Ruslan Pisarenko

THE INTEGRATION OF JASPERREPORTS TO MHG ERP SYSTEM

Bachelor's Thesis
Information Technology

May 2010

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** 24.05.2010
Author(s) Ruslan Pisarenko	**Degree programme and option** Information Technology

Name of the bachelor's thesis

The integration of JasperReports to MHG ERP System

Abstract

MHG Systems is a provider of ERP systems for businesses working in Bioenergy area. The MHG ERP Reporting module is an important part of the ERP system. The module is implemented in Java programming language and is built on Java EE framework. The main function of the module is generation of Excel documents, based on data from the database. The Excel documents featuring reports on the different activity of ERP users and are implemented in a form of Excel tables. The original module is utilizing JXLS framework to provide Excel functionality.

The aim of the project was to research and prove the possibility for usage of JasperReports framework for generation of Excel documents, as the JXLS framework proved to be unreliable and unstable. The module needed to be modified to enable this functionality. The theoretical part of the thesis describes the concepts and technologies used during the implementation of the application. The utilized technologies include MySQL database management system, Glassfish application server and Java Enterprise Edition technologies, such as Servlet API, Java Server Pages, Java Server Faces, IceFaces and Java Beans. The Spiral method of software development was used to develop the application. The practical part describes the prototyping process of application development. The development of the project also included creation of the test database and ERP classes that the module is communicating with. Because the source code of the Reporting module is confidential, the later stages of development are described in general manner, though all the principles and techniques on which the module is built are described in great detail.

As a result of the project a showcase MHG ERP Reporting module was created. The module was developed to demonstrate the complete functionality on example of one report type. During the development process the principle of enabling advanced JXLS functionality with JasperReports framework was developed. This method can be useful for any project that built around migration from JXLS to JasperReports framework involving advanced JXLS templates.

The project showed the possibility of migration from JXLS to JasperReports framework with retention of full functionality. The functionality of MHG ERP Reporting module was further advanced with introduction of graphical elements in Excel reports.

Subject headings, (keywords)

Java EE, JasperReports, JXLS, Servlet, JSP, JSF, Java IceFaces, MySQL, GlassFish

Pages	**Language**	**URN**
70	English	**URN:NBN:fi:amk-2010060111109**

Remarks, notes on appendices

Tutor	**Employer of the bachelor's thesis**
Matti Juutilainen	MHG Systems Oy

CONTENTS

1 INTRODUCTION

MHG Systems Oy is a Finnish company developing Enterprise Resource Planning (ERP) systems for businesses working in the Bioenergy field. After successfully completing my professional training there I was given an opportunity to write my Bachelor's Thesis for the company.

MHG Bioenergy ERP system is a full scale ERP system utilizing modern technologies like mobile communications, the Internet, real-time maps, satellite-based location information, and CO_2 monitoring. MHG ERP allows the use of new empowering operational models, accurate cost monitoring and real-time management of enterprise resources, biomasses and delivery chains. ERP is built on modular principles, so different modules can be added and removed, to make the software fully adjustable to customer's needs.

An important feature of the system is the MHG ERP Reporting module. The module is implemented in Java programming language and is built on Java EE framework. The main function of the module is generation of dynamic Excel documents, based on data from the database. The Excel documents are featuring reports on the different activity of ERP users and are implemented in a form of Excel tables. The process of Excel report generation consists of 3 main stages: the module receives the request for report to be generated, the user is able to set additional report parameters through JSF (Java Server Faces) user interface and the report is generated and returned by a specialized Servlet. During the process of report generation the data is queried from MySQL database and a predefined template is used for report layout. The original reporting module was based on JXLS framework. In the original version there were some shortcomings and the reporting module proved to be the least reliable part of the ERP. MHG Systems keeps high standards for its software, so there was made a decision to research the opportunities to remake the entire reporting module. The JasperReports framework was proposed as an alternative to JXLS.

The main goal of the final thesis is to research and study MHG ERP Reporting module, learn about JasperReports and provide a suitable solution on how to refine and rework the Reporting module so that it will be more reliable and stable. The aim of the thesis project is to develop a real life ERP module that will be able to create Excel documents. The fact that the thesis was done for a private company introduced several limitations and extra

requirements: the program has to be fully functional, there is strong supervision over the development process from the company side, every part of the solution should be tested, memory leaks should be fixed, the solution should follow good programming practices and the security should not be compromised. The main limitation for this final thesis was the confidential source code. The later stages of development process are discussed in general manner, though all the principles and techniques on which the module is built are described in full detail. The thesis project is required to comply to following MHG ERP architecture requirements: Java EE edition should be used as an Enterprise application framework, GlassFish v 2.0 as an Application Server, MySQL as a Relational Database Management System, the user interface was to be implemented with Java Server Faces, though during the project this requirement has changed, and the final interface is implemented with IceFaces framework. JasperReports was proposed as a replacement to JXLS as an Excel report generating framework.

When I started writing my final thesis, I already had one and a half years of Java programming experience. I had good knowledge of Java programming language, had worked with Java SE (Standard Edition) and had accomplished my first two big projects – the porting of MHG Mobile application to Blackberry phones (Java Micro Edition) and Java 3D project for my Animations class. The thesis was my first experience with Server side programming, so as a starting point I have learned the basics of Client/Server architecture and HTTP protocol. As a next step, I learned the Java EE principles and technologies, like Servlets, JSP, and JSF. During the development process the company has changed its presentation layer framework– it switched to Java IceFaces, an Ajax based extension to JSF framework. As a result, I had to make some extra changes to my program. Because the module heavily utilizes databases to generate reports, I had to learn JDBC and SQL principles, to build sophisticated queries. I learned both JXLS and JasperReports frameworks that are used for Excel report generation. One of the project's major requirements is the retention of complete functionality in the modified module. The main problem is that the Reporting module was built around JXLS utilizing all features of this framework, and the two frameworks, though having many things in common, use different approaches to generate report templates. To sustain full functionality, a new principle of transferring advanced JXLS functionality into JasperReports based application was developed.

In the following chapters I am going to write the theory that I learned for this project, and then I will describe the process of making the program.

2 INTRODUCTION TO ENTERPRISE COMPUTING

Nowadays the term Enterprise software or Enterprise application is heavily used in IT industry. It is widely known, that software of this type is often very complicated and can perform a number of functions. Enterprise software is usually heavily modified to suit the needs of the company. Reesy (2000, 8) states that enterprise solution is a solution that identifies common problem domains within a business and provides a shared infrastructure for solving those problems. It also means higher level of abstraction from business culture and a common approach to business problems, while incorporating meaningful distinctions.

Farley el al (2006, 4-5) provides different definition. He states, that enterprise application is about combining separate applications, services and processes into a unified system that is greater than the sum of its parts. Farley also points out the most common characteristics of Enterprise software, such as heterogeneous network of systems and variety of hardware that application is being executed on. This could be Servers of different types, personal computers running different operating systems, as well as other types of devices (often mobile devices) that companies use. Other common traits of enterprise applications are: involvement of different standards and protocols that, for example allow communication between systems built using different architectures, as well as high complexity and large scale of applications.

A common trait of Enterprise Application is utilization of Distributed Computing techniques. As Mukhar el al (2006, 2) points out, such kind of applications require large amount of computing power and may need to run on several computers connected to each other. So, the software needs to be partitioned into functional pieces and deployed on the appropriate hardware platforms. Such approach is called Distributed Computing.

ERP (Enterprise Resource Planning) System is a good example of full scale Enterprise Application. Bidgoli (2004, 707) defines ERP as an integrated computer-based system used to manage internal and external resources of an enterprise, built on software architecture whose purpose is to facilitate the flow of information between all business functions inside the boundaries of the organization and manage the connections to outside entities. Usually such systems use a centralized database and are normally built on a common computing platform (in our case MHG ERP is built on Java EE platform). ERP systems consolidate all business operations into a uniform and enterprise wide system environment.

2.1 Software composition architectures

Every software application can be broken down into three logical layers: *Presentation layer*, *Business rules layer*, and *Data access layer*. Presentation layer deals with displaying the data to the user and handles user interaction with the software. Business rules layer, that is also called Middle tier, deals with application logic and handles important processing. Data access layer is responsible for reading and writing data. (Mukhar, 2004, 4).

Simple software applications are designed to run on a single computer as illustrated in Figure 2.1. All services the application provides: the user interface, the persistent data access, and the logic that processes the data input by the user - are executed on the same computer. *Single tier* systems are easy to manage, but they can't handle multiple users, and they do not provide any means to share data across the enterprise. (Mukhar el al, 2004, 5).

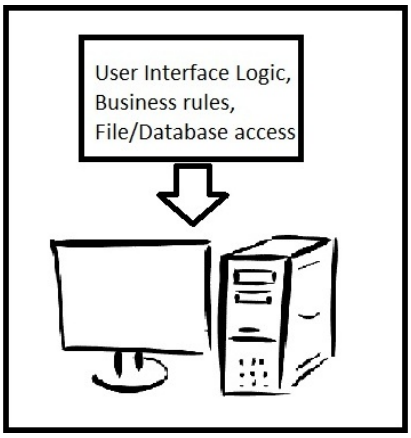


Figure 2.1. Single-tier architecture

Two-tier system architecture is more advanced software architecture, allowing separation of functions between two entities: a Client and a Server. As Yadav & Singh (2009, 1) states, a Client is any process that requests specific services from the Server process. A Server is a process that provides requested services for the Client. Client and Server processes can reside on the same computer or on different computers connected through network. Any architecture that involves Clients and Servers can be referred as Client/Server architecture. As shown in the Figure 2.2, in Two-tier systems Data access components are segregated from the rest of the application logic. The main benefit of this approach is the ability to provide a central database server to share some of the processing and to allow multiuser access to data. The disadvantage of this approach is that business logic is

hardcoded into the application and in a case of business logic modification, all the Client applications have to be updated. This is a costly and inefficient process. (Mukhar el al, 2004, 5-6).

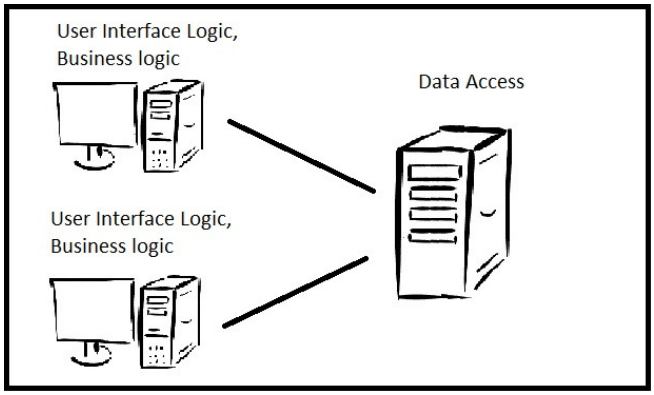
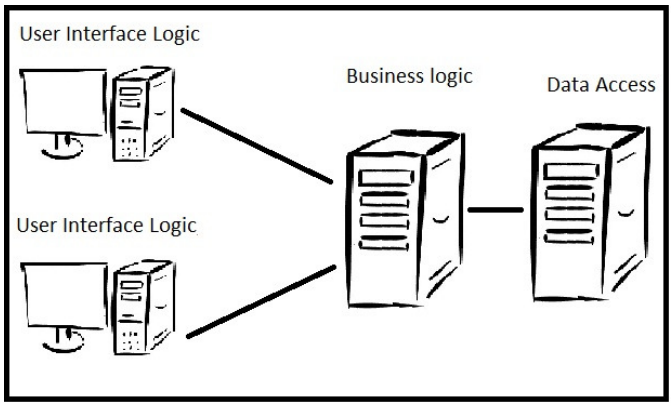


Figure 2.2. Two-tier architecture.

The majority of modern enterprises are utilizing *Multi-tier* architecture. Figure 2.3. shows typical multi-tier application that consists of three main tiers. Unlike the two-tier architecture, *Business logic* tier is executed on a Server apart from workstation. This type of architecture makes complex programs easier to understand, modify and maintain, because different tiers are not tight together. Tiers communicate to each other following individually defined rules. This allows easy modification or even migration to a different framework of one tier, without virtually any modification of other tiers. Also this allows centralized control and modification of business logic – it is much easier to modify business logic if it is running as a separate tier on a dedicated Server, than if it was built into Client applications. It is worth of notice that multi-tier architectures are not bound to any specific number of tiers, and developers can create as many tiers, as they need to. (Mukhar el al, 2004, 6-9).

Figure 2.3. Multitier Architecture.



2.2 Client and Server Types

There are two types of Clients, depending on the amount of computations performed by the client machine/process. A *Fat Client* architecture places more application functionality on the Client machine(s). (Yadav & Singh, 2009, 4). This is being achieved by developing an application running on a Client that can exchange data with Server. Java EE Client can be a console (text) application written in Java, or a GUI application written using the Java Foundation Classes (JFC) and Swing or AWT (Abstract Window Toolkit). (Mukhar el al, 2006, 11). A *Thin Client* is a Client that does minimum computations. Web browser is a typical Thin Client. (Yadav & Singh, 2009, 4).

There are several types of Servers. Some most common types are Web Server, Application Server, Database Server, File Server, DNS Server, Mail Server, Fax Server, Print Server and Transaction Server. This list is far from complete. In the context of undertaken project, three types of Servers will be discussed: Web Server, Database Server, and Application Server. (Yadav & Singh, 2009, 2-4).

Web Servers are the most common type of Servers used. The most popular Web Server is the Apache HTTP Server (April 2010 Web Server Survey). It was built to handle HTTP (Hyper Text Transfer Protocol) protocol, though due to possibility of customization, it is not limited to that protocol. Web Servers are one of the two cornerstone technologies of the Internet, together with networking technologies they are the foundation for the modern web. Web Server stores and provides access to data. It processes requests from Thin Clients such as a web browser to send a response containing requested documents. Such servers share documents across intranets, or across the Internet. (Yadav & Singh, 2009, 3, 167). Most modern Web Servers can both serve static files and create responses based on application computations. For example, Apache HTTP Server can be used as a forefront to applications that generate customized responses to Clients' requests. (Laurie, 2002, 1-4). This can be achieved, by appending modules, through which server software is connected to programs. Installing a PHP module is a good example of such approach. (Laurie, 2002, 330-331, 336).

