Fachhochschul-Diplomstudiengang
Bioinformatik
4232 Hagenberg, Austria

Development of a Web-based application for managing and analyzing real-time PCR experiments

Diplomarbeit

zur Erlangung des akademischen Grads
Diplom-Ingenieur (Fachhochschule)

eingereicht von

Stephan Pabinger

Betreuer: DI Dr. Robert Molidor, TU Graz
Begutachter: DI (FH) Peter Kulczycki

Juni 2006

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, mich auch sonst keiner unerlaubten Hilfe bedient, und diese Arbeit weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

ORT, DATUM	STEPHAN PABINGER

Abstract

Real-time PCR is one of the most important methods used in functional genomics. As one of the leading assays it has a large dynamic range, great sensitivity, and avoids the need for post-PCR processing. The requirement for analyzation, in oder to get significant results, produced a number of different analyzers, which mostly run in a separate software. Drawing conclusions is often done by comparing analyzed results of different experiments and usually consists of dealing with several printed sheets that are hard to handle, store, and organize.

This thesis describes the design and implementation of an application capable of managing and analyzing all relevant real-time PCR data. Based on the three tiered J2EE platform it is developed as a Web application using a model driven architecture approach. It allows the storage of general component descriptions, plate definitions, fluorescence measurements, and analyzer results.

Files produced by PCR machines contain information about the plate and the detected fluorescence measurements. The developed application integrates a parser that is able to read in those files and therefore provides a rapid and error-free insertion of important data. A newly developed system using Web 2.0 features allows to graphically view the inserted fluorescence measurements.

Already existing analyzers were integrated into the system and can be launched from a central point in the application. A plug-in manager ensures that new analyzers can be added to the system without having to change existing parts in the application. Results generated by analyzers can be easily compared and exported to files for further analyses. User interactions are managed using the Model-View-Controler framework Struts and Message Driven Beans are used to perform long lasting business tasks, which run in the background.

The incorporated parser and the possibility to analyze one's date using several different analyzers in combination with an user-friendly interface provides an application capable of managing, storing, comparing, and analyzing real-time PCR experiments. All these features combined in one single application make up an unique tool the certainly improves the daily work of biologists.

Keywords: *real-time PCR, qPCR, analyzers, Web 2.0, J2EE*

Kurzfassung

Real-time PCR ist eine der wichtigsten Methoden im Bereich der »functional genomics« und zeichnet sich durch den Entfall von PCR Nachbehandlungen, ihrem großen dynamischen Bereich und ihrer hohen Sensitivität aus. Da es notwendig ist detektierte Daten zu analysieren, sind eine Reihe von Analyseprogrammen verfügbar. Bestimmte Aussagen können oft nur durch das Vergleichen von Resultaten getroffen werden, was oft mit ausgedruckten Datenblättern durchgeführt wird. Diese sind meistens schwer zu handhaben, archivieren und organisieren.

Diese Diplomarbeit beschreibt das Design und die Implementierung einer Applikation die für die Verwaltung und Analyse von real-time PCR Daten erstellt wurde. Basierend auf der dreischichtigen J2EE Plattform wurde sie mit Hilfe der Model-Driven-Architecture entwickelt. Sie ermöglicht das Speichern von generellen Komponenten, Platten-Definitionen, Fluoreszenzmesswerte und Analyse-Ergebnissen.

Dateien, die durch die PCR Maschine erzeugte werden, enthalten aufgezeichnete Fluoreszenzwerte und Informationen zur verwendeten Platte. Durch die Integration eines Parsers ist es möglich, Informationen, die diese Dateien enthalten, schnell und fehlerfrei einzulesen. Ein neuartiges System, welches die Vorteile von Web 2.0 nutzt, ermöglicht die graphische Darstellung der eingelesen Fluoreszenzmesswerte.

Existierende Analysatoren wurden in das System integriert und können von einem zentralen Punkt in der Applikation gestartet werden. Ein eingesetzter Plug-in Manager ermöglicht ein einfaches Hinzufügen neuer Analyseprogramme, ohne dass existierende Teile des Systems verändert werden müssen. Durch eine übersichtliche Darstellung können analysierte Resultate einfach verglichen und mithilfe eines Exportmechanismus in Dateien abgelegt werden. Benutzerinteraktionen werden durch das Model-View-Controler framework Struts behandelt und der Einsatz von Message Driven Beans dient zur Abarbeitung lang andauernder Aufgaben.

Der eingebaute Parser und die Möglichkeit Daten von verschiedenen Programmen analysieren zu lassen in Kombination mit einer benutzerfreundlichen Oberfläche ergibt eine Applikation, die zur Verwaltung, Speicherung, Auswertung und Analyse von real-time PCR Experimenten dient. All diese Eigenschaften wurden in einem einzigen Programm kombiniert und bilden ein Werkzeug, das zweifellos die tägliche Arbeit von Biologen erleichtern wird.

Schlüsselwörter: *real-time PCR*, *qPCR*, *Analyse*, *Web 2.0*, *J2EE*

Acknowledgments

First of all, I want to thank *Dr. Robert Molidor* for the many insightful conversations during the design and development stages of the application, and for the many helpful comments and suggestions on the thesis's text.

Special thanks go to *Prof. Zlatko Trajanoski* for giving me the possibility to do my diploma thesis at the Institute for Genomics and Bioinformatics—TU Graz.

Then, I want to thank my supervisor *DI (FH) Peter Kulczycki* from the Upper Austria University of Applied Sciences in Hagenberg for his advises and design suggestions on writing this paper.

I would like to thank *Katharina Wimmer* and *Robert Smith* for proofreading my thesis and for giving me many valuable writing hints.

Finally, I would like to thank my family for always supporting me during my whole life.

Stephan Pabinger

Contents

Chapter 1

Introduction

A gene is a hereditary unit consisting of a DNA sequence that determines a particular characteristic in an organism. Since the *Human Genome Project* has discovered the complete human genome sequence, the composition of each gene is known. However the sequence alone is worthless without knowing the corresponding functionality.

Functional genomics is a field of molecular biology trying to describe genome functions by making use of the vast amount of data. Amongst other high-throughput techniques like DNA micro-arrays and mass spectrometry, real-time PCR has become a major method for answering the newly arisen questions.

Polymerase chain reaction (PCR) is a molecular biology technique for replicating DNA sequences. Real-time PCR is an advanced PCR that measures the amount of generated DNA during the duplication. By observing the amplification process one is able to calculate the original amount of DNA and therefore draw conclusions about the gene's functionality.

Data generated by real-time PCR experiments need to be analyzed before conclusions can be drawn. A lot of different algorithms have been developed that have unique ways of correcting the collected data and mostly run as a separate software tool. Moreover many comparisons between results of experiments are needed in order to get the desired answers.

All these difficulties equate to a demand for a centralized system capable of managing and analyzing real-time PCR data. The application developed here is able to administer all the relevant output from these experiments, including definitions of components, experimental setup, and results. The ability to read in files containing real-time PCR data allows a quick and error-free integration of measurements.

The analysis of real-time PCR data has always been a complex task since each software tool runs as a separate application. Giving the user the possibility to submit the generated data to a number of different analyzers from one centralized systems greatly facilitates work with real-time PCR.

This thesis has been subdivided into six[1] chapters and three appendices. Two chapters provide information about the biological and programming background. Four chapters are about the developed application describing design, implementation, and results.

Chapter 2 gives an overview about the biological background. It starts with descriptions of DNA and mRNA, introduces the reader to functional genomics, and finally describes the methodology of real-time PCR experiments. Those already familiar with this subject may skip this chapter.

Chapter 3 describes the current state-of-the-art in Web-based programming. It gives an overview of the standards used, the platform utilized, current Web technologies, and the motivation for using Web 2.0 features.

Chapter 4 describes libraries that were developed at the »Institute for Genomics and Bioinformatics—TU Graz«.

Chapter 5 provides a detailed description of the necessary requirements and presents the developed model.

Chapter 6 presents the implementation developed here. It focuses on the integration of the parser used, the analysis process, and the graphical display of measurements.

Chapter 7 discusses the results of the developed application and gives a short outlook of suggestions for further improvements.

Appendix A attached is the previously compiled »User Requirements Document«.

Appendix B shows the complete entity diagram used for developing the application.

Appendix C provided are tasks and results of the conducted usability test.

[1]Not including Chapter 1 on the preceding page (›Introduction‹)

Chapter 2

Biological Background Information

In this chapter information about the biological background of this thesis is provided. It gives insight into DNA and mRNA, describes the field of functional genomics, and explains the principle of PCR. Moreover the idea and functionality of real-time PCR is illustrated.

2.1 DNA, mRNA

Lodish *et al.* describe the deoxyribonucleic acid (DNA) in [Lodish *et al.*, 2000] as a »storehouse, or cellular library, that contains all the information required to build the cells and tissues of an organism«. Chemically it is a long polymer consisting of nucleotides. There are four types of nucleotides within the DNA: adenine (A), cytosine (C), guanine (G), and thymine (T). The DNA is usually composed of two complementary strands, which are held together through hydrogen bonds. Each base can only pair with one predetermined counterpart (A-T, T-A, C-G, and G-C) which are then referred to as base-pairs. The shape of DNA is normally a double helix, but it can also build triplex and quadruplex forms.

Genetic information is stored within the DNA and is inherited by the offspring of an organism. A gene is considered to be a segment of the DNA sequence, which codes the structure of proteins. Proteins can be accounted as the working molecules of a cell that put the program of activities, encoded by genes, into practice. They are, amongst other things, involved in cell regulation, metabolism, and hormones and play a big part in the immune system.

The information flow from the DNA to a protein can be summarized as »DNA makes mRNA makes proteins«. Messenger ribonucleic acid (mRNA) is a single strand of nucleotides consisting of A, C, G, and uracil (U), which is the counterpart of thymine. It is build up during *transcription*, which is the process whereby information, contained in a section of DNA, is transferred to a newly assembled piece of mRNA. The newly build mRNA is now an exact complementary copy of the gene sequence stored in the DNA. These mRNAs are then transported to ribosomes which translate them into proteins.

2.2 Functional Genomics

In 2003 the Human Genome Project finally announced the completion of their work. Over the course of thirteen years it discovered the full human genome sequence which consists of approximately 3 billion base pairs and 20,000 - 25,000 genes (see [Human Genome Project, 2004]). Since about 50% of these discovered genes have known functions, the task is now to understand these functions and the interplay of genes with proteins [Sebastiani *et al.*, 2003]. The goal of functional genomics is to answer these questions by developing and applying technologies that take advantage of the growing amount of sequence information [Fields *et al.*, 1999]. The term genomics refers to the analysis of genomes whereas functional genomics describes the global approaches to understand the functions of genes and proteins.

Functions of genes can be determined by the activity of the corresponding protein whereas this process of becoming active is called *gene expression*. Because this procedure consists of copying DNA code into mRNA molecules, the abundance of produced mRNA is a measurement for gene expression.

Functional genomics uses a number of different methods. The two most prominent practices are the micro-array technology and the polymerase chain reaction (described in Chapter 2.4 on page 8). Moreover mass spectrometry and electrophoresis are used to find answers to questions provided in the field of functional genomics.

All these methods try to answer fundamental questions in molecular biology. By applying them, one wants to identify genes that are differentially expressed in two conditions. Further they allow the study of the temporal evolution of gene expression profiles, for example to survey the expansion of a tumor including the development of metastases. Moreover general regulatory mechanisms of cells can be analyzed, which can give insight into the process of adaptation to different environmental condition.

The *micro-array technology* allows to »simultaneously measure the relative expression level of thousands of genes within a particular cell population or tissue« [Sebastiani *et al.*, 2003]. Its major concepts are hybridization and reverse transcription.

Generally *hybridization* is the process of pairing two single strands of either DNA or RNA according to their bases. Double stranded DNA separates at a characteristic temperature and binds back to their counterparts when the temperature is lowered again. *Reverse transcription* is the process of producing a complementary strand of a mRNA which has been isolated out of a cell. This reverse-transcribed copy is called copy or complementary DNA (cDNA). Moreover double-stranded cDNA can be again reverse-transcribed into a complementary copy called cRNA. The collection of cDNAs from cellular mRNA builds up the cDNA library of a cell. By using hybridization it is possible to pair a cDNA with an mRNA, given that they have complementary sequences.

By applying the micro-array technology one tries to find out what expression level certain genes have in a particular cell. Therefore a number of molecules, which bind to the molecules of the cell, are placed on a special prepared carrier. Generally there are two methods realizing the binding procedure between the cell molecule, called *target*, and the tethered sequences, called *probes*:

- By hybridizing a labeled cDNA, which represents the cellular mRNA, to cDNA sequences
- By hybridizing a labeled cRNA to short specific segments, called synthetic oligonucleotides, that are also referred to as oligos

In both cases the targets are labeled with fluorescent[1] dyes and the abundance of active mRNA is measured by the emission intensity of the fluorescent dye signal. The outcome of this analyzing process is a digital image which is produced by a scanning laser device. This image has to be processed by specialized programs that translate the intensity of each hybridization signal into a table with numerical measures. After applying numerous bio-informatic algorithms, that, e. g. remove the background noise, the data is ready to be analyzed.

2.3 Polymerase Chain Reaction

The polymerase chain reaction (PCR) is a rapid, inexpensive, and simple molecular biology technique for replicating specific DNA fragments of small amount [Elrich, 1989]. It is done without using a living organism and amplifies DNA molecules in an exponential manner. PCR is commonly used in medical and biological research

[1]Fluorescence is a luminescence (light that is not generated by high temperatures alone) in which the molecular absorption of a photon triggers the emission of a lower-energy photon with a longer wavelength. The energy difference between the absorbed and emitted photons ends up as molecular vibrations or heat, which is visible as light.

laboratories for a variety of tasks, such as the detection of hereditary diseases, the cloning of genes, paternity testing, and DNA computing.

Basic components that are used during a PCR are:

DNA template This can be either a DNA or a cDNA molecule containing the region of the fragment to be amplified.

Primers Primers are short, artificial DNA strands that determine the DNA fragment that should be amplified. Usually they are not longer than 50 base pairs and its nucleotide sequence is complementary to the beginning and ending of the DNA fragment to be amplified. During the PCR they bind to the DNA template and act as starting points for the DNA polymerase, which synthesise the new DNA strand. The determination of a primer's length is crucial for the design of a PCR experiment. Primers that are too short may bind to several positions resulting in non-specific copies. On the other hand the length of primers is limited by the temperature required to melt it (defined as the temperature at which half of the primer binding sites are occupied), which is proportional to the length of the primer. All these considerations make primer design an important part of a PCR experiment which is supported by various computer programs (see [Abd-Elsalam, 2003]).

DNA polymerase A DNA polymerase is a protein that binds to a single DNA strand and creates the missing complementary strand of the template. Unfortunately, during the PCR process human DNA polymerase is destroyed when it is heated over 65°C. The Taq polymerase, named after the hot-spring bacterium *Thermus aquaticus*, lives in geysers at a temperature of over 110°C. Therefore it can survive temperatures over 90°C which makes it an ideal polymerase for PCR experiments. However it has a relative high error rate (1 in 10,000 bases) which can not be corrected due to the lack of a correction mechanism.

dNTPs Deoxynucleotides-triphosphates (dNTPs) are used by the DNA polymerase to assemble the new strand. There are four types of dNTPs (dATP, dCTP, dGTP, dTTP), one for every nucleotide occurring in the DNA strand.

A PCR normally exists of three steps, forming a cycle, that are repeated between twenty to thirty-five times. The steps in each cycle are:

Denaturation This step is usually performed at 95°C about 1-2 minutes. Due to the high temperature the double-stranded DNA melts apart into two single strands allowing the added primers to bind to them. To ensure that both the template DNA and the primers are completely separated and single strand only, a long denaturation step is often performed right before the initial cycle.

Annealing After denaturation the mixture is cooled down to 45-60°C and stays at this temperature for 1-2 minutes. This allows the added primers to bind to the complementary regions of the template DNA. The temperature depends on the primers and is normally 5°C below the melting temperature of the primers.

Elongation The last step, also called *extension*, is carried out at 72°C for about 1-2 minutes given that the Taq polymerase has been used. The chosen time is dependent on the used DNA polymerase whereas the temperature is based on the length of the DNA fragment to be amplified. During this step the DNA polymerase binds to the template DNA, where the primers are attached, and builds a complementary strand. It extends the primers by adding nucleotides in the order in which they can pair. By extending the final elongation step one can ensure that any remaining single stranded DNA is completely copied.

Figure 2.1 displays the steps performed during a normal PCR cycle.

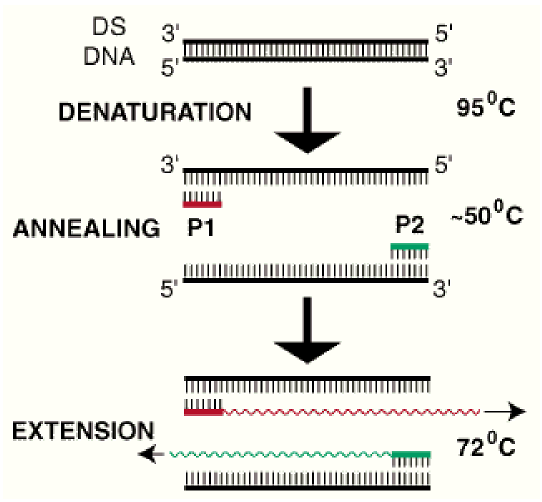


Figure 2.1: The three steps that are normally performed during each cycle are shown. First the DNA is denatured which enables primers to bind to the single strands during the annealing step. Now the DNA polymerase can extend the missing second strand and the cycle starts again. This figure was taken from [Florida Museum of Natural History, 2006].

PCR experiments are carried out in a machine called *thermocycler*. These machines can carry plates consisting over 300 wells each containing a separate reaction, which greatly increases the number of reactions that can be done simultaneously. Its major function is the up and down regulation of the temperature which is needed to perform a PCR.

Being one of the most frequently performed experiments in today's laboratories PCR is part of many modern analysis processes.

- **Genetic fingerprinting** is a technique used to identify a person by comparing his or her DNA with a given sample. It is applied for, i. e., forensic purposes and paternity testing.
- **Cloning genes** describes the process of isolating a gene from one organism and then inserting it into another organism. PCR is used to amplify the gene which can then be inserted into the other organism, where it is easier to study its functionality.
- The **comparison of gene expression** is used to estimate changes in the amount of a gene's expression of, e. g. differently treated organisms.

Although PCR is a very robust laboratory technique it has some problems and limitations.

Non specific priming As already mentioned, the design of the appropriate primers is crucial to the success of a PCR experiment. Primers may bind to positions where they are not supposed to, leading to unwanted sequence amplifications. The non specific binding of primers can be decreased by choosing the proper annealing temperature although it can never be completely extinguished.

Size limitation Using PCR it is possible to amplify DNA sequences up to three thousand base pairs. Longer sequences causes the DNA polymerase to fall of due to its limited life time.

2.4 Real-time PCR

Wong and Medrano state in [Wong and Medrano, 2005] that »real-time PCR has become one of the most widely used methods of gene quantitation because it has a large dynamic range, boasts tremendous sensitivity, can be highly sequence-specific, has little to no post-amplification processing, and is amenable to increasing sample throughput«. It is a technique that collects data during the PCR process using different fluorescent chemistries and therefore puts together detection and amplification in one single step. Because real-time PCR is used to quantitate templates it is also referred to as quantitative PCR (qPCR) or quantitative real-time PCR (qRT-PCR). Real-time reverse transcription PCR (real-time RT-PCR) is a variety of real-time PCR where RNA instead of DNA is used as template.

Data that is generated during the PCR can be used to determine the point at which fluorescence intensity is greater than the background fluorescence signal. This point, commonly referred to as C_t or crossing point (CP), gives information about the initial quantity of target DNA in the starting material and is inverse proportional to the used amount. The possibility to measure data right as it occurs allows the determination of the initial quantum without the need for post-amplification manipulation. Combined with the dynamic range of 7 to 8 log orders of magnitude real-time PCR has revolutionized the field of measuring gene expression. Drawbacks are the requirement for expensive equipment and reagents and the appliance of normalization techniques to achieve accurate results.

A typical PCR has four major phases. It starts with a *linear ground phase* at which the fluorescence emission is below the background. The *early exponential phase* describes the stage of a PCR where the fluorescence emission is significantly higher than the background. The corresponding cycle, determined through a set threshold, is the C_t or CP value and is used to calculate experimental results. In the upcoming *log-linear or exponential phase* the PCR product doubles in each cycle, provided that ideal reaction conditions are available. Finally the *plateau phase* signals the stage at which reaction components become limited and the fluorescence intensity is for no further need.

Real-time PCR can either be performed as a two-step reaction, where the reverse transcription of RNA happens in one tube and the actual PCR amplification occurs in another tube, or as a one-step reaction combining these two steps into one single tube. Two-step real-time PCRs ensure that always the same template amount is used when different PCR assays are performed on dilutions of a single cDNA. However there are increased opportunities of DNA contamination using this type of protocol. One-step real-time PCRs are described as less sensitive than two-step protocols but they minimize experimental variation, because all enzymatic reactions occur in a single tube.

Generally there exist two types of real-time quantitation.

Absolute Quantitation The aim of absolute quantitation is the determination of concentrations using a *standard curve*, which provides a linear relationship between C_t value and initial amount of total DNA. This curve is calculated by serially diluted standards of known concentrations and enables the determination of the concentration of unknown samples. However this method requires that all standards and samples have approximately equal amplification efficiencies to produce accurate results.

Relative Quantitation It uses either an external standard or a reference sample, called calibrator, to measure the changes between samples, whereas results using a calibrator are expressed as target/reference ratio.

In contrast to relative quantitation, absolute quantitation allows the comparison of multiple plates or runs. C_t values from relative quantitation are only accurate when compared within one PCR. Unfortunately absolute quantitation is considered to be more labor-intensive, because of the need for reliable standards.

Due to the diminution of PCR components and the reduction of DNA polymerase activity during a PCR, amplification *efficiencies* are not ideal and calculations may therefore overestimate the starting concentration. The amplification efficiency of the reaction varies from being relatively stable in the early exponential phase and gradually declining to zero. Consequently it is important to calculate the efficiency of a PCR experiment which is most accurate when the analysis is based on raw data.

Material acquired from different individuals usually varies in tissue mass or cell number, experimental treatment, or RNA quantity. *Normalization* tries to correct these sample-to-sample variations of gene expression data. This is done by using a control gene that should have the same expression regardless of the sample's origin or treatment. One example of controls are housekeeping genes, which have stable expressions and have been employed as controls in gene expression assays. Using multiple housekeeping genes and calculating a normalization factor from the geometric mean of their expression can improve the quality of normalization and is considered as the most accurate method [Wong and Medrano, 2005].

