systems: software systems, hardware systems, and real-world organizations. UML offers nine diagrams in which to model systems:

- **Use Case diagram** for modeling the business processes
- **Sequence diagram** for modeling message passing between objects
- **Collaboration diagram** for modeling object interactions
- **State diagram** for modeling the behavior of objects in the system
- **Activity diagram** for modeling the behavior of Use Cases, objects, or operations
- **Class diagram** for modeling the static structure of classes in the system
- **Object diagram** for modeling the static structure of objects in the system
- **Component diagram** for modeling components
- **Deployment diagram** for modeling distribution of the system.

Refer [29] to get more information about modeling and its importance.

Currently there are many UML modeling tools available in the market. To name a few, Rational Rose, Poseidon, UML2COM, Together, OptimalJ, QuickUML**,** iUML**,** Magic Draw etc.,.

For StocksDB, Magic Draw Community Edition (free ware) is used. The main reason for choosing Magic Draw is because of its good compatibility with AndroMDA. Moreover, it has very user friendly interface and it is easy to use and learn.

3.4.1 MagicDraw

MagicDraw [22] UML was developed in Lithuania by an American-managed company named No Magic. Written in Java, it is therefore available for a wide range of platforms. Showing its multi-platform basis, launching and using the product results in multiple independent windows spread neatly around the screen. Diagrams can be printed and saved individually as JPEG files. The class definitions (including methods, attributes, and their associated documentation) can be written to an HTML file.

There is good tutorial available at the official site of MagicDraw which is enough to get started [22]. To know more about the UML, try the OMG's primer at [29]

It should be kept in mind that "UML will never generate the functional code, but it does generate the architectural shell":

3.5 Generation of the Java Code from the UML model

AndroMDA [1], is an open source MDA generator that reads XMI files and uses cartridges to build the various types of Java code. Changing persistence mechanisms, for example, is merely a matter of changing a cartridge.

Figure 3.9 gives an overview of the development cycle in AndroMDA. On the input side it takes an UML model and builds a corresponding metamodel using the Metadata Repository (MDR), an Open Source project of "Netbeans". The instantiated objects of the model are sent to Velocity [43], an Open Source scripting engine which can generate any text document from the model depending on the generation rules (cartridges) [8].

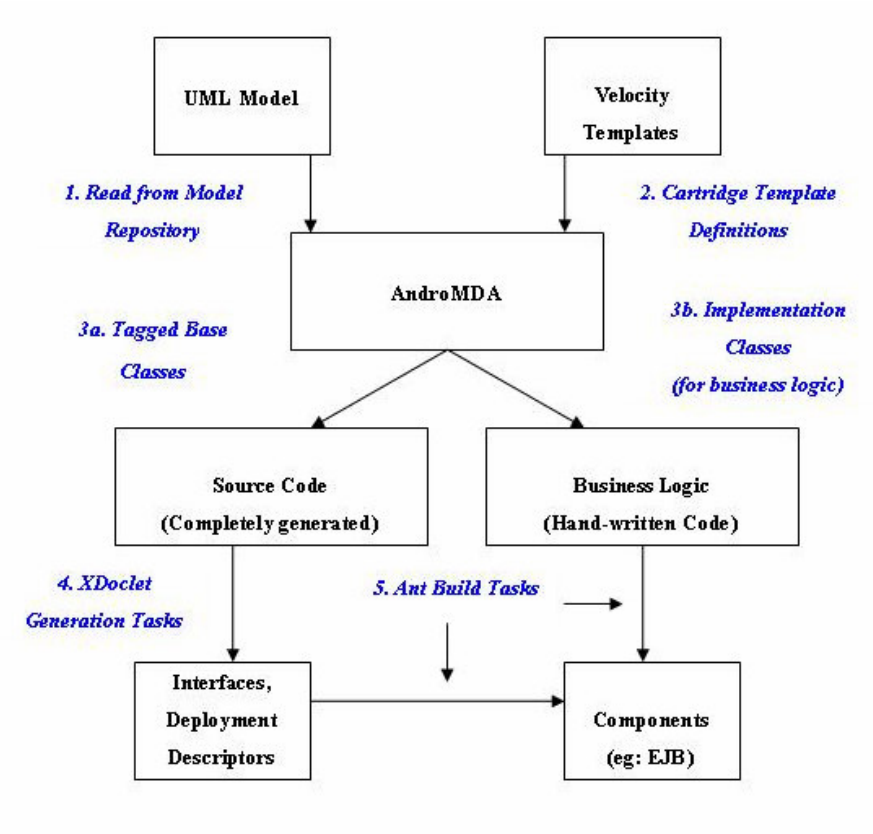


Figure 3.9 AndroMDA code generation process [8]

AndroMDA evolved from the UML2EJB project and cartridges for EJB (Enterprise Java Bean), cartridges for the O/R mapping tool Hibernate as an alternative persistence technology and

cartridges for the Struts framework for Web applications Bpm4Struts [35] are currently available. The Struts part was decided to develop manually as the custom cartridges for Struts are not available yet. Struts will be covered in the following sections.

The EJB and Hibernate cartridges generate java source code tagged with additional information readable for the XDoclet code generator framework. XDoclet processes the source code, generates interface and deployment descriptors and completes the code generation process. AndroMDA 3.0 is the latest version and AndroMDA 2.x is used for **StocksDB** as the cartridges are developed in 2.x and moreover, it suits the current need well.

Modeling is all about communication. When using the UML all the tools needed are provided in order to specify, visualize, construct, and document the elements of a software-intensive system. Even though the UML is very well-defined, there might be situations in which the language has to be bended or extended in some controlled way to tailor it to specific problem domain in order to simplify the communication of our objective. This is where the UML extension mechanisms come in.

3.5.1 Extensibility Mechanisms

The extensibility mechanisms allow customizing and extending the UML by adding new building blocks, creating new properties, and specifying new semantics in order to make the language suitable for specific problem domain. There are three common extensibility mechanisms that are defined by the UML: stereotypes, tagged values, and constraints.

Stereotypes

A Stereotype is a UML model element that is used to classify other UML elements. Certain Stereotypes are already defined in UML. A user can also define his custom stereotypes.

Stereotypes drive code generation [1]. They are a kind of "labels" that are attached to modeling elements to classify them. For example: If a class PrimerService is modeled, which contains all the business methods implementation for the entity Primer, then it can be tagged with a `<<Service>>` stereotype. AndroMDA sees this stereotype, looks into its internal dictionary of available code generation components (called "cartridges") and finds the EJB cartridge. For example, if in that cartridge, two templates correspond to this stereotype: SessionBean.vsl and SessionBeanImpl.vsl. Then, AndroMDA takes the internal representation of PrimerService and calls the template processing engine twice, using this representation as input. As a result, two

output files are generated, called PrimerServiceBean.java and PrimerServiceBeanImpl.java. The other commonly used stereotypes are <<Entity>> for class, <<PrimaryKey>> for attribute, <<FinderMethod>> for operation, <<EntityRef>>, <<ServiceRef>>, and <<ExceptionRef>> for defining the dependencies between classes.

Tagged Values

A Tagged Value is a name-value pair denoting a characteristic of a Model Element. Some Tagged Values are predefined in UML but equally to Stereotypes they can be user defined. Tagged values can be defined for existing model elements, or for individual stereotypes, so that everything with that stereotype has that tagged value. It is important to mention here that a tagged value is not equal to a class attribute. Instead, a tagged value can be regarded as being a metadata, since its value applies to the element itself and not to its instances.

Graphically, a tagged value is rendered as a string enclosed by brackets, which is placed below the name of another model element. The string consists of a name (the tag), a separator (the symbol =), and a value (of the tag), {Name = Value}. For example all the finder methods in StocksDB is tagged with {@andromeda.service.standardfinder=true}. In the custom (velocity) templates, this tagged value is looked upon, and based upon the Boolean value, an implementation for the finder method is provided.

Another example worth mentioning here is the {@andromda.service.standardoperations=true}. If this tagged value is "true", then the implementation code for the 'CRUD' operations (Create, Read, Update, and Delete) for the entity beans are automatically generated by the AndroMDA.

Constraints

A Constraint is a condition or restriction attached to a Model Element. Some Constraints are predefined in UML others may be user-defined. Constraints are expressed as text within braces ({ }). No constraints are used in StocksDB.

Design Patterns

When designing and building different applications, we continually come across the same or very similar problem domains. This leads to find a new solution for the similar problem each time. To save time and effort, it would be ideal if there was a repository which captured such common problem domains and proven solutions.

In the simplest term, such a common solution is a **design pattern**. The repository or place of reference that contains these patterns is a **design pattern catalog**. A design pattern describes a proven solution, from experienced hands, for a recurring design problem.

There are many design patterns but the following discussion will be limited to the patterns which are adapted in StocksDB.

Session Facade

Session Facade encapsulates business-tier components and exposes a coarse-grained service to remote clients. Clients access a Session Facade instead of accessing business components directly. A Session Facade is implemented as a session enterprise bean. It provides clients with a single interface for the functionality of an application or application subset. It also decouples lower-level business components from one another, making designs more flexible and comprehensible.

Transfer Object/Value Object

The Data Transfer Pattern is one such pattern. Clients usually require more than one value from an enterprise bean. To reduce the number of remote calls and to avoid the associated overhead, it is best to use Transfer Objects to transport the data from the enterprise bean to its client. The Transfer Object encapsulates the business data. A single method call is used to send and retrieve the Transfer Object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the Transfer Object, populate it with its attribute values, and pass it by value to the client.

The value object contains a snapshot of the data (a row) in the database, but it won't dynamically reflect changes to that data. The custom template generates the value objects for StocksDB application. When designing the UML model, for all the entity classes an associated Value Object class is modeled tagged with the stereotype <<ValueObject>>. AndroMDA then generate a Value Object class for that entity with accessor ("getter") and mutator ("setter") methods for all the attributes of that entity.

Business Delegate

In distributed applications, lookup and exception handling for remote business components can be complex. When applications use business components directly, application code must change to reflect changes in business component APIs. These problems can be solved by introducing an

intermediate class called a ***business delegate***, which decouples business components from the code that uses them. The Business Delegate pattern manages the complexity of distributed component lookup and exception handling, and may adapt the business component interface to a simpler interface for use by views

With the above overview, all the technologies and methods which are used to generate the EJBs are covered in brief. But, till now only half the river is crossed. The next half will be to see how efficiently the EJBs can be integrated with the Web interface. This is done with Struts framework.

3.6 Web Applications with Struts

A Web application is defined as a program that resides on a Web server and produces static and dynamically created pages in a markup language (most commonly HTML) in response to a user's request. The user makes the request in a browser, usually by clicking a link on the Web page. Figure 3.10 shows a high-level view of Web architecture.

A Web container is a program that manages the components of a Web application, in particular JSP pages and Java Servlets. A Web container provides a number of services, such as

- **Security:** Restricted access to components, such as password protection
- **Concurrency:** The capability to process more than one action at a time
- **Life-cycle management:** The process of starting up and shutting down a Component.

Apache Tomcat is an example of a Web container (servlet engine) — an open-source implementation of the J2EE Java Servlet and JavaServer Pages (JSP) specifications. Within JBoss server, Tomcat is the Web container.

3.6.1 Struts

As Web applications grow in size, the size of development projects grows as well, and it becomes more and more critical to support modular application design and parallel development. The challenge lies in meeting a few unique requirements. The methodology for Web application development is still in its early stages, and new technologies continue to emerge. Struts framework is one method of Web application development.

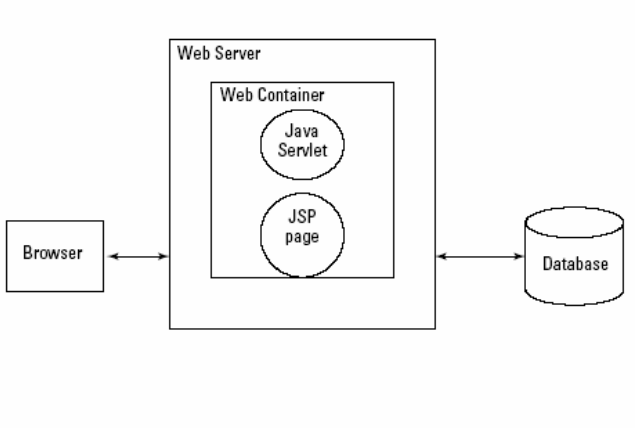


Figure 3.10 Web-architecture with its components residing in the Web Container is shown [48]

An application framework is a skeleton of an application that can be customized by the application developer [48]. Struts is an application framework that unifies the interaction of the various components of a J2EE Web application, namely Servlets, JSP pages, JavaBeans, and business logic, into one consistent whole. Struts provide this unification by implementing the Model-View-Controller (MVC) design pattern.

3.6.2 Model-View-Controller

The MVC architecture is a way of decomposing an application into three parts: the model, the view and the controller [13]. It was originally applied in the graphical user interaction model of input, processing and output.

- **Model**: The data and business logic
- **View:** The presentation
- **Controller:** The flow control

Each of these layers is loosely coupled to provide maximum flexibility with minimum effect on the other layers.

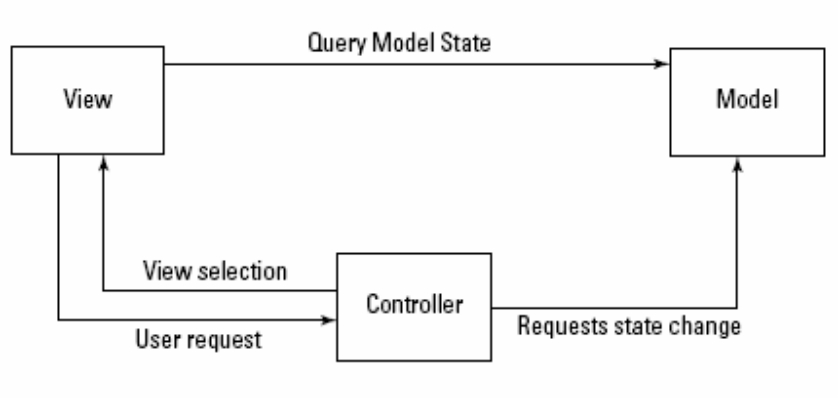


Figure 3.11 Relationship among three layers Model, View, and Controller is shown [48]

How Struts enforces the MVC pattern

The architecture of Struts provides a wonderful mechanism that, when followed, ensures that the MVC pattern remains intact.

The Struts Controller

The primary controller class is a Java Servlet called the ***ActionServlet***. This class handles all user requests for Struts-managed URLs. Using information in the configuration files, the ***ActionServlet*** class then gets the appropriate ***RequestProcessor*** class that collects the data that is part of the request and puts it into an ***ActionForm***, a Bean that contains the data sent from or to the user's form. The final step of the Controller is to delegate control to the specific handler of this request type. This handler is always a subclass of the ***Action*** class. Figure 3.12 shows how Struts uses the MVC pattern. The ***Action*** subclass is the workhorse of the Controller. It looks at the data in the user's request (now residing in an ***ActionForm***) and determines what action needs to be taken. It may call on the business logic of the Model to perform the action, or it may forward the request to some other View. The business logic may include interacting with a database or objects across the network or may simply involve extracting some data from an existing JavaBean. After the necessary action has been performed, the ***Action*** subclass then chooses the correct View to send back to the user. The View is determined by the current state of the Model's data (the model state) and the specifications you defined in the Struts configuration file.

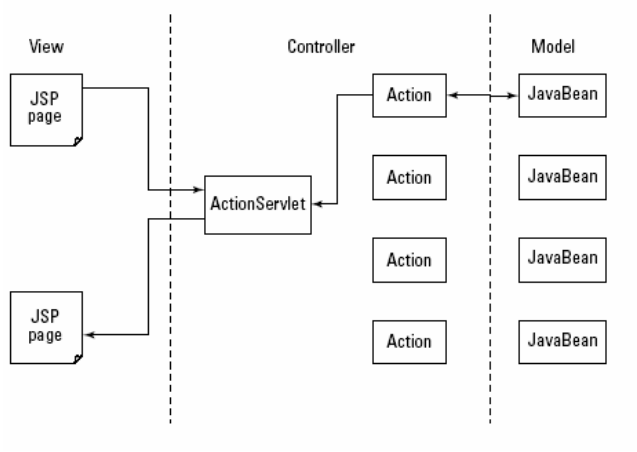


Figure 3.12 The Struts use of MVC pattern [48]

The Struts View

Struts do not provide, nor is it dependent on, a specific presentation technology. Many Struts applications use JSP (JavaServer Pages) along with the Struts tag library (Struts and Struts-EL), JSTL (JSP Standard Tag Library), and JSF (Java Server Faces).

The Struts Model

Nothing in Struts dictates how to construct the Model. However, the best practice is to encapsulate the business data and operations on that data into JavaBeans. The data and operations may reside in the same class or in different classes, depending on the application. The operations represent the business logic that your application is defining. Operations may be the rules that should operate on a particular business entity.

The ***Action*** subclass initiates any action required to handle a user's request, thereby creating the connection with the Model. When formulating a response, the Controller may pass some or all of the Model data to the View through the use of the ***ActionForm*** Bean. Although this Bean is a data container, it should not be considered part of the Model but rather just a transport mechanism between the Model and the View.

The Struts configuration file

The Struts configuration file performs an important role in structuring the Struts application. Although it is not really part of the Model, View, or Controller, it does affect the functioning of the three layers. The configuration file allows defining exactly which of Action subclasses should

be used under what circumstances and which ActionForm should be given to that Action subclass. So part of the Controller interaction is specified in the configuration file. In addition, when the Controller decides which View to return to the user, it chooses the particular View according to specifications in the configuration file. Thus the configuration file actually defines many of the connections between the MVC components.

Using the MVC pattern gives many advantages:

Greater flexibility: It's easy to add different View types (HTML, WML, XML) and interchange varying data stores of the Model because of the clear separation of layers in the pattern.

Best use of different skill sets: Designers can work on the View, programmers more familiar with data access can work on the Model, and others skilled in application development can work on the Controller. Differentiation of work is easier to accomplish because the layers are distinct. Collaboration is through clearly defined interfaces.

Ease of maintenance: The structure and flow of the application are clearly defined, making them easier to understand and modify. Parts are loosely coupled with each other.

Other technologies used in developing StocksDB are:

3.7 PERL

PERL (Practical Extraction and Reporting Language): Perl is an open source, stable, cross platform programming language. Perl has many features which are unequalled in any other mainstream programming language, primarily for text and file manipulation. Perl scripts are used in **StocksDB** to generate some Struts code.

3.8 Validation Framework

The Jakarta Commons Validator framework [42] is used for validating the user input. It is an open source project that was created by David Winterfeldt and is part of the Jakarta Commons subproject. The Commons project was created for the purpose of providing reusable components like the Validator. It offers many benefits. A few of the benefits include:

- A single definition of validation rules for an application.

- Validation rules are loosely coupled to the application.
- Server-side and client-side validation rules can be defined in one location.
- Configurations of new rules and/or changes to existing rules are made simpler.
- Supports Internationalization.
- Supports regular expressions.
- Can be used for both Web-based and standard Java applications.
- Promotes a declarative approach rather than a programmatic one.

More information about these components and a guide which explains in detail how to integrate the Validator framework into the application can be found in the following url: [5]

3.9 Display Tag

The display tag library [9] is an open source suite of custom tags that provide high level web presentation patterns which will work in an MVC model. The library provides a significant amount of functionality while still being simple and straight-forward to use.

3.10 JDOM

JDOM [20] is a library which provides Java-based solution for accessing, manipulating, and outputting XML data from Java code.

3.11 EFetch

Entrez Programming Utilities are tools that provide access to Entrez data outside of the regular web query interface and may be helpful for retrieving search results for future use in another environment [11]. EFetch, one of the utility tools, retrieves records in the requested format from a list of one or more unique identifiers

Next section gives an overview of RDBMS and MySQL.

3.12 Relational Database Management System

The relational database was born in 1970 when E.F. Codd, a researcher at IBM, wrote a paper outlining the process. He proposed a set of 12 rules for identifying relationships between pieces of data. Codd's rules formed the basis for the development of systems to manage data. Today, Relational Database Management Systems (RDBMS) are the result of Codd's vision.

A database management system (DBMS) is software that allows databases to be defined, constructed, and manipulated. Popular commercial DBMS sold as RDBMS include Oracle, Microsoft SQL Sever, Sybase SQL Server, IBM's DB2, and Microsoft Access. MySQL, PostgreSQL, and Firebird are free RDBMSs.

In order to "talk" to the database, some sort of software is needed. Whether it comes with the server or the code has to be written by ourselves, this software is essential for database communications. Although there are innumerable methods of retrieving and storing data, the following are the most common:

SQL (Structured Query Language) the most common data access method.

ODBC (Open Database Connectivity).

Native methods, which are rarely used.

3.12.1 MySQL

MySQL [27], a product of MySQL AB is a full fledged Relational Database Management System like Oracle or Microsoft SQL Server. It is an open source software application; its use within the terms of GPL (General Public License) is free.

MySQL offers several key advantages: Reliability and Performance, Ease of Use and Deployment, Freedom from Platform Lock-in, Cross-Platform Support etc.

Given below are different flavors of MySQL.

MySQL database server & standard clients:

- MySQL 4.0 -- Generally Available (GA) release (recommended)
- MySQL 4.1 -- Gamma release (use this for new development)

MySQL 4.1 is used for StocksDB. The mysql website quotes new special features of version 4.1. The feature which is of particular interest from StocksDB point of view is the support of subqueries and derived tables. A subquery is a `SELECT` statement nested within another statement. A derived table' (an unnamed view) is a subquery in the `FROM` clause of another statement. The support of subquery is essential for the implementation of search feature in StocksDB, and so MySQL 4.1 is sought.

Refer to [28] for other features of MySQL 4.1.

3.12.2 DeZign

"DeZign for Databases"[23] is a database development tool using an entity relationship diagram. It visually supports the lay out of the entities and relationships and automatically generates SQL schemas for most leading databases.

3.13 Integrated Development Environment (IDE)

The entire development was carried out in Borland's JBuilderX, Enterprise edition [19]. It is a cross-platform environment for building industrial-strength enterprise Java applications. The build system in JBuilder provides the flexibility necessary to support today's complex enterprise build processes. JBuilder provides full support for Apache™ **Ant 1.5** [2] and Ant-driven build processes. Apache Ant is a Java-based build tool. In theory, it is kind of like Make in C++.

3.14 User Management and Access Control

The User Management and Access Control, a software mechanism for flexible user access control was integrated in to StocksDB. Please refer the Master Thesis report '**Design and Development of a user management system for Molecular Biology Databases**, by Dieter Zeller [10] for further information.

With this, an introduction of the methods used in the development of **StocksDB** is completed.