HTTP is the standard protocol for transmitting HTML files or other data. Client communicates with Web Server using HTTP requests and responses. There are several types of HTTP request methods. GET and POST request methods are the most common. GET method means that the browser sends a formatted character string to the server, and the server returns the content identified by that string, which is known as a Uniform Resource Identifier (URI). Additional information that is being passed to the server is appended to the URL.

User is able to see and modify the information in the browser address bar and because of that he can send an arbitrary string or record strings sent to the Server. If the request is sent using POST method, the request can include a message body, and the server should pass this message body to the resource in the URI for processing. POST requests are typically generated by users submitting a form through their web browser. Forms can be used with either GET or POST requests, although they are usually used with POST requests, as information passed by the POST method can't be easily observed. When sending HTTP response, the Server locates the resource identified by the URI and returns that resource as part of an HTTP message to the web browser. If web page is requested, the browser displays the web page. If the resource requested is a server-side application, the server needs to interpret the URI as a request for a server-side program, format the request parameters in a form the program recognizes, and pass the request to that program. (Mukhar el al, 2004, 228-235).

Database Server provides Clients with access to data. As Mukhar el al (2004, 5) states, significant applications take advantage of a Database Server and access persistent data by sending SQL (Structured Query Language) commands to a Database Server to save and retrieve data. If running as a standalone server, it shares the data residing in a database across a network. Compared to File Server, which also operates at the Data Access Layer, Database Server has more efficient data management protocol. (Yadav & Singh, 2009, 3). For more details about Database Server please refer to section 2.3.

Compared to the Web Servers, which main function is to handle HTTP requests and return static HTML pages and sometimes other files, Application Servers function as a base or resource providers for the programs that are running on top of them. Application Servers provide their programs with things like communication facilities to talk to other computers, management of database connections, the ability to serve web pages, and management of transactions. (Mukhar el al, 2006, 1). Application servers were built around the need of corporations, research institutions, and other large organizations to process big amounts of data. They are not bound to HTTP protocol and many of them utilize for example COBRA (Common Object Request Broker Architecture) to communicate with each other and their clients in terms of objects, but not requests. Application Servers are much more computation oriented, than Web Servers, and in this respect, can be compared with mainframes. (Yadav & Singh, 2009, 1-5).

Compared to Application Servers, mainframes have different architecture, and more resemble a LAN or a cloud of servers, than a single PC based server. The other key

difference is the extent of the separation of data processing task. In mainframe systems the entire processing takes place on the mainframe and usually primitive terminals are used to display the data screens. These terminals have no autonomy. Client/Server environment provides a clear separation of server and client processes, both processes being autonomous. Yadav & Singh (2009, 5).

Glassfish server that is used in the project is an application server developed by community of developers, with strong support from Sun Microsystems. As any Java EE (Enterprise Edition) compliant server, Glassfish is essentially a set of containers. In an Application Server, web and business components that represent different types of classes and file types used by application, exist inside containers and interface with the Java EE infrastructure through well-defined interfaces. (Mukhar el al, 2006, 9-13). Glassfish consists of containers defined in Java EE and includes containers for both web and business components and a built-in Web Server.

Glassfish web container has its origins in Tomcat, a Java EE partial implementation Server, developed by Apache foundation. A noteworthy change in GlassFish is Jasper, the JSP (Java Server Pages) compiler, which compile much faster (informally 10x faster) than the competitors. (Pelegri-Llopart, 2007, 10). JSPs will be discussed in section 3.5. One of the most important features of Glassfish is the Grizzly framework that was designed to manage long lasting HTTP connections that are essential to support Ajax technology, especially when server Push effect is needed. (Pelegri-Llopart, 2007, 13). Ajax technology will be described in detail in section 3.7.

Glassfish was released in May 2006, and played an important role in adoption of Java EE 5, because it was one of the first its compliant implementations. It was also a big step for Sun Microsystems, as it was their first open source Server. Java EE 5 and GlassFish have had a very symbiotic relationship: the strength of the Java EE 5 specification has increased the value of GlassFish and the availability of GlassFish has validated Java EE 5. (Pelegri-Llopart, 2007, 1-11).

2.3 Introduction to databases

As was described in section 2.1, when the software system is divided into tiers Data Access tier is almost always separated from the Business logic and Presentation tiers. This allows providing clients with centralized data access. With the exception of Files servers, Data access tier is almost always is implemented as a Database server. Thomas (2002, 3) states that most companies use databases to store information and model business processes.

Databases allow companies to store, organize and retrieve large amounts of data. O'Donahue (2002, 6) defines a database as a "structured collection of meaningful information stored over a period of time in machine-readable form for subsequent retrieval." Modern databases can store trillions of entities and comprise up to several petabytes of data. (Top 10 Largest Databases in the World). There are several database types such as *Hierarchical* databases, *Network* databases, *Object* databases, *Object-relational* databases and *Relational* databases. Hierarchical databases store data in records, only parent child relationships can exist between the records. Hierarchical databases provide fast retrieval of data, but slow write operations. Network databases are similar to Hierarchical databases, but allow modeling more complex relationships between entities, such as many-to-many relationship. Object databases are used to store native programming objects and custom data types. Object-relational databases are a type of Relational databases that store objects as custom data types. Relational databases are a type of database that stores all data in tables, and allows to specify the relationships between entities to model real life processes. (Thomas, 2002, 4-5).

Relational database is the most popular type of database. As O'Donahue (2002, 6) states, in practical terms a database must form part of a system that provides for the management of the data it contains. Databases exist in the context of Database Management Systems. Relational Database Management Systems are the most common type of DBMS used by Enterprises. (Thomas, 2002, 12). In RDBMSs there are two basic storage structures: tables and indexes. Data is stored in a form of tables. As shown on figure 2.4, tables consist of rows, which are also called records and columns, also called attributes. In a purely relational model, rows are unique, so this guarantees consistent access to the same records and avoids data corruption. (Thomas, 2000, 14). Indexes are not required, but are utilized to improve the RDBMS performance. Indexes are used to rank or sort records based on a column in the table. The indexed column is known as key. In the Figure 2.4. the N column represents table key. (Thomas, 2000, 14-15).

N	Name	Surname
1	Jack	Smith
2	Bob	Blackstone
3	John	Conner

Figure 2.4. Relational database table

The most important characteristics of RDBMSs are *data integrity* and the availability of *common access language*. Relational databases incorporate data integrity rules to protect data against corruption. This is accomplished by providing entity and referential integrity. Duplicate rows can lead to errors in data retrieval. Entity integrity ensures that the data in the tables remain unique. To ensure entry uniqueness the primary keys are used. Primary key is a type of index that is unique for every attribute in the table. Also there is a possibility to define a composite primary key that is a unique combination of several attributes. In general, when combined with a table name, primary key acts as a pointer to a particular record. In Figure 2.4. the entries in N column represent record's primary keys. Most of RDBMS contain tables that are related to one another. This makes relational integrity an important issue for correct database operations. As shown in Figure 2.5, Table relationships are created with a help of foreign key. (Thomas, 2002, 15-17).

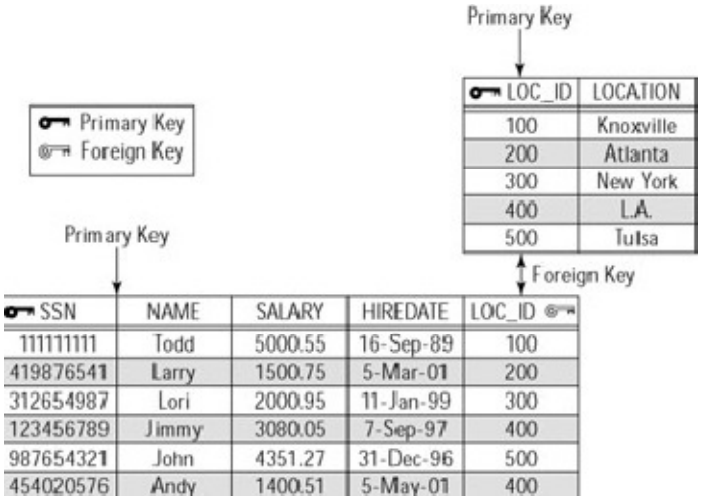


Figure 2.5. Tables relationship in RDBMS (Thomas, 2002, 17)

Foreign keys are primary keys from one table, used in the other table. In this way records in different tables can be connected. Foreign keys minimize database size by limiting data duplication. The establishment of relations between tables allows to model real life systems and processes. Table relationships are the main defining feature of Relational Databases. There are three types of table relations: one-to-one relationship, one-to-many relationship and many-to-many relationship. The types of table relationships are shown in Figure 2.6. The simplest is one-to-one relationship – when record in one table has only one matching record in another table. It is seldom used in modern databases, but for example, this method can be utilized to split extremely large tables into several smaller tables. The most

common practice in databases is one-to-many relationship. It occurs when record in one table matches several records in another table. Many-to-many relationship is not directly implemented in RDBMSs, but can be done indirectly through several one-to- many relationships. (Thomas, 2000, 18).

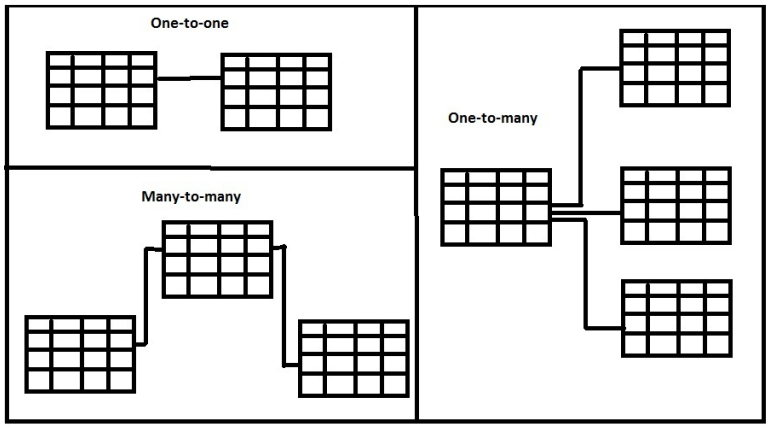


Figure 2.6. Types of table relationships.

The availability of common access language is an important characteristic of RDBMS. All modern relational databases use Structured Query Language (SQL), first developed by IBM in 1970s. It consists of several sublanguages. Data Definition Language (DDL) is used to create, alter, and drop tables and indexes. Data Manipulation Language (DML) is used to insert, update, and delete data. Data Query Language (DQL) is used to query the database using the SELECT command. Data Control Language (DCL) is used to grant and revoke user privileges and to change passwords. SQL also includes Transaction Control Commands, which are used to start, commit, or rollback transactions. Statements from all of these sublanguages can be used together. (O'Donahue, 2002, 25) SQL is a non-procedural, fourth generation language, which means that it is not possible to create a sequence of statements and it is much simpler then languages of previous generations. Databases from different vendors can have minor differences in SQL syntax.

The most important SQL sublanguages in terms of data modification are DML (Data Manipulation Language) and DDL (Data Definition Language). DML is a name for subset of SQL commands used to manipulate records in relational database. To protect against mistakes DML implements a system of transactions, this means that no statements, and changes that they introduce, will be applied before COMMIT statement is executed. If the instructions were wrong, they can be cancelled by executing ROLLBACK statement, unless COMMIT statement has been already issued. (Thomas, 2000, 21). Data Definition Language (DDL) is used to build, manipulate, and destroy database structures. It allows creating

database infrastructure such as tables and indexes to store data. With help of DDL it is possible to build constraints such as primary and foreign keys. Unlike DML, DDL does not participate in transactions, so it would not be possible to rollback such statement. Usually DDL statements require special permissions to be executed. (Thomas, 2000, 25).

In order to comply with MHG ERP architecture requirements MySQL Relational Database Management System in conjunction with JDBC (Java Database Connectivity) will be used as a Data Access tier technology.

MySQL is a multi-threaded SQL Database Server with a Client/Server implementation. Its speed, robustness, and ease of use make it a popular database for sites with dynamic content. PHP3, C, C++, Java, and Perl access MySQL directly through language specific APIs. (Compaq Active Answers, 2000, 6). It is an open source RDBMS distributed under GPL (General Public License). MySQL is highly portable – runs on many flavors of UNIX and Windows operating systems. (Matthews, 2003, 5-6). MySQL Database Server, named mysqld - is the program that actually manipulates databases. Client programs don't do that directly; rather, they communicate to the Server using queries written in SQL. The client programs can be installed locally or on the other machines that are connected to MySQL Server, because MySQL is an inherently networked database system. The interaction between a Client and Database Server usually consists of establishing a connection, receiving SQL queries from the client, performing database operations by RDBMS, and sending the results to the client. (DuBois, 2002, 28).

3 JAVA ENTERPRISE EDITION TECHNOLOGIES

Java EE – Java platform Enterprise Edition – is one of three major Java platforms. Java platform is “the programming language, its APIs, and a virtual machine specification that, taken together, define an entire programming and runtime environment” (Oaks & Wong, 2004, 15). Sun Microsystems, the creator and main developer of Java technology has recently changed the conventions in naming Java platforms, and this can be a reason for confusion. Java platform Enterprise Edition was known as J2EE. Same is true for Java SE (J2SE) and Java ME (J2ME). Because the changes of the platforms are not as immense as when the Platform 2 was introduced, documentation and information for platforms named in old convention is valid for the corresponding platforms named in new convention. (Java EE at Glance).