Major vendors of real-time PCR systems are:

- **Applied Biosystems** Products: ABI PRISM 7000, 7700 Sequence Detection System; 7900HT Fast Real-Time PCR System.
- **Roche** Products: LightCycler 480 Real-Time PCR System, LightCycler 2.0 System
- **Bio-Rad** Products: MyiQ Single-Color Real-Time PCR Detection System, iCycler Thermal Cycler

Chapter 3

Software Development Technologies

This chapter gives insight into the various standards defined by the Object Management Group. The Java 2 Platform, Enterprise Edition and its sundry components are described. Some established and brand new Web technologies are specified, which are followed by a detailed description of code generation methods. Finally, used development tools and the JFreechart library are discussed.

3.1 Standards

All standards presented in this section have been published by the Object Management Group (OMG). It is an »open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications« [Object Management Group, 2006]. Their membership includes nearly all of the large companies in the computer industry and any company may participate in the OMG standards-setting process.

3.1.1 Model Driven Architecture

In 2001 the OMG released their first version of the Model Driven Architecture (MDA) framework. Designed as an approach for using models in software development its three main goals are interoperability, reusability, and portability which are achieved by separating the business and application logic from the underlying platform [Miller and Mukerji, 2003].

MDA is based on the following OMG standards: the Unified Modeling Language, the Common Warehouse Metamodel, the Meta-Object Facility, and the XML Metadata Interchange. A Platform Independent Model (PIM) which has been generated by using these standards can be realized through the MDA on any platform.

MDA stages procedures and tools for:

- defining a platform independent system
- defining platforms
- picking a particular platform for the system
- transforming the system specifications into one for a particular platform

3.1.2 Unified Modeling Language

The Unified Modeling Language (UML) is used to specify, document, and visualize models of software systems. The design of a good model has always been an important part of software development. It lets one work at a higher level of abstraction, it helps concentrating on end-user needs and provides a big picture of the software all before a single line of code has been written [Object Management Group, 2005].

Currently UML 2.0 defines thirteen types of diagrams that are divided into three categories:

Structure Diagrams are used to model the static structure of the system. They focus on elements of a system and the relationships between them. Important diagrams are the Class Diagram, the Component Diagram, the Package Diagram, and the Deployment Diagram

Behavior Diagrams describe behavioral features of a system or business process. This includes Activity, State Machine, and Use Case Diagrams.

Interaction Diagrams are concentrating on object interactions and are derivatives of Behavior Diagrams. Essential diagrams are the Sequence Diagram, the Communication Diagram, and the Timing Diagram.

3.1.3 XML Metadata Interchange Format

The Object Management Group in [Object Management Group, 2006] defines the XML Metadata Interchange Format (XMI) as follows:

> XMI is a model driven XML Integration framework for defining, interchanging, manipulating, and integrating XML data and objects. XMI-based standards are in use for integrating tools, repositories, applications, and data warehouses. XMI provides rules by which a schema can be generated for any valid XMI-transmissible MOF-based metamodel.

Meta-Object Facility (MOF) is an extensible model-driven integration framework for defining, manipulating, and integrating metadata and data in a platform-independent manner. Its standards are used for integrating tools, applications, and data.

The original intention of XMI was to enable an easy interchange between UML-based modeling tools. However it can also be applied to software and databases. Currently two different XMI versions are available whereas there is one main difference between them: Version 1.2 uses DTDs whereas Version 2.0 uses XML Schemas to validate the XML document.

3.2 Java 2 Enterprise Edition

The Java 2 Platform, Enterprise Edition (J2EE) technology provides a component-based approach to the design, development, assembly, and deployment of enterprise applications [Bodoff *et al.*, 2001]. It uses a muti-tiered distributed application model, meaning that application logic is divided into components according to its function. These components can be installed on different machines corresponding to their J2EE tier they belong to.

The J2EE defines four different components:

- Client tier components running on the client machine
- Web tier components running on the J2EE server
- Business tier components running on the J2EE server
- Enterprise information system (EIS) tier software running on the EIS server

Although four different tiers are defined, J2EE applications are generally referred to as three-tiered, because they are usually distributed over three locations. A component can only be executed if it has been assembled into a J2EE module and deployed into its container. Containers are the gateways between a component and the platform-specific functionality that supports the component. Important containers are:

- **Enterprise JavaBeans container** It administers enterprise beans and runs on the J2EE server.
- **Web container** Running on the J2EE server it manages the execution of JavaServer Pages sites and servlet components.
- **Application client container** It handles the execution of application client components and runs on the client.

A J2EE module is an assembly of one or more components including their related files and an XML based deployment descriptor that specifies how to assemble and deploy the unit.

3.2.1 Client Tier

The J2EE platform supports many types of clients whereas the coarsest differentiation is between Web-based and non-Web-based clients [Kassem and the Enterprise Team, 2000]. Web-based J2EE applications use Web browsers to download Web pages and applets to the client machine. Web sites can either be static or dynamic Hypertext Markup Language (HTML), Wireless Markup Language (WML) or Extensible Markup Language (XML). In contrast to Web-based clients, application clients run on a client machine and typically have a graphical user interface created from Swing or Abstract Window Toolkit (AWT) Application programming interfaces (API)s. They directly access enterprise beans running in the business tier [Bodoff *et al.*, 2001].

3.2.2 Web Tier

In a J2EE application the Web tier is mainly responsible for creating (dynamic) content. It processes HyperText Transfer Protocol (HTTP) GET and POST requests and translates them into business logic language. Using the results from the business tier it typically creates HTML or XML content and sends it back to the client. Moreover it manages interactions between clients and application business logic and determines which page is displayed next. Web components are either servlets or JSP sites or a combination of both technologies.

3.2.2.1 Servlet

Initially Java servlets were designed to gratify the need for dynamic Web content. They are written in the Java programming language and extend the functionality of a Web server. Hosted in a Web container they are accessed via the request-response programming model. Servlets are platform independent and can output any content type, usually HTML or XML.

3.2.2.2 JSP

A JavaServer Pages (JSP) site is a textural document that describes how to create a response object from a request object for a given protocol [Roth and Pelegri-Llopart, 2003]. It services requests as a servlet which is generated and compiled whenever a request is mapped to a new JSP page.

Generally a JSP page consists of elements and template data. Elements are processed by the JSP container, whereas template data is everything not known to the JSP translator.

Features of JavaServer Pages are:

Directives Directives are messages for the JSP container that influence the translation of a JSP page into the servlet. They do not produce any output but define which additional page should be imported, which tag libraries are used and what is included into the JSP page.

Standard Actions A certain tag with the prefix *jsp* is used to define standard actions. They allow one to perform certain tasks like instantiating objects or accessing JavaBeans without having to write Java code.

Scripting Elements They are used to create and access objects and to perform computations that affect the content generated. There are three classes of scripting elements: *declarations*, *scriptlets* and *expressions*. Declarations are used to declare variables and methods in the scripting language that are available to all other scripting elements. Scriptlets contain an arbitrary code fragment, and an expression element consists of a scripting language expression that is evaluated and commonly converted into a string.

Tag Extension mechanism A tag file is a source file that abstracts a segment of JSP code and makes it reusable via a custom action. Tag Libraries are collections of tag files that encapsulate functionality to be used from within a JSP page. The tag extension mechanism allows users to add custom tags that can be used in any further JSP page.

Template content It is directly transformed into code and typically uses HTML or XML elements. Template data does not affect the dynamic content and is often used to design the layout of the Web page.

3.2.3 Business Tier

The business tier is, in a multi-tier J2EE application, also referred to as the Enterprise JavaBeans (EJB) tier. It is responsible for managing business logic, transactions, concurrency control, and security.

3.2.3.1 Enterprise JavaBeans

An enterprise bean is a server-side component that runs in the EJB container and encapsulates the business logic of an application. Its important role lies in the linkage between the enterprise information system and the client tier, whereas the client always communicates via the exposed component interface [Roman *et al.*, 2002].

There are three different kinds of enterprise beans:

Session beans are used to model business processes or task-flows. Their lifetime is generally equivalent to the length of the client's session and they are non-persistent. Each session bean keeps conversations with a client which are composed of a number of method calls between the client and the bean.

Based on the type of conversation one can differ two subtypes of session beans. *Stateful session beans* keep their conversational state on behalf of an individual client and are therefore designed to serve business processes that span multiple method requests or transactions. *Stateless session beans* keep their conversational state only for a single method call and can therefore serve more than one client.

Entity beans are persistent objects that know how to persist themselves permanently to a durable storage. They store data as fields and have methods associated to access and manipulate these fields.

Message driven beans are special EJB components that can receive Java Message Service (JMS) messages and cannot be directly accessed by a client. JMS is a Java Message Orientated Middleware API for sending and receiving messages between two or more clients. The messages are processed asynchronously and have no return value.

3.2.4 Enterprise Information System Tier

The enterprise information system (EIS) tier covers enterprise infrastructure systems, namely, mainframe transaction processing, database systems and other legacy information systems. Moreover it deals with enterprise information software.

3.2.5 J2EE Patterns

A Design Pattern is a general solution to a commonly-occurring problem. The J2EE Patterns provide a collection of tested and proven answers for J2EE-based problems. Amongst others, Alur *et al.* specify in [Alur *et al.*, 2001] the following patterns:

Business Delegator hides the underlying implementation details of the business service which reduces the coupling between presentation-tier clients and the system's business service. It uses a component called Lookup Service which locates the actual business services.

Session Facade manages and abstracts the underlying business objects and provides a service layer that exposes only the required interfaces. Thus the whole functionality of an application is accessible through one bean.

Service Locator provides a single point of control to centralize service object look-ups. This reduces client's complexity, improves performance by providing a caching facility, and abates redundant lookups, because the same client or other clients can reuse the Service Locator.

Transfer Objects (also referred to as *Value Objects*) are used to assemble related attributes. By transporting Transfer Objects instead of single attributes from an enterprise bean to its client, the number of remote calls and its associated overhead is reduced.

Value List Handler is used to control a search for certain data, caches the results, and provides the results to the client in a result set. The client can access the result set through an offered iterator and obtains Transfer Objects from the cached list.

Data Access Objects (DAO) abstract and encapsulate all access to the data source and manage the connection with the data source to obtain and store data. In combination with the Abstract Factory pattern, described by Gamma *et al.* in [Gamma *et al.*, 1995], only an *abstract* DAO factory object is provided that can construct various types of concrete DAO factories. This allows to change underlying data access mechanisms, because each factory supports a different type of persistence storage implementation.

3.3 Web Technologies

Todays Web applications have reached a level of complexity that makes it impossible to create them without a solid design and a good knowledge of current technologies. Frameworks provided by various organizations try to help and guide developers building a reliable, easy to maintain, and up-to-date Web application. Almost every year a new technology is invented that revolutionizes the Internet and comes up with a new way of using the World Wide Web.

The preceding section describes one of the most used Web application frameworks and will give insight into the newest and currently most discussed Web technology—Ajax. Traditional Web applications, also referred to as Web 1.0, have a very distinct request-response model that greatly differs from rich desktop applications. Web 2.0 tries to dispense the visible back and forth and introduces things, like drag-and-drop, zooming, and on-the-fly evaluation that have only been known from standalone programs. Furthermore a Java library that improves Ajax will be explained.

3.3.1 Struts

The Struts framework implements the popular Model 2 Architecture framework which provides a unified infrastructure upon which Internet applications can be based. Generally there exist two approaches for building Web applications using JSP technology called JSP Model 1 and Model 2 architectures.

The first one lets the JSP page handle all of the processing of the request and uses it for displaying the output to the client. In contrast the second approach tries to use the JSP page only for displaying the content by including servlets that handle requests and perform front-end processing. By using the Model 2 architecture a clear separation of the business logic, presentation, and request processing is given which is often referred to as a Model-View-Controller (MVC) pattern [Cavaness, 2002].

The Model represents data objects that are manipulated and presented to the user. Serving as the screen representation of the Model, the View presents the current state of data objects. The Controller defines the way the user interface reacts to the user's input and is the component that manipulates the Model. Figure 3.1 on the next page shows a common illustration of the MVC paradigm. The View sends user actions to the Controller which determines the next View or informs the Model that the state has changed. The Model processes queries from the View and sends notifications to the View that the state has changed.

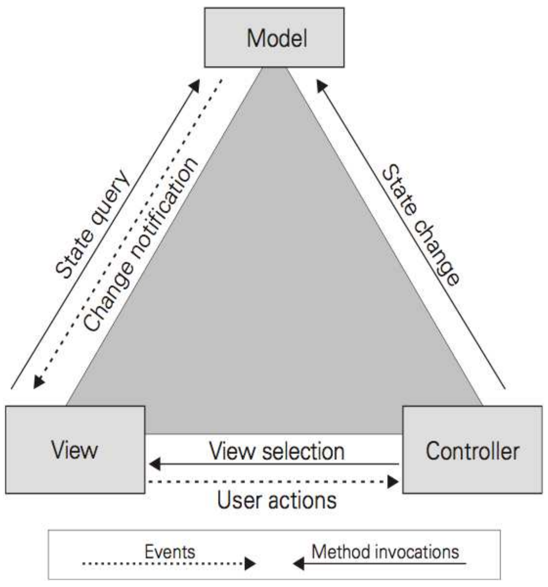


Figure 3.1: Shows a common illustration of the MVC paradigm. The Controller manages user actions sent by the View, informs the Model of a state change, and determines the next View. Queries sent by the View are processed by the Model which informs the View about state changes. This figure was taken from [Husted *et al.*, 2003].

3.3.1.1 The Controller

Struts uses a Java servlet to act as a controller that receives input from the client, invokes business operations, and coordinates the views returned to the client. It extends the `ActionServlet` and processes and delegates requests to Struts Action classes according to the specified mapping. The Action class perform functions, such as authorization, logging, and invoking business methods and sends back the response to the client.

3.3.1.2 The View

The presentation of data is done by a combination of HTML sites, JSPs, custom tag libraries, and ActionForm objects. JSPs make up the majority of View components and are used to present data with the help of custom tag libraries. Included in the framework are the Bean, HTML, Nested, Logic, and Tiles tag libraries [Struts, 2006]. ActionForm objects are used to pass client data back and forth between the user and the business layer.

3.3.1.3 The Model

In a multi-tier application the Model view is normally covered by EJBs. Due to the fact that the Struts framework does not provide any specialized model components, JavaBeans—commonly referred to as data transfer object or value objects—are returned from session beans and used within the Web tier.

3.3.1.4 Features

The Struts framework provides many features that help building up state-of-the art Web applications. Three very important parts are:

Tag Libraries Struts takes advantage of the JSP tag library function and delivers a set of optimized tag libraries. Using the *bean* and *nested* library allows one to access JavaBeans. The *html* library works closely with ActionForms and renders HTML tags. Conditional output generation, looping, and application flow management can be realized with the *logic* tag library.

Validation User input can be validated by a defined method in the various ActionForms. A better way is to move it out into an XML file that can be modified independently of the code. This technique is realized by the Struts Validator framework. Moreover it provides many standard routines that can be easily extended.

Internationalization Cavaness defines in [Cavaness, 2002] internationalization as »the process of designing your software ahead of time to support multiple languages and regions, so that you don't have to go back and reengineer your applications every time a new language or country needs to be supported.« Struts determines the preferred locale and uses it to look up text and resources from resource bundles. They are ordinary text files and can be accessed both in JSPs and Java code.

3.3.2 Ajax

Ajax stands for »Asynchronous JavaScript and XML« and describes a Web development technique that tries to eliminate major disadvantages of common Web applications in comparison with desktop programs [Telerik Corporation, 2005].

Major drawbacks of common Web applications are:

Poor Interactivity After each interaction with the server the user has to wait until the page has been reloaded.

Inefficiency Sending the complete form data from the browser to the server and receiving the full HTML markup of the page is in most cases highly inefficient, since only a small part of the interface is actually changing.

Low Usability Having to reload always the complete page when only minor changes have been made may confuse the user.

Simplistic Interfaces Because of the stateless HTML protocol and the need to transfer the complete page to the server Web applications tend to have very jejune graphical interfaces. Complex user interactions and on-demand updates are not realizable which are a trademark of common desktop applications.

Ajax tries to improve all these weaknesses by trying to make the communication with the server asynchronous, meaning that data is transfered and processed in the background. It uses a JavaScript engine that is loaded on the client during the initial page load and works as a middleman sending only relevant data to client and server [Zammetti, 2005]. Data is sent in form of XML using the `XMLHttpRequest` component which is a JavaScript object. Microsoft's Internet Explorer uses an ActiveX object to create the `XMLHttpRequest` component whereas other browsers can directly instantiate it [McLaughlin, 2006]. The upcoming Internet Explorer 7 is supposed to natively support the `XMLHttpRequest` object.

Moreover this engine is also responsible for rendering the interface the users sees and handles things that do not involve the server, such as simple data validation or small navigation tasks.

Figure 3.2 on the following page shows a diagram which displays a complete lifecycle of an Ajax Web form. An initial HTTP request is sent by the client to the server. The servers transmits back the HTML response and initiates the start of the JavaScript Ajax engine on the client. If the client needs data from the server it uses JavaScript calls that are processed by the Ajax engine which uses the `XMLHttpRequest` object to communicate with the server. This conversation is done asynchronously in the background and as soon as the Ajax engine receives an answer it uses JavaScript to manipulate the Web page.

Advantages of Ajax are:

Interactivity Tasks like updating or deleting records, expanding Web forms, or returning simple search queries are mostly executed on the client without the need of reloading the full page. This means that the user does not have

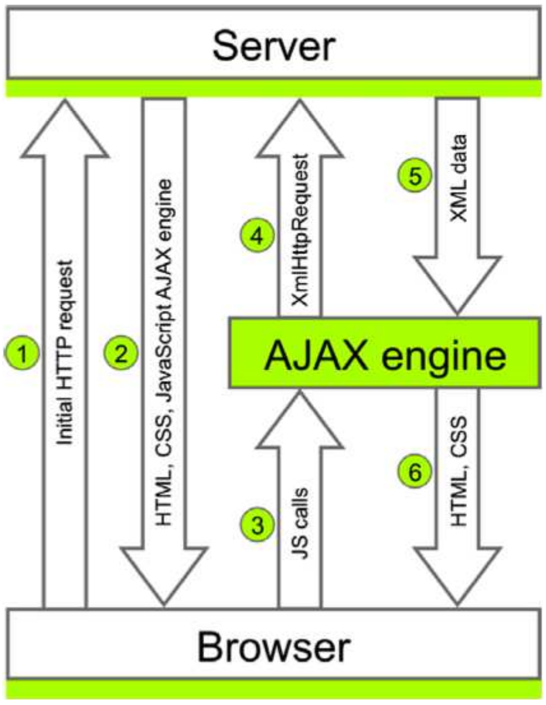


Figure 3.2: Displayed is a complete life-cycle of an Ajax-enabled Web form. The client sends an HTTP request to the server (1) which is responded and inducts the start of the JavaScript Ajax engine (2). Further requests are managed by the Ajax engine which processes JavaScript calls (3), communicates with the server using the `XMLHttpRequest` object (4), and updates the page (6) using XML data received from the server (5). This figure was taken from [Telerik Corporation, 2005].

to stare at a blank screen waiting for a response which dramatically increases usability.

Efficiency Only small request and responses are sent between the client and the server which permits the development of more interactive applications.

Portability Due to the fact that Ajax uses only features that are established and well-documented it runs in all major browsers.

Problems that arise when using Ajax:

JavaScript Developing a Web application which uses Ajax requires excessive use of JavaScript that has the drawbacks of a scripting language and is very difficult to debug.

Breaking the Page Paradigm By using Ajax a Web page no longer holds constant data which brings up two important problems: Neither the back button nor book marking will work any longer. Since most users are very conversant with both techniques, developers have to think of different mechanisms for overcoming these two issues.

Feedback notification Without the usual occurrence of a blank screen between user interactions it is important to find other ways to inform the user that something is going to change. Otherwise users will get confused and are likely to handle the software in a way it was not intended to be.

Browser that currently support Ajax:

- Microsoft Internet Explorer version 5.0 and above
- Gecko-based browsers like Mozilla, Firefox, Camino, Netscape
- KHTML API based browsers like Konqueror version 3.2 and above, Apple Safari version 1.2 and above
- Opera version 8.0 and above

3.3.3 Direct Web Remoting

DWR, standing for »Direct Web Remoting«, is a Java open source library which greatly improves programming of Ajax Web sites allowing to use Java functions on a Web server directly in a Web page [Getahead, 2005].

It consists of two main parts:

Java Servlet running on a server to process requests and sending back responses.

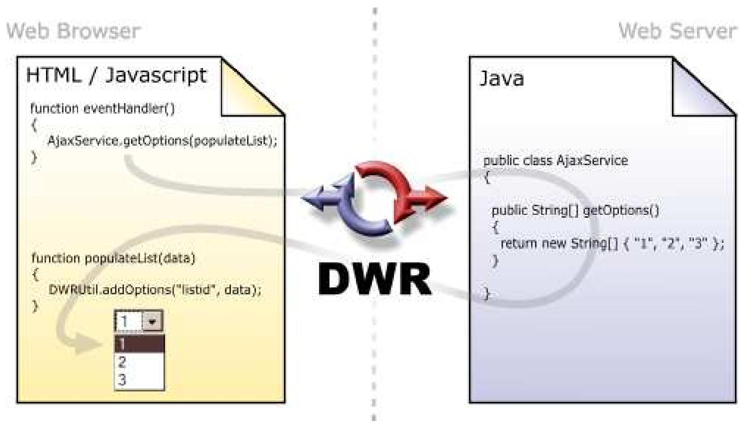
JavaScript used to send requests to the Servlet and for dynamically updating the Web page.

The core idea behind Direct Web Remoting is to dynamically generate JavaScript that is based on real Java classes. A so called »eventHandler«, representing an AjaxService, is created which matches server-side code. It is used to handle all the converting of parameters and return values between Java and JavaScript.

By Using DWR a Web application developer is not limited to the functionality JavaScript provides, but can take full advantage of the Java programming language. This includes complex calculations, calling of business methods, generating content, and the usage of the Java Exception handling mechanism. Instead of having to write complex mapping interfaces and algorithms, all these methods can be accessed using the JavaScript object that is created by DWR.

Furthermore DWR can be used in combination with a Java servlet and provides support for Struts Actions.

Figure 3.3 shows how DWR is used to dynamically alter the content of a selection list. First it listens to a JavaScript event like `onchange`, which calls a function of the JavaScript object provided by DWR. This function is executed on the server and sends back the altered list as an XML object. DWR processes this object and makes it available for JavaScript which uses it to change the selection list. All these activities are done asynchronously in the background and do not require a page reload.



```
function eventHandler()
{
  AjaxService.getOptions(populateList);
}

function populateList(data)
{
  DWRUtil.addOptions("listid", data);
}
```

```
public class AjaxService
{
  public String[] getOptions()
  {
    return new String[] { "1", "2", "3" };
  }
}
```

Figure 3.3: This diagram shows how DWR can alter the content of a selection list as a result of some JavaScript event. This event calls a function provided by DWR which is a mapping of the actual Java function on the server. DWR wraps the request, sends it to the server where it is executed, and processes the response object. Another JavaScript function is used to alter the Web page according to the received data. This figure was taken from [Getahead, 2005].