4 Results

This chapter describes what has been achieved with the master thesis based on the Software Development Life Cycle (SDLC). First, it gives an overview of the SDLC, and later a walkthrough of the development cycle of StocksDB, with reference to the different phases of SDLC, is given.

4.1 SDLC WATERFALL

Small to medium database software projects are generally broken down into six stages:

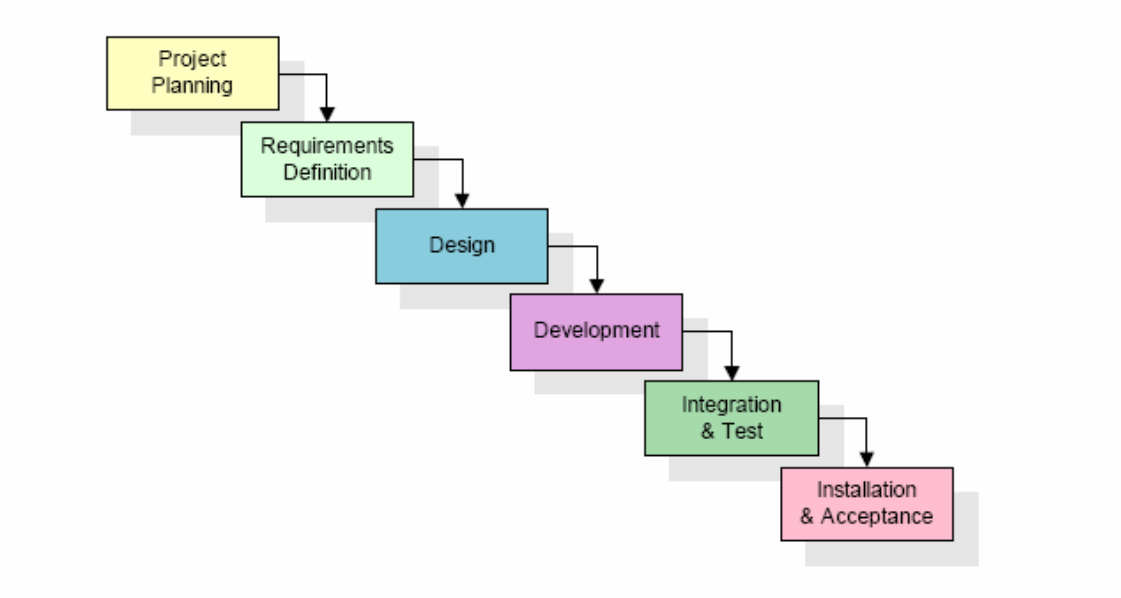


Figure 4.1 The six phases of Software Development Life Cycle is shown [33]

The relationship of each stage to the others can be roughly described as a waterfall, where the outputs from a specific stage serve as the initial inputs for the following stage. During each stage, additional information is gathered or developed, combined with the inputs, and used to produce the stage deliverables. It is important to note that the additional information is restricted in scope; "new ideas" that would take the project in directions not anticipated by the initial set of requirements are not incorporated into the project. Rather, ideas for new capabilities or features that are out-of-scope are preserved for later consideration. (Refer to the Outlook section is Discussion chapter)

4.1.1 PROTOTYPE APPROACH

Generally in any SDLC, the development team, to clarify requirements and/or design elements, may generate mockups and prototypes of screens, reports, and processes. Although some of the prototypes may appear to be very substantial, they are generally similar to a movie set: everything looks good from the front end but there's nothing in the back.

When a prototype is generated, the developer produces the minimum amount of code necessary to clarify the requirements or design elements under consideration. No effort is made to comply with coding standards, provide robust error management, or integrate with other database tables or modules.

The prototype approach was followed for the StocksDB too. Initially, after the requirement definition phase, rather than showing the user a paper sketch of the model or design, a prototype of the StocksDB was build with static html pages. All the interface screens and the view screens were designed in the first stage. The project flow or the response of the application to the actions of the user was simulated then with static html pages. This gave the user a better understanding of the future system and also gave me as a developer a rough picture of the logical flow of the application. This approach has enabled to get the maximum input from the end user in the initial stage itself. It also helps the user to get acquainted to the final system faster as he already had an overview of what is in store for him in the future.

A WALK-THROUGH THE PHASES OF STOCKSDB DEVELOPMENT

4.2 FIRST PHASE: PLANNING

The planning phase establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches. During the very first meeting an overview of the project was given by Gerhard. Then with subsequent meetings with Christine and Anne gave us a clear view of "**What to develop?**" and discussions with Gerhard helped to get an idea of "**How to develop?**" StocksDB.

The technologies that are used for developing the project is pre-defined as our institute already had developed a custom framework and standardized the development process. The technologies and tools are discussed in detail in the **Methods** chapter.

4.3 SECOND PHASE: REQUIREMENTS DEFINITION

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as critical inputs, outputs and reports. A user class hierarchy is developed and is associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.

The output of the requirements definition is the requirements analysis/definition documentation. (Refer Appendix A)

4.4 THIRD PHASE: DESIGN

The design phase takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that programmers may develop the software with minimal additional input.

For StocksDB, the design phase was carried out in two steps.

1. Designing the database schema or ER (Entity-Relationship) diagram.

2. Designing the UML model.

4.4.1 Designing the database schema/ER Diagram

As an overview about the database technologies and methods is already given in the second chapter, here only the key points which are crucial for a good database schema design is given.

1. Identify all of the **entities** in the model (persons, places and things about which the information is stored). For example Primers, Targets, Plasmids etc.,

2. For each entity, identify all of the **attributes** (fields of data stored for each entity). For example Primer Name, Primer Type, Primer Direction etc.,

3. For each entity, identify a key attribute (**primary key**). For example: primerId.

4. Identify all of the meaningful **relationships** between the entities. For example: Fridge-Rackshelf-Box.

5. Determine the **cardinality** of each relationship (one-to-one, one-to-many, many-to-many)

6. Consider the impact of **mandatory or optional requirement** for each of the relationships and try to understand what business rules are enforced depending on whether or not a relationship is mandatory or optional. (include zero or not)

7. If there are any many-to-many relationships in the model, **resolve the many-to-many relationships** by adding a new entity between the original entities having the many-to-many relationship and put the "many" side of the 2 resulting new relationships on the side of the new entity and the "one" side of the 2 resulting new relationships on each of the sides of the original entities.

8. **Duplicate** the primary key on the "one" side of each one-to-many relationship in the entity and put it in the entity with the "many" side of the relationship. The duplicated field is known as the "**foreign key**" on the "many" side of the relationship and the "primary key" on the "one" side of the one-to-many relationship. For all one-to-many relationships, the primary key on the "one" side must be duplicated in the entity on the "many" side. On the "many" side, it is known as a foreign key. NOTE: All one-to-many relationships are represented with a primarykey/foreign key pairing.

9. Remember, some of the original **entities** may not be needed and others that are not identified at first might be added later. Similarly, different new **relationships** may arise from the newly added entities.

10. Bear in mind that some of the original entities might be better represented as **attributes**.

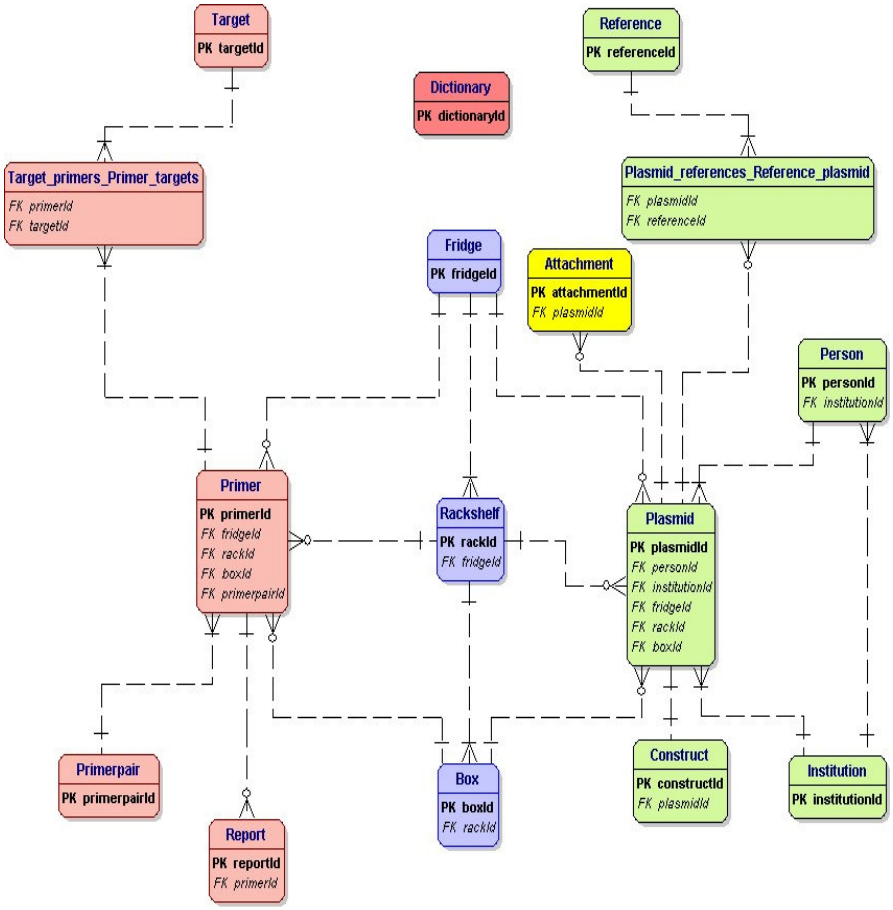


Figure 4.2 The Database Schema/ER Diagram of StocksDB. Location tables (blue), Primer tables (pink), Plasmid tables (green), Attachment table (yellow) and Dictionary table (dark pink). This diagram was created with DeZign [23]

A detailed explanation about the data stored in these tables and their relationships is given below.

Tables

1. **Location Tables** (*The blue ones*)

   Location tables refer to the exact physical location where the entities are stored. These tables store their respective container id, name, and a short description. For example: Box Table stores boxId, boxName and boxDescription. When the user is viewing the information about any entity, then apart from the specific properties of the entities, the location information is also displayed. Regarding their relationships,

   - one Fridge can have many Rack shelves – One-to-Many

- one Fridge can have many Primers/Plasmids – One-to-Many
- one Rackshelf can have many Boxes- One-to-Many
- one Rackshelf can have many Primers/Plasmids- One-to-Many
- one Box can have many Primers/Plasmids- One-to-Many

2. Primer Tables (*The pink ones***)**

2.1 Primer

Primer tables are the most complex tables because of their tight coupling amongst themselves. All primer specific data are stored in Primer table. Though there are separate interfaces to add cloning primers and qRT primes, both the primers are stored in one common table. Obviously, when a single primer is added, a single row is created and when a primer pair is added two rows are created in the table, one for the forward primer and the other for the reverse primer with two different unique id.

Both the cloning primer and the qRT primers have label information and furthermore, qRT primers have the system information and hybridization probe information. Apart from the primer specific data attributes, the two attributes primer type (cloning/qRT) and primer direction (forward/reverse) identifies or categorizes the primers. The attribute primer Tm is not entered by the user but is calculated from the primer sequence. The primary key field, the primerId is also the tubeId from the user's point of view. This Tube Id will be used to label the samples in the physical location to uniquely identify sample tubes.

2.2 Target

One primer can act on many targets. For example, there is a high probability that a 22 bp primer which can act on human insulin gene can also act on its homologues bovine insulin gene. On the other way around, a Target can be acted upon by one primer or many primers. So the relationship between primer and target is modeled as many-to-many. The relationship is resolved by a new intermediary table '**Target-primers-Primer-Targets'** which stores the primary keys of both the tables Primer and Target.

2.3 Primer pair

Some primer data are separated as primer pair attributes because these data will be available only when two primers act on a single target as a pair, one in forward direction and other in reverse direction. When two primers are added simultaneously, then the product size and the product's Tm is calculated and stored in primer pair table. When a single primer is added, then primer pair table will not get a row entry. One important point to note here is the relationship between the primer and primer pair table. Actually it should be modeled as many-to-many. But it has been modeled as many-to-one. The reason behind that is when the tables are related as one-to-many, as explained earlier, the foreign key is stored in the 'many' side of the relationship. As the forward primer Id and reverse primer Id is already stored as an attribute in the primer pair table, the relation on the primer pair side as 'one'. This even simplifies certain method implementations as what we get out of many-to-many relation is a 'collection' of value objects, but with a one-to-many or one-to-one we get a single value object which is easier to handle than a collection.

2.4 Report

The user can add a report about a primer after his experiment. As many users will be using the same primer and may submit individual reports about the primer, the relationship is modeled one-to-many ie one primer can have many reports.

3. Plasmid Tables (*The Green Ones*)

3.1 Plasmid

Plasmid table stores information about the plasmid like *E.coli* strain, antibiotic resistance, size of the plasmid in base pairs etc., apart from the location details. The tables construct, institution, person and reference are identified as sub entities and therefore separate tables are created for them and they are related to the main entity Plasmid by one-to-many relationship. Similar to the Primer table, the plasmidId will be used as a tubeId to identify the sample tubes.

3.2 Construct

Artificial plasmids are usually constructed to cater specific needs of genetic engineers. For example: pBR322. It contains the ampicillin resistance (Ap^R) gene of the natural

plasmid RSF 2124 and the tetracycline resistance gene (Tc^R) of the natural plasmid pSC101Construct. So we store the information associated with the plasmid construct, namely the basic plasmid name and the insert name in construct table, which is related to the plasmid with one-to-one relationship.

3.3/3.4 Person and Institution

The two tables Persons and Institution are related to the MTA, which stands for **Material Transfer Agreement**. It is a customary procedure to sign a simple agreement between the two institutions, one being the recipient of the biological material and the other being the supplier. The MTA document defines the terms and conditions for transferring the material.

Apart from the MTA, the contact details of the person who have supplied the material, is also of interest for future reference. So the person info is stored in the person table and the institution info in the institution table. This separation is necessary because it is quite common for persons to move from one institution to the other. The institution and person table are related by one-to-many relationship. The institution and plasmid, and person and plasmid are related by one-to-many relationship as one institute/person can supply many plasmids.

3.5 Reference

A plasmid can have many references associated with it. And sometimes one reference can refer to many plasmids. So obviously the relationship between them is to be modeled as many-to-many. This relationship was resolved with an associative table **Plasmid_references_Reference_plasmids**.

4. Attachment Table (*The Yellow One*)

The attachment table is common to all entities. It will store the information associated with the attached file like the file name, the type of attachment-- it can be an insert sequence, reference, MTA, or plasmid MAP. The MIME type of the file is also stored as it will be used to display the file in a fool-proof manner. Right now, with the current version of StocksDB, only the entity plasmid has attachments and primer doesn't have any. So the plasmid and attachment tables are related with one-to-many relationship. In

future, if there are other entities with attachments, then that entity too can be related to the attachment table without any problem.

5. **Dictionary Table (***The Dark Pink One***)**

The web interface has several fields which are of select type (drop-down menus/combo-boxes). One way to display the options is to hard code them in the JSP page itself. If it is done that way, then the options become very rigid and fixed. The Manufacturer field is taken as an example to make the point more clear. Take for granted that two manufacturers 'Amersham' and 'Invitrogen' are hard coded in the JSP page. If in future, the institute has decided to buy the stocks from a third manufacturer, then the JSP page has to be edited. This is not efficient and not advisable. So all the attributes or properties which are rather fixed but have the potential to change in future are identified, and stored in dictionary table. Every time when the web interface is displayed to the user, the contents of select field is fetched from this dictionary table. If the user wants to add a new option, for example a new Manufacturer, then the same can be done through a separate interface. In this way, the application is more flexible and there is no need to change the application's source code in the future.

In case of primer, the labels, tags, manufacturer, and system info are moved to dictionary. And in plasmid, the *E.coli* strain and the antibiotic resistance fall into dictionary. The other select options of location fields are not fetched from dictionary but rather directly from the location tables, fridge, rackshelf and box as mentioned earlier.

The dictionary table has three attributes namely:

- **dicId:** The primary key field, dictionary Id.
- **dicType:** All the entries in the table are grouped based on their type and an unique code is assigned for each group. For example: PRSYS for System, PLECS for Plasmid *E.coli* strain. And in the application this field is used to fetch all the values in that group.
- **dicValue:** This field stores the actual value of the entry. For example: An entry with a dicType PRSYS can have dicValue LUX or TaqMan or SybrGreen.
- **dicDescription:** A short description about the entity.

dicId	dicType	dicValue	dicDescription
1	PRSYS	LUX	
2	PRSYS	TaqMan	
3	PRSYS	SybrGreen	
4	PRALB	Biotin	
5	PRFLB	FAM	
6	PRFLB	ROX	
7	PRFLB	Cy3	
8	PRFLB	Cy5	
9	PRAML	c6	
10	PRAML	c7	
11	PRTAG	His	
12	PRMAN	Amersham	
13	PRMAN	Invitrogen	

Table 4.1 The Dictionary table with the predefined values for the entity primer is shown

With the completed database schema, the next section covers in detail about designing the UML model.

4.4.2 Designing the UML model

The UML model is designed by mapping the ER Diagram to the UML. As a brief overview about UML and the UML design tool **MagicDraw** is already given in the Methods chapter, this section addresses only some practical points in creating the UML model with MagicDraw.

1. Sketch out the ER diagram.

2. Be sure of the primary key attribute, all other attributes and there types.

3. Follow a convention in naming the attributes. Keep in mind that if an attribute called primerName is created, then in the table the column name will be PRIMER_NAME.

4. Also don't end table names with plurality ('s'). For example if the table names are PRIMER and TARGET, and suppose they have one to many relationship, then in order to access TARGETS from PRIMER table, a'getTargets()' will be generated. But if the table is named as PRIMER and TARGETS, then a method 'getTargetss' with 'ss' will be generated which is not so aesthetic.

5. Sketch out the package hierarchy. This will be more helpful to build the tree with MagicDraw. Building a tree is most crucial, as finally the XML file is build out of the

containment tree. So build all the packages and the classes in the containment tree of project pane.

6. Use stereotype <<Entity>> for all tables and for all session beans use <<Service>>. Apart from this, for StocksDB other custom stereotypes like <<ServiceFactory>>, <<GlobalConstants>>, <<BusinessDelegate>>, <<ServiceBusinessInterface>> and <<ValueObject>> are used.

7. There are stereotypes even for attributes and operations (methods). Use stereotype <<PrimaryKey>> for primary key field.

8. For finder methods use stereotypes <<FinderMethod>> and tag it with

   {@andromda.service.standardfinder=true}

9. Tag all the entity classes with {@andromda.service.standardoperations=true} to get the "CRUD" method implementations.

10. If the cascade delete implementation for the tables is to be implemented, then tag the relationship with {@andromda.cascade-delete=true}.

11. Establish the relationships between the entities. Pay particular attention to the cardinality.

12. Establish the dependency between the entity and the service classes with stereotype <<EntityRef>> and establish dependency between entity/service and the exception class with stereotype <<ExceptionRef>>

13. Though the entire model can be created in one view, it is better to separate the views into Domain model, Value Objects, and Service Dependencies etc. It is necessary to understand that though multiple views are created there is only one tree. For example, the Primer entity can be in all the five views, but the containment tree will have only one Primer entity.

A successful code generation solely depends on the model. So, considerable time should be spent on mapping the ER diagram to the UML model giving due attention to all the above points. A more detailed explanation about the UML models, stereotypes and the value objects is given in Thomas Truskaller's Master Thesis report [8] and an excellent tutorial covering the above topics in detail is available too.

The UML models designed for the StocksDB application are shown in the following pages.