The primary programming language for Java platforms is Java programming language. The three platforms, Java SE, Java EE and Java ME differ in their available APIs and purposes. Java Standard Edition is oriented on workstations and personal computers. One of Java SE most important features is the ability to support programs running inside a web browser – Applets. Java Micro Edition has a smaller libraries and a lot of special APIs, such as MIDlet API and others, also utilizes different configurations and profiles. It is targeted at mobile and embedded devices – devices with limited memory and processing power. (Topley, 2002, 9-11)

At the base of Java EE lies the Java Standard Edition, which provides the core on which Java EE APIs are based, as well as the JVM (Java virtual machine) that executes compiled code. The major building block of Java EE application is component, which, according to McGovern (2003, 429) "is a set of interfaces and classes that have been assembled into a single package to provide a specific functionality". There are four types of components: *Application clients*, *Applets*, *Web* components and *Server* (*Business*) components. All those components run within specific containers. Application clients and Applets are, in general, Java SE components that run within JVM, though they can access some standard Java EE services. (McGovern, 2003, 3-7). Mukhar el al (2006, 11-13) defines Business (Server) components and Web components as server-side components that rely on the Java EE framework. Java EE provides support for the server-side components in the form of server-side containers. Containers handle all of the details involved with starting up services on the server side, activating the application logic, and cleaning up the component. Java EE compliant Application Servers actually provide components with the implementation of server-side containers. (McGovern, 2003, 43-45).

Java EE was created to address the need of enterprises to create a standard method of development and deployment of Enterprise Applications. Though J2SE had many enterprise level APIs, there were still too many things undefined, such as persistence, transaction management, security, and so on. As Mukhar el al (2006, 3) points out, there was a need to write distributed applications and applications consisting of several layers that are scalable, robust, secure, and maintainable. In 1998 Sun Microsystems released J2EE – a first version of platform that defines a multi-tier architecture for enterprise information systems. (McGovern, 2003, 5).

Java EE was build with scalability in mind. McGovern (2003, 46) defines a scalable application as a scalable piece of software that "is thought to be able to cope equally well with one client or a much larger number of concurrent Web clients, without requiring

complete redesign of the software." Java EE compliant implementations provide all of the needed infrastructure so the application can support the expansion of the company. Compared to other Java platforms, Java EE is ideally suited for developing complex Server applications. It has such components as Servlets, Java Server Pagers, Java Server Faces, Enterprise Java Beans and others in its arsenal. These technologies will be described in connection with tiers that they were designed to represent in the order from Data Access tier to Presentation tier.

3.1 Java Database Connectivity

JDBC (Java Data Base Connectivity) is a SQL-level API that allows embedding SQL statements as arguments to methods in JDBC interfaces. To do this in a database-independent fashion, JDBC requires database vendors to develop a runtime implementation of its interfaces. (Reese, 2001, 13). These runtime implementations are also called Database Drivers. Database Drivers define methods and properties for sending, retrieving, and obtaining status information about the database as well as extracting data. (Thomas, 2002, 45). By switching between different database drivers it is possible to interact with databases from different vendors such as Derby DB, Oracle or MySQL without virtually any changes in code.

As Figure 3.1 shows, there are four types of JDBC drivers. The Type 1 driver is a bridge driver, for example JDBC-ODBC bridge that allows Java programs to interact with a database using widely available ODBC drivers. The main drawback of this type of driver is that it is slow and may require additional configuration. Type 2 driver consists of Java classes that work in conjunction with the non-Java native drivers, provided by the database vendors. They convert JDBC requests to native DBMS calls. Such type of diver works considerably faster than Type 1 driver but requires specific installation and configuration of the machine that Java programs are executed on. (McGovern, 2003, 582-583). Reese (2001, 27) describes Type 3 driver, so called JDBC-Net driver as a driver written entirely in Java that provides a client with generic networking API that is translated to specific database access API on the Server level. This kind of driver is very flexible, allowing connection to several databases at once. Unlike other types, these drivers are provided by server vendors, and not by DBMS vendors. The downside of the Type 3 diver is the need to purchase vendor implementations of such diver. The rest of the driver types implementations are usually available free of charge. Type 4 driver is a pure Java driver, that communicates to database through a network protocol built into a database engine. This is the most direct Java solution. (Reese, 2001, 27). Mukhar el al (2004, 327) states that Type 4 driver is the most common database driver type.

Mukhar also points out that there is a divide in opinions regarding the comparison of driver speeds. The fastest driver types are Type 2 and Type 4. When those two types are generally compared, either Type 2 can be considered faster as it is using native code to execute queries. Type 4 driver can be considered faster because it does not need to use another layer, particularly Java Native Interface, a Java API to perform native method calls, to execute queries. (Mukhar el al, 2004, 327).

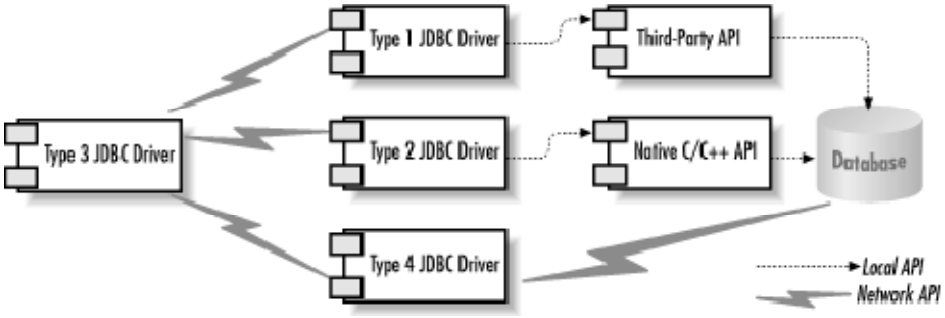


Figure 3.1. JDBC Driver types (Reese, 2001, 27)

Java has an extensive API for adding and managing database drivers. (O'Donahue, 2002, 116-117). When there are different databases present on a server, the desired one can be accessed by passing its URI when establishing a connection. Different RDBMS require different sort of parameters such as login name and password in different forms to access database, so some difference between accessing different databases does exist.

3.2 Java Beans

Sometimes it is being said that the defining technology of Java EE is the EJB – Enterprise Java Beans. (Mukhar el al, 2004, 2). This is a technology of Business logic tier – the tier that holds most of Application Server's functionality. EJBs were not used in the undertaken project. The Business rules tier was implemented in the form of Servlets interacting with and manipulating Java Beans.

Java Bean is a Java class of special form that has a defined structure and defined ways to declare its methods and variables. Following special design patterns, every Java Bean is required to have a default constructor, private class variables, that are called properties, and accessor and mutator methods for each variable. Through the use of special APIs, properties of Java Bean can be accessed by their name. Java Bean is an approach to create an Object Oriented language class, which features are so strictly defined and standardized, that "it can

be manipulated visually in a builder tool". (Englander, 1997, 10-13). Java bean is a way to build data centric Java classes.

Like Java Beans, EJBs have a common architecture defined by EJB Specification, and in fact both this entities have a similar composition, with properties and accessor and mutator methods, but EJBs require extensive infrastructure to support their functionality. Unlike Java Beans, EJBs are not Java objects, but Java EE components. As any Java EE component, EJBs require a specific container to be executed. EJB container is the foundation of every Java EE compliant Application Server. Container provides all the necessary infrastructure for EJBs to execute. There are three types of Enterprise Java Beans: Entity Beans, Message-Driven Beans and Session beans. (McGovern, 2003, 429-436). Session Beans that will be used in the project are not EJBs. They are simple Java Beans that are part of IceFaces framework.

3.3 Servlets

Servlets are the foundation for most web application technologies in Java EE, and it is special Servlets that are running underneath JSPs and JSFs. In the undertaken project Servlets are used as foundation for all business logic and interaction with databases. In modern Java EE books Servlets are not described as a primary Presentation tier technology and are often briefly mentioned, due to the relative complexity in their utilization as a tool to produce dynamic HTML pages. This is the case, when we are talking about thin client type applications. Judging on my previous experience with Java ME, Servlets are best server-side technology to deal with HTTP protocol requests from fat clients. My Java ME application was a fat client and Servlets were receiving and interpreting HTTP requests that I was sending to the Server. McGovern (2003, 51) describes Servlet API as "one of the oldest and best-established APIs of the J2EE platform. It was on the scene years before J2EE was available, and thanks largely to its success Java at the server-side became not only an option but a serious contender for the best technology for development of the Web applications."

The Java Servlet API provides a standard way to extend the functionality of a Server that uses a protocol based on requests and responses. Originally, Servlets were developed as a Java–based replacement for CGI (Common Gateway Interface) scripts that were the dominant technology for extending Web Server functionality when Java Servlets appeared. (Farley & Crawford, 2004, 115). In the early days of adoption, Servlets were used as the main middle-tier Java technology. A typical Servlet used to be a mixture of business logic with Presentation tier code embedded into it. (Mukhar el al, 2004, 229).

It is a common practice to use web browsers (thin clients) as a GUI for interacting with Enterprise Applications. This is very convenient, because most of modern PCs have browsers preinstalled, and there is no need to install specific clients for each web application a person might use. It is also more convenient to generate HTML pages, than to send a Java Applet to the client, because Applets have limitations in terms of security permissions, and HTML pages are much faster to download, than an Applet (which is, though it runs inside a web browser, still a fat client). (McGovern, 2003, 77-78).

To generate dynamic HTML pages, a Web server needs to run a Servlet engine also called Servlet container. Application Servers are usually bundled with a Servlet container. If a user requests for a page that has to include some dynamic data, for example a data from a database, this request is forwarded to Java Servlet that will accommodate the request, build an output HTML page dynamically, and pass that page over to the user with the help of the Web Server. When the user's web browser displays the received page, it does not know if that page was a static HTML page or generated by a Servlet. (McGovern, 2003, 78). Although all the processing can occur within the Servlet class it-self, often, in order to separate logic into several layers, helper classes or other web components such as EJBs are utilized, to perform the business logic processing, leaving the Servlet free to perform the request and response processing.(Mukhar el al, 2004,229).

Servlets are running inside Servlet containers, that provide them with all the needed infrastructure and control Servlets life cycle. As Figure 3.2 shows, Servlet lifecycle consists of subsequent calls to its three main methods: *Init*, *Service*, and *Destroy*. Init method is called when the Servlet is loaded by container either on the Server startup, or when the Servlet was requested for the first time – this depends on the configuration. Service method is called to handle user requests; it defines the type of request and redirects it to the appropriate method to handle the request. Destroy method is called by Servlet container to notify Servlet that it is about to be removed. (McGovern, 2003, 81).

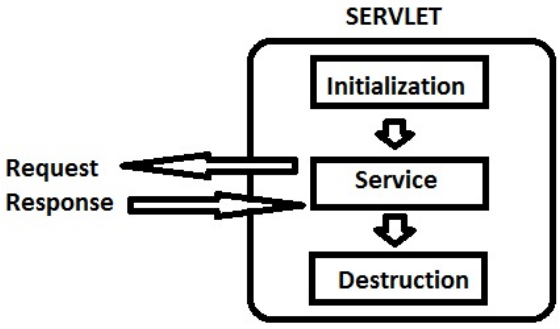


Figure 3.2. Servlet Life Cycle

Servlets allow Enterprise applications to have a web based front end. But, because of the complexity and size of Java EE and high functionality of Servlets and JSPs some developers never need to use the server side technologies like EJBs. So, there were developed Servlet Containers that can run on their own, becoming Java EE partial implementations. There were developed methods to deploy Web applications on the Java EE partial implementation Application Servers. Apache Tomcat is, probably, the most famous Java EE partial implementation. There was developed a WAR (Web Archive) format for easy deployment of web applications. This format is a part of Servlet API, and is a JAR (Java Archive) file, that contains all the files needed for the web application to execute. (McGovern, 2003, 51).

There can be several WAR files, thus several web applications, deployed on the application server and there is a need to provide some degree of separation of resources between those applications. Each web application is interfacing with a Servlet Container through a Servlet Context, which is a unique execution domain for each application. Servlet communicates with Servlet Container, by calling methods on Servlet Context. (McGovern, 2003, 84-85).

In a situation, when a Servlet can't handle user's request, but has some information on where it can be handled, two techniques can be used. If the capable Servlet or JSP is inside the same Servlet Context, the request can be forwarded to it. This type of Request Dispatching is performed on the server-side using original request and response objects, and the end user will not be informed that his request was serviced at a different location. If the destination of the redirection is not in the same Servlet Context, the redirect message with a new URL is sent back to the Client, which connects to this URL, thereby creating a new pair of request/response objects. The later method is considerably slower. (McGovern, 2003, 85-88). In my project only forwarding method is used.

For many Web Applications it is important to recognize the Client though several requests. One of the main shortcomings of HTTP protocol is its stateless nature – inability to remember the user over several requests performed during Client's Session. Session is a coherent conversation consisting of individual Client-Server interactions that allows overcoming HTTP stateless nature and recognizing user over several requests. (Mukhar el al, 2004, 74). McGovern (2003, 88) defines Session as a logical task that a user tries to accomplish on a web site. Session tracking allows Server to remember user's input and to carry it from page to page. Servlet API provides several ways to track users. This can be done

either by storing session information either on the Client or on the Server side. The difference between these two methods is that when session is stored on the Server side, much less session information is sent to the Client. (Mukhar el al, 2004, 75). Storing Session on the Client side is accomplished by either using Cookies, by URL rewriting, or by adding hidden fields to HTML pages. When Client requests service from a Server, all session information that Client possesses is sent with the request. (McGovern, 2003, 88-89). When Session is stored on the Server side, a special session object is created. The session object that is associated with a client is assigned a specialized session ID that is being sent to the Client in a form of Cookie or URL rewriting. When the Server receives session ID, the Client can be recognized. (McGovern, 2003, 89-90).

Cookies are small files, holding session information that Web Server sends to Client to store. Every time user is visiting a web site, browser sends all corresponding Cookies to the Server, as a part of request. The advantages of Cookies are the ability to survive Server crashes, and lighten the load on Server's memory. The disadvantages of Cookies include the possibility to be observed by the user, and to be disabled by web browser. URL rewriting allows the session ID to be attached to the end of the URL string to help Servlet container to identify the Client for subsequent requests. The disadvantage of this method is that it is also highly visible. Appending hidden fields to HTML pages is a good alternative, though it can also be observed by looking at HTML page code. (McGovern, 2003, 89-91).