3.4 Code Generation

Writing and maintaining large business software systems has always been a very labour-intensive and error prone job and includes highly repetitive and monotonous tasks. The automation of software production using a code generator speeds up software development, improves quality, and reduces maintenance efforts.

Generally there are two types of code generators. A *closed* generator offers a fixed and optimized solution that can not be modified or adopted. On the contrary an *open* generator has no limitations in the generated code, but provides only basic functionality that often has to be adopted and extended by the user.

A UML based code generator reads a UML model and generates platform specific code out of it. An example of a UML based code generator is AndroMDA.

3.4.1 AndroMDA

AndroMDA is an open source code generation framework based on the MDA paradigm. By taking one or many models (normally UML models stored in XMI) in combination with AndroMDA plug-ins, the user is able to produce any kind of custom component. The framework is not limited to a number of specific programming languages, because by writing or customizing plug-ins it can produce any file in any language needed [AndroMDA, 2005b].

Currently AndroMDA is mostly used by developers working with J2EE technologies. It is able to generate code for Hibernate, EJB, Spring, Web Services, and Struts and can setup a new J2EE project from scratch, given that a UML model exists. AndroMDA is started using Maven or Ant, although it is recommended to use Maven, because most tools come with a Maven plug-in. Apache Ant is a build tool that uses Java classes and XML-based configuration files. It calls out a target tree, where sundry tasks get executed [Apache Software Foundation, 2006a]. The Apache Software Foundation defines in [Apache Software Foundation, 2006b] Maven as »a software project management and comprehensive tool«. It can manage build processes, reporting, and documentation from a central piece of information.

Figure 3.4 on the following page displays how Ant generates a project using AndroMDA and XDoclet (see Chapter 3.4.2 on page 28).

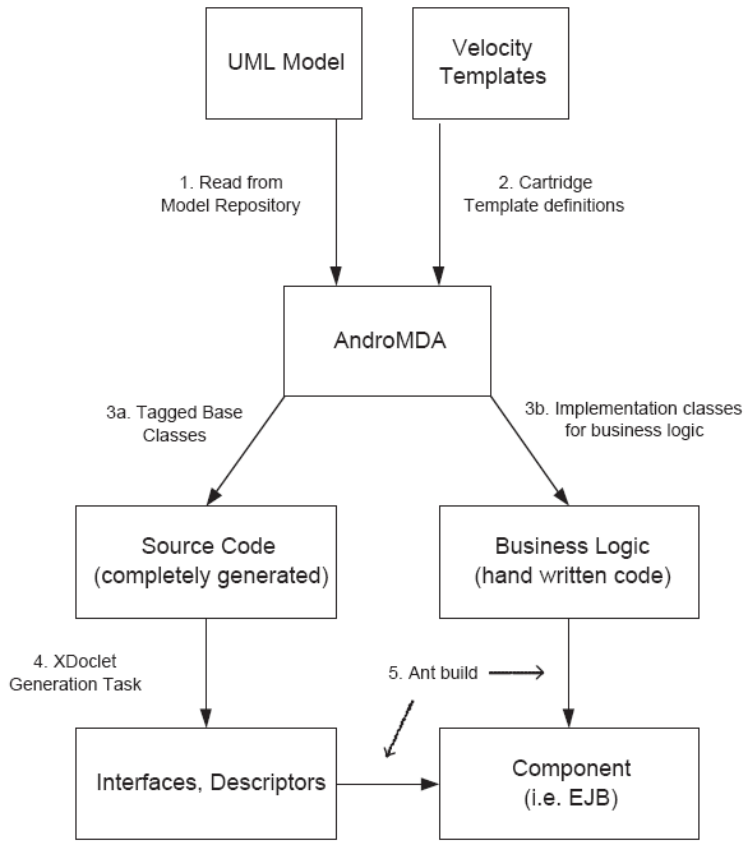

UML Model
Velocity Templates
1. Read from Model Repository
2. Cartridge Template definitions
AndroMDA
3a. Tagged Base Classes
3b. Implementation classes for business logic
Source Code (completely generated)
Business Logic (hand written code)
4. XDoclet Generation Task
5. Ant build
Interfaces, Descriptors
Component (i.e. EJB)

Figure 3.4: Presented is a flowchart of a build process using Ant, AndroMDA, and XDoclet. First AndroMDA reads in the UML model and the cartridge definitions. Next it processes the model and generates on the one hand source code that does not need manual adaptation and on the other hand source code that needs to be modified. Now XDoclet generates interfaces and deployment descriptors out of the generated code. Finally Ant builds the source files and creates a component. This figure was taken from [Truskaller, 2003].

Major advantages of AndroMDA are:

- The project model does reflect the implemented code.
- It is community driven guaranteeing that there will always be fast and free support.
- It is *open source* and *modular* which helps adapting it to new projects.

The actual code generation process starts with mapping the Mental-Model (MM) to a Platform Independent Model (PIM) using a more formal language such as UML.

This PIM is not bound to any existing platform, can be re-modeled at any time, and is a good way to communicate ideas to others. Next this model is transformed into a Platform Specific Model (PSM) by using plug-ins called cartridges which control the generation of each file.

3.4.1.1 Cartridges

Cartridges are used to specify how the UML model should be translated into actual code. It processes specified stereotypes (i. e., `Entity`, `Enumeration`, etc.) or model elements that meet certain conditions (i. e., dependency to a `Service`) [AndroMDA, 2005a].

Currently AndroMDA delivers ten different cartridges which, amongst others, are used for EJBs, Hibernate, Struts, and Spring. Technically a cartridge in AndroMDA is a jar file, zip file, or a directory. It contains Velocity templates, see Chapter 3.4.1.2 (›Velocity Templates‹), Java classes, and a special descriptor file named `andromda-cartridge.xml`.

By applying them to ones project they automatically create session beans, message driven beans, entity beans, value objects, and rudimentary Web pages including all necessary Struts Java classes and configuration files.

One of the advantages of AndroMDA is that the user can remodel the PIM at any time and the underlying code is changed automatically. However this creates the problem that manually altered code is lost whenever a new build is executed. The cartridges used in this project address this issue by dividing code into two separate folders.

Manual Folder It holds all files that may be modified by the user, like JSP pages, Struts files, derivatives of session beans, and many more.

Generated Folder This folder keeps files of value objects, entity beans, etc. and is completely deleted and rebuilt whenever AndroMDA is executed.

3.4.1.2 Velocity Templates

Velocity is a Java-based template engine. It permits anyone to use a simple yet powerful template language to reference objects defined in Java code [Velocity, 2005b].

It is commonly used in various fields [Velocity, 2005a]:

- Web applications
- Source code generation
- Automatic emails
- XML transformation

AndroMDA uses Velocity templates for the code generation process. In the `andro-mda-cartridge.xml` the mapping from stereotype to Velocity template is defined. The template takes an instantiated model and outputs the previously defined content.

3.4.2 XDoclet

XDoclet is an open source code generation engine that enables Attribute-Oriented Programming for Java, meaning that special JavaDoc tags can be added to one's code. These source files are parsed by XDoclet that generates elements like XML descriptors and/or source code files according to the specified JavaDoc tag.

Originally XDoclet was intended to be a tool for generating EJBs, but it developed into a general-purpose code generation engine. Currently it can only be used as part of the Ant build process. It ships with a set of modules that, e. g. generate, based on a EJB implementation source file, interfaces, value objects, Struts Forms, and deployment descriptors [XDoclet, 2005].

XDoclet consists of five components:

XDoclet Engine provides a template engine that transforms a template into a file. It uses a Jarkata Ant task to invoke the templates and a loader to discover and plug in modules.

XJavaDoc Engine is a rewrite of Sun's JavaDoc engine that works five times faster and suits better to XDoclet purposes. It scans the source code and uses an API to extract information from tags.

Subtasks are classes that tell the XDoclet engine what template to invoke and how it should be called.

Templates consist of static and dynamic parts. Static parts are directly rendered whereas dynamic parts are substituted by content provided by tag handlers.

Tag Handlers are sub classes of `xdoclet.TagHandler` and are invoked by the template engine. A tag is an XML element satisfying the following style: `<namespace>.<tag name>`.

3.5 Development Tools and Libraries

The following chapter will introduce the main development tools used during the thesis and will give a description of the powerful open-source library JFreechart.

3.5.1 JFreechart

JFreeChart is a free chart library for the Java platform and part of the JFree software projects created by David Glibert and Thomas Morgner. Other projects are JFreeReport, Pixie, JWorkbook, JFreeDesigner, and the free general purpose Java class library JCommon used in the JFreeChart and JFreeReport projects [Gilbert, 2002].

JFreeChart is designed for use in applications, applets, servlets, and JSPs [Gilbert, 2005]. Distributed with the complete source code the following charts are available: pie charts, bar charts, line charts, scatter plots, time series charts, Gantt charts, meter charts, symbol charts, wind plots, combination charts, and many more. It is written entirely in Java allowing it to run on any implementation of the Java 2 platform.

Some of the core features of JFreeChart are:

- Export to PNG and JPEG
- Tool tips
- HTML image map generation
- Annotations
- Interactive zooming
- Chart mouse events

The drawing of charts is completely managed by JFreeChart. It »achieves this by obtaining data from a Dataset and delegating the drawing to a Plot object (which, in turn, delegates the drawing of individual data items to a CategoryItemRenderer or a XYItemRenderer, depending on the plot type)« [Gilbert, 2005].

All datasets that are used by JFreeChart are defined by interfaces which makes it easy to implement one's own dataset. Of course, there are several default classes available like AbstractDataset, CombinedDataset, DefaultCategoryDataset, Default-TableXYDataset, JDBCCategoryDataset, JDBCXYDataset, XYSeriesCollection, and many more.

Charts and Plots created with JFreeChart can be customized in many ways:

- add/change title
- set background color/background image
- smooth charts using anti-aliasing
- change colors for series
- set axis labels

3.5.2 MagicDraw

»MagicDraw is a visual UML modeling and CASE tool with teamwork support« [No Magic Inc., 2006]. It is a multifunctional tool that facilitates analysis and design of Object Oriented systems and databases by providing database schema modeling, Data Definition Language (DDL) generation, a code engineering mechanism, and reverse engineering facilities.

Due to the fact that it is written in Java it is platform independent and can be run on any system where Java 1.4 or 1.5 is supported. There are five different editions available whereas the Community Edition is free and intended for non-commercial projects. All of the current versions fully support UML 2.0.

3.5.3 Eclipse

Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software. The word *Eclipse* is often used for referring to the Eclipse Software Development Kit (SDK), which is a Java integrated development environment (IDE). It is a combination of several Eclipse projects, including Platform, Development Tools, and Plug-in Development Environment.

The Eclipse workbench provides the structure in which tools interact with the user. It is based on editors, views, and perspectives and supports multiple user workspaces to store projects. Projects can be placed under version and configuration management with an associated team repository whereat support for CVS repositories is included [Eclipse, 2006]. One of the core features of Eclipse is its Plug-in Architecture which allows easy integration of new plug-ins leading to a vast amount of free extensions available in the Internet.

Chapter 4

Institute Libraries

In this chapter the used parser for real-time PCR data is explained in detail. Furthermore a detailed description of the incorporated analyzers is given. Finally the user gains insight into the Genome Usermanagement.

4.1 Parser

Modern real-time PCR thermocyclers allow the storage and exportation of numerous files, including raw fluorescence data and background subtracted data. Some of this files are available as text files, others are stored as binaries. The function of the parser is to read those files and provide a generic interface to access all stored information.

The principle design of the parser consists of two parts. On the one hand it provides an interface consisting of basic classes that are meant to be used to access the functionality of the parser. On the other hand it has several implementation classes, which realize the parsing. Because of the fact that every real-time PCR vendor has other types of files, the parser has to act differently according to their produced data. However the results of the parsing process are all accessed through the same generic interface.

The parser provides information about several things:

Well information General information about each well, including the user defined well name, omitted status, and used master-mix are provided.

Detectors It is possible to access the name and the concentration of each detector used for producing fluorescence emission.

Hardware, software Information about the used hardware and software, including name and version are supplied.

Thermocycler profile The parser allows the software to readout the performed thermocycler profile. This incorporates information about duration and temperature of each step and the number of repeats of each cycle.

Raw fluorescence data The complete fluorescence spectra, produced during the PCR process, can be parsed, which may be used for further analysis and evaluation. Because all this information is not stored in one file, it is necessary to specify additional export files in order to get access to the complete fluorescence data. Moreover it is possible to read in background corrected files whose data is needed by some analysis algorithms.

Dissociation data In order to characterize the generated PCR products it is common to perform a dissociation curve analysis. After the completed PCR the temperature is slowly increased up to 95°C. The increase in temperature causes PCR products to undergo denaturation, a process accompanied by a decrease in fluorescence emission. The presence of different PCR products is reflected in the number of first-derivative peaks. These curves are processed by the parser and made available through a generic interface.

4.2 Analyzers

Real-time PCR has revolutionized many aspects of genomic research, but without the proper analysis methods the generated results are useless. Therefore several methods for analyzing real-time PCR results have been developed. Some of them were implemented in Java and made available for in-house use.

All these classes extend the same base class that provides a single method for analyzing PCR data.

```
ResultData calculate (RTPCRData data);
```

It takes as input a `RTPCRData` object, which contains a two dimensional array holding all wells and their corresponding PCR data. Moreover an optional heading and description can be specified for each well. The `ResultData` object contains the results of the calculation process and specifies C_t value, efficiency, correlation, and starting amount (describing the quantity of the initial template) for each well. Because not every algorithm computes all these properties, only the calculated ones are set.

Usually software modules shipped with PCR instruments display the amplification curves and provide methods to subtract the background signal. However they do not provide a control mechanism concerning the threshold used for calculating the C_t value, which is either set manually by the user or determined by the software.

The following methods provide alternative or better solutions to analyze results generated by real-time PCR.

4.2.1 AnalyzerSoFar

AnalyzerSoFar implements the algorithm described by Wilhelm in [Wilhelm, 2003] and Wilhelm *et al.* in [Wilhelm *et al.*, 2003]. SoFar stands for »Software For the Analysis of Real-time PCR data«. It operates on raw fluorescence data and calculates C_t value, efficiency, and starting amount. First the data is smoothed using *splines*, which is a special function defined piecewise by polynomials. The maximal bend of this function is used to determine the start of the exponential phase, which is described through an exponential or sigmoid function. The specification of the optimal threshold is performed within the exponential phase using all analyzable standards.

4.2.2 AnalyzerRutledGene

This analyzer operates on the raw fluorescence data and returns only the starting amount of the used template. It has been originally developed by Rutledge, described in [Rutledge, 2004]. The principal idea behind the algorithm is to fit the data to a four-parametric sigmoid function, which is used to determine the starting amounts of the used templates. In order to achieve accurate results, the plateau phase of the amplification is not included into the fitting process. To calculate the initial quantity neither standard curves nor threshold based C_t values are needed. Instead, it uses the derivative of the sigmoid curve function.

4.2.3 AnalyzerMiner

AnalyzerMiner implements the model described by Zhao and Fernald in [Zhao and Fernald, 2005]. It operates on the raw fluorescence data and calculates C_t value, efficiency, and starting amount. First the algorithm fits a four-parameter logistic curve to the data, using it for the determination of the exponential phase. Next it uses a simple exponential function to model the exponential phase which is done through an iterative non-linear regression algorithm. The C_t value is set as the end point of the exponential phase, which is the first positive second derivative maximum of the logistic model, and the efficiency is determined using a weighted average of the exponential curve. Once these two values have been computed, they are used to calculate the starting amount.

4.2.4 TAQAnalyzer

This analyzer is based upon the »Target Analysis Quantification« (TAQ) explained by Ostermeier *et al.* in [Ostermeier *et al.*, 2003]. It uses raw fluorescence data and calculates correlation, efficiency, and the initial starting amount of the template. First the algorithm log-transforms the data and then fits a linear regression equation to quantify the reaction efficiency and the initial number of target molecules. The efficiency is reflected by the slope of the equation, whereas the starting amount is calculated by comparing linear regression equations of known concentrations to the newly computed curve.

4.2.5 LinRegAnalyzer

The LinRegAnalyzer operates on background corrected files and calculates efficiency, correlation, and starting amount. Described by Ramakers *et al.* in [Ramakers *et al.*, 2003] it uses log transformed values to construct a slope with at least four and no more than six data points with the highest correlation to the original curve. The search for the best slope is done using a linear regression algorithm.

4.3 Genome Usermanagement

The Genome Usermanagement is a Java based authentication and authorization system originally developed by Dieter Zeller for molecular biology database systems. It is designed to be used in »web-applications, Microsoft Windows domains, and Unix/Linux servers« [Zeller, 2005].

Developers can register their application in the centralized usermanagement system which is administered using a Web interface. Each Web application gets its own unique password and only if the correct password is transmitted upcoming requests are processed.

By using the Web interface all applications, users, resources, and access rights can be managed. Users can be assigned to different groups and may be members of institutes. Through the usermanagement it is possible to control the rights of each entity and give different access rights to users, institutes, or groups. Moreover it is possible to inherit access rights allowing a finely granulated control mechanism.

The developed tag libraries are designed to be used within JSPs that either check the login status of the user or examine the access rights of the logged in user. Therefore it is possible to assign certain tasks only to users that are part of certain groups, and, i. e., allow only them to delete entries.

Within Java code the usermanagement can be access through an API which provides the full functionality. Moreover it is fully integrated into the code generation process which enables a quick construction of a fully working prototype whenever a new project is developed.

Chapter 5

Requirements and Design

Determining the requirements of a system is an important process during the production of a new application. Only a clear picture of the required functions can lead to the development of an adequate design. This chapter describes the discovered requirements and gives a detailed description of the established design.

5.1 Detailed Requirements

The primary goal of this thesis is to develop an application which is able to handle all data accumulated during a real-time PCR experiment. Moreover it should provide a centralized system to manage, view, and analyze results produced during the experiment.

Because of the advantages that a MDA approach offers (described in Chapter 3.1.1 on page 11), the system is based on this architecture. Through the usage of a code generator, one is able to develop a flexible and easy to maintain system which represents the developed model.

Amongst others, the following four requirements were determined:

Web-based Biologists working in laboratories often perform their job at several different places. Therefore it is very convenient for them to have access, almost everywhere, to the system they depend on. Web-based systems satisfy this need because the system can be used from every computer having a connection to the local network or Internet.

Parse thermocycler data When performing a PCR experiment a lot of data has to be entered into the thermocycler. In order to avoid the overhead of entering data twice and to reduce the risk of making errors it is convenient to import the thermocycler data into the application. Moreover the results of the PCR process build up a huge amount of data which can only be entered efficient through the use of a parser.

Incorporate several analyzers As already mentioned in Chapter 2.4 on page 8 and Chapter 4.2 on page 33, raw real-time PCR data has to be analyzed to get significant and meaningful results. The incorporation of *several* analyzers allows the user to easily perform and compare results of different analyzers without having to use different programs for each algorithm.

Graphical display of data In modern user-friendly interfaces data has to be displayed graphically, because it is often more meaningful than pure raw-data tables. Moreover biologists are accommodated to graphs displayed by the PCR software.

5.2 Typical Workflow

Real-time PCR experiments are always performed within borders of a general protocol. The different stages of this protocol have been analyzed and major steps were extracted. These steps were used to define a characteristic workflow when using the developed system. Figure 5.1 on the following page visualizes these steps trying to clarify the overall view of the application.

A typical workflow of the application consists of the following nine steps:

Create experiment Experiments are top-level domains that are mainly used to group different runs, which improves the navigation and usability.

Create run A run represents an actual experiment. It contains information about the used thermocycler, including its temperature profile, and specifies the plate, which directly represents the plate used in the PCR machine.

Define properties The conduction of a real-time PCR experiment includes a variety of components. In order to get a detailed and complete definition of the experiment, properties and definitions of these components have to be inserted into the system. This includes, amongst others, the definition of cDNAs, primers, and detectors.

Perform experiment Ideally, the experiment should be performed after the previous steps have been carried out.

Specify file After the real-time PCR experiment has been performed, the generated file is uploaded into the system and associated with the previously created run.

Parse file Having attached a file to a run, it is possible to parse the file. This process enters the plate definition and the corresponding data into the system.

Display plate and charts Once a file has been successfully parsed, the complete plate definition and spectral information is accessible. By displaying, for example, amplification and dissociation charts, the success and performance of the experiment can be controlled.

Analyze Having verified that the experiment was successful the generated data has to be analyzed to get the information originally needed.

View/export results Finally, results generated by the analysis step have to be evaluated and are used for designing further experiments. Moreover conclusions are drawn based on these findings.

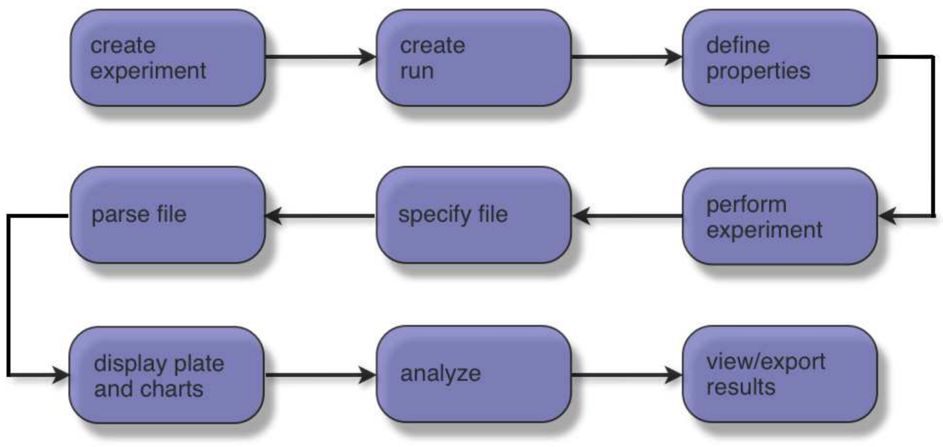


Figure 5.1: Presented is a typical workflow of the developed application. It starts at the top left corner with creating an experiment and ends at the bottom right corner with exporting and viewing the generated results.

5.3 UML Diagram

Based on the analysis process, the determined workflow, and the constituents of a real-time PCR experiment a number of UML diagrams were developed. Generally they can be divided into diagrams for general services, entity specifications, and diagrams needed for creating reports.

5.3.1 Entity diagram

The entity class diagram represents all persistent entities and their relationships. The complete diagram is shown in Figure B.1 on page 88. It can be separated into three major parts.