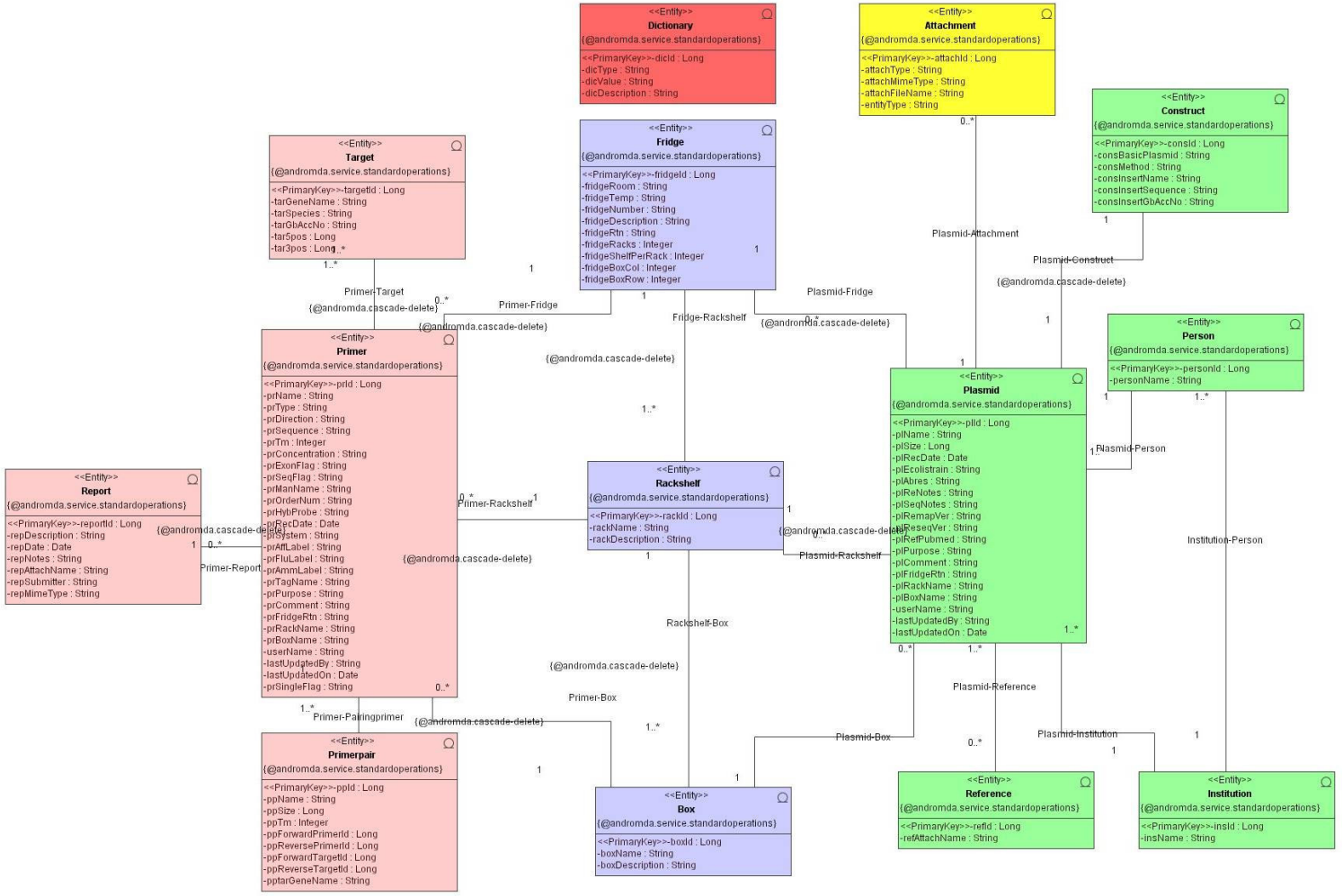


Figure 4.3 The Domain Model. The entities as classes with attributes and relationships are shown (Location tables-blue; Primer tables-pink; Plasmid tables-green; Dictionary table-dark pink; Attachment table-yellow)

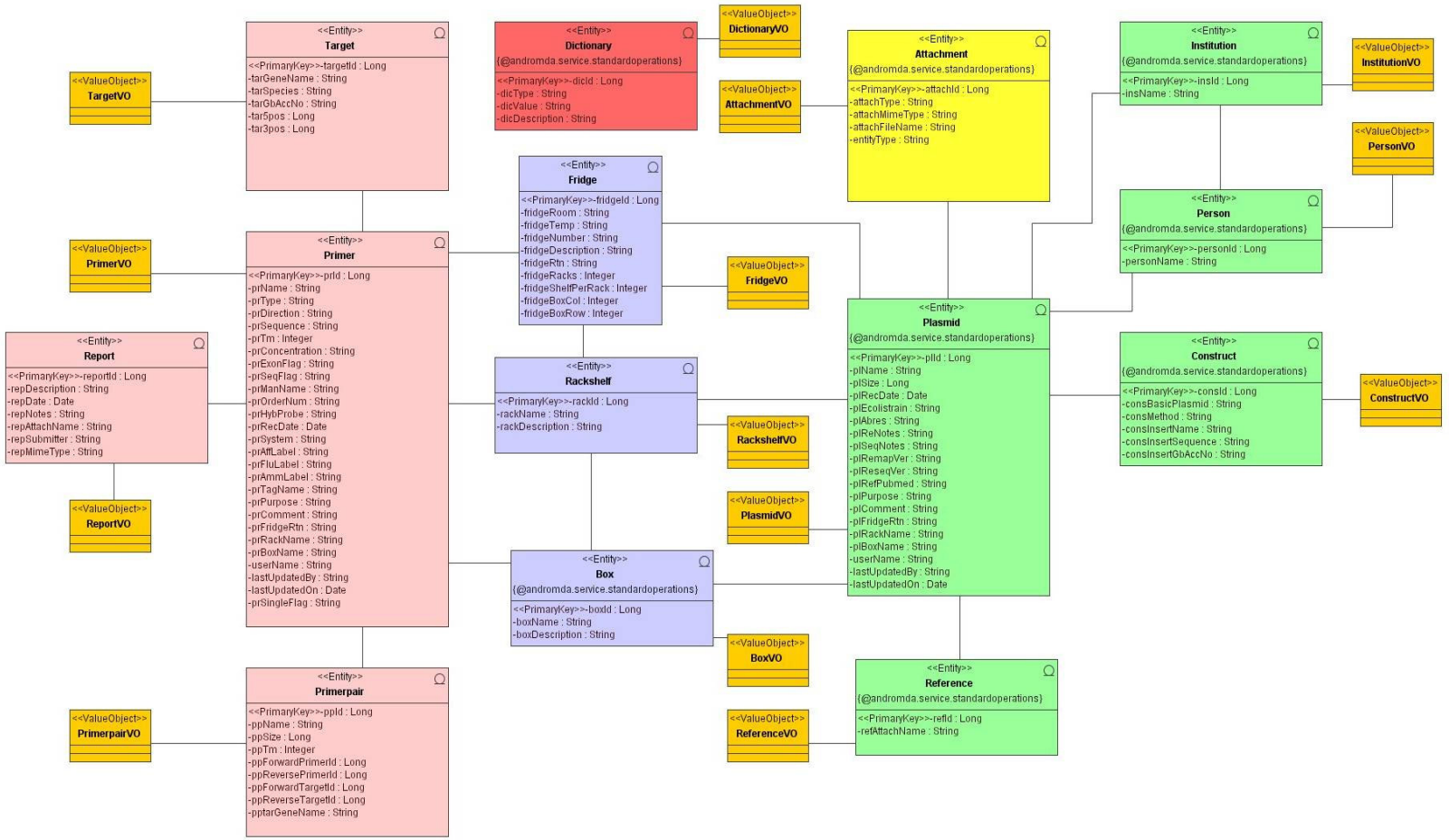


Figure 4.4 The Value Object Model. The entities as classes with attributes, relationships, and value objects are shown (Location tables-blue; Primer tables-pink; Plasmid tables-green; Dictionary table-dark pink; Attachment table-yellow)

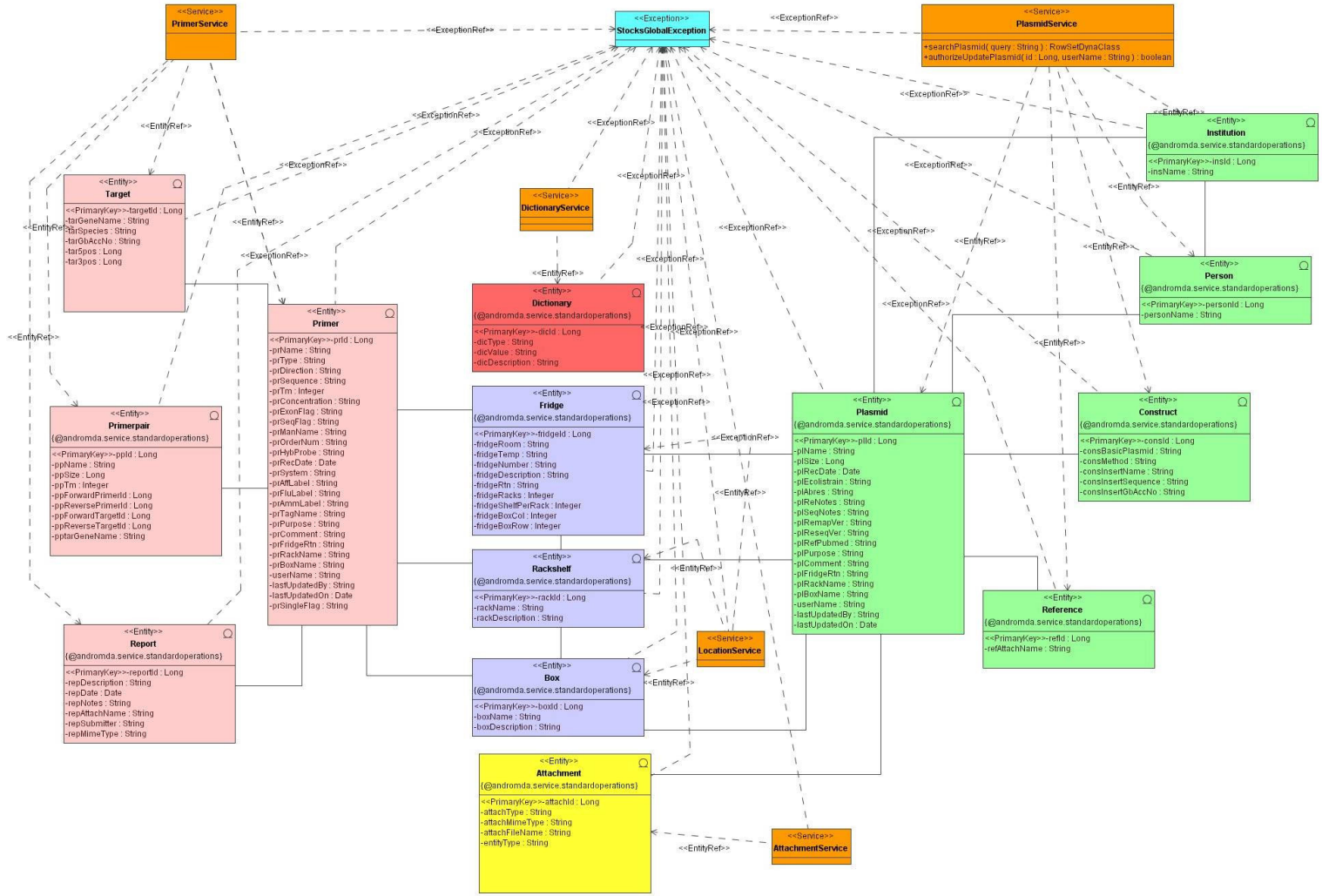


Figure 4.5 The Service Dependencies. The entities as classes with attributes and relationships are shown (Location tables-blue; Primer tables-pink; Plasmid tables-green; Dictionary table-dark pink; Attachment table-yellow)

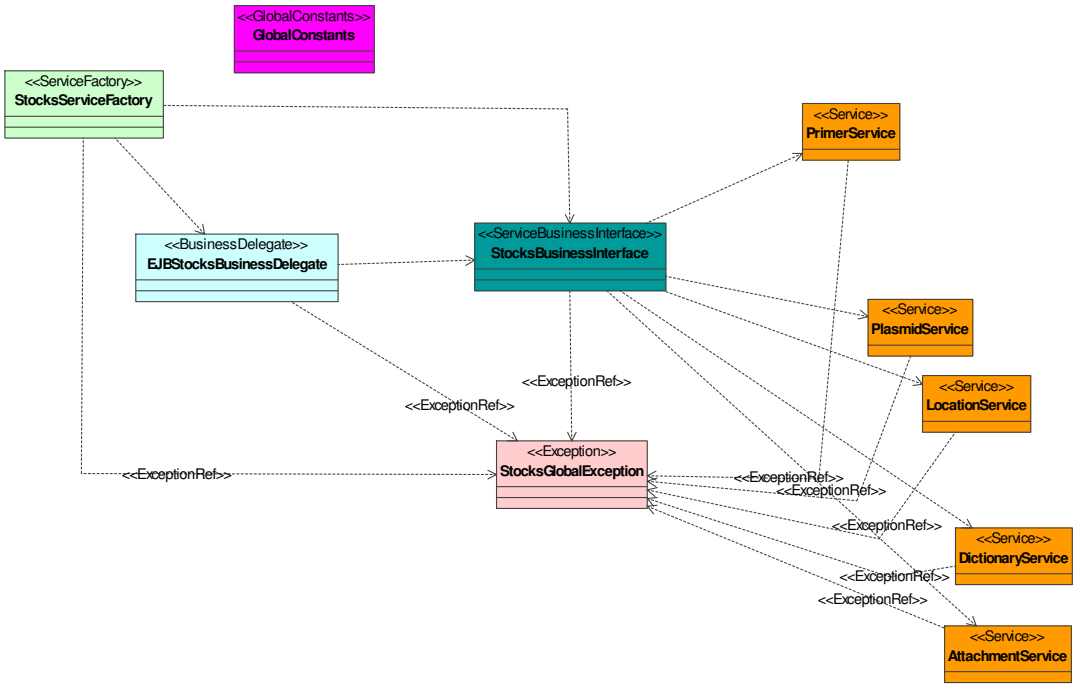


Figure 4.6 Delegate and Locator Patterns

4.5 FOURTH PHASE: DEVELOPMENT

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, data management forms, data viewing formats, and specialized procedures and functions. The outputs of the development stage include a fully functional application that satisfies the requirements and design elements previously documented, a test plan that describes the test cases to be used to validate the correctness and completeness of the application.

The development phase is divided into two stages.

1. Generation of session beans and entity beans from the UML model.

2. Integration of the beans with web interface developed with Struts framework.

4.5.1 Generation of session beans and entity beans from the UML model

An EJB application consists of a number of enterprise beans that handles the business logic and data. Each enterprise bean consists of a number of components that is either created by the developer or by tools provided with the EJB container. The EJB components based on EJB 2.0 specification can be viewed in Figure 4.7 and is described further below.

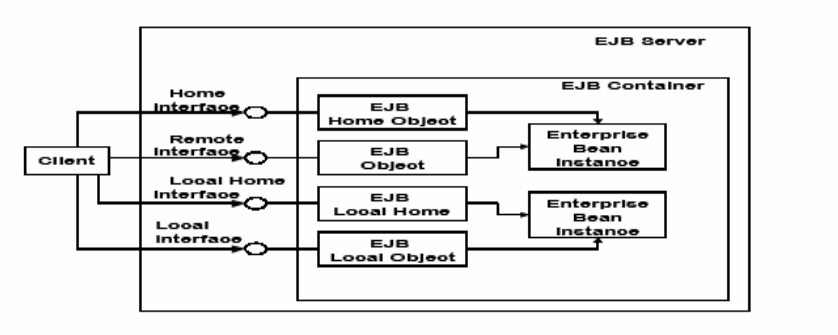


Figure 4.7 Shows how the client access the beans through a set of interfaces [3]

Bean Class (Enterprise Bean Instance)

The Enterprise bean class implements the business logic for the bean. This is where most of the coding is done in an EJB application. The bean class is not accessed directly by the client; instead the remote or local interfaces of the bean are used to call these methods.

Which files are generated?

AndroMDA with its current template set follows a particular pattern to generate the classes which are necessary for a complete Enterprise Java Bean. Currently, there are templates for session beans and for entity beans. The entity bean class is taken as example. When a class Primer is modeled and given it the stereotype `<<Entity>>`, the following classes generated by AndroMDA using the EJB cartridge

PrimerBean-the bean class

AndroMDA generates the following items for this class: [1]

- import statements for referenced classes that are contained in another java package

- a class header with all the necessary definitions, so that XDoclet can generate all the interfaces, value object class and deployment descriptor. This includes ejbref's to other beans and the EJB-QL statements for the finder methods.
- for each attribute: one abstract getter and setter with @tags to generate it as a CMP 2.0 persistent field
- for each method: one abstract method with the same signature, to be implemented in the implementation class, with a throws-clause that lists the exception class for this component
- for each persistent association to other entity beans: one abstract getter and setter with @tags to generate it as a CMR relationship
- one pair of ejbCreate()/ejbPostCreate() methods with all attributes necessary to initialize the bean (except the primary key)
- one pair of ejbCreate()/ejbPostCreate() methods that accept the value object as an argument
- for each dependency that is marked with an <<EntityRef>> or <<ServiceRef>> stereotype: one lookup method for the home interface of the other class.

PrimerBeanImpl- the implementation class

AndroMDA generates the following items for this class:

- a simple header without @tags, letting this class derive from PrimerBean
- for each method: one concrete method that can be implemented here
- the other EJB-related methods that are implemented here: setEntityContext, unsetEntityContext, ejbRemove, ejbLoad, ejbStore, ejbPassivate, ejbActivate.

PrimerBeanCMP – CMP support class

Normally, this class would be unnecessary. However, JBoss does not handle derived classes 100% correctly and complains when the leaf class does not contain the EJB-related methods mentioned above AND all the abstract getters/setters. So this class collects the items once more.

All the above also applies to session bean classes with the following exceptions:

Stateless session beans do not have attributes.

Stateless session beans do not need CMP support classes.

So, for the entity bean Primer, the following classes generated by AndroMDA tasks.

- Primer.java
- PrimerBean.java
- PrimerBeanCMP.java
- PrimerLocalHome.java

The implementation class PrimerBeanImpl.java is generated only once. This is the only class were the method implementations are coded.

It is worth to note here that no business method/logic is implemented in the entity bean class. All the business implementations are done in the Session bean implementation class. The primary reason for this is to have a clear separation between the data layer and the business logic layer. Moreover, by moving the business logic implementation to the session bean class, one session bean can be made to service many entity beans.

For session bean PrimerService, which is a bean serving the entity bean Primer; the following classes are generated by AndroMDA tasks

- PrimerService.java: Remote interface for PrimerService
- PrimerServiceBean.java: The session bean class
- PrimerServiceHome.java: Home interface for PrimerService
- PrimerServiceLocal.java: Local interface for PrimerService
- PrimerServiceLocalHome.java: Local Home interface for PrimerService
- PrimerServiceUtil.java: Utility class for PrimerService containing the lookup methods. Example getHome() and getLocalHome().
- PrimerServiceBeanImpl.java: The manual implementation class

All the business logic for the entity primer is implemented in this class. Similar to this there are session beans to service other entities: Plasmid-PlasmidService, Location-LocationService, Attachment-AttachmentService, and Dictionary-DictionaryService.

The Business Methods

A brief description of some important business methods of StocksDB are given below.

PrimerService: Services Primer tables (Primer, Target, Primerpair, and Report table)

1. PrimerServiceBeanImpl

`calculateTm()`

This method calculates the melting temperature (Tm) of the primer sequence and the product sequence. The melting temperature is the temperature point at which 50% of the DNA has been separated into single strands. The Tm is calculated as follows:

Tm = 4*(G+C)+2*(A+T) [49]

`fetchSequence()`

This is one of the crucial methods of StocksDB as there are many methods dependent on the successful execution of this method. The goal of this method is to fetch a nucleotide sequence from the NCBI, GenBank database given an Accession Number or the GenBank Identifier (gi Number).

Initially the SeqHound API was used for fetching the sequence. SeqHound is a database of biological sequences and structures. SeqHound also stores additional information related to each of these sequences. SeqHound can be freely accessed through the website [34], or through a remote API or by installing SeqHound on our own server. But SeqHound has some limitations. The sequence is not fetched by making a direct remote connection to the GenBank server but rather fetched from the SeqHound server. Moreover, it does not update its database every day. So the most recently submitted sequences cannot be accessed. To overcome these shortcomings, the EFetch [11] tool from Entrez Programming Utilities is used for this purpose.

The sequence and the species information are retrieved from the returned XML file using JDOM. This method returns a String array with two elements namely, the sequence and the species.

`formatSequence()`

This method reformats the XML file returned by the EFetch to resolve some problems with the header line which contains the document type definitions. This method returns a String (the XML file is returned as String object)

`calculateProductSize()`

If the primers are added as Primer Pairs, then the product size is calculated from the 5' and 3' positions where the primers bind the target. The Tm is then calculated for the product sequence and all these information are stored in Primer pair table. This method returns a String, the product sequence.

The method is described with the following illustration. Normally the DNA sequence is written as given below:

5′-TTCTGCGGAGCCTGCGGGACGGCGGCAGCGGCGCGAGAGGCGGCC-3′

3′-AAGACGCCTCGGACGCCCTGCCGCCGTCGCCGCGCTCTCCGCCGGG-5′

A pair of primers, one primer acting in the forward direction (green) and one acting in reverse direction (red), is shown below

5′-TTCTGCGGAGCCTGCGGGACGGCGGCAGCGGCGCGAGAGGCGGCC-3′

3′-AAGACGCCTCGGACGCCCTGCCGCCGTCGCCGCGCTCTCCGCCGG-5′

For example, consider that the first nucleotide of the forward primer is at 3rd base position over the target sequence and the first nucleotide of the reverse primer is at 39th base position.

The length of the amplicon (the amplified product) is then from base 3 to base 39. If we include both the starting bases, the final product length is 37 base pairs.

`searchPrimer()`

This will be discussed in more detail in the special features of StocksDB section.

2. LocationServiceBeanImpl

`generateVirtualFridge()`

Virtual Fridge will be discussed in the special features of StocksDB at the end of this chapter.

With the successful implementation of the bean part, all necessary components are now there to deploy the bean. The advantage of the J2EE architecture should be appreciated here. If an application developer is working on the EJB components his work is completed after he has successfully deployed the beans. A web developer can also simultaneously work on the interface when the beans are being developed by the application developer. This is possible only because of the clear separation of all the layers. In a single developer scenario, this could be done only one after the other.

The next step is to integrate the Enterprise Java Beans and the web interface.

4.5.2 Integrating the EJBs and the Web interface

The MVC architecture and Struts framework has been described in great detail in the Methods chapter. As mentioned earlier, the data and the operations on it falls into the Model layer. The previous section covered how we accomplished that task with the help of AndroMDA. The following section is restricted to the View and the Controller layers.

The interface

In the design phase of StocksDB, it was mentioned that a prototype of the web interface with static html pages was build on the basis of the user requirement. Instead of creating all the JSP pages once again, it was decided to use the same html pages to save time. So the next task was to convert the static html pages with html tags to JSP pages with struts tags. PERL comes handy to do this.

Short PERL scripts were written to convert the html file with html tags to JSP file with the struts tags. Though considerable time has to be spent on coding the PERL scripts, compared to coding all the web interface pages once again with struts tags manually, it is very minimal.

The form bean definitions in the struts-config file are also generated by PERL scripts. As the types of the beans were defined as 'DynaValidatorActionForm', there is no need to code a separate ActionForm class manually with all the getter and setter methods. To generate beans dynamically, at first all the properties and their types should be mentioned in the form-bean definition tag of the Struts-config file. It should be noted here that each JSP page will have an associated form-bean definition. The PERL script can generate this in no time. The input is a JSP file and the output is a form-bean definition tag for that particular form with all the properties

and type set. PERL scripts are also used to generate the validation.xml [ref validation framework] file from the JSP file.

It was mentioned earlier in the Methods chapter that 'The **Action** subclass is the workhorse of the Controller'. So now the action classes have to be coded to do all the magic. A custom Action class has to be written inheriting the LookUpDispatchAction, which is an abstract Action that dispatches to the subclass mapped execute method. This is useful in cases where an HTML form has multiple submit buttons. The button name is specified by the parameter property of the corresponding ActionMapping. To configure the use of this action in the struts-config.xml file, an entry has to be created like this:

```
<action path="/test"
type="org.example.MyAction"
name="MyForm"
scope="request"
input="/test.jsp"
parameter="method"/>
```

The value of the request parameter named "method" will then be used to locate the corresponding key in ApplicationResources. For example, the ApplicationResources.properties might have the following:

```
button.add=Add Record
button.delete=Delete Record
```

And the JSP file would have the following format for submit buttons:

```
<html:form action="/test">
<html:submit property="method">
<bean:message key="button.add"/>
</html:submit>
<html:submit property="method">
<bean:message key="button.delete"/>
```

```
</html:submit>
</html:form>
```

The custom **Action** subclass must implement both getKeyMethodMap and the methods defined in the map. An example of such implementations is:

```
protected Map getKeyMethodMap() {
      Map map = new HashMap();
      map.put("button.add", "add");
      map.put("button.delete", "delete");
      return map;
}
public ActionForward add(ActionMapping mapping,
                         ActionForm form,
                         HttpServletRequest request,
                         HttpServletResponse response)
throws IOException, ServletException {
// do add
      return mapping.findForward("success");
}
public ActionForward delete(ActionMapping mapping,
                            ActionForm form,
                            HttpServletRequest request,
                            HttpServletResponse response)
throws IOException, ServletException {
// do delete
```

```
            return mapping.findForward("success");
        }
```

In this way, in a single action class multiple methods can be implemented.