Servlet Session tracking API uses one or a combination of previously mentioned technologies, though, only session ID is either sent as a Cookie or appended to URL, so all the session information is stored on the Server in a form of HTTP Session object.(McGovern, 2003, 91-93).

Because of Servlet API's high functionality and abilities, Web Applications that are using Servlets for both Business and Presentation tiers came out complicated and hard to maintain. The reason for that is the lack of logic separation in layers or tiers. Another problem is that Presentation level developer has to have Java language programming skills to developed Servlet generated HTML pages. Those problems and lack of possibility to use GUI editor to develop HTML pages led to further development of Servlet platform. GUIs can't be used because HTML code is embedded in form of Strings and it would be impossible to create a reliable software that will be able to recognize and render the page, until the Servlet is executed. To overcome those shortcomings Java Server Pages technology was developed. It did not replace but complemented Servlet API.

3.4 Java Server Pages

Java Server Pages is a Java EE framework used to generate dynamic web pages. JSP page is an HTML page with some Java code embedded into it. Compared to client side scripting technologies, like embedding JavaScript to a HTML page, JSPs are executed on the Server side, so Client receives pure HTML page modified by Java Scriptlets included in the JSP. These Java Scriptlets are fully functional Java expressions and can, for example access data from the entire application and can utilize server-side resources such as databases, EJBs and everything that Java EE framework provides. This extremely powerful solution allows to differentiate the roles in creating JSPs: web designer can design HTML page with GUI tools and embed there any content he would usually put in HTML page, like plug-ins, or for example JavaScript, while the Java Scriptlets that will modify this page according to the request data will be developed by a professional Java programmer. (Mukhar, 2004, 113-115).

This approach is similar to what other server-side scripting technologies like PHP, ColdFusion and ASP are about, though JSP is the only solution, that it is using a real object oriented programming language in Scriptlets. Each of those systems has its own good and bad sides, and comparing them is the same as comparing different programming languages, in the end they are just tools and everything depends on the purpose of application and the skills of programmer. PHP, for example, doesn't have such extensive libraries and is not well suited for complex multi-tier applications as JSP and ASP is a proprietary platform and was built for Windows OS. On the other hand, JSPs, as well as ColdFusion for example, have a reputation of being slower than PHP and ASP. (Converse el al , 2004, 3-4, 26-33). Each of these platforms is very different, and was developed for different types of applications. JSPs are ideal as a front end of a large scale Enterprise application that is using several different operating systems to run Servers on – it will allow interaction with the rest of the application and programmer will not have to learn a new scripting language to bring this functionality to HTML page.

In terms of functionality JSPs are similar to Servlets, though Servlets can be described as an opposite to JSP: Java code with some HTML markup embedded. JSPs are built on Servlet API technologies. JSP pages are being executed inside a special container - JSP container. A JSP container is a piece of software that loads and manages JSP pages. The JSP container translates the HTML and Java code into a Java code source file. This file is then compiled into a Java class that is executed by the Server. The class file created as a result of

JSP compilation is known as the JSP page implementation class which is a Servlet. (Mukhar el al, 2004, 44-45) JSPs and Servlets have the same functionality, because they are based on the same API. The main difference between those two frameworks is the development methods. Another feature is that JSP can be requested directly from clients' web browser, by their URLs. In general, the first request of the JSP will take longer than all single subsequent ones because of this generation process, but after initialization Servlet responds immediately. (McGovern, 2003, 114).

JSP framework has a number of elements that can be embedded in HTML code to provide described functionality. Starting with version 1.2, JSP specification defines special XML tags that enclose the JSP container instructions in the JSP page. Before that, JSP elements were not XML compliant. JSP elements can be of three types: Directive elements, Scripting elements and Action elements. (Mukhar el al, 2004, 46).

Directive elements provide JSP container with information about the page and can be used to include parts of the page (JSP can be assembled from several JSP pages at the runtime) as well as to declare Java packages and tag libraries used. *Scripting elements* are elements that hold the Java code. *Action elements*, as the name suggests, are used to make the page to perform some actions, like, for example, to use specified Java Bean, to include some other page or to add an attribute. There are two types of actions: standard actions, like the ones mentioned above and custom actions that can be defined by developers. (Mukhar el al, 2004, 46-56).

JSPs often contain significant amounts of Java code, which is not an optimal situation for web designer, who has no skills in Java programming language. There was a strong need to, if not to remove, but to hide Java code from web developers. Before the JSP 2.0 Specification, the only way to accomplish this goal was to use custom action tags. This was done by creating a tag library, with a tag library descriptor and appending its reference to JSP page. Tag library descriptor is a XML based file that contains information about classes in the Tag library that actually implement functionality of specified tags. These tags are similar to HTML tags and are much easier to learn and use for a typical web developer than Java programming language. The other useful feature of this approach is the possibility to reuse the once defined code. Over time different JSP standard tag libraries (JSTL), like Struts and special libraries for core and international processing appeared, and the development of custom tag libraries became one of the most important trends in development of JSPs that, in its turn inspired creation of even more sophisticated Java EE presentation layer technologies

such as JSF(Java Server Faces) and IceFaces. (Mukhar el al, 2004, 107, 127-134). JSF will be described in section 3.5 and IceFaces will be described in section 3.6.

With release of JSP 2.0 Specification, apart from enabling new ways, to make the development of Tag libraries easier, another way to hide Java code from designers was introduced to JSPs – the JSP Expression Language. JSP EL is a scripting language that provided some of the basic functionality similar to Scriptlets, especially in terms of fetching a value from a Java Bean or performing some mathematical calculations. It is thought to be easier to use for non programmers. (Mukhar el al, 2004, 113). JSP EL had a great influence on Java EE technologies of the Presentation tier and on Java Report generating technologies. Java report generating technologies will be covered in section 3.7

JSP frameworks' purpose was to make Servlet programming a lot more HTML-like, to hide Java code, to create a framework suitable for a typical web developer. But it wasn't providing standard ways (though, there are community developed frameworks like Struts, that deal with this issue) to solve one of the main Servlet problems – the lack of clear distinction between Middle and Presentation tiers. As well, as if it is a Java developer, who was building JSP page, he has to deal with markup language which is so different from Java programming language. Java programmers' experience in building sophisticated GUIs based on Java Foundation Classes was totally irrelevant.

3.5 Java Server Faces as an example of Model View Controller architecture.

Java Server Faces is a Java EE technology that simplifies the development of Web Applications. Unlike previously discussed technologies, JSF is a supporting technology, so it is intended to be used in conjunction with other Java EE technologies, and can't be used on its own. The primary design pattern of JSFs is Model View Controller (MVC) paradigm. As Figure 3.3. shows, MVC separates an application architecture into three categories of components: *Model*, *View*, and *Controller*. (Mukhar el al, 2004, 166-168). Model is a component that represents application data and processing code. The View is responsible for the presentation of the Model. The Controller is the component that provides reaction to the user's input. (McGovern, 2003, 116). MVC paradigm is very widespread in programming that involves creation of GUIs. McGovern (2003, 9) states that Java EE multi-tier architecture has some correspondence with MVC paradigm. Another example of MVC utilization is Swing framework. Swing is a Java SE API for Graphical User Interface creation that follows MVC patterns. MVC allows high level of independence between different components,

which helps to improve maintainability and allow easy modification of one of the components without virtually any need to modify the rest. McGovern (2003, 116). There were several attempts to bring MVC paradigm to Web Applications. Wadia el al (2008, 4) states that MVC became de facto standard for developing Enterprise Web applications when Struts became popular in 2001. Struts is a framework that utilizes MVC paradigm. It is developed by Apache foundation. When MVC is implemented using Java EE APIs, Servlet acts as a controller for incoming requests from the client, routing requests to a JavaBean or EJB that represents the Model portion of application. JavaBean fetches data from database and then hands it over to the related JSP that acts as a View component. JSP renders the appropriate markup, updates the Model, and issues a response to the client. (Wadia el al, 2008, 4).

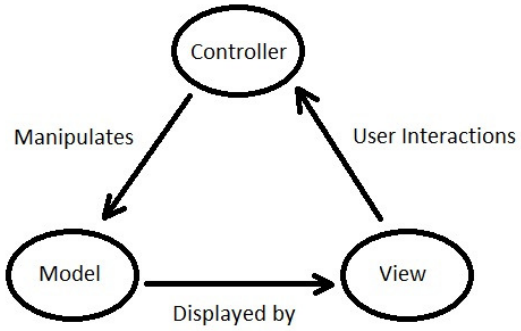


Figure 3.3. Model View Controller pattern

JSF is a standard, released by Sun Microsystems in 2004 to provide a powerful component-based, event driven platform for Web application development in Java. (Wadia el al, 2008, 4). JSF provides more simple, compared to JSP, ways to build UIs (User interfaces). Development of UIs in JSF is similar to GUI development using Swing. JSF framework provides a set of components in a form of reusable objects. (Mukhar el al, 2004, 167-168). JSF components can be embedded inside JSP pages and can render the appropriate markup for HTML or any other output for which a render kit is provided. Render kit is a software module that transforms JSF markup into code that is suitable for the Client. (Wadia el al, 2004, 5).

JSF generates response to Client's request in six phases that comprise a JSF lifecycle: *Restore view, Apply request values, Process validations, Update model values, Invoke application* and *Render response*, as shown in Figure 3.4. During *Restore view* phase JSF implementation builds a tree structure consisting of components that is used to define and

preserve the state of each component. If the user is known, user specific values are restored. During *Apply request* values phase, values that were received via request will be assigned to corresponding components situated in the component tree. *Process validation* phase allows validation of the data that was received via request. If the data is valid, the Model components are updated accordingly, during *Update model values* phase.

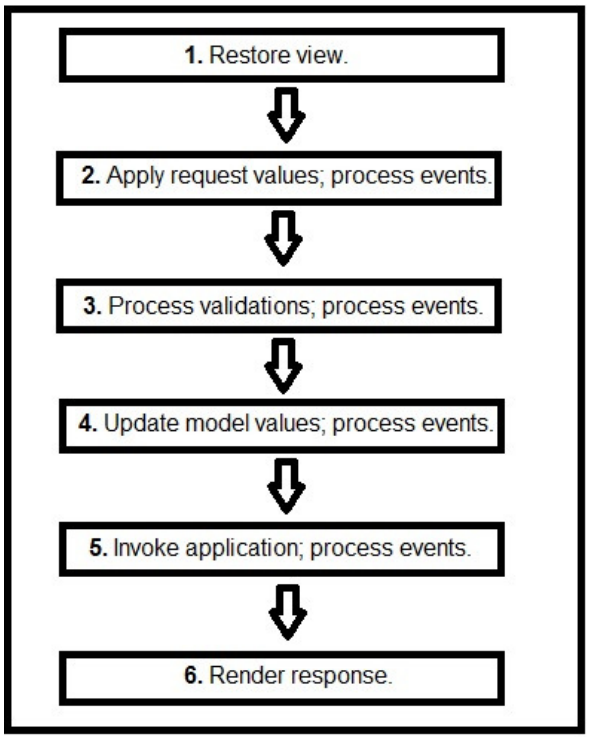


Figure 3.4. Java Server Faces Lifecycle.

During *Invoke application* phase all the events that were generated during previous phases are handled. The final UI response is rendered during *Render Response* phase. JSF lifecycle is the main difference of this framework from other Java MVC Web application frameworks. (Mukhar el al, 2004, 170).

JSF framework provides two of three MVC components: JSF pages are used as a View component, and FacesServlet, a specialized Servlet which defines and handles the flow of the application, acts as a Controller component. (Wadia el al, 2004, 6). JavaBeans are used as a Model component of MVC. JavaBeans that are used by a JSF-enabled application are referred to as *Managed beans*. Managed beans are created and initialized using information stored in configuration files of the application. The properties of managed beans are referenced in JSF enabled JSP pages using either value binding expressions or method binding expressions. Value or method binding expressions are special JSF expressions that allow creation of bindings between JavaBeans and JSF values. (Mukhar el al, 2004,184-185).

Binding means that corresponding parameters in bounded entities will have the same values. Once one of the binding parameters is updated, the second one is updated automatically by JSF framework. Managed beans scope defines its lifetime and accessibility. Managed beans can be configured with one of four different scopes: *None*, *Request*, *Session*, and *Application*. If a bean has a None scope, it can't be accessed from JSF enabled JSP page directly. Beans that have a Request scope are valid only during single request. The Session scope allows Beans to be used through multiple requests by the same user. Application scope Java Beans can be accessed throughout application. (Mukhar el al, 2004, 188).

JSF framework handles inter page navigation by providing navigation rules in a configuration file. The navigation rules can specify which web component initiates the request, which web component handles the response, and which value causes navigation to happen. This way, navigation can be triggered by the return value of the expression. (Mukhar el al, 2004, 203-205). The main restriction of this method of navigation is that it is valid only for JSF enabled JSP pages.

3.6 IceFaces as an Asynchronous JavaScript and XML framework

Asynchronous JavaScript and XML is a combination of dynamic HTML with a JavaScript-based backend communication achieved with XML messages sent between the browser and the server that allows to skip reloading of pages. Ajax allows mimicking the behavior of desktop applications in browser by means of HTML, Cascading Style Sheets (CSS) and JavaScript. All of these technologies can be added individually to usual JSF pages, but because of complexities in development of interactive pages, this approach was not widely adopted. Ajax has several limitations at presenting complex data. They come from the technologies that it is based on. For example, HTML doesn't offer any interactivity or immediate feedback for image manipulations, and requires HTML form to be sent to the server to communicate the changes in order to regenerate the page and display the changes to the client. (Eschen, 2009, 10-11).