- **Run fraction—**Figure 5.2 shows all entities that are associated with a run. This includes hardware, software, instrument setting (represents a thermocycler temperature profile), category, and the relationship to arbitrary experiments. Note that one run can be part of many experiments and one experiment can have any number of runs.

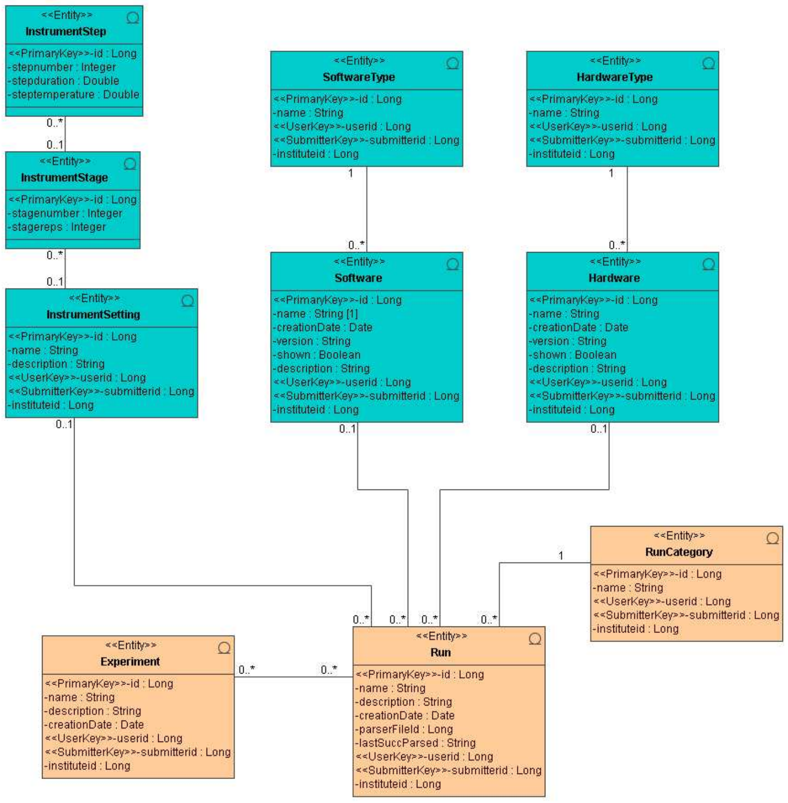


Figure 5.2: Displayed is the part of the class diagram covering the entities that are associated with a run.

- **Well fraction**—The entity well represents the actual reaction room on a plate where the real-time PCR takes place. Therefore it is the central point of the application where everything flows together. Directly associated with a well is the passive reference, the real-time chemistry (also referred to as master-mix), the detector, and various thermocycler data. A detector consists of an arbitrary number of primers that have—according to their type—reporters or quenchers attached. All these entities and their relationships are shown in Figure 5.3.

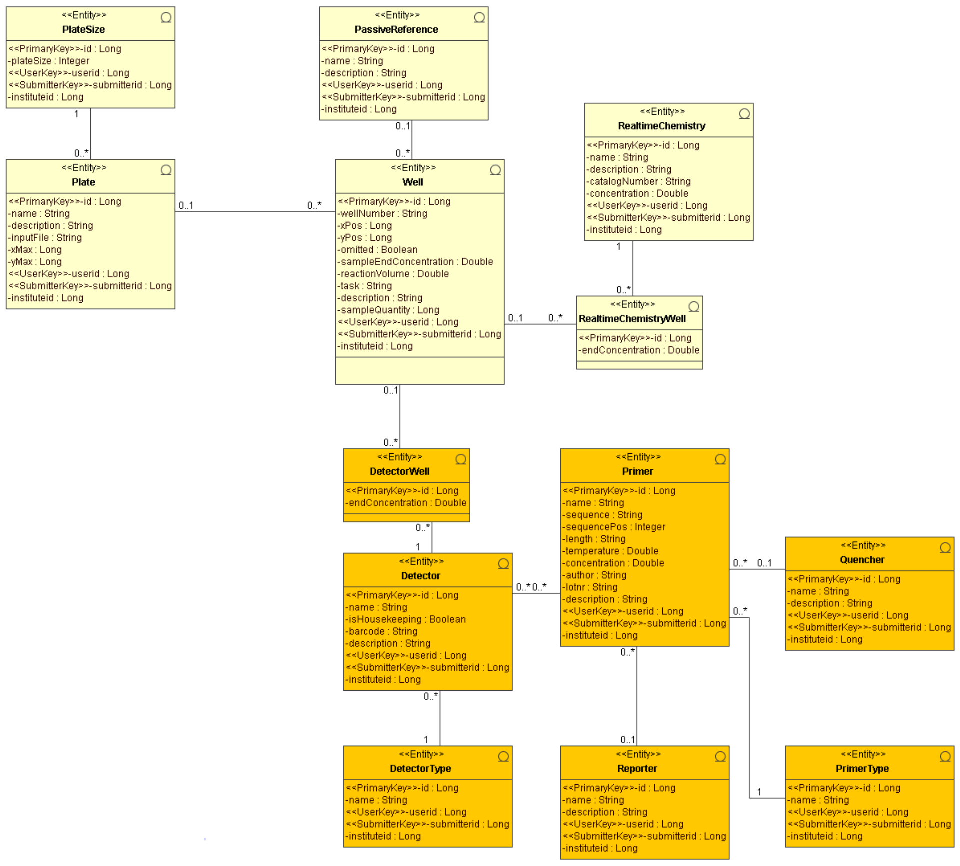


Figure 5.3: Shown are entities that are directly attached to a well, which represents the real-time PCR reaction room.

- **real-time PCR fraction**—Whenever a real-time PCR experiment is executed, the PCR machine collects a huge amount of data. Figure 5.4 on the following page shows the part of the class diagram covering this aspect. It can be separated into dissociation data (raw and derivative data), raw spectral information, amplification data (also referred to as Rn data), and delta Rn data (the background subtracted Rn values). After having applied an analyzer on the real-time PCR data the results are stored in the appropriate entities. There is also the possibility to analyze one well several times using the same or different analyzers.

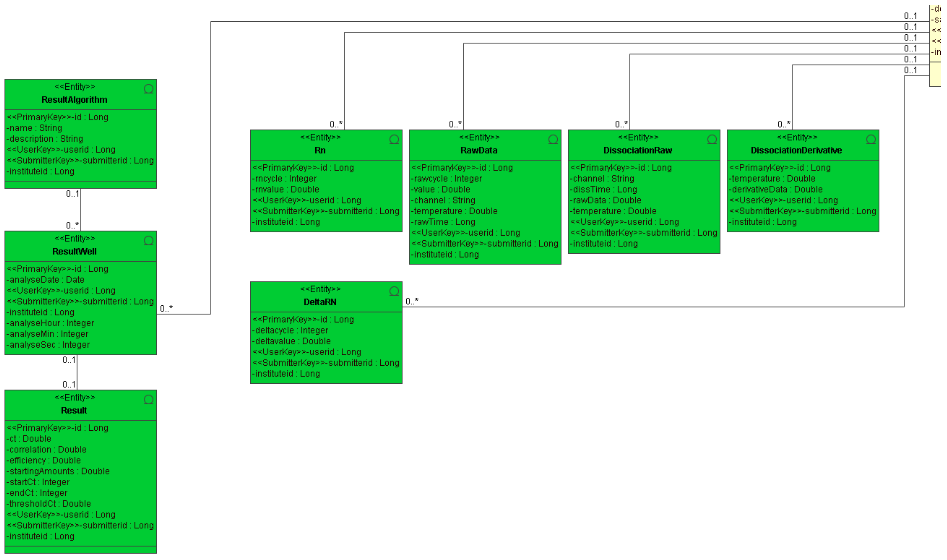
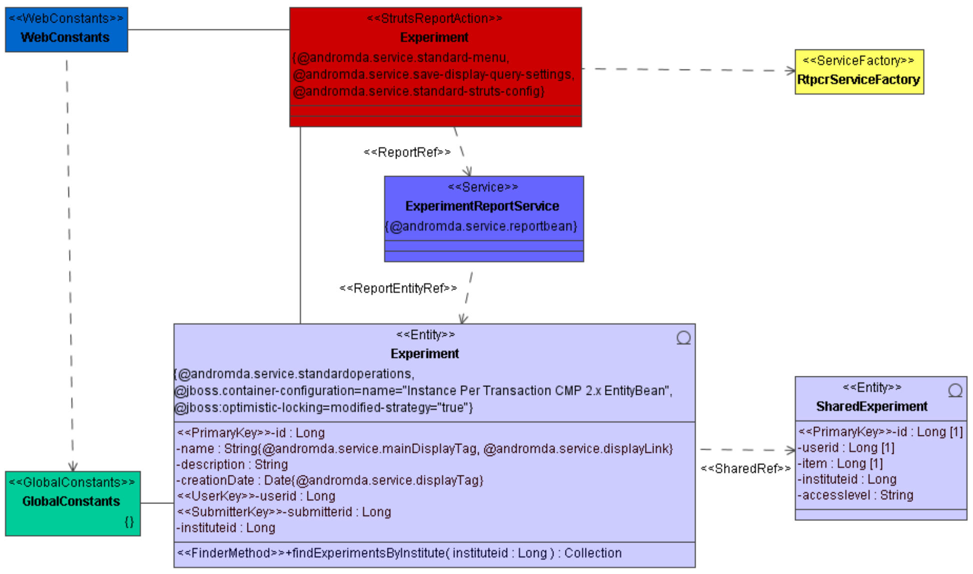


Figure 5.4: Demonstrated are the entities responsible for storing results and spectral information of each well.

5.3.2 Report diagram

Report diagrams are used to model basic »reports« using the code generator AndroMDA. Chapter 6.1 on page 45 describes in detail what exactly is modeled. As an example the report diagram created for the entity *experiment* is shown in Figure 5.5 on the following page. The association to the `GlobalConstants` class produces static variables used throughout the code, whereas `WebConstants` are used for defining `HttpSession` attributes. The `StrutsReportAction` creates Struts Action and Form classes whereat the `ReportService` models a stateful session bean that is necessary for manipulating the entity's data.



<<WebConstants>>
WebConstants
<<StrutsReportAction>>
Experiment
{@andromda.service.standard-menu,
@andromda.service.save-display-query-settings,
@andromda.service.standard-struts-config}
<<ServiceFactory>>
RtpcrServiceFactory
<<ReportRef>>
<<Service>>
ExperimentReportService
{@andromda.service.reportbean}
<<ReportEntityRef>>
<<Entity>>
Experiment
{@andromda.service.standardoperations,
@jboss.container-configuration=name="Instance Per Transaction CMP 2.x EntityBean",
@jboss:optimistic-locking=modified-strategy="true"}
<<PrimaryKey>>-id : Long
-name : String{@andromda.service.mainDisplayTag, @andromda.service.displayLink}
-description : String
-creationDate : Date{@andromda.service.displayTag}
<<UserKey>>-userid : Long
<<SubmitterKey>>-submitterid : Long
-instituteid : Long
<<FinderMethod>>+findExperimentsByInstitute(instituteid : Long) : Collection
<<GlobalConstants>>
GlobalConstants
{}
<<SharedRef>>
<<Entity>>
SharedExperiment
<<PrimaryKey>>-id : Long [1]
-userid : Long [1]
-item : Long [1]
-instituteid : Long
-accesslevel : String

Figure 5.5: Presented is a complete example for generating a »report« of an entity.

5.3.3 Service diagram

The service diagram displays the most prominent services used in the developed application—shown in Figure 5.6 on the next page. The `RunParseFileSevice` is a MDB that is responsible for calling the appropriate parser methods, creating the necessary beans, and having them stored in the database. Also designed as a MDB, the `AnalyserService` manages the calculation of real-time PCR results using the incorporated analyzers. The `PlateService` is a session bean that holds helpful methods used throughout the application.

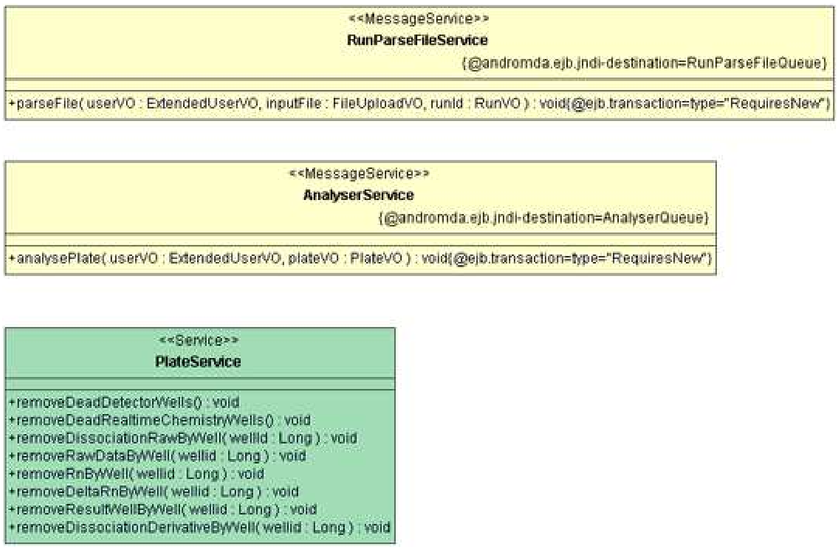

<<MessageService>>
RunParseFileService
{@andromda.ejb.jndi-destination=RunParseFileQueue}
+parseFile(userVO : ExtendedUserVO, inputFile : FileUploadVO, runId : RunVO) : void{@ejb.transaction=type="RequiresNew"}
<<MessageService>>
AnalyserService
{@andromda.ejb.jndi-destination=AnalyserQueue}
+analysePlate(userVO : ExtendedUserVO, plateVO : PlateVO) : void{@ejb.transaction=type="RequiresNew"}
<<Service>>
PlateService
+removeDeadDetectorWells() : void
+removeDeadRealtimeChemistryWells() : void
+removeDissociationRawByWell(wellId : Long) : void
+removeRawDataByWell(wellid : Long) : void
+removeRnByWell(wellid : Long) : void
+removeDeltaRnByWell(wellid : Long) : void
+removeResultWellByWell(wellid : Long) : void
+removeDissociationDerivativeByWell(wellid : Long) : void

Figure 5.6: Shown are the three major services used in the developed application.

Chapter 6

Implementation

Based on the J2EE framework, the application is designed as a Web-based program. As storage system it uses an Oracle database whereas the application itself is deployed on a JBoss application server. The administration of the user interface is managed using the Struts framework in combination with JSPs.

Principally, the implementation of the developed design can be separated into two parts. The first part consists of creating code using the code generation framework AndroMDA, which is based on the modeled UML diagrams. Based on this rudimentary prototype, code has to be added or altered manually which is the second part of the implementation process. Due to the design of AndroMDA the developer is able to refine the used UML diagrams at any time without having to change or backup a single line of self-written code.

The implementation process is more or less determined by the code generator which is used to create an initial, simple prototype. Starting from this initial implementation, features, design corrections, and additional business functionalities are incorporated into the application.

6.1 Report

In this thesis the term *report* describes the sum of all components needed to provide an interface that is capable of displaying, editing, and deleting a certain entry of an entity. Moreover it provides a list showing all created entries according to the users access rights.

The creation of these components is based on a special UML model (see Chapter 5.3.2 on page 42) which has a strict design and needs certain stereotypes and class descriptions. Because of the fact that a report can only have one association to an entity, this diagram has to be designed for each entity where a report is wanted. Once this model has been applied to the AndroMDA code generator, amongst others, the following things are build:

- **JSPs**—On the one hand, it generates a JSP page that is used for displaying all entries of a particular entity. On the other hand a JSP site managing and showing the properties of a special entity is build.
- **Sharing**—Whenever a »shared entity« is attached to the entity, a general sharing mechanism is implemented. It allows the user to share his/her created entity with other users or institutes.
- **Action/Form**—Having defined a `StrutsReportAction` class, the appropriate Struts Action and Form classes of an entity are constructed. By setting different tags the developer is able to control the generation process and is therefore able to customize the applications behavior.
- **Configuration**—Entries in the `struts-config.xml` and `application.-properties` files are created that are necessary to deploy a Struts application.
- **Session bean**—The associated `Service` is accountable for creating a stateful session bean matching the associated entity.

The design of the application comprehends the following reports:

- AnalyserAlgoResult
- AnalyserResult
- CDNACreation
- Detector
- DetectorType
- Experiment
- Hardware
- HardwareType
- InstrumentSetting
- ParserResult

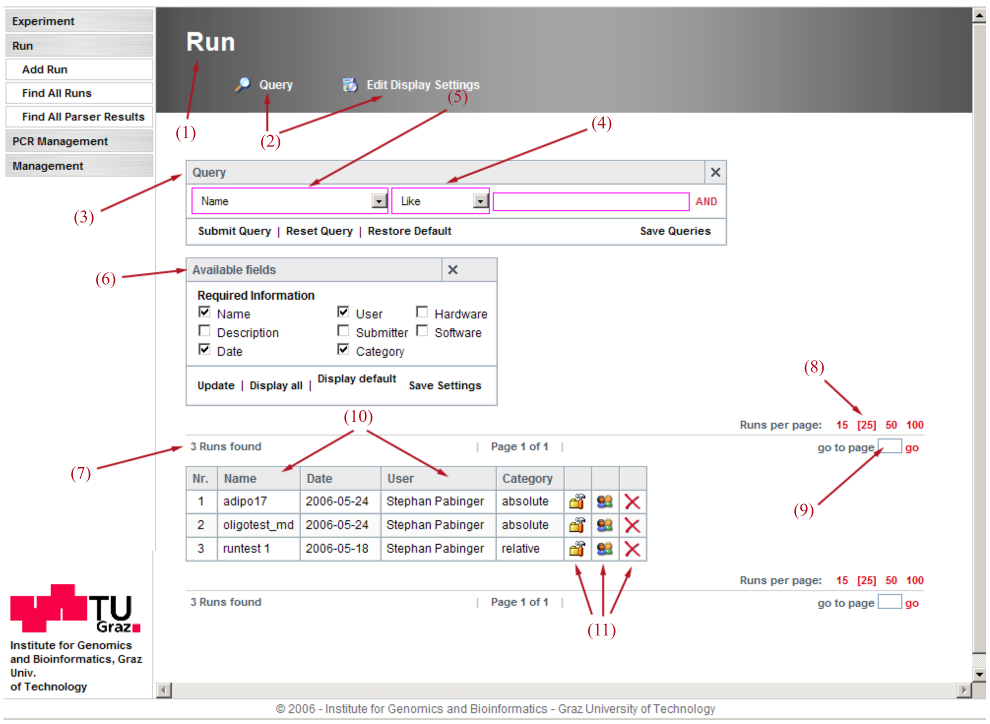
- PassiveReference
- Plate
- PlateSize
- Primer
- PrimerType
- Protocol
- ProtocolType
- Provider
- Quencher
- RealtimeChemistry
- Reporter
- Run
- RunCategory
- Software
- SoftwareType
- Well

6.1.1 List view

Each report defines a JSP page used to display all entries of a certain entity that belong to the user. Figure 6.1 on the next page displays such a JSP site showing the entity *run*.

On top of each page is a header (1) displaying an identifier (normally the entity's name) and providing links to *query* and *display setting* (2) interfaces. Once the user has pressed the query link a new mask (3) pops up giving the user the possibility to search for certain entries. The system provides a number of different operators (4) and generally all properties (5) of an entity are available for querying. Furthermore queries can be saved and used in further sessions. Edit Display Setting (6) displays an additional table underneath the query interface and gives the user the facility to customize the table showing the list of all owned entries. Again, these settings can be saved for each user enabling a highly customizable user interface. By default the query and display setting interfaces are hidden when a user navigates to such a list view.

The list itself is divided into pages which consist of either 15, 25, 50, or 100 entries. Above and below the list is a navigation bar which shows the number of found entries (7), the actual page, the overall number of pages, and the selected number of items per page (8). Moreover the page provides the possibility to directly jump to a certain page (9). On top of the list is a header which specifies the different columns of the table (10). These columns reflect the chosen display setting and change according to the user's preferences. By clicking on the name of the column the list is sorted corresponding to the selected property. In addition to the values of the chosen properties each entry has an edit, delete, and sharing button (11), given that sharing is enabled. By using these buttons the user can edit a selected entry, share it with other users, or delete it from the system.


Experiment
Run
Add Run
Find All Runs
Find All Parser Results
PCR Management
Management
Run
Query
Edit Display Settings
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
Query
Submit Query | Reset Query | Restore Default
Save Queries
Available fields
Update | Display all | Display default
Save Settings
TU Graz
Institute for Genomics and Bioinformatics, Graz Univ. of Technology
© 2006 - Institute for Genomics and Bioinformatics - Graz University of Technology

Query: Name | Like | AND

Required Information

☑ Name	☑ User	☐ Hardware
☐ Description	☐ Submitter	☐ Software
☑ Date	☑ Category	

Runs per page: 15 [25] 50 100

3 Runs found | Page 1 of 1 | go to page go

Nr.	Name	Date	User	Category			
1	adipo17	2006-05-24	Stephan Pabinger	absolute			✗
2	oligotest_md	2006-05-24	Stephan Pabinger	absolute			✗
3	runtest 1	2006-05-18	Stephan Pabinger	relative			✗

Runs per page: 15 [25] 50 100

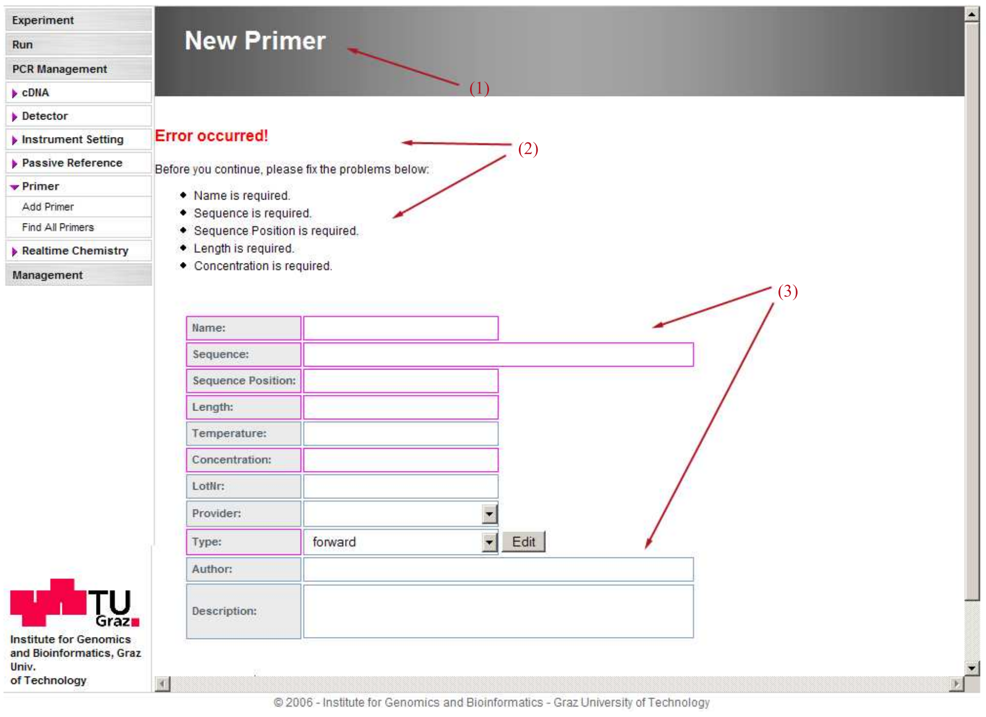
3 Runs found | Page 1 of 1 | go to page go

Figure 6.1: Shown is an example of a JSP site displaying a list of stored entries. It contains a header (1) and links (2) to enable querying (3,4,5) and manipulating the look of the list (6). Moreover it shows how many entries were fetched (7), the number of items per page (8), and provides the possibility to jump to a certain page (9). The list itself is described by table headers (10) and each entry has buttons to edit, share, and delete it (11).