StocksDB has the following Action classes. The name of the action classes is suggestive of what it does, but if still interested to know more about what these Action classes do and what are the methods in these Action classes, please refer to the StocksDB documentation.

1. CloningPrimerAction
2. CloningPrimerPairAction
3. QrtPrimerAction
4. QrtPrimerPairAction
5. PlasmidAction
6. DetailPrimer
7. DetailPlasmid
8. RudPrimer (Read,Update,Delete)
9. RudPlasmid
10. RudFridge
11. ViewAction
12. LocationAction
13. LabelAction
14. SearchAction

Still, a method worth explaining here is the addPrimerPair() method in the DetailPrimer Action class.

```
addPrimerPair()
```

During the initial phases of prototype development, the application was designed to add primers in single. All the primers have to be added individually, even though they are pairs. But this design gives additional overhead to the user as the user is forced to repeat certain entries twice if the primers were ordered in pairs. For example, the location details, manufacturer details, date etc. With subsequent discussions, it was found out that rarely the primers are ordered in single. So, it was decided to have a common interface for paired primers. This considerably saves the user's time and computational time. Moreover, calculating product information is straightforward as both the primer information is there beforehand. But there is also a possibility for a primer to be ordered as singles too. This happens if one of the primer pair is not performing as expected and the user proceeds to order only that primer. So, the initial interface wherein a single primer can be added was retained. Now the application is flexible enough to input the primer data either in single or in pairs.

Now, the next challenge is to pair the single primers. The method should help the user to pair the primers in an easy way. When the action to add primer pair is triggered, the method first picks up all the targets over which the primer acts. As a next step, all the primers which are acting on these targets in opposite direction are picked up. Finally, all these targets and primers are displayed to the user as drop down menus. The application then dynamically displays the primers depending on the target selected. The user can then select any combination of Target and primer to form a primer pair.

4.5.3 Special features of the web application

1. Dynamic/Dependent Drop-Down Combo Box

Drop-Down Combo Boxes are used for letting the user select one item out of many. In Struts this is accomplished using the Select, Option, and Options tags. The idea in a combo box is for the user to select one value out of many, and for struts to assign the selected value into a variable in the form bean. The initial value of this variable is used to pre-select one of the options. Select can be used with:

- A collection of strings, such as an arraylist.
- A collection of beans, each of which has string fields.
- Hard coded values.

- A combination of the above.

In many situations, their might be two combo boxes, with the contents of the second dependent on the value selected in the first. That is, each item in the first combo box is associated with a list. To handle this, it is necessary that the first combo box do a submit when it changes, so that there is a chance to update the second. It is done with:

```
<html:select property="prFridge"

Onchange ='document.forms[0].submit()'>

<html:options collection="fridgeCollection"

property="fridgeId" labelProperty="fridgeRoom"/>

</html:select>
```

In this example, the property "fridgeId" of the beans in fridgelist will be assigned to the property "prFridge" of the form bean and the "labelProperty" property is the name to display. During the first time entry into the action class, the prFridge string property of the form bean will be null or blank. In subsequent submissions the prFridge property of the form bean will contain a valid fridgeId, so that by looking up the list associated with that value, the second drop-down combo box can be populated properly.

StocksDB has many associated combo-boxes. This feature helps to reduce the list of options and also prevent the user from choosing wrong options. Two such examples are given below.

Fridge-Rackshelf-Box

For example, consider that the institute has five fridges and each fridge with 20 racks and each rack with 12 boxes, then, imagine a situation where the user has to choose his box from a huge list of 1200 boxes. To make the user's life simpler, the interface was designed in such a way that when the user choose a particular fridge, then only the racks available in that fridge are displayed in the rackshelf options, and subsequently when he choose a rack, the boxes available in that rack alone are displayed. Now the user sees only 12 boxes instead of 1200 boxes. This saves the user's time drastically.

System-Labels

The next example is from the qRT interface.

Figure 4.8 Before the System is selected, all the labels are 'Undefined'

Figure 4.9 After the System is chosen (eg: LUX) for the forward primer, the System of reverse primer is also set to 'LUX ' system and the label values are automatically set depending on the System property

The qRT primers can belong to a list of Systems like LUX, TaqMan, SybrGreen etc., and each system can have specific labels like Fluorescent labels, Amino Modifying labels or Affinity label. Not all the labels are applicable for all the Systems. If this is designed in a usual way, then there is a high chance that the user will choose the wrong combination of the systems and the labels. To avoid this, the System and label fields are made as dependent fields. Also, when adding two qRT primers as a pair, if the user selects a system for the forward primer, then the system of the reverse primer is also set to the same system, as there is no way that a primer pair can be formed with two different systems.

2. Validation

Every application has a responsibility to ensure that only valid data is inserted into its repository. Attempts to insert or update data that do not meet the criteria should be detected as soon as possible and rejected. This detection usually occurs in several places throughout an application; the presentation tier might perform some level of validation, the business objects typically have business-level validation rules, and the DBMS as well usually does its own validation.

The Benefits of Using the Validator

- The Validator framework [42] which is used for StocksDB offers several benefits over the more conventional method of defining validation rules within the source code of an application.

The most important characteristic of the Validator is its inherent support for pluggability. The Validator allows us to define our own validation routines and easily plug them into the existing framework. There are several components that make up the Validator framework.

- Validators
- Configuration Files
- Resource Bundle
- JSP Custom Tags
- Validator Form Classes

When using the Validator framework with Struts, there are two configuration files used. One is called validator- rules.xml and the other is validation.xml. The validator-rules.xml file acts as a template, defining all of the possible Validators that are available to an application. This file need not be coded. It is available with the struts binary download and can be directly used, unless there is a need to define our own validator rules.

The validation.xml file is where the individual Validators defined in the validator-rules.xml are coupled to components within the application.

StocksDB has implemented both content and type validations. The example mentioned below says that the property named 'prOrderNum' is a mandatory property and so it is required and it should be filled. And in case the field is not filled, then the message value associated with the key "label.orderNumber" will be displayed. The key-value pair should be defined in the resource bundle.

```
<field property="prOrderNum" depends="required">

<arg0 key="label.orderNumber"/>

</field>
```

The second example says that the field is a mandatory field and also a content check has to be made. The 'mask' is a variable name which takes java regular expression as its value. The data entered by the user should satisfy the regular expression rule. The regular expression syntax in the example means that the value can contain only the characters a,t,c,n,A,T,C,G,N from the beginning to the end

```
<field property="prSequence" depends="required,mask">

<msg name="mask" key="errors.sequence.mask" />

<arg0 key="label.sequence"/>

<var>

<var-name>mask</var-name>

<var-value>^[atcgnATCGN]*$</var-value>

</var>

</field>
```

3. Tool Tips

Tool tips are short aiding text that appears just when the mouse is rolled on over a component (eg: a Text Field). Though the component's labels are descriptive of the contents to be entered by the user, additional tip increases the user's comprehension. It also helps to avoid corrupt data. For example, if a field is labeled as 'Sequence', then there is a high chance that the user gets confused whether to enter Nucleotide or Protein sequence. A short tool tip with a text message 'The primary structure with A,T,C,G or N' avoids this confusion. StocksDB has provided useful and descriptive tool tips for all the web components in the screens.

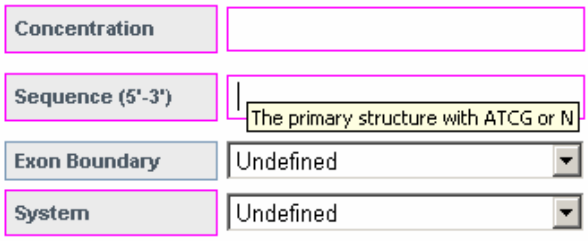


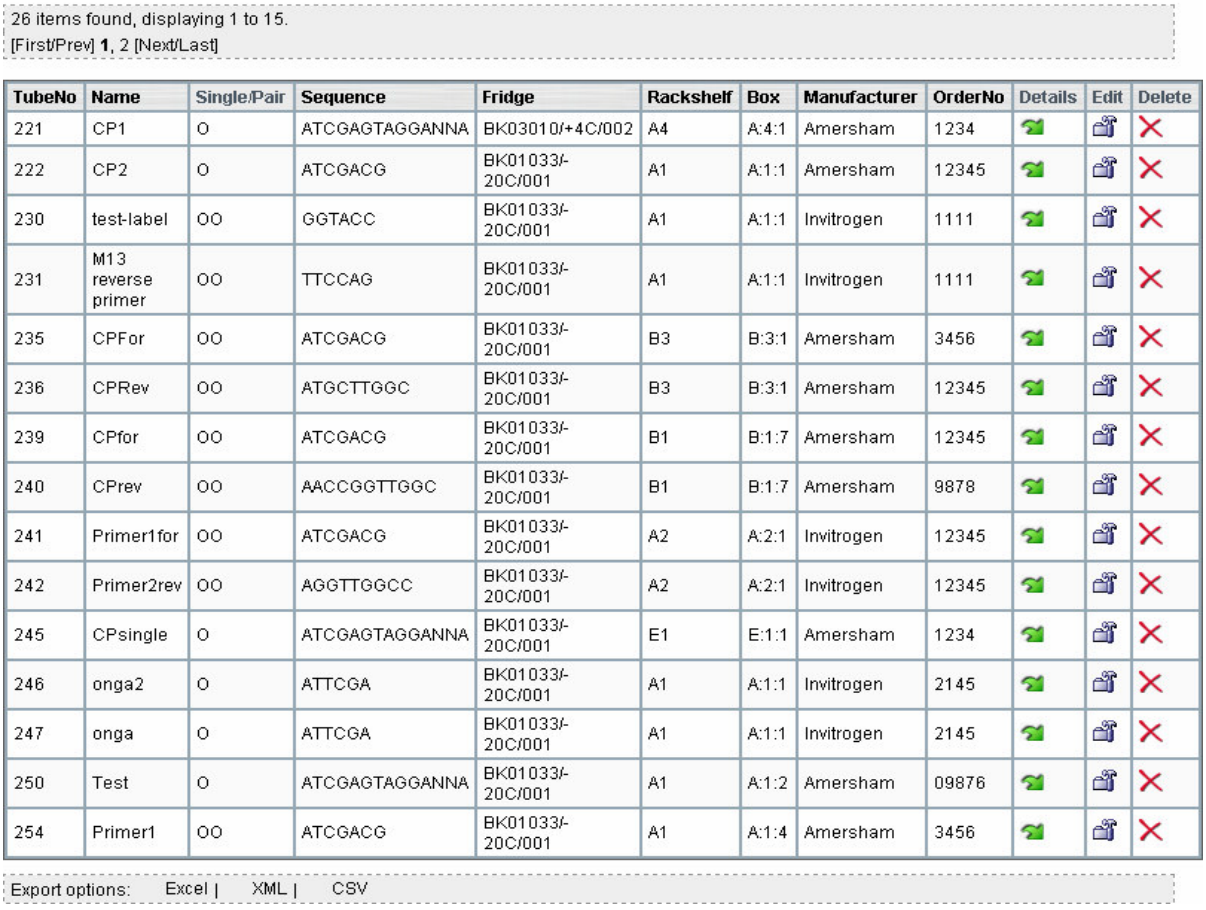
Figure 4.10 A tool tip with a short aiding text for the Sequence Field

4. Differentiating Mandatory and Optional Fields

In StocksDB, for each entity the attributes are divided into mandatory and optional attributes. The mandatory attributes are essential attributes and must be filled in by the user. The optional attributes are not-so-important fields and the user can even leave them blank without filling anything. In the web interface, the mandatory fields are given a pink border and the optional fields blue border as shown in Figure 4.10. This helps the user to differentiate the fields. It should be noted here that only the mandatory fields are validated.

5. DisplayTag

The DisplayTag library is used in StocksDB to display the tables in default view.



26 items found, displaying 1 to 15.
[First/Prev] **1**, 2 [Next/Last]

TubeNo	Name	Single/Pair	Sequence	Fridge	Rackshelf	Box	Manufacturer	OrderNo	Details	Edit	Delete
221	CP1	O	ATCGAGTAGGANNA	BK03010/+4C/002	A4	A:4:1	Amersham	1234			
222	CP2	O	ATCGACG	BK01033/-20C/001	A1	A:1:1	Amersham	12345			
230	test-label	OO	GGTACC	BK01033/-20C/001	A1	A:1:1	Invitrogen	1111			
231	M13 reverse primer	OO	TTCCAG	BK01033/-20C/001	A1	A:1:1	Invitrogen	1111			
235	CPFor	OO	ATCGACG	BK01033/-20C/001	B3	B:3:1	Amersham	3456			
236	CPRev	OO	ATGCTTGGC	BK01033/-20C/001	B3	B:3:1	Amersham	12345			
239	CPfor	OO	ATCGACG	BK01033/-20C/001	B1	B:1:7	Amersham	12345			
240	CPrev	OO	AACCGGTTGGC	BK01033/-20C/001	B1	B:1:7	Amersham	9878			
241	Primer1for	OO	ATCGACG	BK01033/-20C/001	A2	A:2:1	Invitrogen	12345			
242	Primer2rev	OO	AGGTTGGCC	BK01033/-20C/001	A2	A:2:1	Invitrogen	12345			
245	CPsingle	O	ATCGAGTAGGANNA	BK01033/-20C/001	E1	E:1:1	Amersham	1234			
246	onga2	O	ATTCGA	BK01033/-20C/001	A1	A:1:1	Invitrogen	2145			
247	onga	O	ATTCGA	BK01033/-20C/001	A1	A:1:1	Invitrogen	2145			
250	Test	O	ATCGAGTAGGANNA	BK01033/-20C/001	A1	A:1:2	Amersham	09876			
254	Primer1	OO	ATCGACG	BK01033/-20C/001	A1	A:1:4	Amersham	3456			

Export options: Excel | XML | CSV

Figure 4.11 A snapshot of the View Screen showing the pagination(at the top) and export(at the bottom) features of DisplayTag library

Given a list of objects, the library will handle column display, sorting, paging, cropping, grouping, exporting, smart linking and decoration of the table in a nice and customizable xhtml style. More information can be found in [9]

6. Detail-Edit-Delete Features

Default View-Detailed View

Default view displays only the important fields which are used to identify, locate or order the primers. For a more detailed view, which displays all the attributes of the entity, the user has to click on the 'Details' icon in the default view screen. (As shown in Figure 4.11). As a matter of fact, the detailed view screen should be similar to the input screen. But there are few sub entities which are related to the main entity with many-to-one relationship. For example, the main entity primer can have many targets. It is not possible to know how many targets will be there for each primer beforehand and provide as many fields in the initial input screen itself. The design should allow adding as many targets as the user wishes. In StocksDB, all the sub entities which are related to the main entity by one-to-one relationship will be added when the main entity is first submitted. And all the sub entities with one-to-many relationship will be added through separate interfaces by following the 'Details' icon. The example sub entities of primer which can also be added later are the targets, primer pairs, and reports.

Edit

The user can possibly enter a data with wrong spelling or altogether irrelevant data. The application has provided the edit feature, where by the mistakes can be rectified. The implementation is not straightforward though. Some entities are very tightly coupled. So editing one field might have an impact on other related fields too. For example, if the target Accession Number/GI id is edited, then the product length, product Tm, and the species info have to be recalculated and changed. On the other end, editing a primer name or changing the label name is quite simple as it does not have an impact on other attributes. The application also ensures that only the submitter of that particular record has permission to edit the record. Besides, it also stores the date on which the record was updated.

One important point should be noted here from the practical point of view. When a record is to be updated, first the value object is retrieved using the primary key of the record. Then the properties to be edited are once again set to the value object. If the value object is passed to the update method as it is, then all the foreign key values are lost from the corresponding record.

Care should be taken to set all the relationships to the value object once again before it is updated. Whether this is a merit or demerit of the design, is still a question and is left to debate.

Delete

There might be a situation where in spite of all strict validation, error correction and editing, the record might be corrupted beyond repair. The user should be able to delete such records and the feature is implemented in StocksDB too. Before any record is deleted the use is alerted to prevent any accidental deletion. Moreover, additional control support is provided by the user access privileges based on the User Management System [10].

7. Virtual Freezer

It is not uncommon in any laboratory to find a research student searching for the samples desperately and frequently getting confused about the contents of the box. So it was decided to build a virtual freezer, which is actually the replica of the physical fridge. As a first step, one of fridges in the institute is taken as a model system.

A1				A2				A3			
A:1:1	A:1:2	A:1:3	A:1:4	A:2:1	A:2:2	A:2:3	A:2:4	A:3:1	A:3:2	A:3:3	A:3:4
A:1:5	A:1:6	A:1:7	A:1:8	A:2:5	A:2:6	A:2:7	A:2:8	A:3:5	A:3:6	A:3:7	A:3:8
A:1:9	A:1:10	A:1:11	A:1:12	A:2:9	A:2:10	A:2:11	A:2:12	A:3:9	A:3:10	A:3:11	A:3:12
B1				B2				B3			
B:1.1	B:1:2	B:1:3	B:1:4	B:2:1	B:2:2	B:2:3	B:2:4	B:3:1	B:3:2	B:3:3	B:3:4
B:1.5	B:1.6	B:1.7	B:1.8	B:2:5	B:2:6	B:2:7	B:2:8	B:3:5	B:3:6	B:3:7	B:3:8
B:1.9	B:1.10	B:1.11	B:1.12	B:2:9	B:2:10	B:2:11	B:2:12	B:2:9	B:3:10	B:3:11	B:3:12
C1				C2				C3			
C:1:1	C:1:2	C:1:3	C:1:4	C:2:1	C:2:2	C:2:3	C:2:4	C:3:1	C:3:2	C:3:3	C:3:4
C:1:5	C:1:6	C:1:7	C:1:8	C:2:5	C:2:6	C:2:7	C:2:8	C:3:5	C:3:6	C:3:7	C:3:8
C:1:9	C:1:10	C:1:11	C:1:12	C:2:9	C:2:10	C:2:11	C:2:12	C:3:9	C:3:10	C:3:11	C:3:12
D1				D2				D3			
D:1:1	D:1:2	D:1:3	D:1:4	D:2:1	D:2:2	D:2:3	D:2:4	D:3:1	D:3:2	D:3:3	D:3:4
D:1:5	D:1:6	D:1:7	D:1:8	D:2:5	D:2:6	D:2:7	D:2:8	D:3:5	D:3:6	D:3:7	D:3:8
D:1:9	D:1:10	D:1:11	D:1:12	D:2:9	D:2:10	D:2:11	D:2:12	D:3:9	D:3:10	D:3:11	D:3:12
E1				E2				E3			
E:1:1	E:1:2	E:1:3	E:1:4	E:2:1	E:2:2	E:2:3	E:2:4	E:3:1	E:3:2	E:3:3	E:3:4
E:1:5	E:1:6	E:1:7	E:1:8	E:2:5	E:2:6	E:2:7	E:2:8	E:3:5	E:3:6	E:3:7	E:3:8
E:1:9	E:1:10	E:1:11	E:1:12	E:2:9	E:2:10	E:2:11	E:2:12	E:3:9	E:3:10	E:3:11	E:3:12

Figure 4.12 A Layout of a Model Fridge. The fridge has five shelves which are labeled A, B, C, D, and E. Each Shelf has three racks which are labeled A1, A2, A3……E1, E2, E3. Each Rack-shelf has 12 Boxes numbered from 1 to 12.

In summary,

Number of levels in each Fridge : 5

Number of Rack-shelf combination at each level : 3

Number of boxes in each Rack-shelf : 12

Total Number of boxes in one Fridge : 12 X 3 X 5 = 180

This model is taken as a general model for all the fridges and the model can be mapped to the actual physical storage location. But the application is not restrained to this particular model. The application will be flexible enough to fit any kind of the fridge model. A separate user interface screen is provided for the user to upload the fridge data which he/she can customize based on the fridge model. The user has to just mention the number of shelves, number of racks per shelf and number of box per shelf. Then the application constructs a virtual freezer from the inputs. The final output from the application as seen through web interface is shown below.

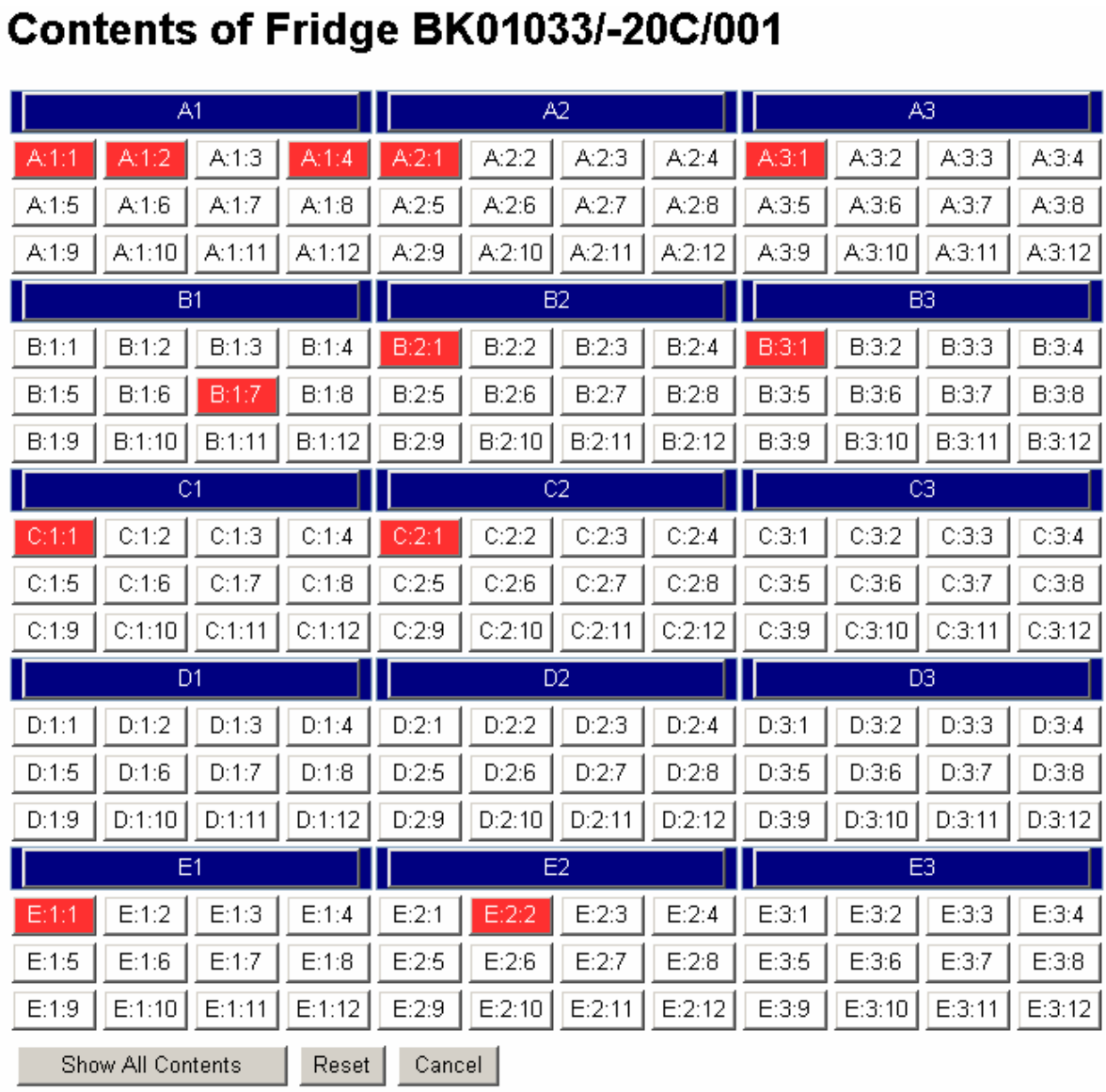


Figure 4.13 A snapshot of Virtual Freezer

The user can click on each of the boxes to see the content of that box. This can be done at different levels ie at the level of box, rack, or even the entire content of the fridge can be visualized. The application also differentiates the empty boxes (white) and filled boxes (red) with specific color codes.