IceFaces, developed by IceSoft Company, have renderers that shield developer from developing code manually. Modern JSF implementation consists of two layers: JSF *reference implementation* and vendor specific *add-ons*. The most well known JSF reference implementations (RIs) are Sun Microsystems RI and Apache MyFaces. Modern Java EE compliant Servers' vendors support both implementations. The IceFaces framework resides in the vendor-specific add-on layer. It is defined as an Ajax extension for Java Server Faces.

IceFaces support the majority of RIs and all of Java EE compliant application servers. (Eschen, 2009, 12). Utilizing technology called Ajax bridge IceFaces enabled JSF pages are fully loaded only once. The updates are limited to specific parts of the page, so complete reload is not needed. Such functionality is achieved by Direct-to-DOM (Document Object Model) technology. Document Object Model is an element of the HTML parsing process and is used by browsers to render the page. Direct-to-DOM technology manages server side Document Object Model that is a copy of the Client side DOM to manipulate web page on the server side. All the changes are done on the server side first, and then Ajax update mechanism updates the client side DOM. (Eschen, 2009, 18, 160). Partial submit is a JSF standard compliant implementation that manipulates the actual state of form during runtime. This allows to have a single event processing per field, so when some input have been added to a field, it sends a partial submit form that is processed on the server side and changes are communicated back to the Client. Field generated events allow implementing dependent components to ease the input for the user and to make web page more interactive. (Eschen, 2009, 160-162).

IceFaces framework supports change of the visualization skins during runtime. This allows to interactively adapt presentation to user's requirements. Skins can be defined by developers in a form of CSS classes. In order to achieve collaborative computing, and deliver server side changes to a group of clients simultaneously, IceFaces utilizes Ajax Push technology. Ajax Push technology is interfering with classic request-response pattern of web containers and replacing Client side rendering, with Server side rendering. Parallel updates of web pages on several Clients are possible due to the ability to multicast web page changes. Figure 3.5. shows Ajax Push technology architecture. Communication between a web browser and a web container is very restrictive. Normally, there are only two allowed communication channels in parallel per web browser. This limitation of channels is protocol related and restricted on the Client side through web browser implementation. Ajax Push technology overcomes this limitation with Ajax Push Server. Ajax Push Server implements global channeling on the server side which allows creation of unlimited amount of virtual channels. (Eschen, 2009, 240). Under a standard Java Servlet model, each connection requires its own thread of execution. Such architecture can have problems when scaled.

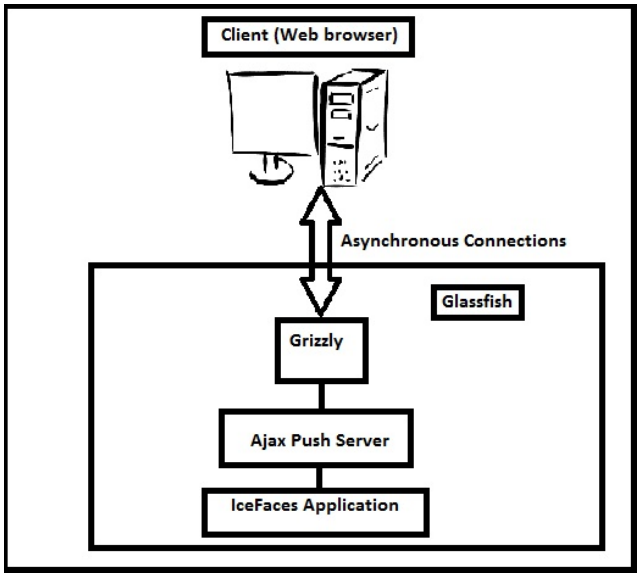


Figure 3.5 Ajax Push technology architecture

Sun Glassfish Grizzly plug-in allows overcoming thread-per-client dependency by using Asynchronous Request Processing mechanism. (Ajax Push). HTTP protocol does not support Server side initiated updates, Server can only respond to Clients request. Ajax Push technology is using long polling, a protocol inversion technique that "involves holding a connection open from the browser Client to the Server with a blocking request, and then fulfilling that request when some state change in the application triggers an update to the presentation." (Ajax Push) This technology allows sending server-side updates to the Client.

3.7 Java Excel generating frameworks.

As Walkenbach (2007, 3) states, Microsoft Excel is a part of Microsoft Office suite and is the most widely used spreadsheet application. A spreadsheet is the computer equivalent of a paper ledger sheet. It consists of a grid made from columns and rows. (Spreadsheet tutorial). The work performed in Excel is stored in the workbook file which is comprised of one or more worksheets, and each worksheet is made up of individual cells. A standard Excel 97-2003 workbook file format is XLS. Office Open XML based XLSX format that was developed by Microsoft is the default Excel file format since Excel 2007. (Walkenbach, 2007, 142-143). Excel is primarily used for number calculations, such as creating budgets, analyzing survey results and for any type of financial analysis needed. Another common application of Excel is lists creation and Excel, because of its row and column layout, allows storing lists efficiently. Excel has built in tools for creating charts,

graphs and diagrams. (Walkenbach 2007, 3). Visual Basic programming language can be used to automate time consuming procedures. (Walkenbach, 2007, 681).

JXLS is a small Java library for writing Excel files using XLS templates and reading data from Excel into Java objects using XML configuration. The latest available version is 0.9.9. JXLS allows creation of complex Excel reports from predefined XLS template that contains formatting, formulas, markup and specific notation to indicate placement of data. These templates are passed to JXLS engine that combines them with specified data and returns a ready report. JXLS reader is a JXLS module that is responsible for reading XLS documents. The main features of JXLS framework are the ability to embed SQL queries into the templates, support of conditional tags which have form similar to JSP tag libraries, and dynamic cell manipulations. Script similar to JSP EL can be used to retrieve data from Java objects. JXLS is based on Apache POI API. (JXLS). Apache POI is an open source framework for reading and writing Microsoft Office Documents. POI provides Java APIs for manipulating various file formats based upon both the Office Open XML standards and Microsoft's OLE(Object Linking and Embedding) 2 Compound Document format that older XLS format is based on. Right now, POI provides full support only to Microsoft Excel files. (Apache POI project).

JasperReports is an open-source Java class library designed to aid developers with the task of adding reporting capabilities to Java applications. It is primarily used to add reporting capabilities to Web applications, but has no dependencies on Java EE. A typical workflow while creating reports with JasperReports , as shown on the Figure 3.6, consists of creating JRXML template, compiling JRXML template into Jasper template, filling the template to generate the report and exporting the filled report. Reports are generated based on predefined templates. JasperReports templates are XML files that have JRXML extension. JRXML files can be created manually or with iReport. iReport is a UI tool for creation and compilation of JasperReports. It can come in a form of standalone application, or as an Integrated Development Environment plug-in. JRXML files are compiled into JasperReports native binary templates, known as Jasper files. Jasper files are then combined with the data and filled reports are created. JasperReports filled reports are known as JasperPrint files. JasperPrint files can be then further exported to PDF, RTF (Rich Text Format), Excel, HTML, XML, CSV(Comma Separated Values) and to plain text formats. (Heffelfinger, 2006, 221-228). JasperReports utilizes either Apache POI API or JExcelApi framework to export filled reports in Excel formats. (Heffelfinger, 2006, 21). JExcelApi is an open source java API enabling developers to read, write, and modify Excel spreadsheets dynamically.

Compared to POI, JExcelApi is a more modern and mature API. It can also handle images and supports copying of charts. (Java Excel API - A Java API to read, write, and modify Excel spreadsheets).

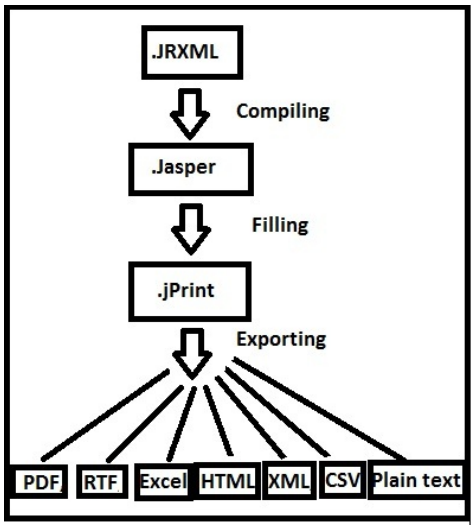


Figure 3.6. JasperReports workflow.

The main features of JasperReports are flexible report layout, the ability to present both graphical and textual data, support of multiple data source types, the ability to include subreports, and to export data to multiple formats. As figure 3.7. shows, Flexible report layout means the ability to separate report into several sections, such as report *title*, that appears only once at the top of the report, *page header*, that appear at the top of every page, *detail* section that contains primary report details, *page footer*, that appears at the bottom of every page and summary section to conclude the report. All of those sections can be optional, and some special sections like background section, for adding background images can be added. (Heffelfinger, 2006, 122-124). Images can be added to any section of the report. JasperReports allows Databases, Arrays or Collections of Java Objects, XML and other sources to be used as a source for dynamic data. Developers are provided with a mechanism to create custom data sources. When Databases need to be accessed, SQL queries can be added to the templates. (Heffelfinger, 2006, 79). JasperReports template can be included into another JRXML template as a subreport, so they can be developed or modified independently. JasperReports can generate dynamic charts by using JFreeChart framework. JFreeChart is an open-source library for creating professional-looking charts including, but not limited to 2-D and 3-D pie charts, 2-D and 3-D bar charts, and line charts. (Heffelfinger, 2006, 122-124).

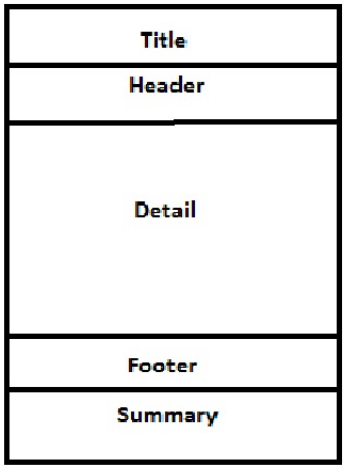


Figure 3.7. Flexible report layout elements.

4 PRACTICAL IMPLEMENTATION

MHG ERP Reporting module was developed over four years ago. The two main elements of the module are: JSF page for extra parameter input and a Servlet that is responsible for preparing the data and calling JXLS methods to generate the Excel reports. The reports are generated from the pre-designed XLS templates stored on the Server.

For the purpose of making the project and learning current implementation, I was given a zip folder containing all the MHG ERP Reporting module files. The folder contains four subfolders: *UI*, *Servlet, Templates* and *Language files*. UI folder contains two files: a JSF page responsible for user interface of the Reporting module and a Backing Bean that communicates user interface instructions to the ReportServlet. ReportServlet is the main class of the reporting module. Together with ReportHelper java class, they reside in the Servlet folder. The templates folder contains JXLS templates in a form of Excel files. There are thirteen templates, one for each reporting case. Because MHG Systems has projects going on in several countries of the world, one of the main features of the ERP is full internationalization. The Reporting module is supposed to produce reports in several languages. The Language files folder contains the language files in a form of Java property files. There are language files for ten languages, but, because the module was built with internationalization in mind any number of languages can be supported if specific property files provided. At the beginning of the thesis I was confused with the contents of the folders as I was expecting it to be an executable module out of the box. But, as my project manager

explained me, in MHG Systems they have a philosophy that every module should be built independently, so each module can be updated or modified without any need to modify the rest of the ERP. That's why I was given limited amount of files, just enough to understand how the module is interfacing with the system and what kind of API it provides. As have been agreed, during my thesis I will create my own demo database for development purposes, so the module will be data independent. The real database testing will be done as the last step during final testing process. This model of development is based on the black box principle. McGovern (2003, 821) describes black-box as a programming paradigm when the internals of the classes are not exposed to the rest of the application. The interaction with the application happens through well defined interfaces in form of methods that can be accessed from other classes. In terms of Java this means that these methods should have either public or private access specifiers.

I was provided with a rough plan of actions by my project manager. The first step was to study the original Reporting module. The second step was to study and research Jasper Reports and based on this knowledge to start the implementation. We have agreed that I will be sending e-mails to my project manager every two to three weeks, describing the progress and once in five to six weeks, I would hold a presentation in the company describing current progress and challenges.

4.1 Methodology

The defining factors for the choice of research methods were the lack of experience with Java EE and the demand from the company to see the results. Because I was working autonomously, the best way for company to learn about my progress was to see the demonstrations of the prototypes and the explanations of the work done during scheduled meetings. I was recommended to develop the module through extensive prototyping.

During my previous computer programming experience, I have developed personal research method principles and a way to estimate the approximate stage of the project in respect to its completion, as well as designing project architecture based on future knowledge. Being a young programmer it was hard to forecast the completion date of the project, because every time I started a project, I was facing a framework that I had no experience with. The challenge was to plan a project schedule based on the knowledge that has not been accrued yet. In this respect it was impossible to plan how long it will

approximately take to learn the new technology or programming language and how many problems will there be to solve during the implementation process. The experience was not extensive enough to predict the development process of a project. I have researched the general principles of Agile and Extreme programming. But both this methodologies were developed for working in a team. The Spiral Model of programming provided a good general model of software development, but was lacking the forecasting methodology. As the research was brief I didn't find the needed answers. I was learning Java programming by myself, working alone at home and have found an individual approach in project planning. It is influenced by the ideas of Exponential growth of Information Technology advocated by several Information Technology philosophers and future forecasters. Ray Kurzweil (2005, 132) states that because of the structure of human brain, learning is an evolutionary process. And evolutionary process "in general progresses in an exponential fashion." (Kurzweil, 2005, 22). Over the course of completed projects I have found a common pattern that towards the end of the project the knowledge increases in a fashion similar to exponential. As the knowledge increases, the learning process becomes faster though the technologies that are being learned become more complicated. Based on the experience of two big projects that I have undertaken, and proved by the experience of my final thesis I have developed a graph showing the increase in complexity level of acquired knowledge and programming methods over time as shown on the Figure 4.1.