6.1.2 Detail view

The second JSP page that a report creates is used to add, view, edit, or delete a specific entry. Because the design created by the code generator is very simple almost every page needs manual revision. Amongst other things, it is necessary to define the mandatory properties, create associations to other entities (mostly using comboboxes), and change the general appearance of the page.

As an example, the *primer* JSP page is displayed in Figure 6.2. On top of the site is a header (1) that displays the actual purpose of the page. Next, occurred errors (2) are shown that present a detailed description of arisen problems. The actual masks of the properties (3) vary from entity to entity. Generally boxes in pink are mandatory properties whereas grey boxes mark optional attributes.



New Primer

Error occurred!

Before you continue, please fix the problems below:

- Name is required.
- Sequence is required.
- Sequence Position is required.
- Length is required.
- Concentration is required.

Name:	
Sequence:	
Sequence Position:	
Length:	
Temperature:	
Concentration:	
LotNr:	
Provider:	
Type:	forward
Author:	
Description:	

© 2006 - Institute for Genomics and Bioinformatics - Graz University of Technology

Figure 6.2: Presented is a typical JSP page that is used for adding, editing, viewing or deleting a certain entry. Usually it consists of a header (1) and the mask for editing or viewing the properties (3). Additionally errors (2) are displayed on top of the page whenever they occurred.

6.1.3 Struts Action/Form

Each used Action class extends the `LookupDispatchAction` which enables the calling of different methods when an HTML form has multiple submit buttons. The `LookupDispatch Action` is an abstract Action, mapping a special parameter property to the appropriate method implemented in the subclass.

Typically a generated Action class provides twenty-three different methods. This includes methods for manipulating the display settings of the corresponding JSP page, controlling the scrolling mechanism required for viewing already created entries, enabling the sharing of entries, and saving queries defined by a user. Amongst all these methods, three methods are of major interest when it comes to adapting the generated code. The `findAll` method is used to create a list storing all entries of a certain entity belonging to a user. In order to improve the performance of the system the list holding all entries consist only of the entities ids. For each new request a new list storing the actual beans of an entity is created, which minimizes the overhead of storing non-needed beans. Entries are created, edited, or deleted using the `createEdit` method. It is more efficient to combine these three procedures into one method, because a lot of identical code is used by all of them. The different procedures are chosen trough a defined Form property. The `viewEdit` method sets the Form properties used to display a certain entry. Moreover it is used to create lists storing the different entries of a drop-down list.

Each generated Struts Form is a Java bean extending the `ActionForm` class. It consists of the entity's properties and their getter and setter methods. Furthermore a few additional methods are defined. The `validate` method is used to control the user's input and generates error messages that appear on the according JSP page, whenever something went wrong. Form properties are filled using the `setForm` method which uses a *value object* as input parameter. Moreover methods that control the display setting of the corresponding JSP page are specified.

6.1.4 Stateful session bean

The stateful session bean of a report controls the business logic that is necessary for retrieving, manipulating, and adding entities. It is stateful because of the fact that each client creates a list storing all own entries of an entity. This list is unique for each client and is maintained over multiple conversations demanding for a *stateful* session bean. It is used for fetching elements of the entity, controlling the business logic of the JSP scrolling mechanism, and implementing the sharing mechanism.

6.2 Parser

The incorporation of the parser is a central part of the developed application. Without the possibility to automatically fill in a huge amount of data the system would be nearly unusable. The functionality is best illustrated by stepping through a typical parsing job.

6.2.1 Run (JSP/Action)

After a file has been attached to a run the parser can be started by clicking on a button appearing in the run JSP site. This invokes the `createEdit` method in the `RunAction` class which calls the section responsible for the parser. Because of the fact that each parsing job creates a new plate, the old plate associated with the run is deleted. Having completed this task a message is built up which is sent to the MDB—`RunParseFileService`. The message contains the following parameters:

- **`userVO`**—The user value object contains information about the logged in user. It is needed to create valid entries into the database concerning the ownership and sharing mechanism.
- **`runVO`**—It is a value object of the actual run holding all information of the entity.
- **`fileVO`**—This value object contains the path of the file that is attached to the selected run and acts as a source for the parser.
- **`boolCDNA`**—Initially, a plate is specified using the PCR machine's software module. This application provides the possibility to define a name for each well on the plate. In some cases the biologist sets this name equal to the used sample (which is a cDNA), in other cases the name of the used detector is entered. The Boolean `boolCDNA` reflects a combo-box which needs to be set before the parser is started and tells the system if the name of each wells is equal to the used sample name.

6.2.2 RunParseFileService

Once the sent message has been received by the `onMessage` method of the MDB `RunParseFileService`, the transmitted parameters are unpacked again. In order to avoid writing corrupt data into the database a global variable has been defined, which signals the success of the parsing job. Only if nothing went wrong things are written into the database.

The following cases are defined that lead to a not successful parsing job:

- the parsing itself went wrong
- used hardware was not found in the system
- used software was not found in the system
- used cDNAs were not found in the system
- used detectors were not found in the system
- passive references were not found in the system

Moreover each parsing job creates a `ParserResultVO` that contains, amongst other things, errors, information about problems, and details about the parsed hardware, software, and instrument setting. This value object is stored in the database and presented to the user.

The MDB `RunParseFileService` defines the following methods:

parseFile This method starts the parser and stores the result into a temporary variable—`UnifiedParsersRTPCRData rtdata`. This variable contains everything that is stored in the used file. Next appropriate methods are called to build up value objects that are then written into the database. Furthermore this method creates a `ParserResultBean` which is used to collect information during the next steps that are required to build up the `ParserResultVO` object.

addToDatabase Its purpose is to write the generated value objects into the database, but it only starts writing them if the parsing job was successful. Otherwise it creates a new empty plate and attaches it to the used run. This has to be done, because the old plate was deleted when the MDB was called and it is necessary that each run has an association to a valid plate.

parseInstrumentSetting The instrument setting reflects the temperature profile used in the thermocycler. Principally it consists of an arbitrary number of stages, which are also referred to as cycles. Each stage is made up of one or many steps, whereas each step has a duration and a temperature attribute defined. The aim of the `parseInstrumentSetting` method is to search for an instrument setting in the system that is identical to the current one. If it finds an appropriate entry it simply creates a reference to the existing instrument setting, given that the run has no instrument setting attached. If it does not find the instrument setting in the system the method creates a new value object. This object is inserted into the database and associated with the transmitted run.

parseSoftwareHardware This method is used to search for existing hardware and software units that are identical with the current ones. It first checks if the run has already a hardware or software attached and otherwise sets the proper links to the found entries. Given that it did not find the fitting entries it creates an entry in the `ParserResultBean` object and sets the success of the parsing job to `false`.

parsePlateData Provided through the parser are the size of the used plate and the source file. The method checks if the found plate size is already existent in the system or, if not, creates a new one. Then it associates the corresponding size with the plate and sets the input file.

parseWell Aim of the method to create all wells and their attributes. The parser generates an array holding all wells and their respective attributes that were stored in the file.

The following things are specified:

- **Well position** In order to display wells in a table, it is necessary to specify their exact position. Therefore the x and y coordinates are stored for each well.
- **Well number** The well number is a human friendly label specifying the position of a well on the plate. In most cases it is done by writing the first coordinate as letter and appending the second coordinate as digit (e. g. C6).
- **cDNA** If the user has specified that the names of wells are equal to the used cDNA names, the method tries to set the appropriate reference. Therefore it is necessary that each cDNA has been specified before the parser was started. For each found well name the application checks if the corresponding cDNA is existent in the system using the `findCDNACreation` method. Having found a missing one it adds the name to the `ParserResultBean` and sets the success to `false`. Found cDNAs are associated with the corresponding well.
- **Task** The task of a well displays the purpose of the performed PCR experiment.
- **Omitted** A well can be set to omitted if it should be excluded by the PCR machine. The parser signals omitted wells by either setting the well name to »EMPTY« or excluding them from the list. Therefore the `parseWell` method first creates a list of all wells and sets their omitted status to `true`. Only if a valid well name has been found this status is changed to `false`, guaranteeing that all wells have the correct omitted status.
- **Detectors** Each well has one or more detectors associated. Name and concentration are identified by the parser and the system searches for

them in the database. Found detectors are associated with the well and the used concentration is specified. If the used detector is not available in the database its name is added to the `ParserResultBean` and the success is set to `false`.

- **Passive reference** Passive references, also referred to as master-mix, need to be specified before the parser is started. If the identified passive reference is available in the system, the `parseWell` method creates an association to the entry. Otherwise it adds the name to the `ParserResult-Bean` and sets the success to `false`.
- **Dissociation data** The dissociation curves are available as raw and derivative data sets. Available attributes are channel, temperature, duration, and measurement time-point.
- **raw data (spectral information)** The raw data reflects the pure measured fluorescence signal. Stored attributes are channel, duration, temperature, and time-point.

findDetector This method is used to locate a detector in the system. It either returns the found value object or a null pointer.

findCDNACreation It is a helper method that is used to find a specific cDNA. It either returns the found value object or a null pointer.

6.2.3 JSP—ParserResult

Once the parser has finished its job the user needs to get informed about the completion of the task. Due to the fact that a MDB works asynchronously and is not able to handle the `HTTPSession` object, there is no way to send a message to the client that the bean has finished its work.

Therefore a system has been developed that checks every fifteen seconds if a new parser result is available. The corresponding JSP page implements a JavaScript function (`updateParserResult`) which is repeatedly called using the `setInterval` method. `UpdateParserResult` uses DWR to call a message on the server, which returns `true` if a new result is available. Moreover by using JavaScript the method updates the appearance of the JSP page (see Figure 6.3 on the next page). By clicking on the link a list appears and the user gets a detailed description about the selected parser result.

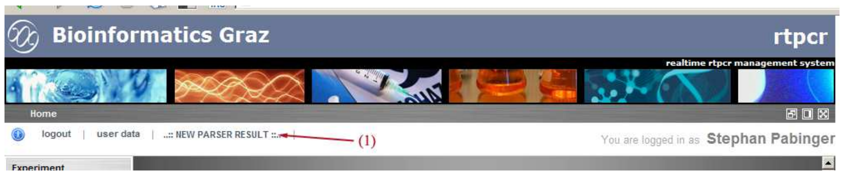

Bioinformatics Graz
rtpcr
realtime rtpcr management system
Home
logout | user data | ..:: NEW PARSER RESULT ::..
(1)
You are logged in as Stephan Pabinger
Experiment

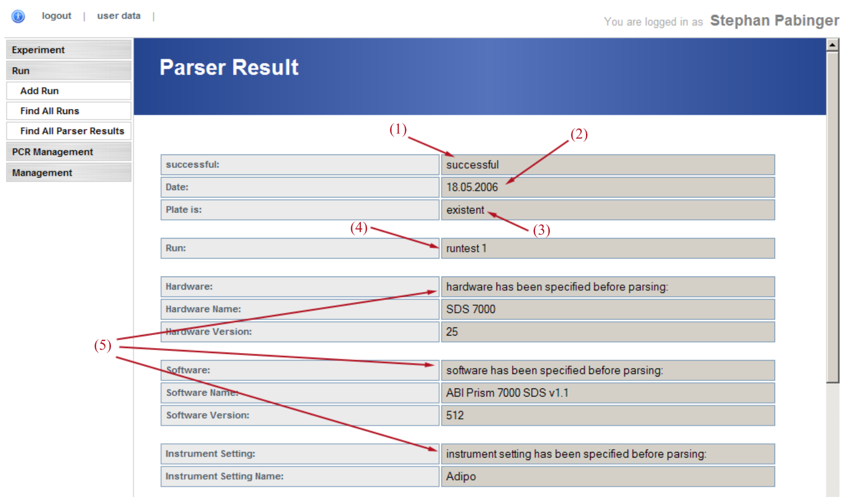
Figure 6.3: Shown is the message (1) appearing when a new parser result is available.

The parser result page presents the relevant information of a parsing task to the user. Figure 6.4 on the following page displays the upper part of the result page. On top of the page a message signals if the job was successful (1). Next, the date (2), the status of the newly create plate (3), and the name of the run (4) are shown. The following section covers information about hardware, software, and instrument setting. Beside each property a message is displayed (5) that tells the user what the parser has detected. The following messages can appear, whereas »`item`« is substituted by hardware, software, or instrument setting:

please add following »`item`« to the system This message appears given that the »`item`« was not found in the system during the parsing process. Moreover a button below the according section is shown which, by clicking on it, creates the missing entry. Since missing instrument settings are added by the parser, this message can only emerge for hardware and software modules.

»`item`« has been specified before parsing This message signals that the user has already set a reference before the parser was started.

found following »`item`« in system This message tells the user that the parsing task has found the appropriate entry in the system and has set a reference to it.



successful:	successful
Date:	18.05.2006
Plate is:	existent
Run:	runtest 1
Hardware:	hardware has been specified before parsing:
Hardware Name:	SDS 7000
Hardware Version:	25
Software:	software has been specified before parsing:
Software Name:	ABI Prism 7000 SDS v1.1
Software Version:	512
Instrument Setting:	instrument setting has been specified before parsing:
Instrument Setting Name:	Adipo

Figure 6.4: Shown is an example of the first part of a parser result page. It presents the success of the parsing job (1), the date (2), the status of the plate (3), the run's name (4), and reports about the detected hardware, software, and instrument setting (5).

The lower part of the result page, shown in Figure 6.5 on the next page, presents information about the newly create plate. Displayed are the identifier (1), the name (2), and the size (3) of the plate. Moreover the user gets the possibility to enter a plate description (4) and by clicking »Update & goto plate« (5) this description is set and the user is linked to the new plate. Once this task has been carried out, only a link to the plate is displayed on further visits. In addition to the used file (6), the status of the plate (7) is shown on the result page.

Covered by the last part of the result page is the section about missing detectors (8), cDNAs (9), or passive references (10). The boxes are used to display a list of needed »`items`«. These »`items`« can be either added manually by the user or by clicking on the button »Add to System« (11), which creates a simple body of the required entries. This mechanism allows a quick generation of relevant entries that can be edited and completed at a later date.

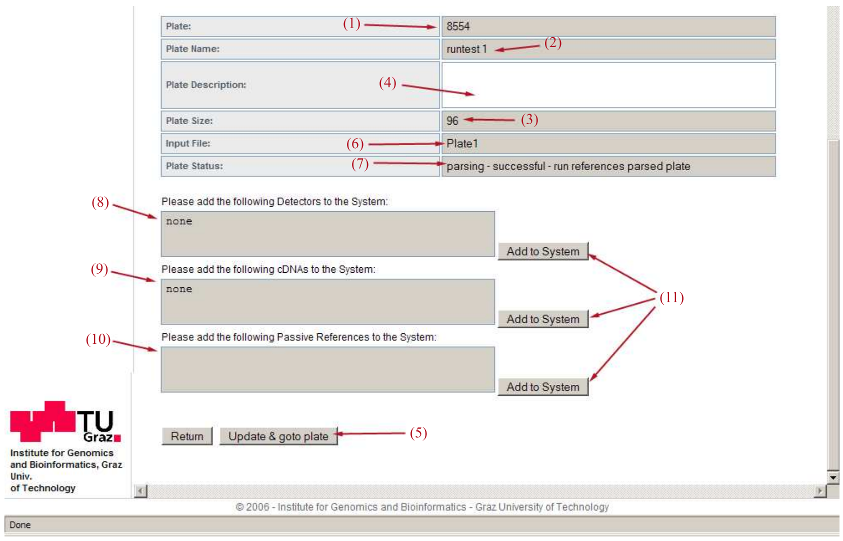


Plate:	8554
Plate Name:	runtest 1
Plate Description:	
Plate Size:	96
Input File:	Plate1
Plate Status:	parsing - successful - run references parsed plate

Please add the following Detectors to the System:

```
none
```

Please add the following cDNAs to the System:

```
none
```

Please add the following Passive References to the System:

Institute for Genomics and Bioinformatics, Graz Univ. of Technology

© 2006 - Institute for Genomics and Bioinformatics - Graz University of Technology

Done

Figure 6.5: Displayed is the second part of the parser result JSP page. It displays information about the generated plate (1,2,3,6,7) and provides the opportunity to set a plate description (4,5). Furthermore the missing detectors, cDNAs, and passive references are listed (8,9,10), which can be added to the system by pressing the corresponding button (11).

6.3 Analyzers

The implementation of the incorporation of analyzers can be divided into two major parts. The first part regards choosing the analyzers and starting them in a MDB. Presenting the results is the main function of the second part.

6.3.1 Choosing and starting analyzers

Once the user has pressed the button to analyze a particular plate, a site is displayed that presents all available analyzers. Figure 6.6 on the following page displays an example of such an overview page. On top it shows the name of the chosen plate (1) that simultaneously acts as a link back to the plate site. Below, a list is displayed that specifies names (2) and descriptions (3) of available analyzers. Each entry offers a checkbox (4) that is used to select the algorithm utilized to analyze the plate, whereas it is possible to select multiple analyzers. Pressing on the »Analyse« button sends the selection to a MDB which starts the analyzation process.

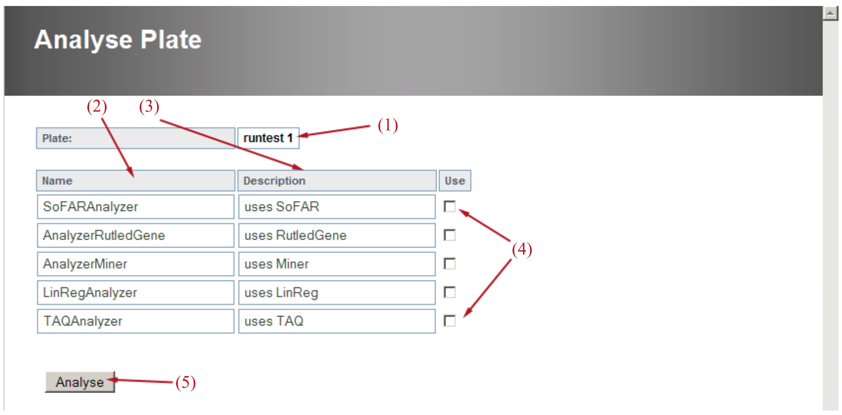

Analyse Plate

Plate:	runtest 1	
Name	**Description**	**Use**
SoFARAnalyzer	uses SoFAR	☐
AnalyzerRutledGene	uses RutledGene	☐
AnalyzerMiner	uses Miner	☐
LinRegAnalyzer	uses LinReg	☐
TAQAnalyzer	uses TAQ	☐

Figure 6.6: Shown is a JSP site used to select different algorithms for analyzing a plate. Displayed is the plate name (1), names (2) and descriptions (3) of the analyzers, checkboxes (4) for selecting them, and a start button (5).

The generation of the list of analyzers is done using the `WebPluginManager`. This class extends the `PluginManager` class which inspects all Java archives (JAR) in a certain directory and searches for classes extending a certain basis class or implementing a specific interface. The used `WebPluginManager` looks for classes extending the `TemplateAnalyzer` class, because all used analyzers extend this base class. On the one hand this method guarantees that the list of analyzers is always up-to-date and on the other it allows a simple addition of analyzers by including them into a JAR.

Messages that are sent to the MDB `AnalyserService` contain a list of selected analyzers, the identifier of the plate, and the user's value object. Once these parameters are unpacked the `analysePlate` method is called. This method performs the following tasks:

- For each analyzer the `findResultAlgorithm` method is called which checks if the current analyzer is already stored in the database. Given that the analyzer was found the corresponding value object is returned. Otherwise a new entry is created and the new value object is returned.
- A new `AnalyserResult` entry is made which is later used to present details about the analyzing process to the user.
- Using the attached spectral information of each well a `RTPCRData` object is built which acts as input parameter for the analyzers. This objects contains an array holding either the Rn or delta Rn measurements of the used wells.

- For each selected analyzer the `calculate` method is called which returns a `ResultData` object. Data stored in this object is examined and written into a `ResultWellVO` that is associated with the corresponding well. Exceptions thrown by analyzers are caught and the error messages are stored in the `AnalyserResult` object and presented to the user.

6.3.2 Presenting results

Having again used a MDB, no mechanism for notifying the user about the completion of the job is available. Therefore the mechanism used for signaling the finish of a parsing task, described in Chapter 6.2.3 on page 54, has been adapted to the analyzer. Figure 6.7 shows the message which is displayed when a new analyzer result is available.

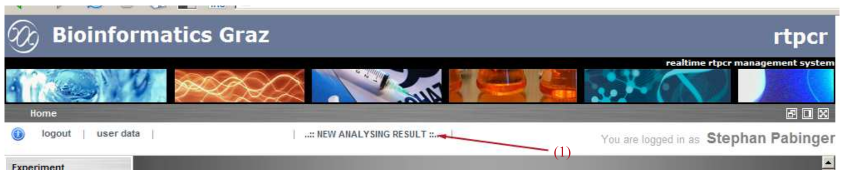

Bioinformatics Graz
rtpcr
realtime rtpcr management system
Home
logout | user data |
| ..:: NEW ANALYSING RESULT ::.. |
(1)
You are logged in as Stephan Pabinger
Experiment

Figure 6.7: Shown is the message (1) appearing when a new analyzer result is available.

If an analyzing task has been performed on a plate, the JSP page displays a button which links to the summary site of performed analyzing jobs. An example of this page is presented in Figure 6.8 on the following page. The business logic needed for this site is performed in the `DisplayAnalyseActionManual` class. It checks every well attached to a plate and searches for analyzer results, because it is possible that only a few wells were analyzed separately. These results are then collected and sent to the JSP page.