The main use of the Virtual freezer is that the user need not go to the actual physical location to see what is in the fridge. The information can be viewed from his desk. Apart from this, the user can also find who has placed a particular sample in a box of interest and when it has been placed. Now the user has all the information about the content of the freezer in his fingertip.

8. Search Features

A database application should not only allow the user to store the information but should also allow him to retrieve information in an efficient way. Actually, the database should aid the researcher to get more intuitive interpretation of the data than just viewing what has been deposited. Keeping this in mind, our search feature was designed. Two different search screens have been designed for StocksDB, one for simple search and the other for advanced.

A screenshot of simple search is given below.

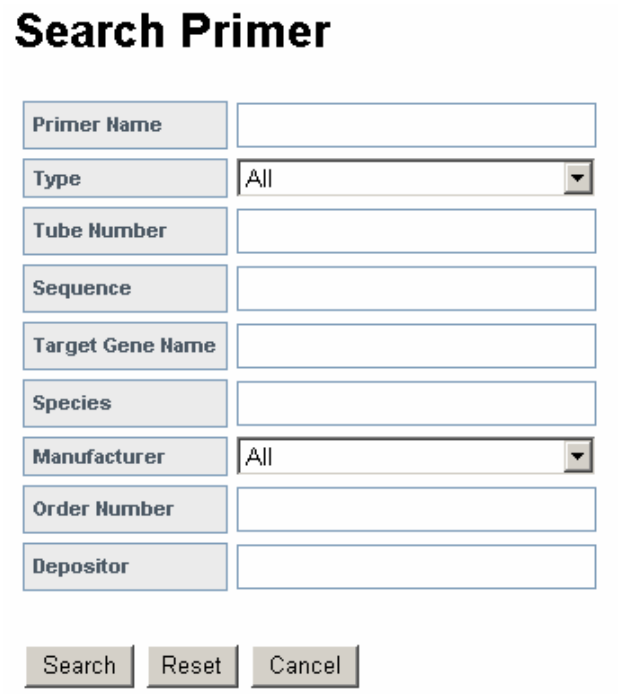


Figure 4.14 A Simple search screen for the Primer

It is more simple and easy to use but less flexible. For example, to search the Primer/Target table for species 'Homo sapiens', then the same can be done with simple search. But in order to search either 'Homo sapiens' or 'Drosophila', then the user must go for an advanced search.

By default all the text entered in the simple search will be searched for likeliness. For example, as search for a plasmid name 'Blue' will match all plasmids with any of the names like 'Blue Script' or 'Blue Script Vector' or 'Plasmid Blue'.

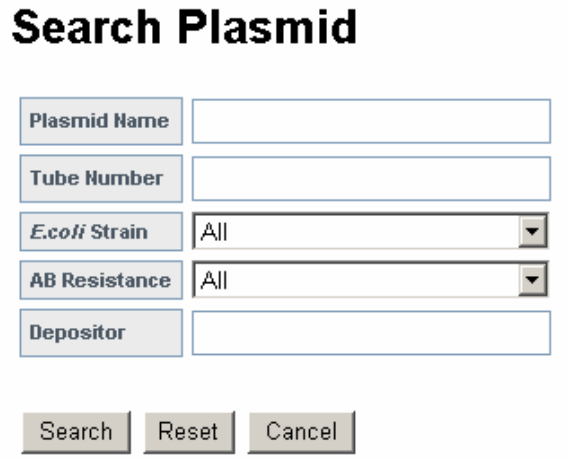


Figure 4.15 A simple search screen for the plasmid

The web interface for advanced search is designed to be as flexible as possible. The user can choose any entity/table in the combo box. Depending on the entity chosen, the columns in the sub entity field changes dynamically.

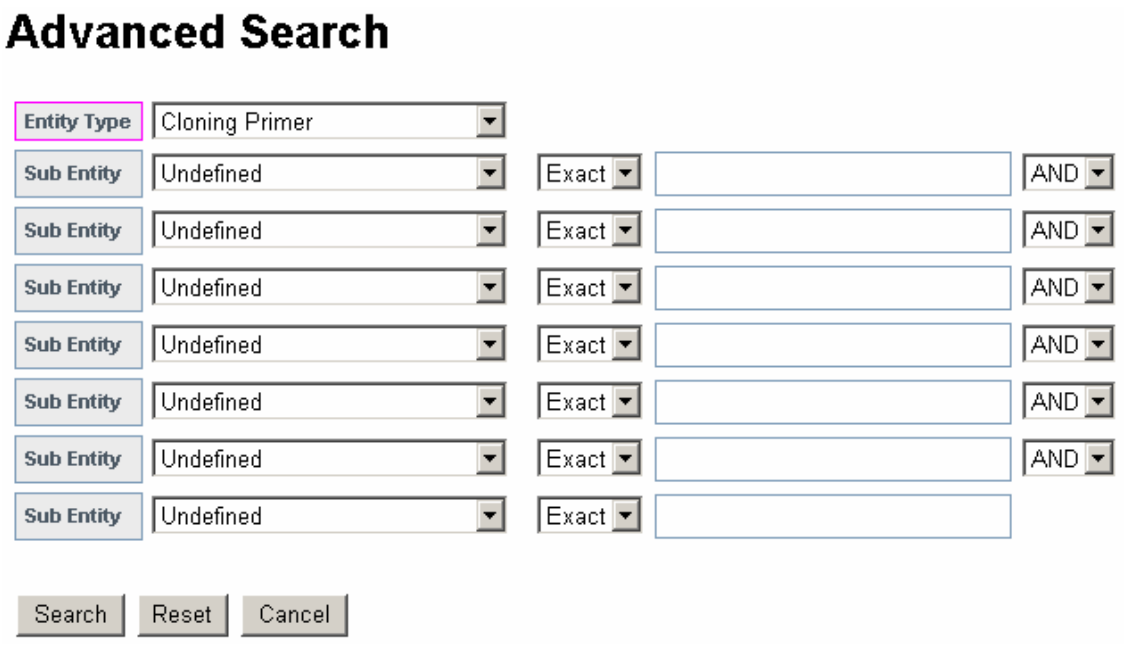


Figure 4.16 Advanced Search screen common to all entities

The user has the flexibility to search for 'exact' match or 'like'. For example if the 'Exact' option is chosen, then 'Homo sapiens' has to be entered if a species search has to be done. If

'Like' is selected, then it is sufficient to enter 'Homo'. Additionally, there are Boolean options 'AND' and 'OR'.

The search is not executed through the EJBQL but rather with SQL with direct JDBC connections, which returns a ResultSet. The reason behind this is that the EJBQL is generated by the AndroMDA based on the 'finder method' declarations in the UML model. These finder methods can only be used to search the database at the most with single parameters. Moreover, EJBQL is limited in its capabilities. SQL is more flexible and advanced in its capabilities. So, internally, the application builds the SQL query dynamically based on the search parameters. Note that the query is not restricted to one table. If the user opts to search in the target fields also, then a 'SQL JOIN' has to be performed.

4.6 FIFTH PHASE: INTEGRATION & TEST

During the integration and test stage, the software artifacts and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software.

Tracking and managing the issues and bugs that emerge during a project is a critically important task. During the Test phase of StocksDB, a list of test cases was prepared and handed over to two researchers Anne and Christine, who are also the scientific advisors for StocksDB. Few bugs which were found during the test phase were subsequently corrected. There were also few suggestions to improve the flexibility of the database. Due to the time limitations, some tasks were categorized as high priority and some were moved to the outlook (refer Discussion chapter). The outlook also includes some points which came up during the requirement definition phase. Those features will be implemented in later versions of StocksDB.

The bugs reported and the suggestions made are updated periodically in the JIRA, a J2EE-based, issue tracking and project management application.

4.7 SIXTH PHASE: INSTALLATION & ACCEPTANCE

During the installation and acceptance stage, the software artifacts, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the institute. After the scientific advisors for StocksDB have

verified that the initial production data load is correct and the test suite has been executed with satisfactory results, then StocksDB will be opened to all our lab personnel.

During the test phase, the database server for StocksDB was MySQL. The final production version uses Oracle server. The application is yet to be deployed in the production server.

There is a common say in the IT field, '**If at first you don't succeed, call it version 1.0**'. Though an honest attempt has been made to make a robust database application, still the current version of StocksDB is named as 1.0. There is always a room for improvement in any software. Similarly the second version (2.0) will soon be released with improved features and functionalities.

4.8 Usability Testing

Usability testing encompasses a range of methods for identifying how users actually interact with a complete system/application. In a typical approach, users — one at a time or working together — use the system/application to perform tasks.

Testing Goals

The goal of this usability testing is to find out whether the system meets its intended purpose.

In a usability test, usually answers to the following types of questions will be obtained from the user:

- Do users complete a task successfully?
- If so, how fast do they do each task?
- Is that fast enough to satisfy them?
- What paths do they take in trying?
- Do those paths seem efficient and clear enough to them?
- Where do they stumble?— What problems do they have?— Where do they get confused?

Keeping the above questions in mind, the test was prepared. The users were asked to perform eight scenarios or tasks. Each of the scenarios is graded in a scale of 1 (lowest) to 5 (highest) with reference to the following five parameters.

Simplicity

- Ability to grasp easily.
- Ability to learn easily
- No redundant functionality.

Forseeableness (Anticipate)

- Expected result.
- Consistency.
- Logical Structure

Transparency

- Where am I?
- Which possibilities do I have?

Efficiency

- Minimal Actions to accomplish the tasks.

Fault-Tolerance

- Design must minimize incorrect input of the user.
- Avoid possibility of incorrect inputs.

Apart from this, the last parameter is the time taken (in minutes) to accomplish each task.

Six people, three biologists and three information technologists, were selected to do the test.

The scenarios and the results are given below. Please refer Appendix C for the the templates and the actual scores.

Scenarios

- Add a single Cloning primer filling all fields.
- Add a qRT Primer Pair filling all fields.
- After successful completion of any of the above two scenarios, try to change/edit the primer name.
- Add additional Target to the single cloning primer which you added in the first scenario.
- Add a Plasmid with Reference as Attachment (you can attach any **Test** document)
- Do a simple plasmid search

- Do an advanced primer search
- Find how many boxes are occupied in Fridge 001 and check the contents of the boxes.

Results

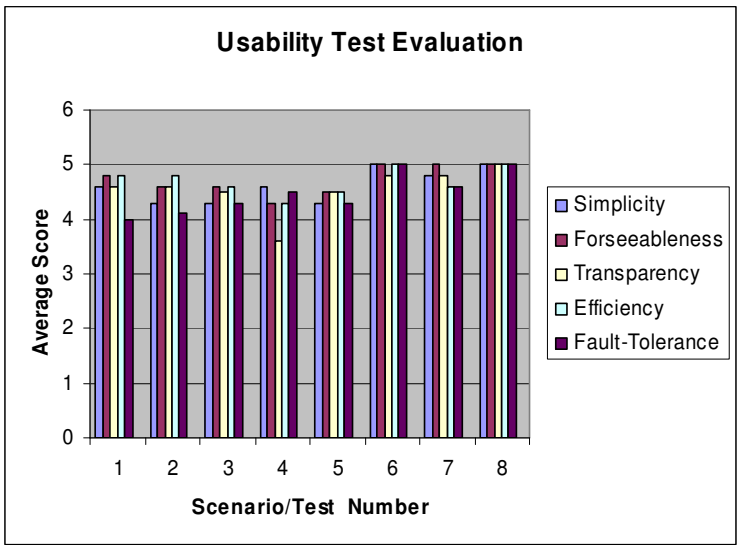


Figure 4.17 Results of the Usability Test

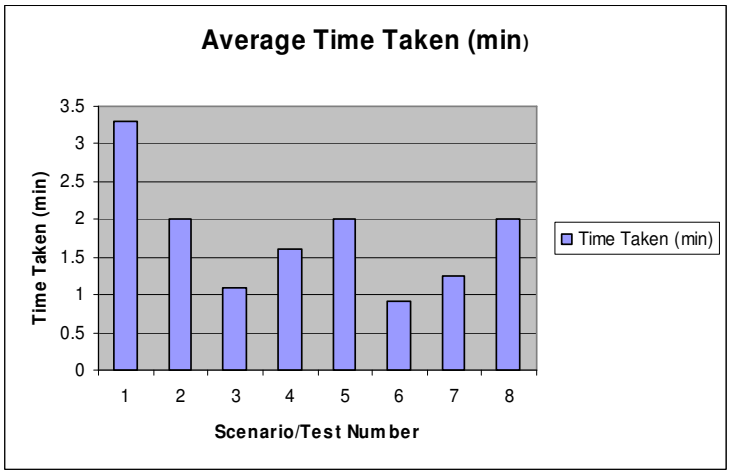


Figure 4.18 Results of the Time Taken to accomplish the scenarios/tasks

Besides the tests with eight scenarios, the users were also asked to grade the overall usability of the application with specific emphasis on four features screen, terminology, system capabilities, and overall usability. Similar to the test done above, the grading was in a scale of 1 to 5.

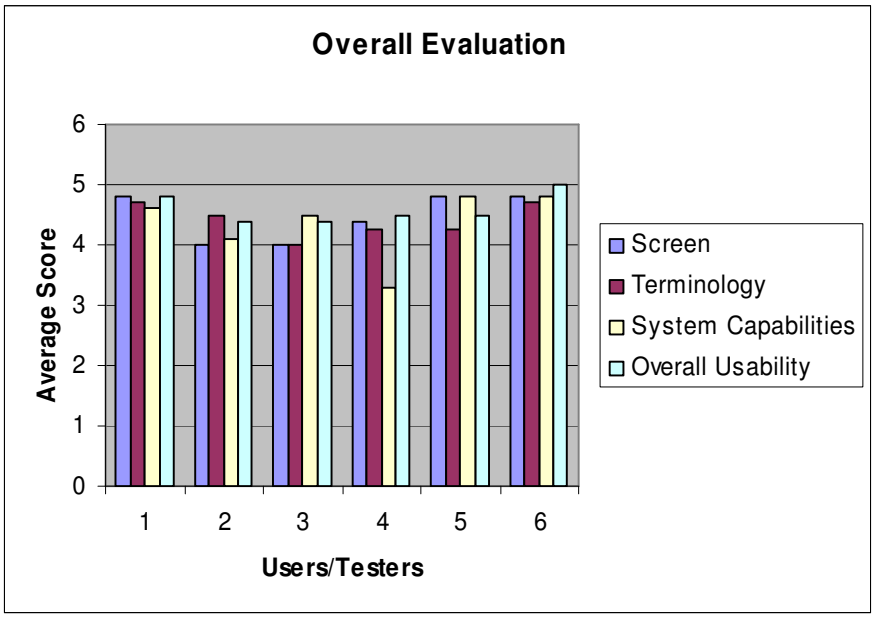


Figure 4.19 Results of the Overall Evaluation

How the test results were used?

The data from all participants is compiled and the each of the user's problems was noted down. The problems were sorted by priority and frequency of the problem. And solutions were developed to fix them. Apart from the problems, the suggestions made by the user to improve the flexibility and user-friendliness of the application was also noted down, sorted by priority, and were implemented based on priority.

Problems that were detected and fixed out of the usability test are:

1. The sequence is not fetched by the EFetch, which is used to fetch the sequence, if the user enters a space after the GenBank Id/Accession Number.

2. The concentration field of the primer interface is not validated for alphabets.

Most of the users had problems with the fourth scenario, which is adding an additional target to the Cloning Primer. To overcome this, a detailed Help page will be provided, which provides a step by step guide to accomplish the tasks.

To sum up, the overall impression of the users about the system is very good.

5 Discussion

5.1 General

In this project, a database application was developed to store the information about the biological stocks used in our laboratory. The information comprises primer related data like the reaction conditions, the actual oligonucleotides etc., and plasmid related data like the maps, constructs, inserts etc.

This database is used to serve the needs of researchers who are interested in both contributing to and taking advantage of the stored information in this area. To make this database available to a wide range of users, the database was implemented as a web client application.

To facilitate easy and flexible database access the Relational Database Management System was adapted. The database was implemented in J2EE architecture to give the system a two-way advantage: First, it had simplified the application development by having the container manage most of the essential work. Second, because of the services provided by the container, the system is more robust, efficient and fail-safe. Additionally, the system can be easily integrated with external applications or databases as and when needed.

The application presents the data to the end-user in an understandable and simple format incorporating all routine database features.

The implementation of the virtual freezer allows the user to track the movement of the stocks within the storage area facilitating easy access to stocks.

Search features were included to view the data in a flexible way by querying the database. The interfaces for querying are customized giving due consideration to the expertise of different users.

To increase the data integrity and reduce the user's workload, the application design minimizes the manual input of the data as and when possible.

Efficient and secured access control to the data is provided by integrating the database application with the User Management System [10]. This prevents unauthorized manipulation of the data.

In future new entities will be added to the system namely the tissues, organisms and antibodies. These features where not within the scope of the current thesis work. But there are few features which are within the scope of this work, but are moved to the outlook due to time constraints.

5.2 Influence of the Technologies

The application was developed using a code generator tool, AndroMDA, the J2EE technologies, and the struts framework.

It is obvious that code generation is powerful and can build useful code, but does it have drawbacks?

Code generation is not without disadvantages. Perhaps the biggest drawback of code generation is that it is difficult to adapt easily. The learning curve is very steep. Initially a lot of time has to be spent in understanding the whole process. It is not just one technology. To have a clear understanding of what is going in and coming out, some basic understanding of all the technologies used by the code generator is required. In the case of AndroMDA, the developer should have an idea about tools like Ant, XDoclet, Velocity, MagicDraw etc., and a good grasp of the J2EE technologies. Otherwise it will be just like a magician's black box. A lot of things can be seen coming out of the box, but without a clue how or where it is coming from.

All the efforts that are put in initially are not without any benefits. The real benefit will be realized once the learning phase is over and the development phase is started. It will be realized that AndroMDA has made the development much easier, faster, and efficient. The merit of the whole framework can be observed when there is a design change mid-way during the development process. To incorporate these changes into the project, it is mandatory that the model need to be changed. Imagine a case where the code generator is independent of the model. Then practically, no one will bother to go back and make changes to the UML model once they have passed the design phase. In short, the model and the application are inseparable. Viewing it from the other side, a small change in the model will reflect on nearly five to six java files in our application. During the development of StocksDB, it was inevitable to make changes to the database schema and to the UML model during the course of development. AndroMDA has taken care of all the issues and made development easier, thanks to the efficient design of the AndroMDA Template Project and the custom cartridges developed by Thomas Truskaller [8].

Integration of the beans with the presentation layer was done with Struts. The Struts learning curve is not too steep, and developers who are familiar with MVC-based Web apps should have no difficulty in adapting to Struts development. But application development in enterprise applications requires discipline. Starting from the struts-config file to the JSP page everything should be restricted strictly to the specifications otherwise it will take some time to figure out the errors, as the error-reporting is not straightforward.

All of the above technologies supported the implementation of the proposed features of StocksDB. In fact, with these powerful, scalable and stable technologies anything and everything is possible to develop.

5.3 Outlook

An application should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear unnecessary. The outlook includes some points which came up during the requirement definition phase and some suggestions after the test phase. These points may be considered as a weakness or restrictions of the application. Due to time limitations, it was decided to implement these features in the next version of StocksDB. The wish list follows:

- Extend the database to include additional entities like organisms, tissues and antibodies.
- A separate web interface to edit/delete/view dictionary table entries.
- A web interface to add another type of Oligo, a simple Oligo which is a not primer and do not have a target.
- A feature/interface to move the contents of one fridge to another fridge.
- While adding a new fridge, there should be a feature to edit the room and temperature information.
- Validating the primer sequence to check if it is complementing with the target.
- Dynamically calculating the 5’ and 3’ position of the primer from the primer sequence and the target.
- Automatic generation of a complete report for a particular entity based on standard templates.
- Generate database statistics dynamically.

5.3 Conclusion

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult. StocksDB was designed keeping in mind the first strategy. The application was developed for our laboratory personnel to efficiently store and accesses the data. A prerequisite for the accessibility of data of different types e.g. primer information, plasmid information, or location information is the standardized submission of different parameters/subsets. This standardized procedure will also ensure high quality of stored data and interoperable input and output of unrelated users. Furthermore uniform data storage and open access could facilitate ordering procedures and would guarantee a complete documentation in terms of an electronic lab journal, which is crucial for any successful modern laboratory.

5.4 Acknowledgments

I would like to specially thank my supervisor Univ. Prof. Zlatko Trajanoski for giving me an opportunity to work in his lab. I am greatly indebted to my project advisor Gerhard Thallinger and scientific advisors Anne Krogsdam and Christine Paar, who have spend endless hours in supervising and guiding me throughout the course of the project. I like to thank Bernard Mlecnik and Jürgen Hartler for their valuable tips during the development of the project. I am also thankful to our course coordinator Marcel Scheideler and the entire staff of Institute of Bioinformatics and Genomics for their support and encouragement.