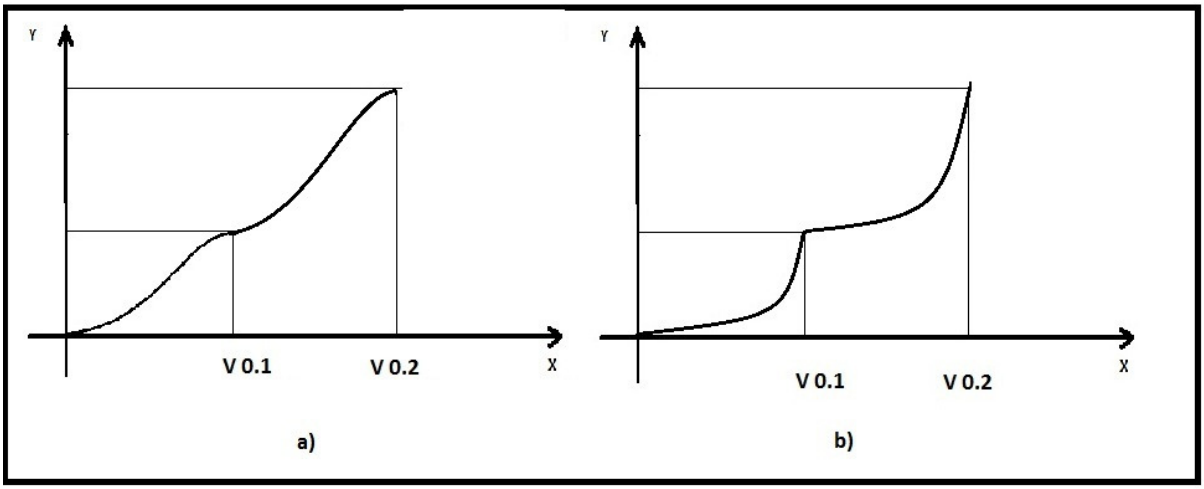


Figure 4.1. a) the dependence of complexity of acquired theoretical knowledge over time b) the dependence of complexity of programming techniques over time

From what can be seen in the figure, the progress of learning and programming is uneven and depends on the stage of development. In the beginning of each stage the learning

is slow, as it is important to understand the general principles of technology, to learn general facts about alternative methods and acquire basic knowledge. Over time though, the theoretical knowledge increases exponentially, until the knowledge is enough to start implementation of a prototype. Then, more time is spent on programming, than on learning, so the learning process slows down. As for programming, there is a need to program during the learning process, but often the programs that are done during that time are simple exercises that are introduced for better understanding of a topic. Such exercises are usually small, seldom containing more than a few classes, though their level of complexity grows over time. When the programming phase of the development of prototype is reached the acquired knowledge can be implemented and the complexity of program grows exponentially. This way of thinking in terms of exponential growth helped in forecasting and planning the project as well as in dealing with complex challenges. In the process of overcoming challenges, I had a clear understanding, that it is important to acquire more knowledge, because the more knowledge is acquired, the more complicated problems can be solved.

Because of the project presentation schedules and lack of knowledge to define final design, I was developing it in a series of incremental steps. This method of software development is known as Spiral Method of Software Development. Boehm (Spiral Method in Software Development, Computer, 5/1988) defines spiral model as a model that reflects the underlying concept that each cycle involves a progression that is done in the similar sequence of steps. As shown on Figure 4.2. Spiral model consists of a set of incremental operations. Each cycle consists of determination of objectives, learning phase, alternatives evaluation, risk evaluation, prototyping phase and review of the project phase. The main advantage of the spiral model is that software is developed in evolutional way.

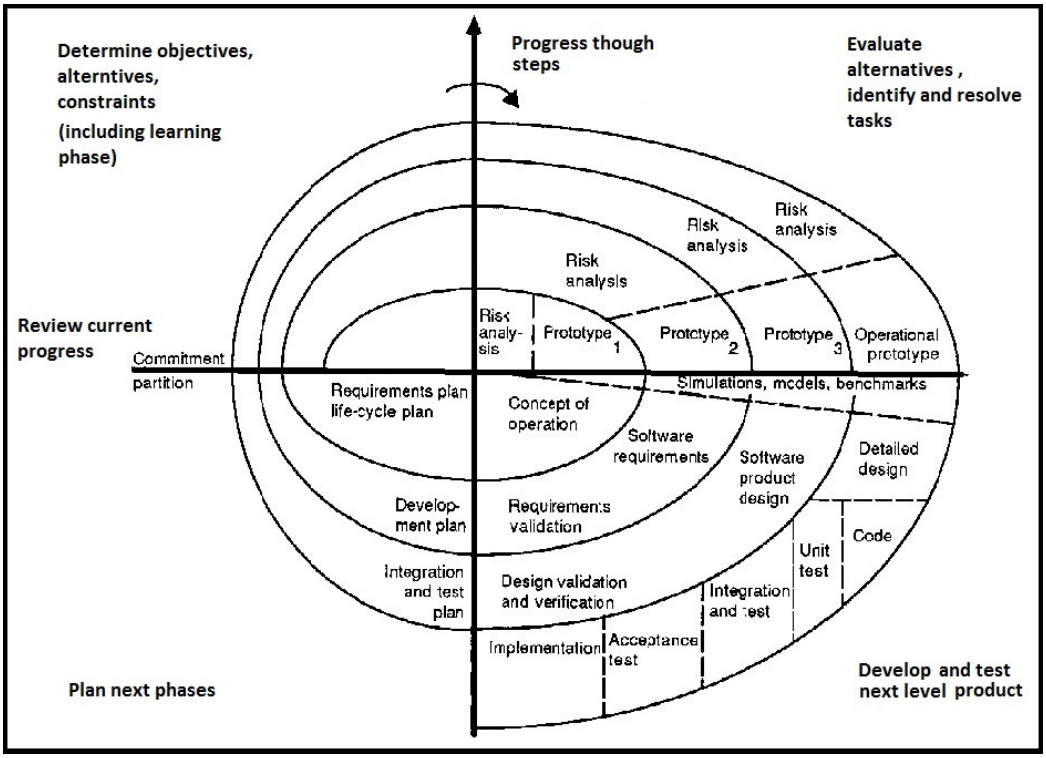


Figure 4.2. Spiral Model of Software Development (Spiral Method in Software Development, Computer, 5/1988, p 64).

Boehm (Spiral Method in Software Development, Computer, 5/1988, p 69) describes key advantages of Spiral model as development model that focuses on creating reusable code elements and allowing to incorporate software quality objectives into software product development because this method emphasizes identification of objectives and constrains during each round. Boehm also states that Spiral approach allows early recognition and elimination of errors and unattractive alternatives. The main disadvantage of Spiral model is the reliance on risk assessment expertise, so the ability of developers to identify and manage sources of project risk is the main concern for reliability of this method. (Boehm, 1988, 70). The main risk factors of the project were the lack of knowledge and experience for forecasting the project flow, the need to plan project architecture based on future knowledge and the need to rapidly create working prototypes based on technologies learned. The personally developed paradigm of Exponential nature of learning process provided methodology to deal with those risk factors. Boehm (1988, 71) states that "risk-driven nature of Spiral Model is adaptive to full range of software project situations" and that "it is particularly applicable to very large, complex, ambitious software systems." From all the software development methods I know, Spiral Model is the most suitable for this project development.

The final architecture and design of the module was evolving during the learning process, corresponding to acquired knowledge and change in architectural requirements. In the end of the project I have spent plenty of time brainstorming different concepts in a search of solution to solve the problem of different architectural approaches in JXLS and JasperReports that arose. In the learning process, I utilized a book “Java 2 Enterprise Edition 1.4 Bible” as my main source and was learning the technologies partially in the order presented in that book. For the purpose of better topic understanding, I was simultaneously learning the corresponding topics from “Beginning Java EE: From novice to professional” and technology specific books. Every time an unclear concept was found I was consulting other books that are listed in the bibliography section. I was also utilizing a lot of unofficial information sources, like online tutorials and programmer forums to get some simple explanations and basic concepts, though the main learning tools were the mentioned books. In the early stages several online video tutorials helped to quickly understand the concept of Server and HTTP protocol.

4.2 Developing JasperReports Technology demonstrator.

Netbeans 6.5.1 is the IDE (Integrated Development Environment) that was used for project development. Netbeans is an Open-Source IDE developed with support from Sun Microsystems. Netbeans provides several useful tools that help to reduce development time such as automatic generation of Ant scripts to start program execution from the IDE, executable project templates and source code editor. Netbeans source code editor has such features as semantic code highlighting, live parsing that allows to recognize and mark errors, as well as tools for showing API documentation and for word completion. Refactoring capabilities of source code editor allow change of class and method names. Important features of the IDE are debagging support, automatic code generation, tools to integrate server and database support, and ability to extend functionality via plug-ins. (Netbeans 6.8 features). Netbeans version 6.5.1 was used throughout the project with the most complete feature bundle installed.

Learning Java EE capabilities of Netbeans was the first step in project development. I have created a Java EE Application under File>New Project>Java EE> Enterprise Application. IDE has generated not a single project, but three projects instead, as shown on Figure 4.3.

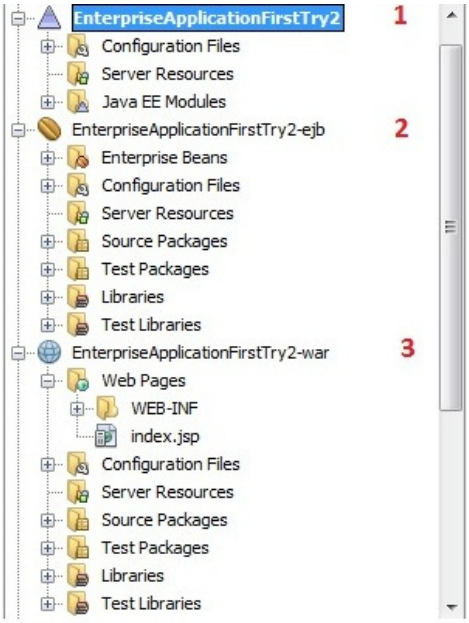


Figure 4.3. Netbeans Project tree.

The important feature of the majority of newly generated Netbeans projects is that they can be executed without any modifications. When application started executing, three separate tabs appeared on the output panel: EnterpriseApplicationFirstTry2 (run) tab, that is the main panel of execution similar to execution panels of Java SE applications, the Java DB Database process tab, and Glassfish V2 tab. The Java DB (Database) also known as Apache Derby DB is an open source RDBMS implemented completely in Java. Java DB and Glassfish application server came as a part of Java EE SDK (Software Development Kit) that was bundled with Netbeans 6.5.1 that was downloaded. The application lunched the web browser where I was able to see the simple "Hello World" message in a form of HTML page. The URL for the page that I could see in the browser was *http://localhost:17330/EnterpriseApplicationFirstTry2-war/* . The URL didn't feature any particular file, but only a Web application directory. Usually, when web browsing is performed, the URL is pointing to an HTML or other type of file. The title of the page was "JSP page". In the Project tree pictured in the Figure 4.3, there is only one file with JSP extension, *index.jsp*. After reviewing the contents of the file that appeared to be HTML markup, I could conclude that this is the file that produced the page that I was able to view in web browser.

The index.jsp file is situated in EnterpriseApplicationFirstTry2.war directory that is representing a Web application. When I tried to run the EnterpriseApplicationFirstTry2-war project, that is a Java EE Web Application project, build failed because the context root was

used by Enterprise application. But when I cleaned the Enterprise Application project, the Web Application project was able to execute and outputted the same web page. It is worth of notice to mention that when the projects were Run the second time they were executing considerably faster, because the Glassfish server and the Database were already running.

Databases and Application Servers can be observed at the Services tab as shown in the Figure 4.4. The Services tab allows to observe and manage resources such as Web Services, Databases, Application servers and other. Under Databases tree, two installed RDBMSs can be observed as well as available database drivers and template databases. Under Server tree, the available servers are represented. The Applications folder of Glassfish server contains all the applications installed on the Server. The EnterpriseApplicationFirstTry2 is contained in Enterprise Applications folder and only Web Application module is deployed. This happens because EJB module is generated without any code and is not executable by default.

For application to be executed on a server it first needs to be deployed on it. This means that the files that contain the application code need to be put into a special directory on a Web Server. Java EE Web applications are deployed in a form of WAR (Web Archive) file. WAR file contains all the Java EE Web application components, project specific libraries and deployment information that is used by the server during the deployment process. (Mukhar el al, 2004, 38).

Inside a WAR file there are several levels of directories. The top directory contains files that can be accessed directly from the client's web browser, such as client side resources, HTML and JSP pages. The META-INF inner directory contains the manifest file that defines files that are contained in the archive. Manifest file is used by JVM. The WEB-INF directory contains files that are used by web container. There are two subdirectories: lib and classes. Lib directory contains extra libraries that are used by the application classes contained in the classes directory. WEB-INF directory also contains web.xml deployment descriptor file. Deployment descriptor contains mapping of deployed objects to actual classes and to parts of URLs to accesses them, as well as session configuration, initial parameters, security and other information. (McGovern, 2003, 103). This allows to configure specific Servlets to respond to specific requests based on request URL. WEB-INF directory also contains the sun-web.xml file that contains the context root parameter. The context root is used to specify the root directory of Web application.