The corresponding JSP page displays the name of the plate (1) which simultaneously acts as a link back to the original plate. The analyzer results are bisected into successful (2) and not successful (3) jobs. In both cases the analyzer's name and the date of execution are displayed. Because of the fact that a plate can be analyzed several times using the same algorithm, even within one day, the date also displays minutes and seconds. This allows a better distinction between the completed jobs. Entries for analyzer tasks that were not successful provide in addition to the analyzer's name and the execution date an error message that shows detailed information about the occurred problem. For each successful task detailed information is available that can be retrieved by clicking on the corresponding »go« button.

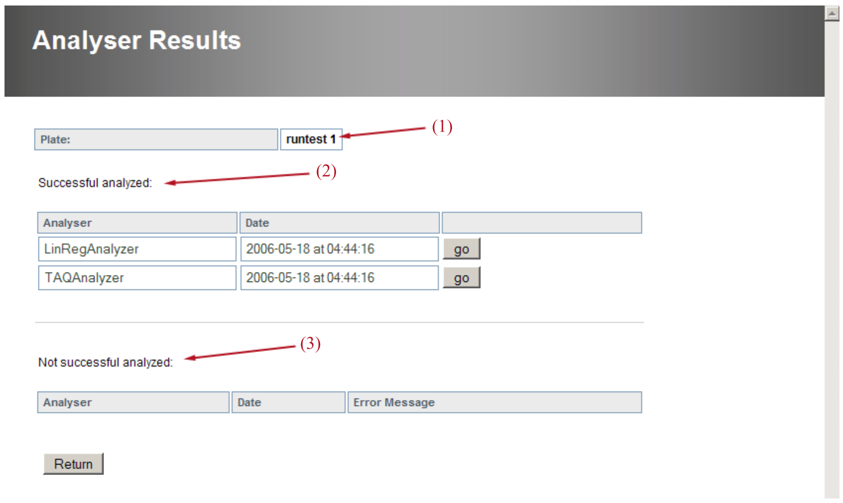


Plate: runtest 1

Successful analyzed:

Analyser	Date	
LinRegAnalyzer	2006-05-18 at 04:44:16	go
TAQAnalyzer	2006-05-18 at 04:44:16	go

Not successful analyzed:

Analyser	Date	Error Message

Figure 6.8: Displayed is the analyzer's result page. It list all successful (2) and not successful (3) analyzing jobs and provides a link back to the original plate (1).

Once the user has selected a certain successful analyzer job, a page is presented that provides detailed information about the completed calculation. An example of such a site is shown in Figure 6.9 on the next page.

The first part of the page displays the name of the analyzed plate (1), again acting as a link back to the origin, the used analyzer (2), and the date of the calculation (3). The second part consists of the computed results which are embedded in a concise table. Each row consists of the well's name (4), its corresponding sample (5), and detectors (6). Moreover C_t value (7), correlation (8), efficiency (9), and the calculated starting amount (10) are displayed.

This table provides a clear and compact list of relevant data of a real-time PCR experiment. To simplify further analysis, an export function is provided (11) that exports the list and additional information into a user defined file. Provided are Microsoft Word and Excel documents, *Comma Separated Values* (CSV) files, and plain text documents.

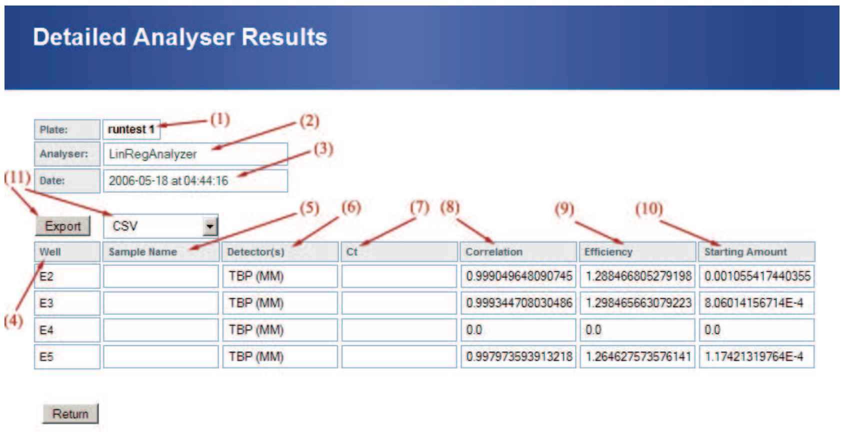

Detailed Analyser Results

Plate:	runtest 1
Analyser:	LinRegAnalyzer
Date:	2006-05-18 at 04:44:16

Well	Sample Name	Detector(s)	Ct	Correlation	Efficiency	Starting Amount
E2		TBP (MM)		0.999049648090745	1.288466805279198	0.001055417440355
E3		TBP (MM)		0.999344708030486	1.298465663079223	8.06014156714E-4
E4		TBP (MM)		0.0	0.0	0.0
E5		TBP (MM)		0.997973593913218	1.264627573576141	1.17421319764E-4

Figure 6.9: Presented is the page displaying details of a certain analyzer task. It shows the name of the plate (1), the used analyzer (2), and the calculation date (3). Results are presented in a table where information about the well (4), its sample (5) and detectors (6), the calculated C_t value (7), correlation (8), efficiency (9), and starting amount (10) is listed. Furthermore result can be exported in a file (11), whereas four different types are available.

6.4 Chart generation

Another major part of the developed application is the graphical display of real-time PCR data. These graphs are based on charts displayed in the various thermocycler software modules. In order to build up a user-friendly interface it was necessary to invent a new system using the ideas of Web 2.0, described in Chapter 3.3 on page 18.

Key features of this new system are:

Web 2.0 based Getting rid of the page reloading cycle and blank screens between user interactions was an important requirement, since it greatly improves the usability of Web applications.

Asynchronous loading Images and additional data are fetched asynchronously in the background using DWR and Ajax. During the loading process a message is displayed informing the user of the current activity.

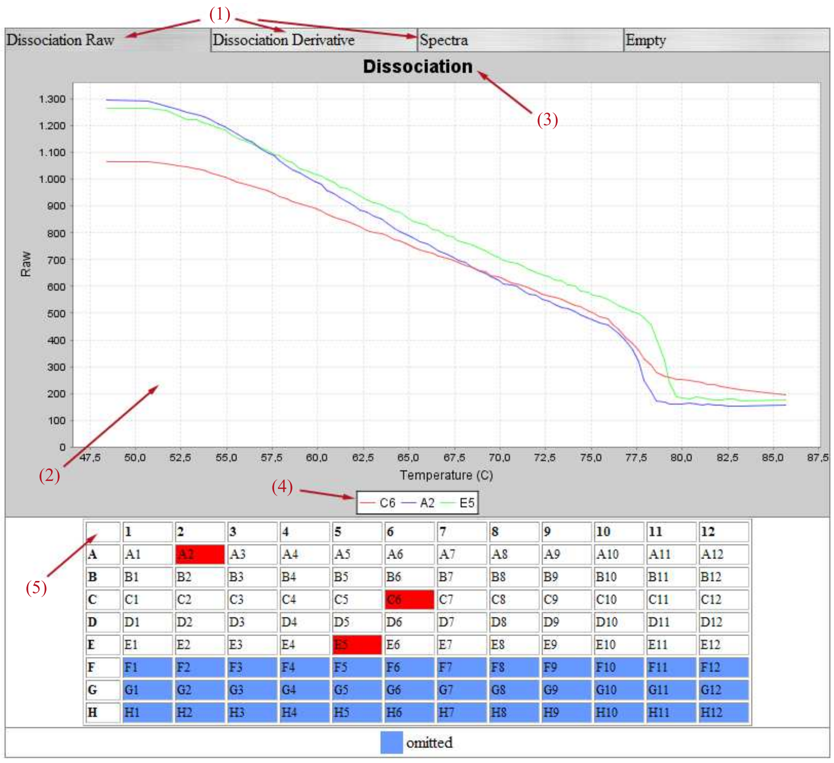
Tabbed charts Because most Web browsers support tabbed browsing users are familiar with this system. Therefore this technique is utilized for switching between charts.

JavaScript functionality A lot of user interactions are handled on the client side using JavaScript. This reduces a lot of client-server communication and provides a better event handling mechanism.

Provide familiarity Offering an interface that is as close to the PCR software as possible improves usability and reduces the need for training, because users are already accommodated to this interface.

The design of the chart page is illustrated in Figure 6.10 on the next page. The top of the page is covered by tabs (1) which are used to switch between the different chart types. Below the navigation bar the actual chart (2) is shown. It is an image using the *Portable Network Graphics* (PNG) format, because of its loss-less compression algorithm. The complete image, including its title (3) and legend (4), is created by JFreeChart (described in Chapter 3.5.1 on page 29). Underneath the figure a table representing the used plate is displayed. It is used for selecting single wells, columns or rows, or the whole plate (5). Omitted wells are colored blue and are not included in the chart. The red background color is used to mark selected wells, which can be deselected by clicking on them. Each time wells are selected or deselected the graph changes according to the current selection.

Figure 6.10 on the next page shows the dissociation curve of three selected wells. In order to identify them a legend is shown which maps the color of the single slopes to the well number. This feature allows a quick analysis of the chart and since the PCR software does not provide a legend it is an important improvement to the standard system.

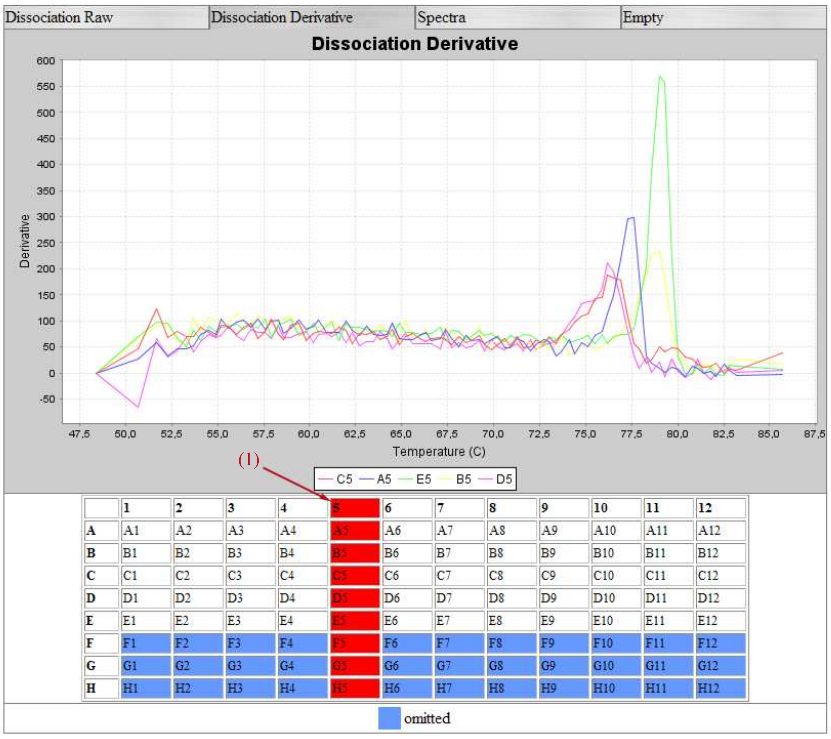


	1	2	3	4	5	6	7	8	9	10	11	12
A	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
B	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
C	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
D	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
E	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
F	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
G	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
H	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12

Figure 6.10: Displayed is the dissociation chart of three selected wells (marked in red). The tabs (1) on the top of the page are used to switch between the different chart types. The chart itself (2), the title (3), and the legend (4) are built using JFreeChart and are combined in one image. Below the figure a grid is displayed representing the layout of the used plate. By clicking on wells they are added or removed from the chart, whereas it is possible to select the whole plate (5), rows, or columns.

The second tab displays graphs of the derived dissociation curve. These slopes are more significant than the raw dissociation data, because the peaks signal the dissociation temperature of the amplified products. Multiple peaks would mean that more than one molecule has been amplified which is mostly a sign that something went wrong.

Figure 6.11 on the following page displays an example of a derived dissociation curve. In this case a complete column has been selected (1), whereas the omitted wells are not included in the graph.

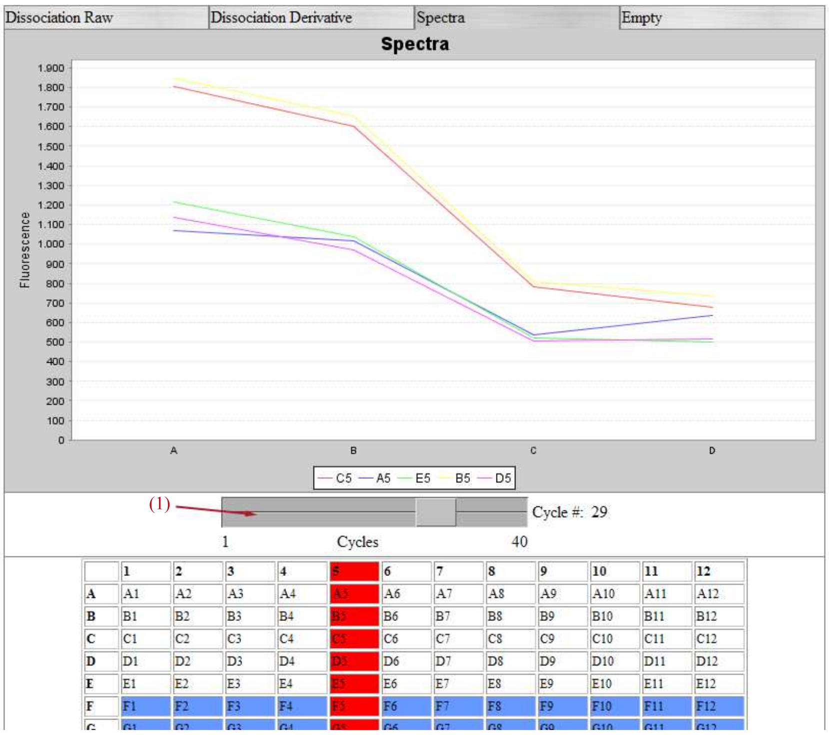


	1	2	3	4	5	6	7	8	9	10	11	12
A	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
B	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
C	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
D	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
E	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
F	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
G	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
H	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12

Figure 6.11: Shown is an example of a derived dissociation chart. Wells of a certain column (1) were selected, whereas the omitted wells are not incorporated into the chart.

The spectra tab displays the pure raw fluorescence signal, shown in Figure 6.12 on the next page. All four channels used by the the PCR system to measure the signal (each channel uses a different wavelength) are displayed in the figure.

The chart always shows one measurement of all channels (labeled as A, B, C, D) at one specific cycle. Using a slider (1) gives the user the opportunity to easily change the cycle number. Moreover, by sliding through the different cycles the progression of the real-time PCR experiment can be examined.



	1	2	3	4	5	6	7	8	9	10	11	12
A	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
B	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
C	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
D	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
E	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
F	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12

Figure 6.12: Presented is a chart showing the full spectral information for a selection of wells. The figure shows the fluorescence signal of all four channels at one specific cycle. A slider (1) is used for going through them.

All the presented charts are generated using a developed procedure. Figure 6.13 on the following page displays a sequence diagram that represents the generated curse of actions.

Plate.jsp

The process is initiated by clicking on the »`Display Parser Results`« button, shown on the plate JSP page, which calls the `display` method in the `ResultActionManual` class.

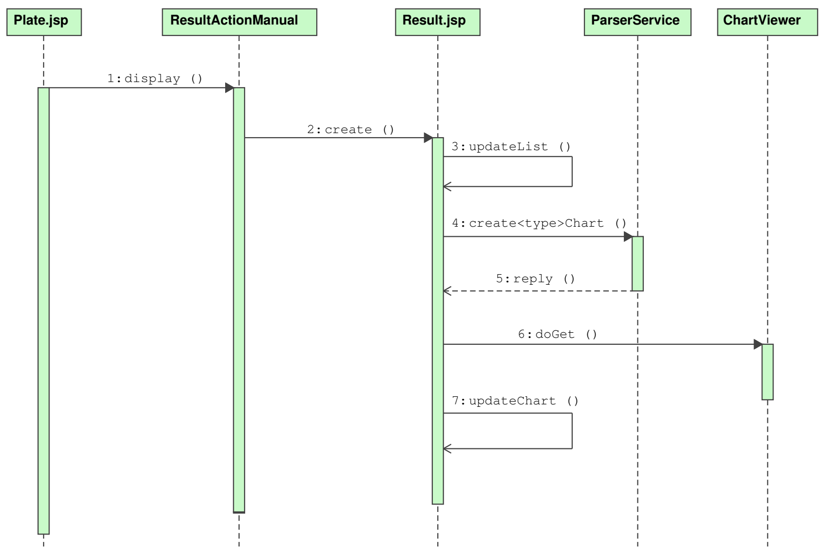


Figure 6.13: Shown is a sequence diagram that represents the course of actions needed to update a chart image

ResultActionManual

`ResultActionManual` extends the `LookupDispatchAction` and implements only the `display` method. This method gathers information about each well associated with the selected plate and builds up a list used for displaying the grid in the corresponding JSP page. This list contains wells which are sorted according to their well number. Since each plate has a defined plate size, missing wells are filled up in this list. Moreover a hash-map with the well's identifier as key and the well number as value is created and put into the `HTTPSession`. It is required to build up the legend of the chart, since only a list of identifiers is sent to the server.

Result.jsp

As soon as the Action has finished its job the `Result` JSP page is displayed, presenting an empty chart since no selection has been made. In order to provide the previously described functionality the page is packed with JavaScript. The selection of the tabs and the complete control mechanism of the grid is realized through JavaScript. An array variable `wells` is storing the current selection of wells which is updated when the user selects or deselects new wells.

Once the user selects a well JavaScript is changing the background color of the cell and adds this well to the `wells` list. Now the JavaScript method `createChart` is called which invokes a method on the server. This message is sent asynchronously using DWR to the `ParserService` class and contains the list of selected wells and the current `HttpSession` object.

ParserService

The `create<type>Chart` method performs the following things:

- Data is fetched out of the database using *Java Database Connectivity* (JDBC) and *prepared statements*. This method is much faster than using Java beans and avoids the creation of needless objects.
- A dataset is created acting as input parameter for JFreeChart.
- The actual chart is generated using the created dataset and setting the corresponding parameters (e. g. legend, title, background color).
- Directly returning the chart and using it in the JSP page is not possible, because JavaScript is not able to handle the produced data. Therefore the chart is stored in the `HttpSession` whereas the actual image production is done by an other Java class.

Result.jsp

Once the server has notified the JSP page about the completion of its work the figure needs to be generated. The source of the image tag in the JSP is a Java servlet—`ChartViewer`— that returns the produced image. In order to refresh it the complete image tag is removed and set again by JavaScript. To activate the refresh process in every browser a random dummy parameter is passed to the servlet.

ChartViewer

The `ChartViewer` servlet uses a generated chart and builds an image out of it. Image and chart generation process have been separated in order to provide a modular and reusable system. The Java class reads out the previously generated chart which is stored in the `HttpSession`. Next it creates an image using the PNG file format and sets the response content type to `image/png`. This allows the browser to interpret the returned object as an image. Moreover parameters in the header tell the browser not to cache the transferred chart. Finally the created image including the generated legend is returned to the browser.

Result.jsp

Once the picture is returned to the JSP page the browser replaces the old image with the new one matching the current well selection. During the exchanging of charts a messages is presented which tells the user that a new figure is created.

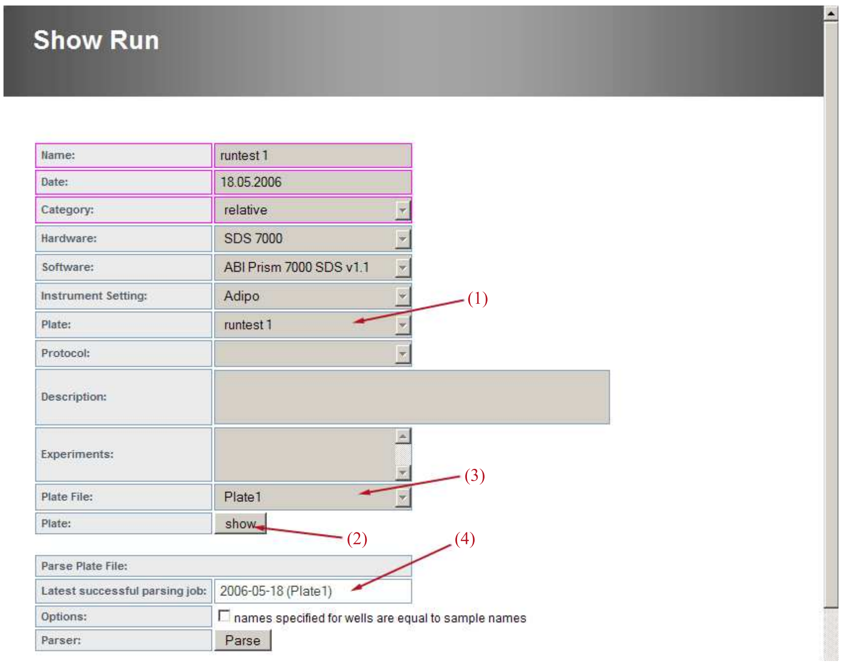
6.5 General webdesign

An important part in the developed application was the implementation of intuitive and user friendly interfaces. Since the code generated by AndroMDA is not able to create interfaces that show relationships to other entities all these connections have to be implemented by hand. Moreover the Web pages are very simple and require a lot of refinement work.

6.5.1 Run

In this page all relationships to other entities are realized using combo-boxes. Hardware, software, and instrument setting can be either specified manually or are set by the parser. The plate's name (1) is equal to the run's name at the last parsing task, whereas the plate can be viewed by clicking on the »show« button (2).

The parsing of a file can be started on this page. Therefore the correct option needs to be set and a file has to be specified (3). Finally the parser is started by clicking on the »Parse« button. Because a file can be parsed several times or another input file may be utilized, the date of the latest successful parsing job is displayed (4). An example of a run JSP page is displayed in Figure 6.14 on the next page.