References:

[1] AndroMDA:
last visited on 2004-10-15,
http://www.andromda.org/

[2] Ant:
last visited on 2004-10-15,
http://ant.apache.org/

[3] Bahman Kalali, Implementation of an EJB Application Using JBoss, Department of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1
last visited on 2004-10-03,
http://www.math.uwaterloo.ca/~bkalali/project.pdf

[4] Bustin SA, Quantification of mRNA using real-time reverse transcription PCR (RT-PCR): trends and problems, *Journal of Molecular Endocrinology* (2002) 29: 23–39

[5] Chuck Cavaness, Using the Validator Framework with Struts, *Article in onjava.com*
last visited on 2004-09-23,
http://www.onjava.com/pub/a/onjava/2002/12/11/jakartastruts.html

[6] Code Generators:
last visited on 2004-10-24,
http://www.onjava.com/pub/a/onjava/2003/09/03/generation.html

[7] Daniel Aparicio, Tefen, Laboratory Information Management Systems: Seven Key Factors to a Successful Implementation, Operations Management Consulting, USA
last visited on 2004-11-11,
http://www.tefen.com/files/articles/LIMS-%20Seven%20Key%20Factors%20Final(1).pdf

[8] Data Integration into a Gene Expression Database, Master Thesis by Thomas Truskaller, TU-Graz, 2003

[9] Display Tag:
last visited on 2004-09-21,
http://displaytag.sourceforge.net/

[10] Design and Development of a User Management System for Molecular Biology Databases, Master Thesis by Dieter Zeller, TU-Graz, 2003

[11] Efetch:
last visited on 2004-11-03,
http://eutils.ncbi.nlm.nih.gov/entrez/query/static/efetch_help.html

[12] EJB architecture image
last visited on 2004-11-10,
http://csci.mrs.umn.edu/UMMCSciWiki/pub/Ochemtutorial/EnterpriseJavaBeans/tier_architecture.jpg

[13] Fast Track to struts, What it does and How, By Nadhir Gulzar, TheServerSide.com
last visited on 2004-11-10,
http://www.theserverside.com/news/thread.tss?thread_id=16336

[14] Gordon K. Smyth and Terry Speed, Normalization of cDNA Microarray Data, To appear in: *D. Carter (ed.), METHODS: Selecting Candidate Genes from DNA Array Screens: Application to Neuroscience*.

[15] Holland PM, Abramson RD, Watson R and Gelfand DH, Detection of Specific Polymerase Chain Reaction Product by Utilizing the 5' 3' Exonuclease Activity of Thermus aquaticus DNA Polymerase, *PNAS* 1991, 88:7276-7280

[16] Java Database Connectivity
last visited on 2004-11-11,
http://java.sun.com/products/jdbc/

[17] JBoss:
last visited on 2004-11-11,
http://www.jboss.org/index.html

[18] JSP:
last visited on 2004-11-11,
http://java.sun.com/products/jsp/

[19] JBuilderX:
last visited on 2004-11-11,
http://www.borland.com/jbuilder/enterprise/index.html

[20] JDOM:
last visited on 2004-11-11,
http://www.jdom.org

[21] LUX System:
last visited on 2004-09-24,
http://www.invitrogen.com/content/sfs/brochures/711_01953LUXBrochure.pdf?brochureid=72

[22] MagicDraw:
last visited on 2004-11-11,

http://www.magicdraw.com/

[23] Managing ER Diagrams

last visited on 2004-11-11,

http://www.datanamic.com/dezign/

[24] Microarray Workflow:

last visited on 2004-10-26,

http://www.ym.edu.tw/excellence/HBP/HBP_CP4/microarray.jpg

[25] Model-driven development of large-scale Web applications*, IBM J. RES. & DEV*. VOL. 48

last visited on 2004-10-14,

http://www.research.ibm.com/journal/rd/485/tai.html

[26] Modeling Systems with UML, *A Popkin Software White Paper*

last visited on 2004-11-11,

http://www.akira.ruc.dk/~keld/teaching/OOP_f02/uml_modeling.pdf

[27] MySQL:

last visited on 2004-11-04,

http://www.mysql.com/

[28] MySQL 4.1 features:

last visited on 2004-11-04,

http://dev.mysql.com/doc/mysql/en/Nutshell_4.1_features.html

[29] OMG Primer:

last visited on 2004-10-24,

www.omg.org/news/pr97/umlprimer.html.

[30] Plasmids:

last visited on 2004-11-09,

http://www.web-books.com/MoBio/Free/Ch9A4.htm

[31] Plasmid & Cloning Workflow:

last visited on 2004-11-09,

http://www.accessexcellence.org/RC/VL/GG/plasmid.html

[32] Real-Time PCR Overview:

last visited on 2004-11-09,

http://cgr.harvard.edu/resource_center/real-time_PCR_overview.htm

[33] SDLC:

last visited on 2004-10-14,

http://www.elucidata.com/refs/sdlc.pdf

[34] Seqhound:
last visited on 2004-09-28,
http://www.blueprint.org/seqhound/
[35] Struts:
last visited on 2004-10-25,
http://struts.apache.org/
[36] Sun/J2EE:
last visited on 2004-11-11,
http://java.sun.com/j2ee/index.jsp
http://www.javaworld.com/javaworld/jw-12-2000/jw-1201-weblogic.html
[37] SybrGreen Molecular Probes:
last visited on 2004-10-14,
http://www.probes.com/handbook/figures/0700.html
[38] Tomcat:
last visited on 2004-10-26,
http://jakarta.apache.org/tomcat/
[39] Tony Yuen, Elisa Wurmbach, Robert L. Pfeffer, Barbara J. Ebersole and Stuart C. Sealfon, Accuracy and calibration of commercial oligonucleotide and custom cDNA microarrays, *Nucleic Acids Research*, 2002, Vol.30, No. 10 e48
[40] Traditional PCR:
last visited on 2004-11-05,
http://www.myxo.uni-saarland.de/files/pdf/RM_SS04_Molekularbiologie7.pdf
[41] UML:
last visited on 2004-11-11,
http://www.uml.org/
[42] Validator Framework:
last visited on 2004-11-08,
http://struts.apache.org/userGuide/dev_validator.html.
[43] Velocity:
last visited on 2004-10-25,
http://jakarta.apache.org/velocity/
[44] Web Application Framework, Craig McClanahan, Larry McCay, Erik Bergenholtz
last visited on 2004-10-22,
http://www.sys-con.com/java/archives/0603/mcclanahan/

[45] Wittwer CT, Herrmann MG, Moss AA & Rasmussen RP, Continuous fluorescence monitoring of rapid cycle DNA amplification, *Biotechniques* (1997) 22: 130–138.

[46] XDoclet:
last visited on 2004-10-25,
http://xdoclet.sourceforge.net/xdoclet/index.html

[47] Yulu Xia, Roland E. Stinner, Ping-Chu Chu, Database integration with the Web for biologists to share data and information, *EJB Electronic Journal of Biotechnology* ISSN: 0717-3458, Vol.5 No.2

[48] Web reference for TaqMan System (RT-PCR)
Last visited on 2004-11-29
http://www.narf.vcu.edu/pcr3.html

Formula

[49] Formula to calcite the Melting Temperature(Tm):

Tm = 4*(G+C)+2*(A+T)

Appendix

Appendix A: User Requirements Document

User Requirement Documentation

Abstract:

This document describes the requirements for the Stocks Database project 'Development of Stocks Database using AndroMDA and J2EE'. It is based on the discussions and meetings carried out with Dr. Christine Paar, Dr. Anne-Margrethe Krogsdam, who specified the requirements and with DI Gerhard Thallinger (Gerhard.Thallinger@tugraz.at), my Project Advisor.

Introduction

STOCKS DB is a database application that store the who, what, where, when, why, etc. information about stocks that are used at the institute. (primers, antibodies, cell lines, plasmids and mouse strains). It should have features to input all the relevant data, view the data in a flexible manner, and also to search the data by querying the database. The application is more or less like an inventory tracking system. In short, the database should assist the efficient management and organization of the data and also provide quick information to the researchers in the laboratory as and when needed.

Current Workflow Without Database Support

The Current Workflow scenario is outlined below,

1. A researcher order an entity or stocks. (From now on we will refer to the stocks as entities) for his experiments.
2. He/She receives the entity.
3. Store the entity in a physical storage place.
4. Make an entry into his/her personal lab book.

Subsequently, from now on, all the associated information about the entity will be entered into his/her lab book. This can include movement of the entities from one physical location to the other, usage of the samples, quality evaluation and reordering the samples.

In this scenario, the data is not stored in one location. The data is spread into many places (lab books) and is stored as per the style and convenience of the individual. Moreover, the data is not available to all the persons at the same time. In short, the data is not uniformly shared. So a need for a database to store this information has raised and subsequently implemented.

Investigation and Analysis Process

A team was formed composed of researchers, a technical advisor and a developer. Dr. Christine Paar and Dr. Anne-Margrethe Krogsdam, the researchers provided the requirements input (will be referred as clients in this document) and DI Gerhard Thallinger, my Project Advisor provided the technical guidance and myself being the developer. A series of interviews were carried out with Christine, Anne and Gerhard and requirements were gathered. Visits were made to the actual physical location where the stocks are stored and to the lab to understand the workflow.

A prototype of the web interface was first build based on the initial requirement and it was continuously modified and refined as more requirements were added from the ideas and suggestions in the series of meetings. Also a prototype simulating the actual application was build to get the feel of the workflow with static html pages.

Out of the five stocks (entities) identified initially for which the information has to be stored in the database, a detailed user requirement analysis has been carried out on two entities namely Primers and Plasmids based on the priority. Only the requirement for these two stocks will be discussed in this document.

Requirement analysis of the other entities will be carried out after the complete development of the database with all the functionalities for the above two entities.

Future Workflow With Database Support

The Future Workflow will be,

1. A researcher orders an entity or stocks for his experiments.
2. He/She receives the entity.
3. Store the entity in a physical storage place.
4. Make an entry into the STOCKS database.
5. The data is available to all the valid users.

If the entities are moved, used, evaluated or reordered, all the information will be updated. The primary benefit of the system being, the data is now centrally available to be shared by all the researchers in the group.

Requirement Categorization

We have categorized the User Requirements as follows:

1. Data Input Requirements
2. Visualization Requirements
3. Functional Requirements

1. Data Input Requirements

As soon as the researcher receives a particular entity, the user must make an entry of a series of data pertaining to the entity into the database. The clients have classified some data as mandatory and some data as optional. Based on the classification, client side validation will be carried out when the application is developed. The web interface will be designed in such a way that the mandatory and optional fields can be easily differentiated.

In the interface design, the fields are grouped depending on the nature or purpose of the data. For example, all the data fields associated with location information form a group, the fields associated with sequence, labels and markers form a group, the data fields associated with manufacturer form a group, and the remaining fields like purpose and comments form a group.

We can classify the data in to two major categories namely Generic Data and the Entity specific data. As the name implies, the Generic Data is common to all the entities and the Entity specific data is much more specific to the entities.

Generic Data:

The six fields Name, Receiving Date, Fridge, Rackshelf, Box, and Tube Number are common for all the entities and are placed as first six entries. These fields will be present in the main input screens for all the entities.

The Generic Data can be further categorized as Identification Data and Location Data.

Note: The fields which are mandatory are shaded in the following tables. The fields which are specified as “Select” are drop down menus. The data is dynamically fetched from the database

and are displayed for the user to select. Apart from the regular options two additional options namely “none” and “undefined” will also be displayed for additional flexibility. By default all the select fields will be “undefined”. Some select fields may also be mandatory fields. If the user has overlooked this field and submitted the form, then the form can be validated with the “undefined” value. Some times all the options listed might not be applicable to the entity. Then the user can go for “none”.

Identification Data:

Field Name	Content	Field Type
Name	Name of the entity	Text Field
Tube Number	A unique number for an entity group, auto-generated. Used for tracking entities.	Text Field (disabled)
Receiving Date	The date of receipt of the entities.	Text Field

Location Data:

Field Name	Content	Field Type
Fridge	The name/description of the fridge	Select
Rack-shelf	The name/description of the Rack-shelf	Select
Box	The name/description of the Box	Select

The three fields Fridge, Rack-shelf and Box are the most important fields associated with the location of the sample.

One of the freezerss in the institute is taken as a model and is given below.

This model is taken as a general model for all the fridges and the model can be mapped to the actual physical storage location. But the application is not restrained to this particular model. The application will be flexible enough to fit any kind of the fridge model. A separate user interface screen is provided for the user to upload the fridge data which he/she can customize based on the fridge model.

A1				A2				A3			
A:1:1	A:1:2	A:1:3	A:1:4	A:2:1	A:2:2	A:2:3	A:2:4	A:3:1	A:3:2	A:3:3	A:3:4
A:1:5	A:1:6	A:1:7	A:1:8	A:2:5	A:2:6	A:2:7	A:2:8	A:3:5	A:3:6	A:3:7	A:3:8
A:1:9	A:1:10	A:1:11	A:1:12	A:2:9	A:2:10	A:2:11	A:2:12	A:3:9	A:3:10	A:3:11	A:3:12
B1				B2				B3			
B:1.1	B:1:2	B:1:3	B:1:4	B:2:1	B:2:2	B:2:3	B:2:4	B:3:1	B:3:2	B:3:3	B:3:4
B:1.5	B:1.6	B:1.7	B:1.8	B:2:5	B:2:6	B:2:7	B:2:8	B:3:5	B:3:6	B:3:7	B:3:8
B:1.9	B:1.10	B:1.11	B:1.12	B:2:9	B:2:10	B:2:11	B:2:12	B:2:9	B:3:10	B:3:11	B:3:12
C1				C2				C3			
C:1:1	C:1:2	C:1:3	C:1:4	C:2:1	C:2:2	C:2:3	C:2:4	C:3:1	C:3:2	C:3:3	C:3:4
C:1:5	C:1:6	C:1:7	C:1:8	C:2:5	C:2:6	C:2:7	C:2:8	C:3:5	C:3:6	C:3:7	C:3:8
C:1:9	C:1:10	C:1:11	C:1:12	C:2:9	C:2:10	C:2:11	C:2:12	C:3:9	C:3:10	C:3:11	C:3:12
D1				D2				D3			
D:1:1	D:1:2	D:1:3	D:1:4	D:2:1	D:2:2	D:2:3	D:2:4	D:3:1	D:3:2	D:3:3	D:3:4
D:1:5	D:1:6	D:1:7	D:1:8	D:2:5	D:2:6	D:2:7	D:2:8	D:3:5	D:3:6	D:3:7	D:3:8
D:1:9	D:1:10	D:1:11	D:1:12	D:2:9	D:2:10	D:2:11	D:2:12	D:3:9	D:3:10	D:3:11	D:3:12
E1				E2				E3			
E:1:1	E:1:2	E:1:3	E:1:4	E:2:1	E:2:2	E:2:3	E:2:4	E:3:1	E:3:2	E:3:3	E:3:4
E:1:5	E:1:6	E:1:7	E:1:8	E:2:5	E:2:6	E:2:7	E:2:8	E:3:5	E:3:6	E:3:7	E:3:8
E:1:9	E:1:10	E:1:11	E:1:12	E:2:9	E:2:10	E:2:11	E:2:12	E:3:9	E:3:10	E:3:11	E:3:12

A Layout of a Model Fridge. The fridge has five shelves which are labeled A, B, C, D, and E. Each Shelf has three racks which are labeled A1, A2, A3……E1, E2, E3. Each Rack-shelf has 12 Boxes numbered from 1 to 12.

In summary,

Number of levels in each Fridge : 5

Number of Rack-shelf combination at each level : 3

Number of boxes in each Rack-shelf : 12

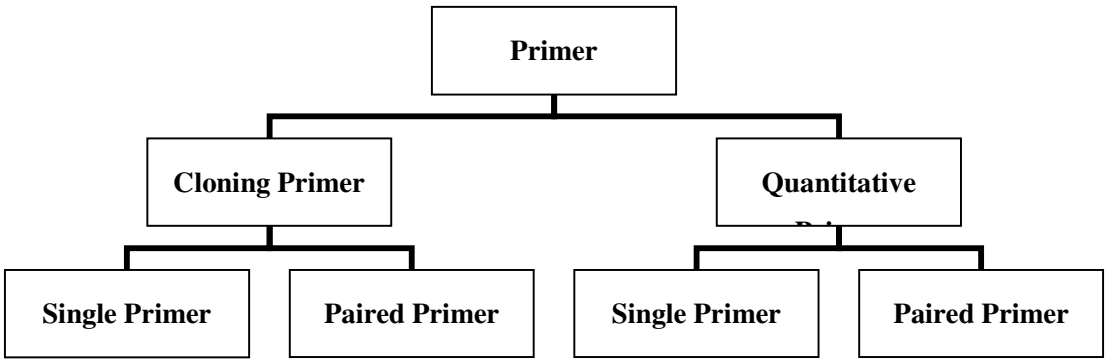
Total Number of boxes in one Fridge : 12 X 3 X 5 = 180

The three fields Fridge, Rack-shelf and Box will be provided as a drop down menu with all the available Fridges, Rack-shelf combinations and Boxes. The menu will be dynamic in the sense that if the user selects a particular Fridge, then the rack-shelf field will display only the rack-shelf combination for that particular fridge and if the user selects a particular rack-shelf, then only the boxes which are available in that particular rack-shelf will be displayed.

Entity Specific Data

Primer

Primers are classified as two types Cloning Primer and qRT PCR Primer for this application. The application will provide separate interfaces for entering either single primers or primers in pairs as the users can order the primers either in single or in pairs.



Fields common for Cloning Primer and qRT Primer

Primer and Sequence Data

Field Name	Content	Field Type
Concentration	The concentration of the sample in pmol/µl- Quantitative measure which tells about the modality	Text Field
Type	Whether Forward Primer or Reverse Primer	Select
Sequence	The primer sequence (only bases A,T,C,G,N are allowed)	Text Field
Tm	Melting Temperature. Will be calculated.	Calculated Field [4*(G+C)+2*(A+T)]
Exon Boundary	Flag which states whether the primer spans the boundary of two expressed regions adjacent only in mature transcripts	Select (Yes/No)
Fit for Sequencing	Flag which states the compatibility with sequence analysis reactions	Select (Yes/No)

Target Data

Field Name	Content	Field Type
Target Gene Name	Gene description & aliases	Text Field
GenBank AccNo.	Unique identifier for a sequence record	Text Field
Target 5' position	Position of first nucleotide w.r.t mRNA/cDNA	Text Field
Target 3' position	Position of last nucleotide w.r.t mRNA/cDNA	Text Field

From the above information the application will fetch the sequence from NCBI databank, calculate the product size and sequence and store the information in the database in separate table.

Tags & Label Data

Field Name	Content	Field Type
Tags	Descriptive marker	Select
Affinity Label	Identifying marker with binding ability	Select
Fluorescent Label	Identifying marker	Select

Manufacturer Data

Field Name	Content	Field Type
Manufacturer Name	Name of the Manufacturer/Supplier	Select
Ordering Number	The sample order number	Text Field

Other Data

Field Name	Content	Field Type
Purpose	Purpose of the primer	Text Field
Comment	Short comment on the primer	Text Field

Fields specific for qRT Primers

Field Name	Content	Field Type
System	Probe Type	Select
Amino modifying label	Modifying marker	Select
Hybridization Probe	Labeled fragment pairing the target gene in between TaqMan primer	Text Field (applicable only if the system is TaqMan)

Plasmid

Apart from the Identification and Location data this entity has the following entity specific data

Field Name	Content	Field Type
Plasmid size	Plasmid size in bp	Text Field
Plasmid Map	A gif image of the plasmid map	Attachment
E.*coli* strain	The strain in which the plasmid is obtained	Select
Antibiotic resistance	The resistance to antibiotic if any	Select
Construction-Basic Plasmid	Construction details	Text Field
Methods	The method of construction	Text Field
Name (Insert)	Name of the insert in the construct	Text Field
Genbank Acc.No	Unique identification number of the insert	Text Field
Sequence (Insert)	Sequence of the insert	Attachment
Verified RE Mapping	A flag which says if it's verified by RE mapping	Select(Yes/No)
Verified Sequencing	A flag which says if it's verified by sequencing	Select(Yes/No)
Notes (RE map)	A short note about the RE map	Text Area
Notes (Sequence)	A short note about the sequencing	Text Area
Reference(Pubmed)	Reference to the PubMed	Text Field
Reference	Any reference document can be attached	Attachment
MTA	Material Transfer Agreement	Attachment
Institution	The Institution from which the plasmid is procured	Text Field
Contact	The contact person in the institution	Text Field

2. Visualization Requirements

View Screen for Primer

Name	Sequence	Location				Manufacturer		Depositor	Details	Edit	Delete
		Fridge	Rack	Box	Tube Number	Name	Order Number				
Primer1	ATCGTTGGCCTCGAC	BK03010/-20°C/#1	A:1	A:1:3	200	Invitrogen	356078 01	Christine			
Primer2	GGCCTCGAC	BK03010/-20°C/#1	A:1	A:1:3	201	Invitrogen	356078 01	Christine			

View Screen for Plasmid

Name	*E.coli* Strain	AB Resistance	Location				Depositor	Details	Edit	Delete
			Fridge	Rack	Box	Tube Number				
pref-1_396RL	BL21	amp	BK04033/4°C	A:2	A:2:3	200	Christine			
PLA-1_396P2	XL1 blue	tet	BK03010/-20°C/#1	B:2	B:2:1	201	Annie			

Most of the view screens will fit into the above models. All the view screens will display only the most important columns. This will be the default view. The other columns will be displayed when the user clicks the Details icon, where all the details specific to that particular entity or row will be displayed.

The application will also provide Edit and Delete options. These options will be enabled based on the status of the user. These features will be integrated into the STOCKS DB application from the already existing in-house application User Management System.