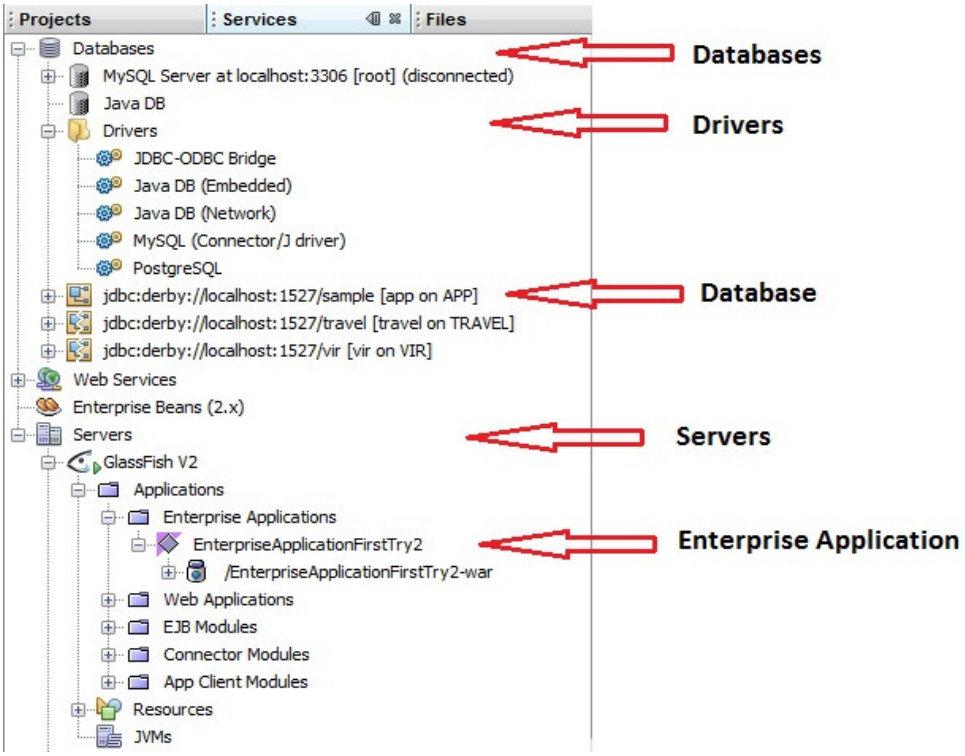


Figure 4.4. Services Tab

To demonstrate the features of JasperReports framework to the company I have created a MHG_JASPER_TEST web application. The goal of this application was to research and practice the creation of JasperReports enabled applications, as well as to define future steps in project development and to experiment with possible structure of the application. The initial idea was to develop a separate class that will handle all the interactions with JasperReports framework, so it can be reused in later iterations of the program. The first step in technology demonstrator development was to build a web application. Unlike the previous application, I decided to use JSF framework to create user interface, because it was used in the original MHG ERP Reporting module.

I have created a new project MHG_JASPER_TEST under File>New Project> Java Web > Web Application and chose Visual Web Java Server Faces as a framework for the project. The initial structure of the application was to include a welcome JSP page that has a link to a Servlet that will generate an Excel report with help of ReportGenerator class. ReportGenerator class was designed to handle all JasperReports specific operations showed in Figure 3.6, so other parts of the program can use JasperReports functionality without dealing with framework specific details.

The final version of ReportGenerator class contains several methods; six of them will be discussed. In order to ensure the availability of all needed templates, I created a

checkForAllTemplates method. This method would create a Hash Map that contains all the available templates. This method would be useful if a particular MHG ERP configuration wouldn't need all the report types. The available report types can be controlled by simply including available templates into the accessible directory. The *checkForAllTemplates* is using underlying overloaded method *checkForTemplate* for individual template check. This method is able to check for template stored in either .jrxml or .jasper format and because of the overloaded form can access files via URL or as a File object. If needed, the method can compile the initial .jrxml template into JasperReports template format that can be used for filling the report with data. The compilation is done by a separate *compileTemplate* method. The *fillReportWithData* method fills the .jasper report template with map parameters and a datasource. During the development this method was overloaded to accept several datasources supported by JasperReports framework. In order to create static report, a version of method was created that accepts empty data source. Because the undertaken project includes communication with a database, the final version of *fillReportWithData* accepts a Database Connection object as one of the parameters. The *fillReportWithData* method returns a JasperPrint object, an object that represents a ready report in JasperReports native format. (Heffelfinger, 2006, 14-16). The *returnExcelReport* and *returnPDFReport* are able to convert the JasperPrint object to Excel and PDF formats accordingly. The methods are able to get an output stream from *HttpServletResponse* objects provided in a form of method parameters. The methods are able to modify the response object to assign it a format specific type, to set a name for the returned file in the response header and to return the reports to the client.

In order to compile the ReportGenerator class, several libraries had to be downloaded and added to the project. Netbeans allows to add library to the environment via libraries wizard, that can be launched through Tools>Libraries. Once the libraries are added through the wizard, they can be added to individual projects. When web application project is built and deployed to the Server, the external libraries are added to the lib folder of the WAR file. The libraries required by JasperReports that are used by MHG_JASPER _TEST project include JasperReports-3.5.2.jar that contains main JasperReports libraries and jasperreports-javaflow-3.6.1.jar that contains libraries for multithreading. For generation of PDF reports iText-2.1.0.jar library was added, and for generating Excel reports, jxl.jar was added. Jxl.jar is a JExcelApi library discussed in section 3.6.

In the process of learning several JasperReports templates were created, to practice different aspects of the report creation. These templates were static reports, filled with empty datasource objects. To fully demonstrate the capabilities of JasperReports framework I had

to create dynamic reports and developed a test database for this purpose. The RuslanDB1 database was created through Services tab> Java DB> Create Database. After database name, username and password were configured through the Create Database wizard, the connection was needed to be established to the database. Then I have proceeded through Create Table wizard, where I was able to create a table named DATATABLE with six columns. The first column, named NUM was set as a primary key and the data type was set as integer. The rest of the columns were defined as varchar fields of various lengths. The final structure of the database can be seen in Figure 4.5.

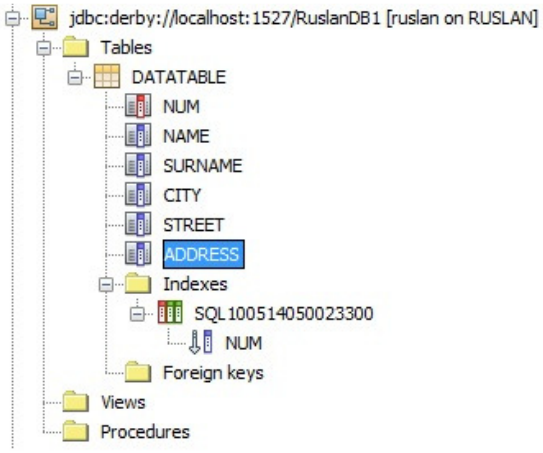


Figure 4.5 Database structure viewed in Service Tab

The database represents an inventory containing the address details about several people. Each person is given a unique identification number, which is used by a database as a primary key. In order to fill the table with data, I have selected Datatable>View data to see and modify the database data. This command launched a Netbeans SQL editor, which allows easy execution of SQL queries. It also provides tools to add a row to the table and to observe the table resulted from query. Figure 4.6 shows the Netbeans SQL editor and the contents of the RuslanDB1 Datatable. Each record holds the inventory number represented in NUM column, the name and surname of a person, hold in NAME and SURNAME tables, as well as address details represented by CITY, STREET and ADDRESS tables. Insert record button allows to add records through add record wizard. It is also possible to add data to the table by executing custom SQL queries. The RuslanDB1 database was used to create first dynamic JasperReports templates.

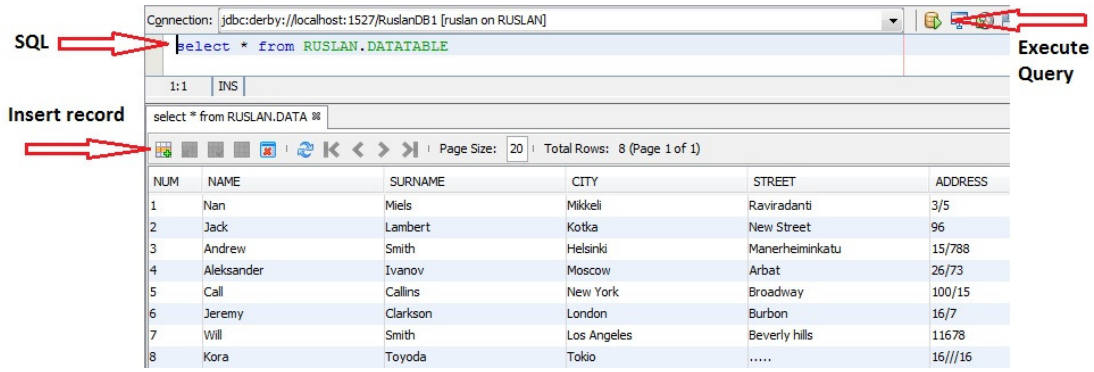


Figure 4.6. Netbeans SQL editor

iReport is the official JasperReports template graphical designer. iReport exists in two flavors: as a Netbeans plug-in and as a standalone program. I was using iReport as a standalone program because it has incompatibility with several other Netbeans plug-ins like IceFaces visual designer. The functionality for both versions is the same, and in fact the stand alone iReport 3.7.1. that was used in the project is based on Netbeans platform. iReport provides all the necessary tools for graphical report template creation, as well as predefined templates, tools to connect to database, to compile and fill the reports. Figure 4.7. shows the iReport user interface and the final design of report template for MHG_JASPER_TEST application. iReport allows to view the template in graphical mode, as well as to switch to XML version. The changes on either side are instantly transferred to the other. The compile report button allows compilation of designed report to the .jasper format. The preview mode allows to fill the report with parameters specified in popping-up windows and by executing a query on a specified database. In this way, the preview mode generates a Jasper Print file, which is then represented in the editor.

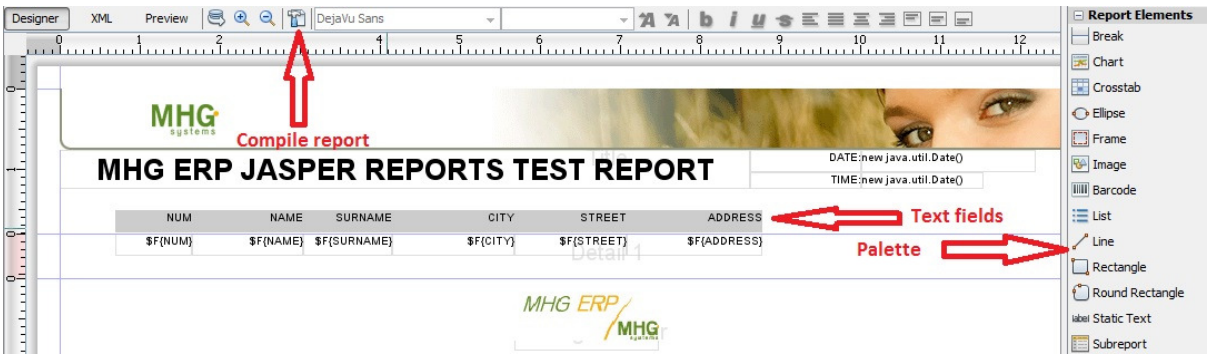


Figure 4.7. iReport featuring a template.

The new template was created under File>New that launches a template wizard. I have chosen to create a report from an empty template and named it report3.jrxml. The

source code of report3.jrxml can be viewed in Appendix A. I have collected several images from MHG Systems web site to make report more visually appealing. The possibility to add images is one of the main improvements over JXLS framework. In order to construct a report layout the palette featuring avaliable elements can be used. Most of the elements of the design used in the report are text fields. Text fields in JasperReports are one of the most important elements allowing to output text dynamically. After the fields were placed in the right places and the design phase of development was complete, I preceded to XML view of the template to hand code the inner working of the report.

To embed SQL query to report template the queryString tag is used. Unparsed character data that is embedded into it allows to write any type of SQL queries needed. The SQL query can be dynamically modified by parameter passed to the template in a form of HashMap. In order to be used, parameters need to be specified before the queryString tag. Unfortunately there is no way to embed logic external to SQL query and parameters that are passed to the template are immutable and no logic can be executed over them either. For the report3 template a simple SQL "Select NUM, NAME, SURNAME, CITY, STREET, ADDRESS from DATATABLE" query was used. Each returned column needs to be defined in a special field tag in order to be used in the template. If SQL query is unable to execute for some reason, JasperReports will return an empty report, or an exception will be thrown. In terms of exception communication, the JasperReports framework has some limitations, and sometime cannot point to the right cause of the problem. For example when there was a mistake in query pointing to the wrong table, the NullPointerException was thrown without specification what caused the exception, and it took time to go through the code to find the real cause of the exception. The data type integrity is ensured by the requirement to specify a type of field. If query returns a column of type different from the corresponding filed type, an exception is thrown.

JasperReports template is separated into several elements depending on the position and function. The most common elements are title, page header, column header, detail, column footer, page footer and summary. (Heffelfinger, 2006, 15). When query is executed, the results of the query are stored in corresponding fields. Each field acts as a list each containing an entire column of values. The detail element of the report is repeated as many times, as there are rows in returned fields. There is no way to access the specific record in the field. The field can be referenced in detail element of the template, and when a new iteration of detail element is added to the filled report, the values of fields are iterated automatically, so each detail corresponds to a row in table that would have been returned by execution of

SQL query. This approach allows easy and fast implementation of simple reports, but is very limiting when there is need to manually iterate fields, or to pick one specific row. And, unlike the usual XML document, XSL (eXtensible Stylesheet Language) can not be used to specify the conditional operators.

In order to be used, fields need to be referenced in textField tags that are situated inside detail element of the template. TextField tags allow to specify the accepted data type of the expression as well as to specify colors, position and size of element. Inside each textField tag is the textFieldExpression tag. TextFieldExpression tag is one of the expression tags, which are very permitting in terms of functionality. It allows to manipulate fields and parameters, as well as variables that are used to reference some calculations with similar to JSP EL language. The content of the textFieldExpression tag is an unparsed character data. It is also possible to embed Java code, in a manner similar to JSP expressions. This means that Java code has to return a value corresponding to the type defined in the textField tag. For example, one of the fields shown in Figure 4.7 is used to output current date generated by Java code. Images can be added in a similar manner by specifying a URL in the imageExpression tag. Because it is possible to reference parameters in expressions, URL can be passed to the template as one of the parameters hold in the HashMap.