Show Run

Name:	runtest 1
Date:	18.05.2006
Category:	relative
Hardware:	SDS 7000
Software:	ABI Prism 7000 SDS v1.1
Instrument Setting:	Adipo
Plate:	runtest 1
Protocol:	
Description:	
Experiments:	
Plate File:	Plate1
Plate:	show
Parse Plate File:	
Latest successful parsing job:	2006-05-18 (Plate1)
Options:	☐ names specified for wells are equal to sample names
Parser:	Parse

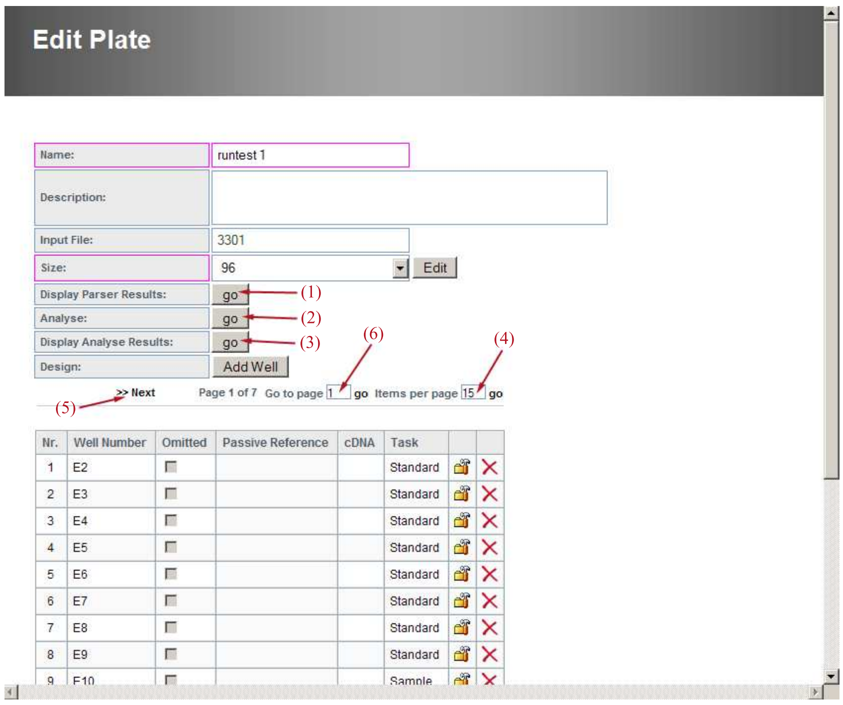
Figure 6.14: Displayed is the interface of a run. Needed for parsing is the definition of a plate file (3) and the correct setting of the parser option. A click on »Parse« starts the parser which sets the name of the plate (1) equal to the run's name and inserts the text to display the latest successful parsing job (4). The generated plate can be viewed by clicking on »show« (2).

6.5.2 Plate

The plate JSP page is divided into two parts. The first part covers general information about the plate and provides links to various additional details. The second part is occupied by a list containing the associated wells. Figure 6.15 on the following page shows an example of this JSP site.

Name, description, input file, and size are the displayed properties of a plate. Clicking on the button next to »Display Parser Results« (1) lets the chart images appear. The button besides »Analyse« (2) starts the analyzing process whose results can be viewed by clicking on the button next to »Display Analyse Results« (3). Pressing »Add Well« adds a new well to the list as long as the plate size is not reached.

The list of associated wells is embedded in a table which is displayed below the general plate information. Each well can be edited or deleted whereas well number, omitted status, passive reference, cDNA, and task are displayed in the table. To provide a clear view the table is spitted into several pages containing as many items as defined by the user (4). The different pages can be accessed by a scrolling mechanism (5) or by directly jumping to a certain page (6).



Edit Plate

Name:	runtest 1
Description:	
Input File:	3301
Size:	96 Edit
Display Parser Results:	go
Analyse:	go
Display Analyse Results:	go
Design:	Add Well

>> Next Page 1 of 7 Go to page 1 go Items per page 15 go

Nr.	Well Number	Omitted	Passive Reference	cDNA	Task		
1	E2	☐			Standard	✎	✗
2	E3	☐			Standard	✎	✗
3	E4	☐			Standard	✎	✗
4	E5	☐			Standard	✎	✗
5	E6	☐			Standard	✎	✗
6	E7	☐			Standard	✎	✗
7	E8	☐			Standard	✎	✗
8	E9	☐			Standard	✎	✗
9	E10	☐			Sample	✎	✗

Figure 6.15: Shown is a plate JSP page. It displays information about a plate and provides links leading to charts (1), to the analyzer results (3), and to the page for starting different analyzers (2). The list of wells can be scrolled (5,6) and customized by the user (4).

6.5.3 Well

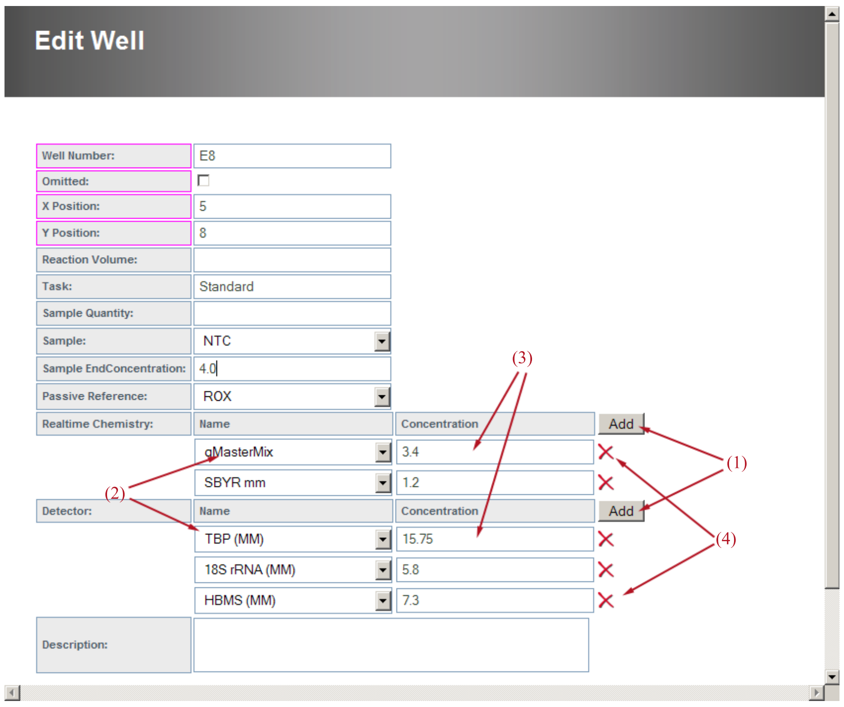
The well JSP page covers all properties of a well, displayed in Figure 6.16 on page 72. These are:

- well number
- omitted status
- x position—needed to locate the well on the plate
- y position—needed to locate the well on the plate
- reaction volume

- task
- sample quantity
- sample—is set by the parser if the user has specified it
- sample end concentration
- passive reference
- real-time chemistry
- detector
- description

Since a well can have more than one real-time chemistry or more than one detector a mechanism has been developed to easily manage these associations. By clicking on the »add« button (1) a new row pops up where the user can select the required real-time chemistry or detector (2) and may specify a concentration (3).

By clicking on the delete symbol (4) the entry is deleted from the list. In order to reduce database transactions these selections are stored in temporary lists which are committed to the database once the user has pressed the »update« button.



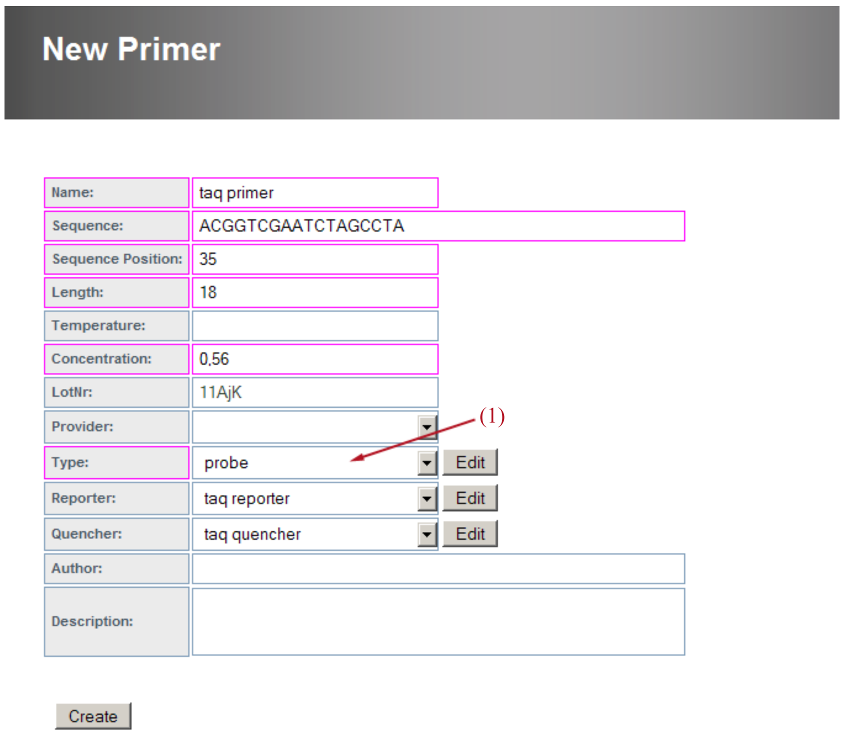
Edit Well

Well Number:	E8		
Omitted:	☐		
X Position:	5		
Y Position:	8		
Reaction Volume:			
Task:	Standard		
Sample Quantity:			
Sample:	NTC		
Sample EndConcentration:	4.0		
Passive Reference:	ROX		
Realtime Chemistry:	Name	Concentration	Add
	qMasterMix	3.4	✕
	SBYR mm	1.2	✕
Detector:	Name	Concentration	Add
	TBP (MM)	15.75	✕
	18S rRNA (MM)	5.8	✕
	HBMS (MM)	7.3	✕
Description:			

Figure 6.16: Displayed is the page for manipulating the attributes of a well. Adding or deleting references to real-time chemistries or wells is done by clicking on a button (1) or image (4). Each row consists of the entity itself (2) and a user defined concentration (3).

6.5.4 Primer

The primer interface allows the definition of the following attributes: name, sequence, sequence position, length, temperature, concentration, lot number (an identification number), primer, type, author, and description. The type of a primer can be *forward*, *reverse*, or *probe* (1). If the user has selected a probe primer a reporter and a quencher need to be specified. The corresponding combo-boxes are only visible when the selected type is set to probe.

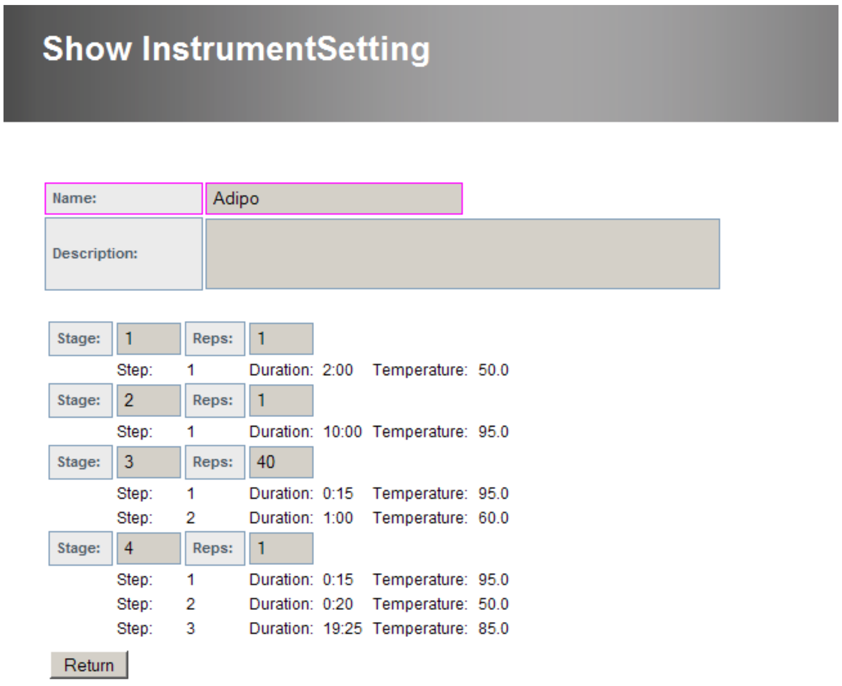


Name:	taq primer
Sequence:	ACGGTCGAATCTAGCCTA
Sequence Position:	35
Length:	18
Temperature:	
Concentration:	0,56
LotNr:	11AjK
Provider:	
Type:	probe
Reporter:	taq reporter
Quencher:	taq quencher
Author:	
Description:	

Figure 6.17: Show is the primer interface. Defining a probe primer (1) requires the definition of a reporter and a quencher, which are only displayed when the type is set to probe.

6.5.5 Instrument setting

General attributes of an instrument setting are name and description. Each instrument setting consists of a sequence of stages which can be repeated several times. A stage consist of several steps, whereas each step has a duration and a temperature specified.



Show InstrumentSetting

Name:	Adipo
Description:	

Stage:	1	Reps:	1			
	Step: 1		Duration: 2:00		Temperature: 50.0	
Stage:	2	Reps:	1			
	Step: 1		Duration: 10:00		Temperature: 95.0	
Stage:	3	Reps:	40			
	Step: 1		Duration: 0:15		Temperature: 95.0	
	Step: 2		Duration: 1:00		Temperature: 60.0	
Stage:	4	Reps:	1			
	Step: 1		Duration: 0:15		Temperature: 95.0	
	Step: 2		Duration: 0:20		Temperature: 50.0	
	Step: 3		Duration: 19:25		Temperature: 85.0	

Figure 6.18: Displayed is the JSP page presenting a certain instrument setting.

Chapter 7

Discussion

The main goal of this thesis was the development of a Web-based application for managing and analyzing real-time PCR data.

Web-based systems have the advantage to be accessible from any computer having access to a network and suit therefore the needs of biologists, since they perform their work at several different places. The application itself is based on the J2EE platform and has been implemented using a MDA approach. In combination with a code generator this method allows the implementation of software that reflects the developed model, is easy to maintain, and simple to extend.

The free available and platform independent J2EE technology supports the development of 3-tiered applications. Employing this architecture provides the following advantages:

- Each tier can be replaced or modified separately without affecting the other ones.
- J2EE containers provide a mechanism that supports simplified scaling of distributed applications. Because they may run on multiple systems, containers can automatically balance load in response to varying demand.
- Since all business logic is performed on the application server, clients require only rudimentary hardware.
- Security can be implemented at multiple levels making it more difficult for a client to obtain unauthorized data.

Although MDA in combination with a code generator has a lot of advantages some drawbacks have to be considered. Whenever a new technology is used it takes some time until the code generator has been adapted to it. This is done either by the vendor of the code generator or by the developer who has to implement new templates, which is mostly a very time-consuming job. Moreover in some cases the model has to be changed in oder to adapt it to the new technology.

Since each real-time PCR experiment produces a vast amount of data a parser has been incorporated that automatically reads in the generated measurements. Once the data is included in the system the user can view the information on graphs that are based on figures displayed in the PCR software. This unique system adds an important part to the usability of the software since graphs are often more significant than pure raw data tables. Moreover it makes use of Web 2.0 ideas that try to remove the dusty behavior of standard Web applications by implementing features known from common desktop programs.

Raw real-time PCR data does not provide meaningful results, since it has to be analyzed first. Usually the biologist has to use several different software modules, has to handle several sheets just to be able to compare the results, and even has to convert the data just to be able to use the desired software. Having now the possibility to use different analyzers (even simultaneously) from one centralized system greatly enhances the, otherwise arduous, job. Moreover the developed system allows an easy integration of new analyzers which helps to keep the analysis process up-to-date. A new analyzer is added to the system by just including its code into a certain JAR file, whereas the analyzer has to extend the `TemplateAnalyzer` basis class. From now on the included analyzer can be used without having to change the code of the application or adding entries to the database.

All these features in combination with the implemented interfaces and business logic build up an unique application that can really ease the work with real-time PCR experiments.

7.1 Usability testing

Usability testing is a method for measuring how well people can use a human-made product for its intended purpose. Therefore a scenario is provided that contains a list of tasks that are performed by the user. An observer watches the progress of the test and takes notes of how well the user executes the given tasks. In order to get to know the user's way of thinking he/she is motivated to think aloud (verbal expressing of thoughts) during the test [Andrews, 2006].

This kind of test ensures that each user performs the same tasks without having used the system before. It guides them through the system and its design allows to test all relevant functions of the application. The *thinking aloud* method makes sure that many usability problems are found whereas only a small number of test users is needed. Moreover it identifies the reason of the occurred problem and eases the process of finding and correcting them.

The performed usability test consisted of eight different tasks. Five users conducted the test, whereas three of them are working as biologists and two are computer scientists. Beginning with relatively easy exercises and ending with uncommon tasks, all primary functions of the software were included in the scenario. To measure the performance of a user executing a task, the following criteria were defined:

Time For each task the time until the user completed (successful or not successful) it was taken.

Errors Errors for each task were collected. An error is either a wrong outcome or a deviation from the optimal path.

Success For each task one or many success criteria were defined.

After the test has been performed the user was asked to fill out an overall questionnaire, accessible in Chapter C.2 on page 92. Figure 7.1 on the next page displays the most important results of this evaluation.

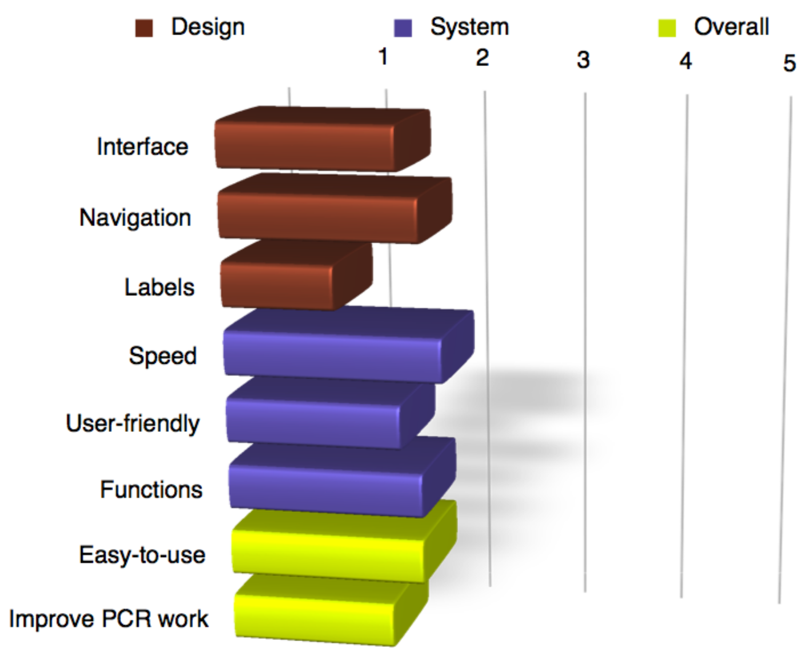


Figure 7.1: Presented is the evaluation of the overall questionnaire. The users were asked to rate aspects of the system from 1 to 5, where 1 is the best mark.

The test has shown that users are able to handle the application and are quite satisfied with the usability. Most difficulties were based on the rather new chart interface, since users are not accommodated with the provided functionality.

7.2 Perspectives

As the usability test has shown the developed application already improves the work of real-time PCR experiments. Nevertheless adding additional features to the system can lead to an even more valuable system.

Identified improvements:

- Incorporation of an improved parser offering the ability to read in additional datasets and attributes
- Ability to choose the dataset where the analysis is performed on

- Integration of the amplification plot
- Adding analyzers that are based on the algorithms used in the various PCR software modules
- Providing a status bar that displays the progress of parser and analyzer jobs

Because of its MDA approach and state-of-the-art software technologies the developed application acts as a perfect starting point for further enhancements.

Appendix A

User Requirements Document

A.0.1 Realtime RT-PCR

Being used as a diagnostic tool both in clinical and research settings, realtime reverse transcriptase polymerase chain reaction *(realtime RT-PCR)* has never been as important as nowadays. Compared to commonly used techniques like Northern blot analysis, realtime RT-PCR can be used to quantify RNA from very small samples. In addition to its great sensibility, which allows detection of products at very low concentration, it has the advantage of producing results immediately.

The general functionality of realtime RT-PCR can be described as followed: The quantity of a RNA-sample is determined by doing a *normal* PCR and comparing the expression level of the target RNA with a control RNA. To perform a *normal* PCR one needs RNA, which is reverse transcribed into cDNA. Moreover dNTPs, a special polymerase and primers are required.
This cDNA is then amplified by doing a cycle of these steps:

1. Denaturation - performed at approx. 95 °C
   During denaturation, the double strand melts open to single stranded DNA.
2. Annealing - performed at approx. 55 °C
   Now the primers bind to the complementary strand of the cDNA and the polymerase starts copying the template.
3. Extension - performed at approx. 72 °C
   The temperature is raised again to 72 °C because this is the ideal working temperature of the used polymerase which now reads the template and adds complementary bases to the strand.

During this process the PCR-machine measures the fluorescence signal in each step and plots the result. The fluorescence signal is emitted by a special chemistry which binds on the cDNA. Examples of such chemistries are: TaqMan, Molecular Beacons, Scorpions and SYBR Green, which is used in our laboratory. The more cDNA is amplified the stronger the fluorescence signal gets.
The following tasks have to be performed during a realtime RT-PCR experiment:

- RNA extraction
  The first step is to extract RNA out of the original sample.
- cDNA synthesis
  Then the RNA is reverse transcribed into cDNA.
- Data acquisition
  Put the cDNA with appropriate primers, dNTPs, ... into the realtime PCR system and start the PCR.

- Normalization
  After a successful run it is necessary to normalize the data in order to get appropriate results.
- Data analysis
  The last step is to analyze the collected data.

A.0.2 Project goal

The aim of this project is to develop a web based database which is able to manage all accumulated data during a realtime RT-PCR experiment. It provides the possibility to insert/upload data according to the steps of a realtime RT-PCR experiment. This database will be incorporated into the existing MARS system that enables reuse of multiple already developed modules.
Due to the fact that realtime RT-PCR experiments can be run with different fluorescence chemistries, different quantification methods, ... sundry normalization methods are incorporated.
Moreover to be able to fully describe samples and its extracts a module using ontologies will be integrated. The design of this model should be as general as possible so it can be used in any other project as well.

A.0.3 Software Environment

The project is based on a three tier architecture using the Java 2 Platform, Enterprise Edition (J2EE). J2EE is a standard for developing multi tier enterprise applications, that uses Enterprise Java Beans (EJB), Servlets, Java Server Pages (JSP) and XML. The three tiers in detail:

- Data Tier - Relational database (Oracle).
- Middle Tier - JBoss application server, manages access to database and interaction with data.
- Presentation tier - Web server in conjunction with a servlet-container using JSPs.

The Open Source project AndroMDA is used to generate deployable code out of UML models.

A.1 Project realization

A.1.1 Introduction

The final software should provide a flexible and easy to use interface and should offer a complete coverage of the entire realtime RT-PCR process. Due to the fact that the realtime RT-PCR experiment follows several well defined steps the software will be divided according to this tasks.

A.1.2 Basic units

Basic units are components which are highly reusable. Therefore they are managed separately.

A.1.2.1 Protocol

Protocols can be loaded into the system. The user needs to specify a name for the protocol, chose a category and may enter a description of the protocol. To ensure that it is readable by the system it has to be in an ASCII format. Additionally the user can add another file (e.g. in pdf, doc format).

A.1.2.2 Provider

Providers are also managed separately. Mandatory fields are name and abbreviation.