3. Functional Requirements

Specific

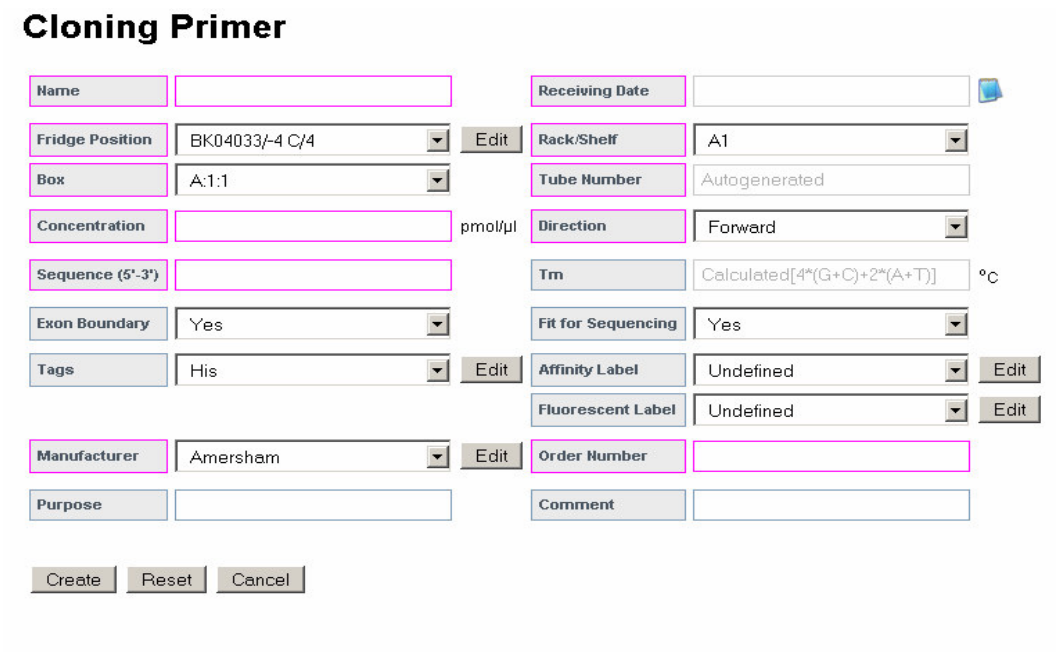
1. Calculation of Tm from the sequence
2. Fetching the sequence from NCBI with the Genbank Id
3. Calculation of product size based on the target positions.
4. Provide links to external databases if applicable (Genbank, PubMed etc.,)

General

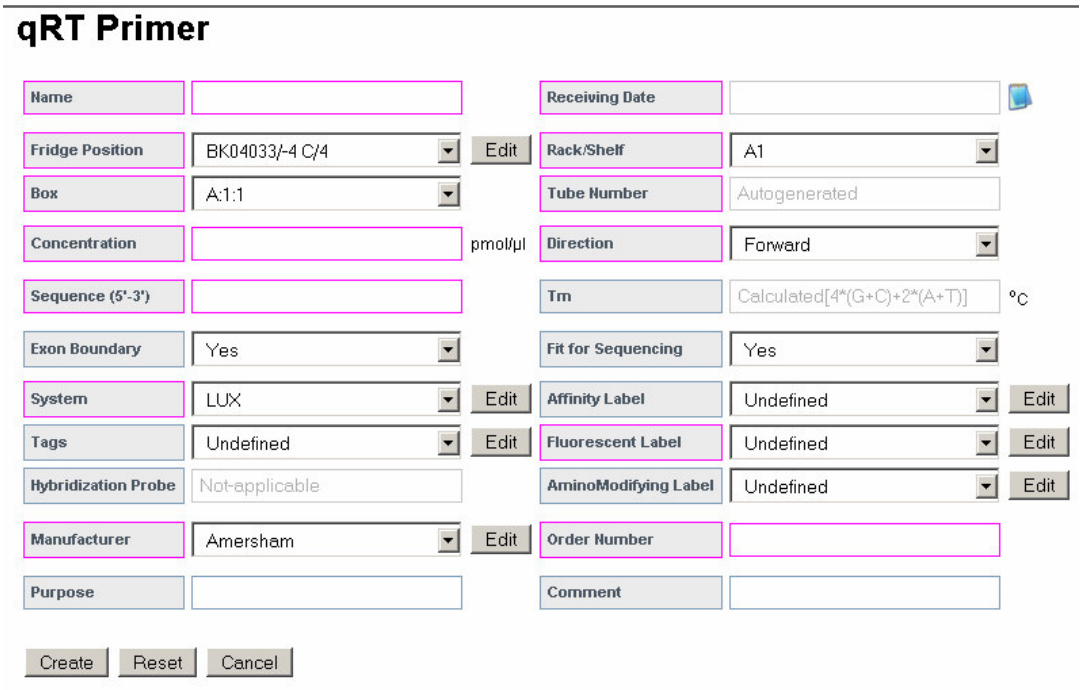
1. Dynamic select options ie the select lists are not hard coded but rather dynamically fetched from the database, which give more flexibility to the application.
2. Strict Validation of the data –data type validation and in some cases content validation. (eg A,T,C,G, N)
3. The application should facilitate all the CRUD implementations ie the Create, Read, Update and Delete features.
4. A record deleted should also delete its associated or related records in a cascading fashion depending upon the relationships.
5. Effective, flexible, and user-friendly search facilities.

The Input Screens

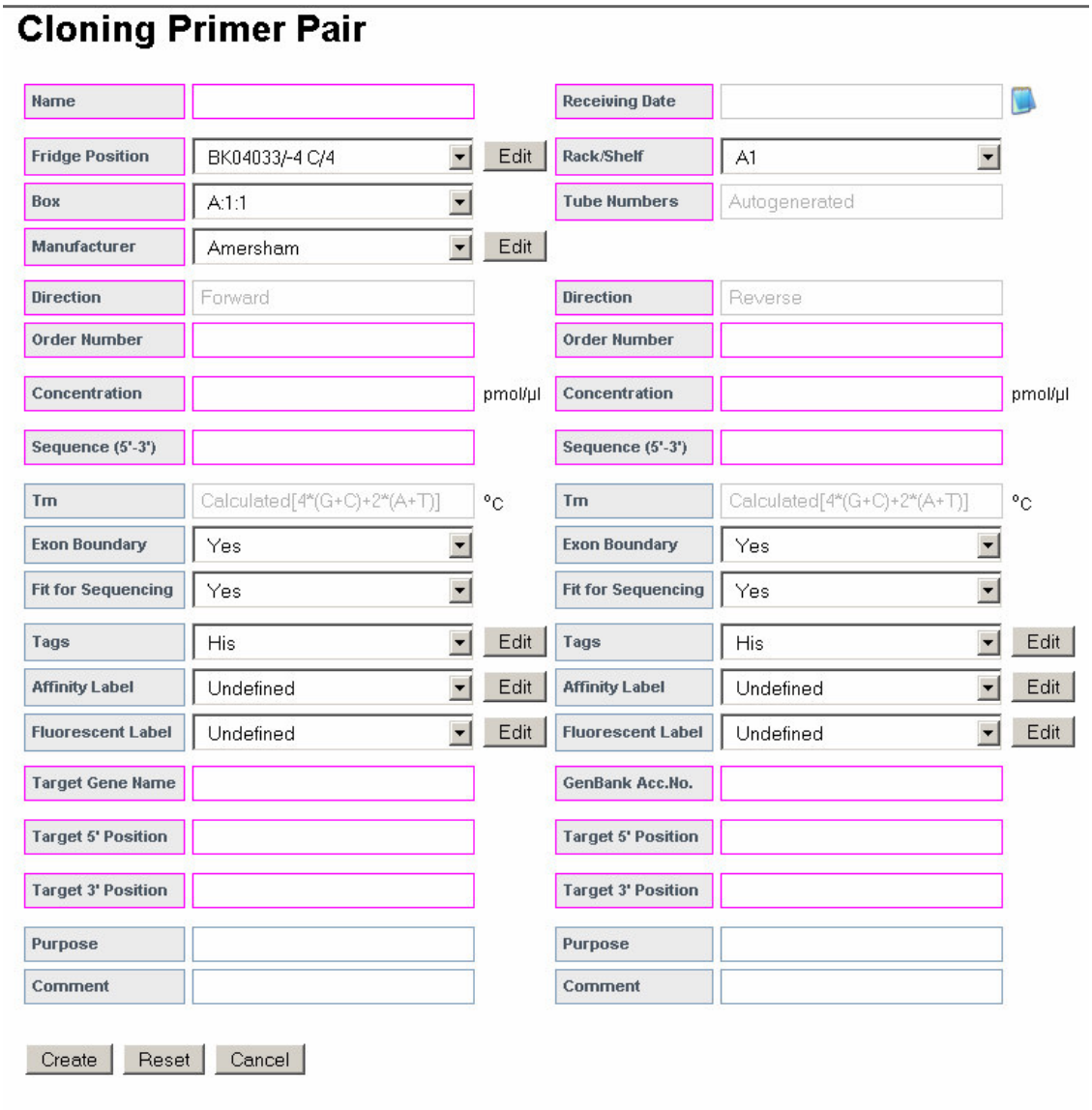
Input Screen for Cloning Primer single



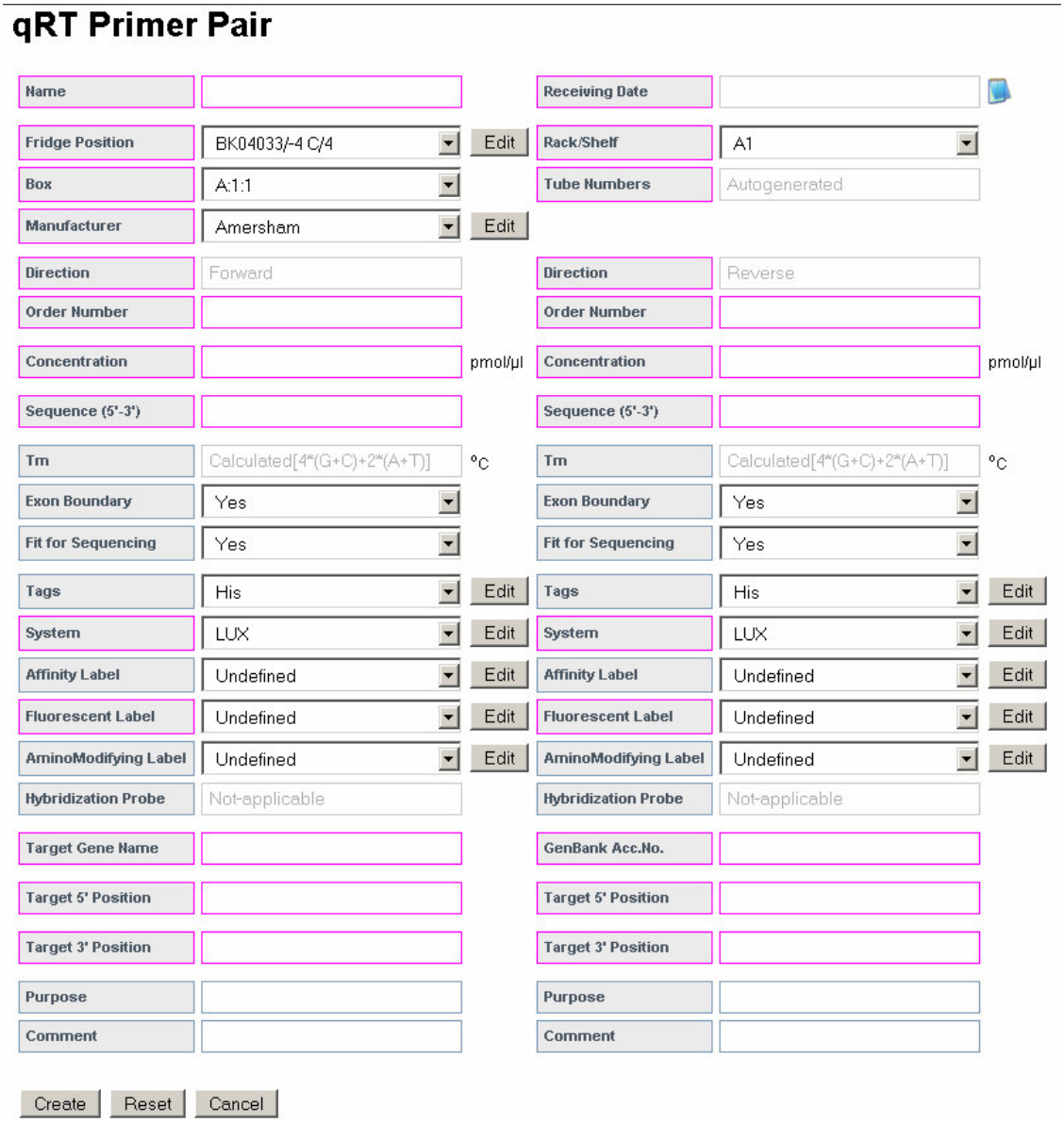
Input Screen for qRT Primer single



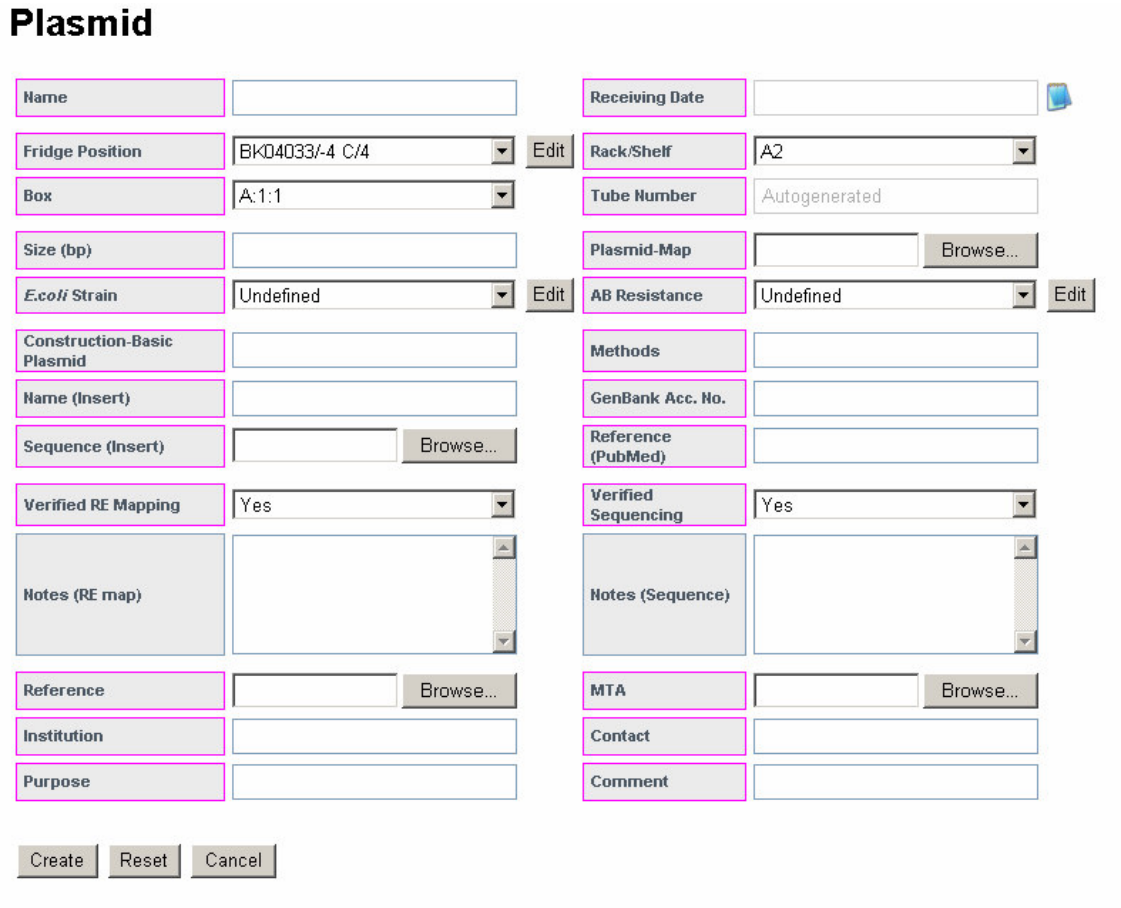
Input Screen for cloning primer pair



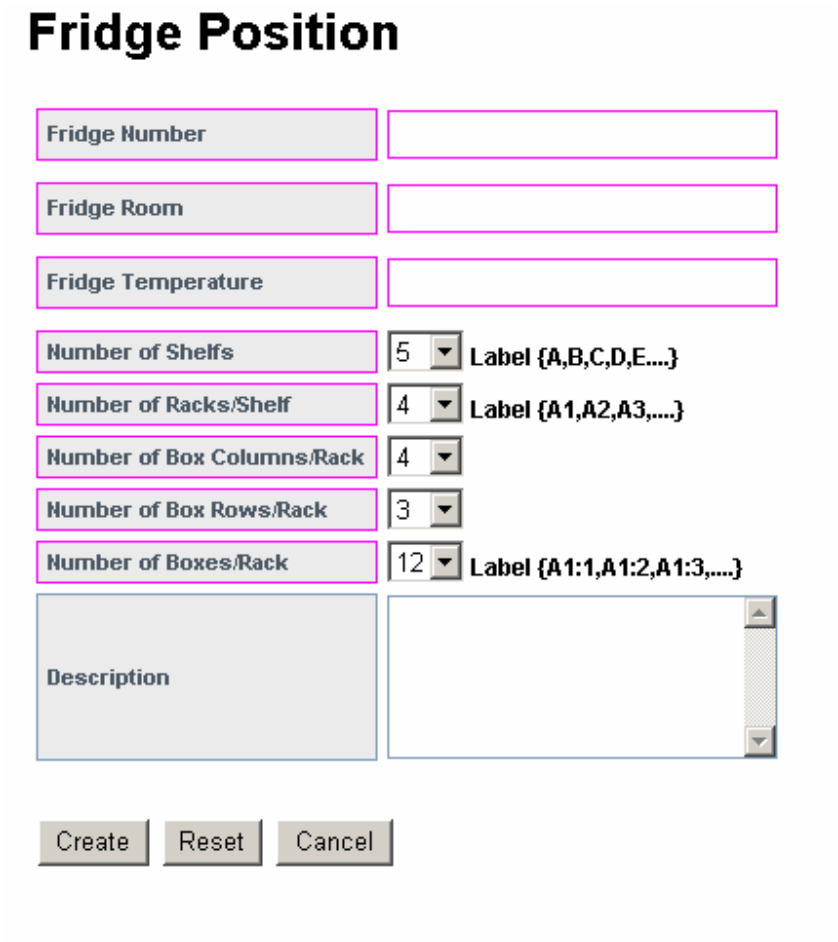
Input Screen for qRT primer pair



Input Screen for plasmids



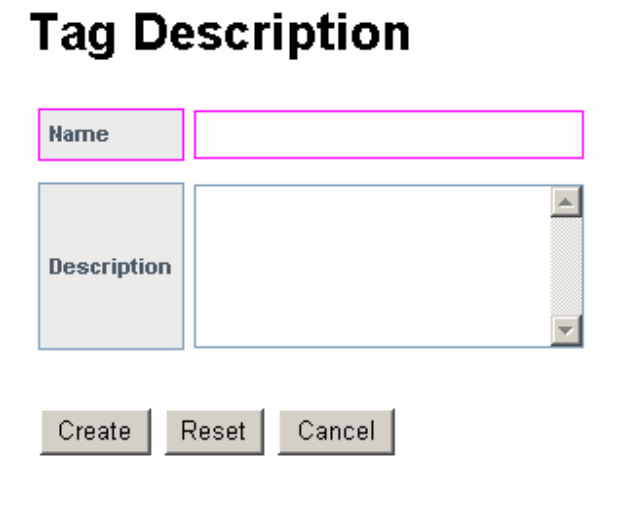
Input Screen for Fridge, Rack-shelf, Box

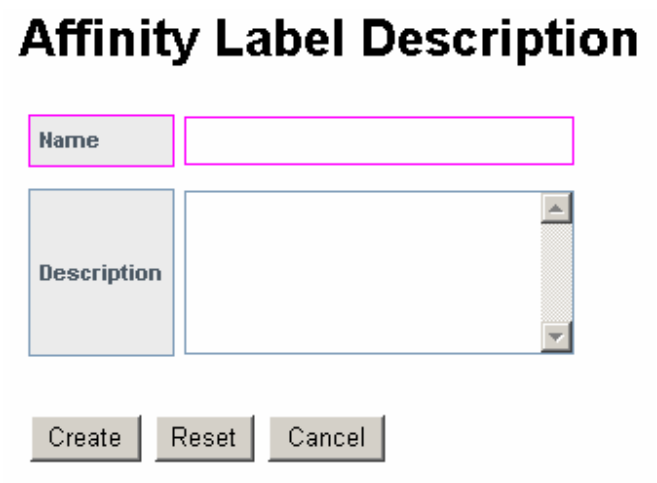


Input Screens for Updating/editing the Select Fields

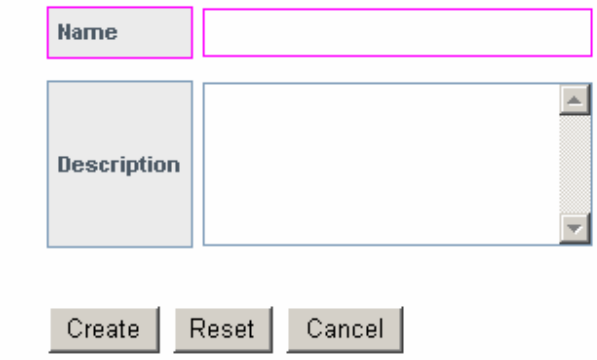
We have provided features for the user to add new data to the all the option fields through separate input screens which are invoked on clicking the 'Edit' buttons.

Field Name	Content	Field Type
Name	Descriptive name to the entity	Text Field
Description	A short note about the entity	Text Area

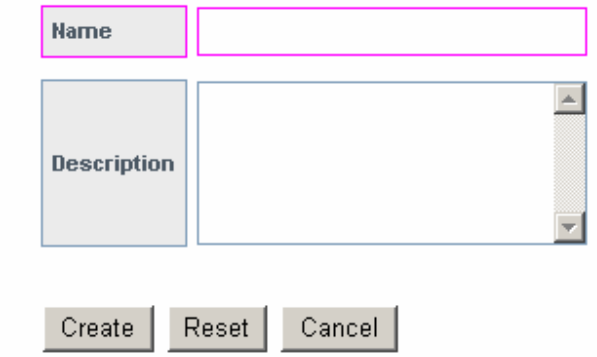




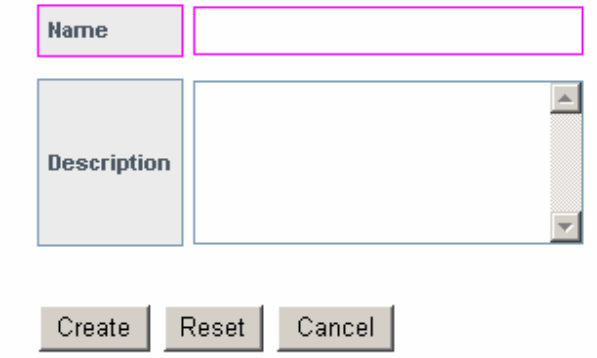
Fluorescent Label Description



AminoModifying Label Description



System Description



Conclusion

We like to strictly stick to the user requirements. But as we proceed with the implementation, based on the implementation requirements and flexibility, few changes might be made to the above requirements.

Finally I like to conclude the User Requirement Analysis documentation with the following quote.

"A user friendly application first requires a friendly user"

&

"Computers follow your orders, not your intentions"

☺☺☺

Appendix B: Installation Instructions

The purpose of this document is to aid the developers who will be upgrading the application and for the developers who will be installing the application in production environment.

1. MySQL Installation

1.1 Download the latest version of mysql from I:\Windows\Application\Databases\MySQL (Version must be 4.1 and above).

1.2 Unzip the file into any directory. The following files would be found in the package (4.1.7 is taken as example):

1.3 Double click on the MSI file. MySQL will be installed in the default install location. When the default installation path is not changed the server is now installed to `C:\Program Files\MySQL\MySQL Server 4.1`. This is the new recommended location for the MySQL server.

1.4 The MySQL Server Instance Configuration Wizard

A checkbox can be noticed on the "Setup Complete Successfully" page. This checkbox is labelled "Configure the MySQL Server now" and is checked by default.

When the checkbox is checked the new MySQL Server Instance Configuration Wizard will be launched after the installation has been completed successfully and you press the [Finish] button.

1.5 How to Use the Configuration Wizard

After clicking next on the welcome page, the type of configuration can be chosen. The "Standard Configuration" should only be used on machines that do not have a MySQL server installed at the moment since it will configure the server to use TCP port 3306 which might conflict with existing servers.

After selecting "Detailed Configuration" and pressing [Next >], the kind of machine should be specified. This will influence memory, disk and CPU usage. Select "Developer Machine".

On the "Database Usage" dialog, the MySQL storage engine can be selected. Select "Multifunctional Database".

On the next dialog, the location for the InnoDB table space can be specified. Keep the default "Installation Path".

On the next dialog, the expected concurrent connections to the server can be specified. Please note that this does not limit the connections, it only assigns an optimal amount of memory to each connection.

Keep the "Enable TCP/IP Networking" turned on in the networking options dialog to allow connections other than named pipes, which is a NT based Windows only feature. The port number is important. It specifies the port on which the server should listen for client connections. The default port is 3306. If a MySQL server is running on this port then change the port number. When pressing [Next >] the Wizard checks if the port is already in use and warns you if it is already taken.

Use the "Best Support For Multilingualism" option in the default character set dialog. This will set the default character to UTF8 which allows the storage of multilingual texts in the database.

The Windows Service dialog is only available on Windows NT bases systems like Windows NT 4.0/2000/XP/2003. To run the MySQL server as a Windows service is the recommended way to run the server on Windows. Please note that the service name, just like the port number, has to be unique. If already a service is installed with the name "MySQL", then choose a different name for the new service. Note that service names are not case sensitive.

The last option dialog is dedicated to security settings. On this dialog you should set a password for the root account, which has full access to the MySQL server.

After all options have been set, press [Execute] on the execution plan dialog. This will setup the new server instance.

The following steps will be applied automatically:

Prepare Configuration

All options and required files will be checked.

Write Configuration File

The MySQL configuration file (`.cnf` file) will be generated and stored in the installation directory (e.g. `C:\Program Files\MySQL\my.cnf`). Please note that Windows hides the file extension `.cnf`.

Start Service

The Wizard installs the Windows service and tries to start up the MySQL server for the first time. If the server startup fails, you can find error logs in the data directory (`C:\Program Files\MySQL\data\*.err`). The configuration file can be modified according to the error message. But please note that this should not occur.

Apply Security Settings

The security settings will be applied if you selected the appropriate options. A TCP/IP connection to the server will be established and the `user` table in the `mysql` schema will be updated. If this step can be completed successfully the server has been configured well.

Please refer to the following article for a detailed MySQL intallation.

http://dev.mysql.com/tech-resources/articles/4.1/installer.html

1.6 Creating the Database and Tables:

Copy the createSequenceTable.sql from the src/sql/ directory of the project to C:\Program Files\MySQL\MySQL Server 4.1\bin>

Say mysql –u 'your user name' –p < createSequenceTable.sql

After entering your password in the prompt a sequences table will be created. This is the only table created manually. This table is used to generate the primary keys.

2. JBoss Installation and Set up

JBoss 3.2.4 was used during the development phase of StocksDB. So the below installation and configuration instructions are for JBoss 3.2.4.

2.1 Installation

Download JBoss from http://www.jboss.org/downloads/index and unpack it in

C:\Java\jboss-3.2.4.

Right click on jboss-3.2.4 directory. Choose property. Select the security tab and choose your user name and grant all permissions to this directory.

Go to:

C:\Java\jboss-3.2.4\bin>

Say: run

JBoss should run without throwing any errors.

To get a live view of the running server, point your browser at the URL

http://localhost:8080/jmx-console

You should see JMX Agent view.

No in the console, say shutdown from C:\Java\jboss-3.2.4\bin>

2.2 Configure JBoss for MySQL

The official JDBC driver for MySQL is called "Connector/J". For StocksDB version 3.0.12 was used. As a first step, download the MySQL Connector for Java (production release) from the below mentioned URL.

http://dev.mysql.com/downloads/

Unzip it into any directory. Inside the directory 'mysql-connector-java-3.0.12-production-bin.jar' file will be found. Place it in C:\Java\jboss-3.2.4\server\default\lib directory.

2.3 Deploying the Datasource

Create a file called *mysql-ds.xml in* C:\Java\jboss-3.2.4\server\default\deploy with the following datasource configuration:

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

<!-- $Id: mysql-ds.xml,v 1.1.2.1 2003/12/12 19:19:56 starksm Exp $ -->

<!--  Datasource config for MySQL using 3.0.9 available from:

http://www.mysql.com/downloads/api-jdbc-stable.html

-->

<datasources>

  <local-tx-datasource>
```

```
    <jndi-name>DefaultDS</jndi-name>

    <connection-url>jdbc:mysql://localhost:3306/stockdb</connection-
url>

    <driver-class>org.gjt.mm.mysql.Driver</driver-class>

    <user-name>prem</user-name>

    <password>premand01</password>

  </local-tx-datasource>

</datasources>
```

which mirrors the database and user information we set up in the previous section. Our aim here is to replace the default datasource in JBoss with a MySQL version, so you have to remove the existing *hsqldb-ds.xml* from the deploy directory or there will be a conflict between the JNDI names of the two datasources. Copy the new file in its place.

2.4 Configuring jms file

Next, copy the file mysql-jdbc2-service.xml from the C:\Java\jboss-3.2.4\docs\examples\jms directory to C:\Java\jboss-3.2.4\server\default\deploy\jms. Do not make any changes to this file.

2.5 Configuring service file

In the C:\Java\jboss-3.2.4\server\default\conf directory, you will find jboss-service.xml file.

The deployment directory is pointed to the jboss in the attribute element as mentioned below. The build directory should be in your project directory.

```
<attribute name="URLs">

   deploy/,file:///D:/Development/Stocksdb/build

</attribute>
```

2.6 Configuring standardjbosscmp-jdbc file

In the C:\Java\jboss-3.2.4\server\default\conf directory, you will find standardjbosscmp-jdbc.xml file.

The datasource element should be as shown below. If it is MySQLDS, change to DefaultDS.

```
<datasource>java:/DefaultDS</datasource>

<datasource-mapping>mySQL</datasource-mapping>
```

Now in the console, say run from C:\Java\jboss-3.2.4\bin>

JBoss should run without any error.

3. StocksDB directory settings

As the server is running in local system, all the attachments made through the web interface of StocksDB application will be stored in local system drive. And the directories are mapped already in the resources file stocksData.properties as shown,

stockAttachments.primer.reports.dir = D:/Development/stockAttachments/primer/reports/

stockAttachments.plasmid.insertsequence.dir=
D:/Development/stockAttachments/plasmid/insertsequence/

stockAttachments.plasmid.map.dir = D:/Development/stockAttachments/plasmid/map/

stockAttachments.plasmid.reference.dir = D:/Development/stockAttachments/plasmid/reference/

stockAttachments.plasmid.mta.dir = D:/Development/stockAttachments/plasmid/mta/

The directory settings can be changed, but the physical directory location should be correctly mapped with above keys in the properties file.

As an example,

Create a directory stockAttachments under D:/Development.

Under stockAttachments create two directories primer and plasmid.

Under primer create directories: reports and insertsequence.

Under plasmid create directories: map, reference, and mta.

The path D:/Development is applicable only for the development environment. It has to be replaced with appropriate path in the production server.

4. Configuring JBuilderX

4.1 Checkout Project

Check out Stocksdb from CVS

Start JBuilderX

Click on Team→ Pull project from CVS. (Module Name: Stocksdb)

4.2 Configuring JBoss server for JBuilderX

Click on Tools → Configure server → Enable Jboss 3.x and Tomcat servers 3.3, 4.0, 4.1

Click on Project → Project properties → Server → choose jboss 3.x and enable the services.

Run → Runtime configuration → New → Choose Name: jboss, Build target: <None>, Type: server, Server: JBboss 3.x and check it as Default.

This will let you to start and stop JBoss as default. Run Project will start JBoss server.

4.4 Build the project

Right click on build and say make. After the build task is completed successfully, start the JBoss server. The server should deploy the Stocksdb.jar and the Stocksdb.war files successfully.

4.5 Start the application

Then login to the StocksDB application through the following url

http://localhost:8080/Stocksdb/

Appendix C: Usability Test Scores

Scenario 1: Add a single Cloning primer filling all fields

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	5	5	5	5	2
Bio2	4	4	5	5	4	2
Bio3	4	5	3	4	3	3
IT1	5	5	5	5	2	7
IT2	5	5	5	5	5	4
IT3	5	5	5	5	5	2
Avg	4.6	4.8	4.6	4.8	4	3.3

Scenario 2: Add a qRT primer pair filling all fields

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	5	5	5	5	1
Bio2	4	5	5	5	5	2
Bio3	3	4	4	4	3	2
IT1	5	5	5	5	2	2
IT2	4	4	4	5	5	3
IT3	5	5	5	5	5	2
Avg	4.3	4.6	4.6	4.8	4.1	2

Scenario 3: Edit/Change the primer name of any of the primers added above

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	5	5	5	5	1
Bio2	5	5	5	5	5	1
Bio3	2	3	2	3	2	1
IT1	5	5	5	5	5	2
IT2	5	5	5	5	4	1
IT3	4	5	5	5	5	1
Avg	4.3	4.6	4.5	4.6	4.3	1.1

Scenario 4: Add an additional Target to the Cloning primer added in Scenario 1

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	3	3	5	5	2
Bio2	4	4	4	4	4	1
Bio3	5	4	2	2	4	1
IT1	5	5	5	5	5	2
IT2	5	5	3	5	4	2
IT3	4	5	5	5	5	2
Avg	4.6	4.3	3.6	4.3	4.5	1.6

Scenario 5: Add a Plasmid with Reference as attachment

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	5	5	5	5	1
Bio2	4	4	4	4	4	1
Bio3	4	3	3	4	3	1
IT1	5	5	5	5	4	5
IT2	5	5	5	5	5	2
IT3	3	5	5	4	5	2
Avg	4.3	4.5	4.5	4.5	4.3	2

Scenario 6: Do a simple Plasmid search

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	5	5	5	5	1
Bio2	5	5	5	5	5	1
Bio3	5	5	4	5	5	0.5
IT1	5	5	5	5	5	1
IT2	5	5	5	5	5	1
IT3	5	5	5	5	5	1
Avg	5	5	4.8	5	5	0.91

Scenario 7: Do an advanced Primer search

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	5	5	5	5	1
Bio2	5	5	5	5	5	1
Bio3	5	5	4	5	5	0.5
IT1	5	5	5	5	5	2
IT2	4	5	5	3	3	2
IT3	5	5	5	5	5	1
Avg	4.8	5	4.8	4.6	4.6	1.25

Scenario 8: Find how many boxes are occupied in Fridge 001 and check their contents

User	Simplicity	Forseeableness	Transparency	Efficiency	Fault-Tolerance	Time taken(min)
Bio1	5	5	5	5	5	1
Bio2	5	5	5	5	5	1
Bio3	5	5	5	5	5	1
IT1	5	5	5	5	5	5
IT2	5	5	5	5	5	3
IT3	5	5	5	5	5	1
Avg	5	5	5	5	5	2

Usability Evaluation of StocksDB								
			1	2	3	4	5	
	Screen							
1	Reading characters on the screen	*Hard*						*Easy*
2	Organization of information	*Confusing*						*Clear*
3	Sequence of screens	*Confusing*						*Clear*
4	Consistency of labels and fields	*Not-Consistent*						*Consistent*
5	Validation	*Not-useful*						*Useful*
	Terminology							
1	Use of Terms throughout the system	*Inconsistent*						*Consistent*
2	Terminology relevant to task	*Disagree*						*Agree*
3	Error Messages Helpful	*Not Helpful*						*Helpful*
4	Help messages on the screen (Tool Tips)	*Not useful*						*Useful*
	System Capabilities							
1	Learning to operate the system	*Difficult*						*Easy*
2	Designed for all levels of users	*Disagree*						*Agree*
3	System Performance (CRUD-Create,Read,Update,Delete)	*Bad*						*Good*
4	System speed	*Slow*						*Fast*
5	System reliability	*Unreliable*						*Reliable*
6	Correcting your mistakes	*Difficult*						*Easy*
	Overall Usability							
1	I am satisfied with how easy it is to use this system	*Disagree*						*Agree*
2	I feel comfortable using this system	*Disagree*						*Agree*
3	It is easy to find the information I needed	*Disagree*						*Agree*
4	The information provided for the system is easy to understand	*Disagree*						*Agree*
5	I like using the interface of this system	*Disagree*						*Agree*
6	This system has all the functions and capabilities I expect it to have	*Disagree*						*Agree*
7	Overall, I am satisfied with this system	*Disagree*						*Agree*

Result of Overall Evaluation

User	Screen					Terminology				System Capabilities						Overall Usability						
Bio1	5	5	4	5	5	5	4	5	5	5	5	3	5	5	5	5	5	5	5	5	4	5
Bio2	5	5	3	4	3	4	4	5	5	4	4	4	3	5	5	4	4	5	5	4	4	5
Bio3	5	4	3	5	3	4	3	4	5	5	3	5	5	4	5	5	5	4	5	3	4	5
IT1	5	4	4	4	5	5	5	3	4	4	4	5	4	4	3	4	4	5	4	5	5	5
IT2	5	4	5	5	5	5	4	5	3	5	5	5	5	5	4	5	5	4	4	5	4	5
IT3	5	4	5	5	5	5	5	4	5	5	4	5	5	5	5	5	5	5	5	5	5	5
Avg	4.4					4.3				4.4						4.6						

Average of Overall Evaluation: 4.4

Glossary

Molecular Biology

5' or 3' end: The nucleoside residues which form nucleic acids are joined by phosphodiester linkages between the 3' Carbon atom of one ribose moiety and the 5' Carbon atom of the next. Therefore each strand of DNA or RNA has a free 3' C at one end and a free 5' C at the other. The free 3' C normally carries a - OH group and the 5' C a phosphate group.

Antibiotic marker gene: A gene that codes for resistance to an antibiotic. It is used to identify when gene transfer has been successful. Following the genetic engineering process, the cells are grown in a medium containing the relevant antibiotic. Only the cells that have been successfully transformed will be able to survive.

Annealing: Formation of double-stranded molecules from two single strands of nucleic acid by base pairing of complementary sequence. Usually achieved by incubation at a favorable temperature (similar to Hybridization)

Application programming interface (API): A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

Base: The purine or pyrimidine component of a nucleotide; often used to refer to a nucleotide residue within a nucleic acid chain.

Base Pair (bp): One pair of complementary nucleotides within a duplex strand. Under Watson-Crick rules, these pairs consist of one pyrimidine and one purine: i.e., C-G, A-T (DNA) or A-U (RNA). However, "noncanonical" base pairs (e.g., G-U) are common in RNA secondary structure.

cDNA: cDNA is DNA copied from messenger RNA (mRNA).

Clone (verb): To "clone" something is to produce copies of it. To clone a piece of DNA, one would insert it into some type of vector (say, a plasmid) and put the resultant construct into a host (usually a bacterium) so that the plasmid and insert replicate with the host. An individual bacterium is isolated and grown and the plasmid containing the "cloned" DNA is re-isolated from

the bacteria, at which point there will be many millions of copies of the DNA - essentially an unlimited supply.

Construct: A "construct" is a vector with a subcloned insert. Expression Construct: The expression vector which contains the coding sequence of the recombinant protein and the elements necessary for its expression.

E. coli: A common Gram-negative bacterium useful for cloning experiments. Present in human intestinal tract. Hundreds of strains of *E. coli* exist. One strain, K-12, has been completely sequenced.

Hybridization: The reaction by which the pairing of complementary strands of nucleic acid occurs. To "anneal" two strands is the same as to "hybridize" them.

Insert: An insert is the DNA carried by a vector into which that DNA sequence has been recombinantly subcloned.

LIMS: Laboratory Information Management System.

Label: Label refers to a reagent, system, or mechanism of differentiating one preparation used in a given expression profiling experiment from others used in the same experiment. Cy3 and Cy5 fluorescent labels, for example, are commonly used to distinguish baseline and experimental preparations in gene expression microarray hybridizations.

mRNA: Mature transcript of RNA splicing which removes introns and joins exons in the primary transcript.

Microarray: A small microscopic glass slide or other solid surface on which thousands of immobilized oligonucleotide probes have been synthesized or robotically deposited in a predetermined array, so that automated recording of fluorescence from each of the positions may be easily determined.

Microarray Technology: A technique that enables parallel assessment of the relative expression of thousands of mRNAs in response to different experimental conditions or in different tissues.

Plasmid: An extrachromosomal, usually circular, double-stranded DNA which is capable of replication within a cell. Common plasmids are pBR322, pGEM, pUC18.

Primer: A small oligonucleotide (anywhere from 6 nucleotides long) used to prime DNA synthesis. The DNA polymerases are only able to extend a pre-existing strand along a template; they are not able to take a naked single strand and produce a complementary copy of it de-novo. A primer which sticks to the template is therefore used to initiate the replication.

Primer pair: Two primers which act over the target sequence in the opposite direction to give a desired product forms a primer pair.

Polymerase chain reaction (PCR): A technique for replicating a specific piece of DNA in-vitro. Primers are added (which initiate the copying of each strand) along with nucleotides and Taq polymerase. By cycling the temperature, the target DNA is repetitively denatured and copied. PCR can be used to amplify RNA sequences if they are first converted to DNA via reverse transcriptase. This two-phase procedure is known as 'RT-PCR'.

Quantitative PCR (QPCR) or Real-Time PCR (RT-PCR): A technique which is used for many different purposes, particularly for quantifying nucleic acids and for genotyping. The peculiarity of real-time PCR is that the process of amplification is monitored in real time by using fluorescence techniques. The information obtained, that is the amplification curves, can be used to quantify the initial amounts of template molecules with high precision over a wide range of concentrations.

Quantitative PCR primer: Primers which are specifically designed in pairs to give a PCR product of predefined base pair size. qPCR can be applied on cDNA obtained by reverse transcription of mRNA. Quantification of the resulting product is then used as a means to quantitate the amount of a given mRNA in the original sample. To obtain optimal sensitivity of the qPCR, special requirements are imposed on the design of the qPCR primers.

Restriction enzyme: A class of enzymes ("restriction endonucleases") generally isolated from bacteria, which are able to recognize and cut specific sequences ("restriction sites") in DNA. There are more than six hundred known restriction enzymes.

Sequence: As a noun, the sequence of a DNA is a buzz word for the structure of a DNA molecule, in terms of the sequence of bases it contains. As a verb, "to sequence" is to determine the structure of a piece of DNA; i.e. the sequence of nucleotides it contains.

Target: In this context, the nucleotide sequences over which the primers can anneal.

Tm: The melting temperature for a double-stranded nucleic acid. Technically, this is defined as the temperature at which 50% of the strands are in double-stranded form and 50% are single-stranded, i.e. midway in the melting curve. A primer has a specific Tm because it is assumed that it will find an opposite strand of appropriate character.

Vector: The DNA "vehicle" used to carry experimental DNA and to clone it. The vector provides all sequences essential for replicating the test DNA. Typical vectors include plasmids, cosmids, phages and YACs.

Information Technology

Architecture: The model(s) that describes how a set of applications will be structured and it also defines the interfaces and design rules for each of its parts. It can be defined as language-independent representation of a given software system.

Attribute: An instance or static variable, usually called a field.

Bean: A reusable Java class which can be combined with other beans to form an application. JavaBeans are lightweight Java objects with accessor and mutator methods (getter/setter) for accessible attribute. JavaBeans must also implement the Serializable interface.

Component: Components are reusable software programs that you can develop and assemble easily to create sophisticated applications eg JavaBeans. Components have two distinct parts: specifications (or interfaces) and implementations.

Container: A runtime entity that provides services to specialized Java components. Services provided include life cycle management, security, deployment and component-specific services for EJB, Web, JSP, servlet, applet and application clients.

DBMS: A database management system (DBMS) is software that allows databases to be defined, constructed, and manipulated.

Deployment: The process whereby a software is installed into an operational environment.

EJB: Enterprise JavaBeans are heavyweight Java components that adhere to the J2EE specification.

EJB Container: An EJB container is a runtime environment for managing enterprise beans. The container hosts and manages an enterprise bean in much the same manner that a Java servlet

engine hosts a Java servlet. The EJB container instantiates and controls the enterprise beans and provides them with system-level services.

Framework: A software framework is a set of cooperating classes that make up a reusable design for a specific class of software.

IDE: IDE is a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger.

J2EE: Edition of the Java platform that targets enterprises to enable development, deployment, and management of multi-tier server-based applications. This edition encompasses many specifications, including EJB, JMS, JSP and Servlets.

JDBC: Java database connectivity. This set of application programming interfaces (APIs) provides a standard mechanism to allow Java applications access a database.

JNDI: Java Naming and Directory Interface (JNDI) defines a generic interface to accessing name and directory services.

JSP: Java Server Pages is an HTML/Java hybrid language for embedding program commands in a web page. These commands are interpreted by the web server prior to returning content to the client.

MDA: Model Driven Architecture is an OMG standard with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform.

Model: A model is an abstraction of the end (or target) system. It serves as a prototype and a proof-of-concept.

MVC (Model / View / Controller): This pattern was originally adopted in Smalltalk to support Graphical User Interfaces. Views support graphical interfacing, controllers handle interaction, and models are the application objects.

Platform: A collection of tightly integrated computing hardware, peripherals, operating system, and middleware upon which an application is built. The application provides some of its functionality by accessing services residing on the application platform through an API.

PIM: A Platform Independent Model (PIM) is one that describes the target system without any details about the specifics of the implementation platform.

PSM: A Platform Specific Model (PSM) describes the target system on its intended platform, such as J2EE, .NET, CORBA, etc.

RMI: Remote Method Invocation is Java's build-in distributed object protocol. With RMI, you can define objects that export their interface, so that they can be called remotely from other applications in a network.

Struts: Open source implementation of MVC pattern that uses servlets and JSP technologies. It consists of framework classes, helper classes, and custom JSP tag libraries, all dedicated for development of Java 2 Enterprise Edition (J2EE) applications based on the Model 2 design pattern.

Transformation: The process of converting a PIM into a PSM is called transformation

UML(Unified Modeling Language): UML prescribes a standard set of diagrams and notations for modeling object oriented systems, and describe the underlying semantics of what these diagrams and symbols mean.

XMI (XML Meta Interchange): An open information interchange model that is intended to give developers working with object technology the ability to exchange programming data over the Internet in a standardized way.

XML: E**X**tensible **M**arkup **L**anguage is an open standard of the World Wide Web Consortium (W3C) designed as a data format for structured document interchange on the web.