A.1.2.3 Software

The unit software manages the used software systems in different experiment steps. The user has to define name, version, type and description of it.

A.1.2.4 Hardware

The unit hardware manages the used hardware systems. The user has to define name, version, type and description of it similar to software systems.

A.1.2.5 Upload Zone

In the upload zone the user manages files that are used in the system. Any file that will be utilized has to be uploaded first and can be analyzed or linked afterwards. The user has to define name, file type (are managed separately), path to the file and eventually a description of it. According to the file type the system tries to find an appropriate parser and stores the data into the database which can now be analyzed by provided applications.

A.1.3 Experiment

The root of every realtime RT-PCR run is an experiment. The user is able to add new runs to an experiment as well as modify and delete existing ones.

A.1.4 Run

It describes the entire process of a single realtime RT-PCR run. To ensure maximum flexibility one run can be used in several experiments. At this stage the user defines the plate layout (96, 384,...) which is not limited to any special design. Because of the fact that plates often differ in only a few parameters (used cDNA,...) the user is able to save a plate design and use it in future runs. Now he/she has to change only a few parameters and does not have to design the whole plate again.

A.1.4.1 Sample description

The first thing the user needs to specify is the used sample. In the first project phase the MARS sample description will be used.
At the moment samples can be annotated in a user-customizable manner. Four different annotation types are provided: enumeration, numbers, integers and free text. In the future it should be possible to annotate samples by using a well defined ontology. This system allows better search results and helps structuring annotations.

A.1.4.2 RNA extraction

Now it is necessary to extract the RNA out of the sample. The description of this process is also adopted from the MARS database. Amongst other things the extraction method, quantity and concentration can be specified.

A.1.4.3 cDNA creation

Before the RNA is transformed into cDNA all existent DNA is erased by DNAse treatment. To describe this step the user can upload a protocol (step is optional).
To perform the PCR one needs to transform the RNA into a cDNA. The used protocol can be uploaded or manually inserted into the system. Moreover the final amount and concentration of cDNA and the dilution factor can be entered.

A.1.4.4 PCR

The following parameters have to be specified when performing the PCR run:

- Master-mix
  Specify the used master-mix by chosing it from a list and insert the accordant concentration. If a self-made mastermix is used the user can specify the corresponding protocol.
- Primer
  Define the used primer (chose from a combobox) and insert concentration and used amount. Information about sequence, length, melting point, author and provider can be stored.
- Realtime Chemistries
  Specify the realtime chemistry (e.g. TaqMan, Scorpions, SYBR, ...) by choosing an entry from a list. The list is managed separately. In addition information about provider, version, ... can be saved.
- Passive reference (ROX)
  Define the used ROX Reference Dye from a given list. If necessary the user can add entries to the list.
- dNTPs
  Specify the used dNTPs, amount, provider.
- cDNA
  Define the amount, concentration and the used cDNA.

A.1.4.5 Results

Most of the data that is necessary for displaying the results is stored into a file which is generated by the realtime PCR system. This file is parsed by the software and the data is stored into the database. Additionally the required data can be inserted manually. The following parameters have to be specified:

- Used software
  The used software is chosen from a list. If necessary the user can add entries to the list.
- Plate design
  According to the previously defined plate the data is either inserted manually or parsed out of the result file.
  Properties for each well are:
  - sample name
  - use
  - detector
  - reporter
  - quencher
  - task
  - quantity
  - color
  - passive reference
  - omitted
- PCR parameters
  Parameters concerning the PCR are either inserted manually or parsed out of the result file. Information about the used temperatures, durations, cycles is stored.
- Melting point
  Define the temperature for the dissociation protocol. Additionally the dissociation figure will be displayed.
- Sample volume
  Specifies the used volume in each well.
- Figures
  Figures that are generated by the realtime PCR system are rebuilt by the software using the result file.
  Relevant for evaluating results are:
  - Amplification Plot
  - Standard Curve
  - Dissociation Figure

Appendix B

Class diagram

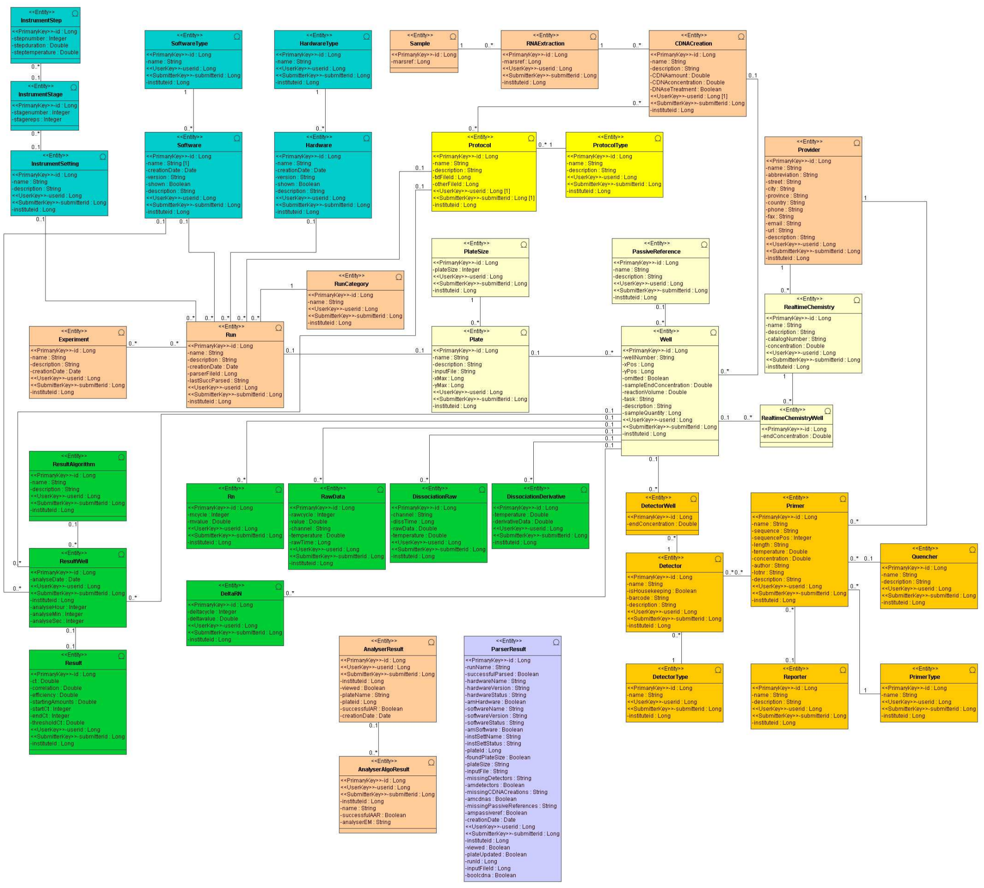


Figure B.1: Presented is the complete entity diagram.

Appendix C

Usability testing

C.1 Task-list and test-score

This list displays each task and shows the collected results. Time is taken in minutes, errors are counted as they occurred (also wrong paths are counted as error), and success is noted in percentages.

1. Add an Experiment filling all fields

	Bio1	**Bio2**	**Bio3**	**IT1**	**IT2**
Time	0.66	0.9	0.66	1	0.9
Errors	0	0	0	0	0
Success	100	100	100	100	100

Table C.1: Score—Add experiment

2. Add a Run and fill out »Name«, »Date«, and »Category« and specify Experiment

	Bio1	Bio2	Bio3	IT1	IT2
Time	2.75	2.25	0.75	0.9	1
Errors	1	1	0	0	0
Success	75	75	100	100	100

Table C.2: Score—Add run

3. Upload a Plate-File and specify it as »Plate«

	Bio1	Bio2	Bio3	IT1	IT2
Time	1	0.75	0.66	1.33	1.4
Errors	0	1	0	0	0
Success	100	100	100	100	100

Table C.3: Score—Upload file

4. Edit and Update newly added Run: specify the newly added Plate and change Name

	Bio1	Bio2	Bio3	IT1	IT2
Time	3	1.25	1.17	2.33	1.5
Errors	2	1	0	2	0
Success	100	75	100	100	100

Table C.4: Score—Edit run

5. Select Run and start Parser

	Bio1	Bio2	Bio3	IT1	IT2
Time	1.75	0.75	0.9	0.83	0.5
Errors	0	0	0	1	0
Success	100	100	100	100	100

Table C.5: Score—Start parser

6. View Parser results

	Bio1	Bio2	Bio3	IT1	IT2
Time	1.17	0.66	0.75	2.33	0.66
Errors	0	0	0	2	0
Success	100	100	100	100	100

Table C.6: Score—View parser results

7. View Plate of added Run and show Well A5

	Bio1	Bio2	Bio3	IT1	IT2
Time	2	0.5	0.9	0.833	2.33
Errors	0	0	0	1	3
Success	100	100	100	100	50

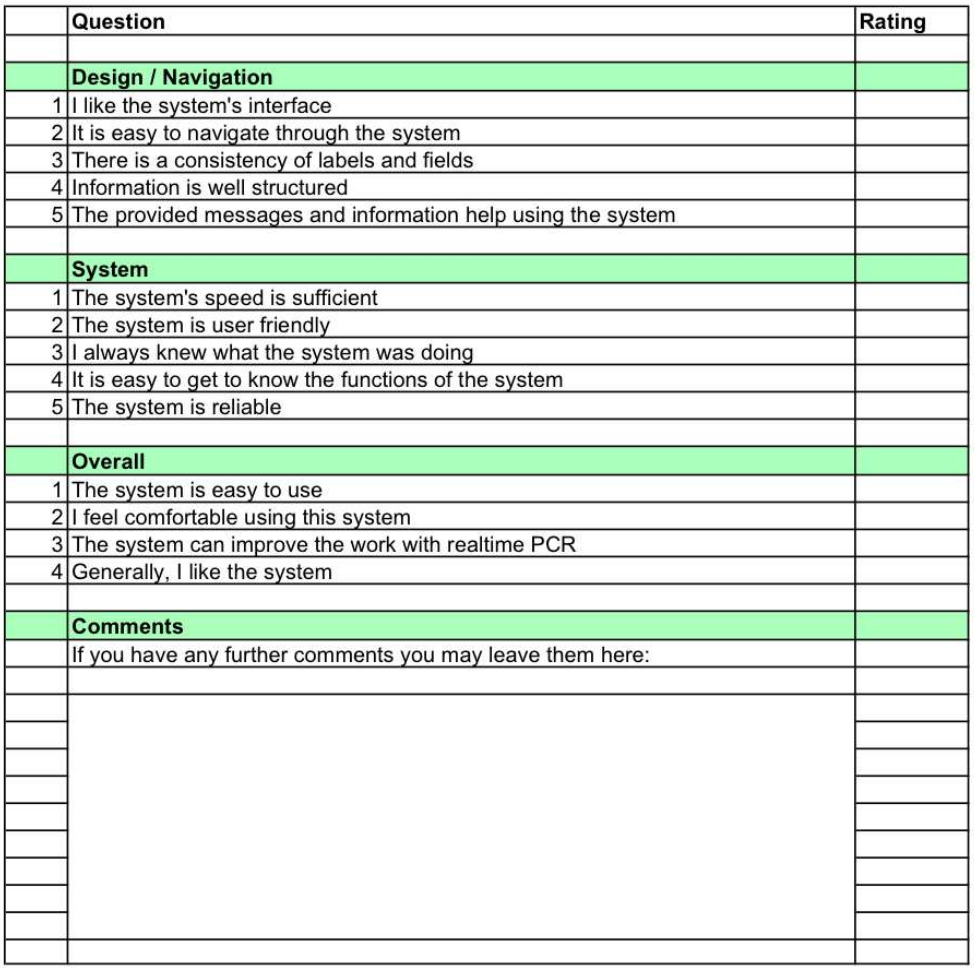
Table C.7: Score—View plate

8. Display »Parser Results« of Plate and
    - View Dissociation of well A6
    - View Spectra of column 4

	Bio1	Bio2	Bio3	IT1	IT2
Time	6.66	3.25	1.5	1.5	0.75
Errors	1	2	0	1	1
Success	100	100	100	100	75

Table C.8: Score—View charts

C.2 Overall questionnaire



	Question	Rating
	Design / Navigation	
1	I like the system's interface	
2	It is easy to navigate through the system	
3	There is a consistency of labels and fields	
4	Information is well structured	
5	The provided messages and information help using the system	
	System	
1	The system's speed is sufficient	
2	The system is user friendly	
3	I always knew what the system was doing	
4	It is easy to get to know the functions of the system	
5	The system is reliable	
	Overall	
1	The system is easy to use	
2	I feel comfortable using this system	
3	The system can improve the work with realtime PCR	
4	Generally, I like the system	
	Comments	
	If you have any further comments you may leave them here:	

Figure C.1: Shown is the overall questionnaire given to each user after completing the tasks.

	Bio1	**Bio2**	**Bio3**	**IT1**	**IT2**
Interface	2	2	1	2	1
Navigation	2	2	1	2	2
Labels/Fields	1	1	1	1	1
Information	2	2	1	2	1
Messages	3	3	2	3	2
Speed	1	3	2	3	1
User friendly	2	3	1	1	1
Systems behavior	3	3	2	2	3
Functions	2	2	1	2	2
Reliability		2	1	1	1
Easy to use	2	2	1	2	2
Comfortableness	1	2	1	1	1
Improve PCR work	2	2	1		1
I like system	2	2	1	1	1

Table C.9: Score—Overall questionnaire

Figures

Tables

Bibliography

[Abd-Elsalam, 2003] Kamel A. Abd-Elsalam. Bioinformatic tools and guideline for PCR primer design. *African Journal of Biotechnology*, 2(5):91–95, May 2003. Accessible through the Web at »`http://www.academicjournals.org/ajb/PDF/Pdf2003/MayPDFs2003/Abd-Elsalam.pdf`« (last visited on May 12, 2006).

[Alur *et al.*, 2001] Deepak Alur, Dan Malks, and John Crupi. *Designing Enterprise Applications–with the Java 2 Platform, Enterprise Edition*. Prentice Hall PTR, New Jersey, 2001.

[Andrews, 2006] Keith Andrews. *Human-Computer Interaction*, 2006. Accessible through the Web at »`http://courses.iicm.edu/hci/hci.pdf`« (last visited on June 3, 2006).

[AndroMDA, 2005a] AndroMDA. *AndroMDA Cartridges*, November 2005. Accessible through the Web at »`http://galaxy.andromda.org/docs-3.1/andromda-cartridges/index.html`« (last visited on April 17, 2006).

[AndroMDA, 2005b] AndroMDA. *What is AndroMDA?*, November 2005. Accessible through the Web at »`http://galaxy.andromda.org/docs-3.1/whatisit.html`« (last visited on April 17, 2006).

[Apache Software Foundation, 2006a] Apache Software Foundation. *Ant*, January 2006. Accessible through the Web at »`http://ant.apache.org`« (last visited on April 26, 2006).

[Apache Software Foundation, 2006b] Apache Software Foundation. *Maven*, April 2006. Accessible through the Web at »`http://maven.apache.org`« (last visited on April 26, 2006).

[Bodoff *et al.*, 2001] Stephanie Bodoff, Dale Green, Eric Jendrock, Monica Pawlan, and Beth Stearns. *The J2EE Tutorial*. Sun Microsystems, Inc, 2001.

[Cavaness, 2002] Chuck Cavaness. *Programming Jakarta Struts*. O'Reilly & Associates, Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo, November 2002.

[Eclipse, 2006] Eclipse. *Eclipse Platform Technical Overview*, 2006. Accessible through the Web at »`http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf`« (last visited on April 24, 2006).

[Elrich, 1989] Henry A. Elrich. *PCR Technology: Principles and Applications for DNA amplification*. Stockton Press, New York, 1989.

[Fields *et al.*, 1999] Stanley Fields, Yuji Kohara, and David J. Lockhart. Funtional genomics. *Proc. Natl. Acad. Aci. USA*, 96:8825–8826, August 1999.

[Florida Museum of Natural History, 2006] Florida Museum of Natural History. *PCR*, 2006. Accessible through the Web at »`http://www.flmnh.ufl.edu/cowries/PCR.gif`« (last visited on May 13, 2006).

[Gamma *et al.*, 1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[Getahead, 2005] Getahead. *DWR: Easy AJAX for JAVA*, August 2005. Accessible through the Web at »http://getahead.ltd.uk/dwr/overview/dwr« (last visited on April 19, 2006).

[Gilbert, 2002] David Gilbert. *The JFreeChart Class Library - Reference Documentation*, June 2002.

[Gilbert, 2005] David Gilbert. *The JFreeChart Class Library - Installation Guide*, December 2005.

[Human Genome Project, 2004] Human Genome Project. *Human Genome Project Information*, 2004. Accessible through the Web at »http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml« (last visited on April 10, 2006).

[Husted *et al.*, 2003] Ted Husted, Cedric Dumoulin, George Franciscus, and David Winterfeldt. *Struts in Action–Building web applications with the leading Java framework*. Manning Publications Co, Greenwich, 2003.

[Kassem and the Enterprise Team, 2000] Nicholas Kassem and the Enterprise Team. *Core J2EE Patterns: Best Practices and Design Strategies*, Volume 1.0.1. Sun Microsystems, Inc, October 2000. Accessible through the Web at »http://java.sun.com/j2ee/1.4/docs/tutorial/doc/J2EETutorial.pdf« (last visited on April 22, 2006).

[Lodish *et al.*, 2000] Harvey Lodish, Arnold Berk, S. Lawrence Zipursky, Paul Matsudaira, and David Baltimore. *Molecular Cell Biology*. W. H. Freeman Company, New York, 4th Edition, 2000.

[McLaughlin, 2006] Brett McLaughlin. *Mastering Ajax, Part 2: Make asynchronous requests with JavaScript and Ajax*, 2006. Accessible through the Web at »http://www-128.ibm.com/developerworks/java/library/wa-ajaxintro2/index.html« (last visited on April 26, 2006).

[Miller and Mukerji, 2003] Joaquin Miller and Jishnu Mukerji. *MDA Guide Version 1.0.1*, June 2003. Accessible through the Web at »http://www.omg.org/docs/omg/03-06-01.pdf« (last visited on April 20, 2006).

[No Magic Inc., 2006] No Magic Inc. *Introducing MagicDraw*, March 2006. Accessible through the Web at »http://www.magicdraw.com« (last visited on April 24, 2006).

[Object Management Group, 2005] Object Management Group. *Introduction to OMG's Unified Modeling Language (UML)*, July 2005. Accessible through the Web at »http://www.omg.org/gettingstarted/what_is_uml.htm« (last visited on April 19, 2006).

[Object Management Group, 2006] Object Management Group. *About the Object Management Group*, 2006. Accessible through the Web at »http://www.omg.org/gettingstarted/gettingstartedindex.htm« (last visited on April 23, 2006).

[Ostermeier *et al.*, 2003] G. Charles Ostermeier, Zhandong Liu, Rui Pires Martins, Rikki R. Bharadwaj, James Ellis, Sorin Draghici, and Stephen A. Krawetz. Nuclear matrix association of the human β-globin locus utilizing a novel approach to quantitative real-time PCR. *Nucleic Acids Research*, 31(12):3257–3266, 2003.

[Ramakers *et al.*, 2003] Christian Ramakers, Jan M. Ruijter, Ronald H. Lekanne Deprez, and Antoon F.M. Moorman. Assumption-free analysis of quantitative real-time polymerase chain reaction (PCR) data . *Neuroscience Letters*, 339:62–66, 2003.

[Roman *et al.*, 2002] Ed Roman, Scott Ambler, and Tyler Jewell. *Mastering Enterprise JavaBeans*. John Wiley & Sons, Brisbane · Chichester · New York · Singapore · Toronto · Weinheim, second Edition, 2002.

[Roth and Pelegri-Llopart, 2003] Mark Roth and Eduardo Pelegri-Llopart. *JavaServer Pages Specification - Version 2.0*, 2003. Accessible through the Web at »http://jakarta.apache.org/velocity/index.html« (last visited on April 22, 2006).

[Rutledge, 2004] R. G. Rutledge. Sigmoidal curve-fitting redefines quantitative real-time PCR with the prospective of developing automated high-throughput applications. *Nucleic Acids Research*, 32(22), 2004.

[Sebastiani *et al.*, 2003] Paola Sebastiani, Emanuela Gussoni, Isaac S. Kohane, and Marco F. Ramoni. Statistical Challenges in Functional Genomics. *Statistical Science*, 18, 2003.

[Struts, 2006] Struts. *The Struts tag-library*, April 2006. Accessible through the Web at »http://struts.apache.org/struts-action/struts-taglib/index.html« (last visited on April 26, 2006).

[Telerik Corporation, 2005] Telerik Corporation. *Using AJAX in Web Applications–A Practical Guide for ASP.NET Developers*, January 2005. Accessible through the Web at »http://www.telerik.com/documents/Telerik_and_AJAX.pdf« (last visited on April 21, 2006).

[Truskaller, 2003] Thomas Truskaller. *Data Integration into a Gene Expression Database*, 2003.

[Velocity, 2005a] Velocity. *Velocity Overview*, 2005. Accessible through the Web at »http://jakarta.apache.org/velocity/overview.html« (last visited on April 21, 2006).

[Velocity, 2005b] Velocity. *What is Velocity?*, 2005. Accessible through the Web at »http://jakarta.apache.org/velocity/index.html« (last visited on April 21, 2006).

[Wilhelm *et al.*, 2003] Jochen Wilhelm, Alfred M. Pingoud, and Meinhard Hahn. SoFAR: software for fully automatic evaluation of real-time PCR data. *BioTechniques*, 34(2):324–332, February 2003.

[Wilhelm, 2003] Jochen Wilhelm. *Entwicklung real-time-PCR-basierter Methoden für die moderne DNA-Analytik*. PhD thesis, Justus-Liebig-Universität Gießen, 2003.

[Wong and Medrano, 2005] Marisa L. Wong and Juan F. Medrano. Real-time PCR for mRNA quantitiation. *BioTechniques*, 39:75–85, July 2005.

[XDoclet, 2005] XDoclet. *Welcome! What is XDoclet?*, 2005. Accessible through the Web at »http://xdoclet.sourceforge.net/« (last visited on April 24, 2006).

[Zammetti, 2005] Frank W. Zammetti. *Ajax using XMLHttpRequest and Struts*, 2005. Accessible through the Web at »http://www.omnytex.com/articles/xhrstruts/xhrstruts.pdf« (last visited on April 21, 2006).

[Zeller, 2005] Dieter Zeller. *Design and development of a user management system for molecular biology database systems*, 2005. Accessible through the Web at »http://xdoclet.sourceforge.net/« (last visited on April 24, 2006).

[Zhao and Fernald, 2005] Sheng Zhao and Russell D. Fernald. Comprehensive Algorithm for Quantitative Real-Time Polymerase Chain Reaction. *Journal of Computational Biology*, 12(8):1047–1064, 2005.