The described template file in form of .jasper file was put to resources folder at the top directory of WAR file, so it can be accessed by the application. To execute the program I have created a ReportServlet class. The *doGet* method is implemented by *doPost* method because there is no form parameters to utilize. The *doPost* method is calling appropriate ReportGenerator methods to load a template file, to fill the report and to generate and return the Excel report. In order for Servlet to intercept the requests, modifications to the web.xml file was made, so every request that is featuring the */report/* directory will be forwarded to the ReportServlet. Figure 4.8. shows generated Excel report as viewed in Microsoft Excel 2007.

Figure 4.8. Excel report generated by MHG_JASPER_TEST application.

The main features of the report are the generated table and the embedded images. It is also worth of notice that there are no grid lines, which are common to excel documents, though each value or image is inside a corresponding line. The embedded top image has also stretched row 1, so it can be fit into it. The current JExcelApi framework doesn't allow embedding images on top of the grid. The size of the Excel report is 45.5 kb. The numbers that are present in the report are stored as text, so, in order to use the numbers in functions, they need to be first converted from text to numbers. After this is done, the numbers can be manipulated like any other numbers in the Excel document. This inconsistency is an issue of JasperReports utilization of JExcelApi. The report can also be opened with Open Office Calc spreadsheet application. When opened with Open Office Calc, the report looks same as when opened with Microsoft Excel, except that in Calc the spreadsheet grid is visible. The numbers are represented in a form of text with an apostrophe sign in front of each number. The number column can be extended, and the numbers will be iterated accordingly, but functions can't be executed on them. It is possible to convert the text to numbers, by executing a VALUE function for the specified columns, but in this way another column needs to be used to store the function. From my current knowledge I can conclude that the only way to convert these tabs to numbers without using other columns is by manually deleting the apostrophe signs in each column. The objective of the project was to generate Excel reports compatible with Microsoft Excel application. The compatibility with Open Office Calc can be subject to further research.

The next step in development of technology demonstrator was the creation of a more sophisticated user interface of the program. The user interface needed to show the design concept of the future application, as well as to provide means for user to choose either an Excel or a PDF as a final report format. Visual web JSF framework was enabled in the web application project when the project was created. The framework includes a collection of specific libraries that were automatically added to the lib directory of the WAR file, as well as the initial JSP page set to be opened in a special UI editor. UI editor is provided via Visual web JSF plug-in, which is included in Netbeans 6.5.1. by default. Figure 4.9. shows the UI of Visual web JSF featuring the final design of the welcome JSF page. JSF UI editor allows adding components to the page by simply picking them up from the palette and dropping them in the page area. It is possible to switch the view of the page from visual to JSP code, as well as to easily access Managed bean. The designed page features several JSF components such as static text, images, table that was added to the page as a design concept and buttons. The buttons are used to start the execution of the report generating logic. When button is pressed, it generates an event that is handled in the Managed Bean by special methods. The initial version of the program had only Excel report generating button and was simply forwarding the request to the Report Servlet.

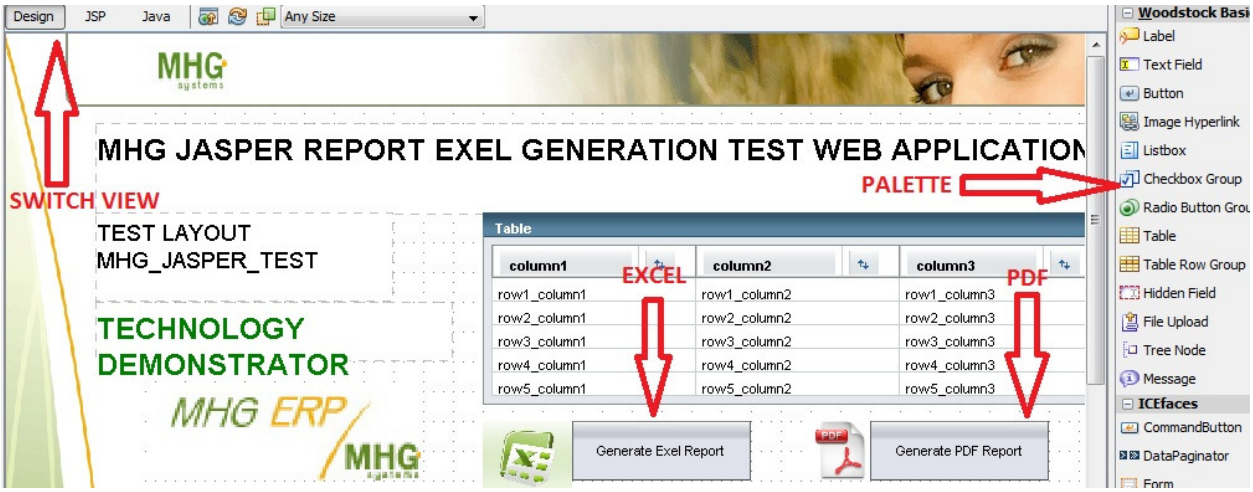


Figure 4.9 Visual Web JSF editor.

In order to extend the functionality and enable the PDF report generation, I have rethought the structure of the application. I have decided to create a class similar to ReportGenerator in functionality, but more abstract. ReportGenerator is handling the entire process of generation of reports, from loading the template to sending the report as a response to the user. I decided to abstract this class in a way, that it is just returning an output stream

containing the report, and the calling class can decide what to do with the report: either to send it to the client, to store it as a file or to do any other operation with the stream. The new class is called the *ExcelReportGenerator*. Despite of the name, it can produce both Excel and PDF reports. It is a simplified version of ReportGenerator class. It receives template as an input stream and returns a generated Excel or PDF report as an output stream. So, the functionality of appending ready report to the response object moved to the Managed bean of the JSF page.

4.3. Building MHG ERP Reporting module

The next step of the project was to develop a real life MHG ERP Reporting module. Because of the confidentiality of the MHG Systems software the development of the module will be discussed in lesser detail, though all the main concepts will be described. The principles and software applications that were used to develop the application are described in section 4.2. The MHG ERP Reporting module execution sequence is described in Figure 4.10.

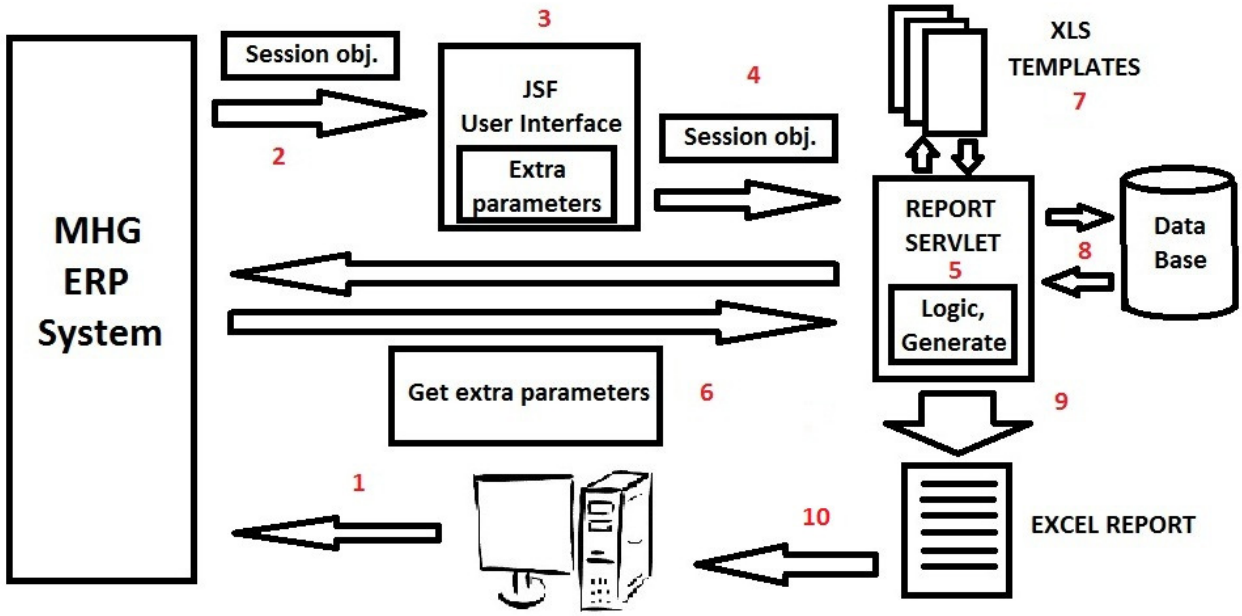


Figure 4.10. MHG ERP Reporting module execution sequence.

As a first step, client is logging into the MHG ERP System. MHG ERP System is consists of many modules, so client is able to manage employees working in the field, to view centralized data, and to communicate data to partners and contractors. When client need a general data over a certain period of time, or he is willing to manipulate data using Microsoft

Excel application, he requests systems to generate a report. When MHG ERP is receiving a request for an Excel Report, it is forwarding the request to the MHG ERP Reporting module. This procedure is marked by number 2. There is also a Session bean that holds customer specific information that is forwarded with the request, so the Reporting module can recognize the customer, his role in the network of partners and access permissions. As a third step, the Reporting module is generating a JSF page, that allows the customer to choose the type of report he is willing to receive and to define some report specific parameters such as time interval of the data to be included and extra information to be used. The JSF page is including the specified parameters into the Session bean and forwarding it to the ReportServlet as a stage 5 of the execution process. The ReportServlet is the main execution class of the MHG ERP Reporting module. As a step 6 of the process, the ReportServlet class is analyzing the data received through the Session bean. Those parameters are incorporated into a Hash Map. Also Servlet is generating several extra parameters that are also being added to the Hash Map and preparing the response object to accommodate an Excel report. Depending on the report type, the map is passed to one of the 13 report specific methods where report specific parameters are being requested from the ERP and some extra logic is applied to the HashMap in order to prepare it for report generation. Once the HashMap is ready, ReportServlet is loading one of the 13 templates, which are filled with JXLS logic. This is a step 7. As a step 8, some of the HashMap parameters are combined with SQL queries that are stored in templates, which are then communicated with the database. As a step 9 the data from the Database is added to the report and the Excel report is generated. Finally, the ready Excel report is appended to the response object and sent to the client that requested the report. This operation is shown as a step 10 of the Reporting module execution sequence.

In order to learn the functionality of the MHG ERP Reporting module, the files that are described in section 4 needed to be assembled in a functional Web Application project. There was a need to create all the ERP classes that are interfacing with the module as well as a test database, so the reports can be generated. As a first step, I decided to create a test database named MHG_APP_DATABASE. The MySQL RDBMS was downloaded and installed on the computer. Netbeans 6.5.1 comes with a built in support for MySQL RDBMS. To configure the database connection I selected Services>Databases>MySQL>Properties. The properties window appeared where I have defined Server host name as localhost, Server port number as 3306 and also configured the username and password. In admin properties tab, I have defined administration tool, start and stop commands as described in MySQL

Netbeans tutorial. (Connecting to a MySQL Database). After the MySQL server was configured I continued with creation of a test database named MHG_APP_DATABASE. The database was created in such a way, that it represents the structure of real life MHG ERP database, or at least all the entities that are utilized by the MHG ERP Reporting module.

MHG ERP Database structure is a very sensitive data, so even the structure of a test database can not be discussed in detail. I am providing a general overview of the created database. In order to create a test database, I have learned the structure of the original MHG ERP database. Based on the SQL queries embedded to13 XLS templates I have created a map of tables and columns of the database, also featuring data types. I have also researched the rest of the provided classes for extra database information and refined the map of the structure. Then I created tables and columns of the database in a way described in section 4.2. The final version of MHG_APP_DATABASE contains 24 tables of total 162 columns. The hardest part was to fill the database with data. Because there are many relations between tables the data consistency was a challenge. To add data I was using Netbeans SQL editor, with up to 10-15 editors open at the same time that were spread among my laptop and external monitors. After the database was filled with data, I have executed all the SQL queries that were embedded in XLS templates. SQL queries in XLS templates are dynamic in their nature and need to include the parameters from the HashMap to be executed. I have defined and recorded the required parameters and values that allow database to return consistent tables. The only drawback of the database is that, due to the fact that adding data to the tables is time consuming, the returned tables are not very large in terms of records, so testing with real life database is still required.

Once the database was created, I proceeded with creating all the necessary classes that MHG ERP Reporting module is interfacing with. There were three types of classes created: the *model* classes, the *logic* classes, and *resource provider* classes. The model classes are implemented in a form of JavaBeans. Such classes represent different logical objects such as Client or Machine and contain class variables to hold the object specific characteristics. The logic classes are able to return collections of model classes or individual model classes. They also hold some calculation logic used by ReportServlet. The resource provider classes are represented by Db and UITextHandler classes. Db class abstracts the connection to a database so the ERP can use different RDBMSs without virtually any changes in the rest of the code. The UITextHandler is used to provide multilingual functionality of the MHG ERP System. It connects to the language .property files and is able to return strings in any language if required property files are provided. Any classes in MHG ERP that are using text for user

interface are referencing methods of this class. Both resource provider classes were implemented in full functionality. There were also classes created that are used for formatting and for unified exception logging. In the final version of the module that is discussed in section 4.4. consists of 9 packages containing 39 classes. The process of creation, testing and refinement of the MHG ERP interface classes was the most time consuming phase of the project. Because of the restricted functionality of described classes due to focus on Reporting module interaction, I called them surrogate classes. So, the first fully functional Web Application project was named MHG_ORIGINAL_APP_SURFILES.

When the initial request is sent to the MHG ERP Reporting module, it already contains a set of values transferred by a Session bean. In order to set the initial values to the Session bean I have created an InitServlet class. This class creates a Session bean and then adds parameters to it in a form of model classes initialized with specific parameters, initialized resource provider and format classes. The WAR file also needed to be modified, so InitServlet intercepts all requests sent to the URL with .kkk extension. When executed, the MHG_ORIGINAL_APP_SURFILES sends user a welcome.jsp page. This is a simple page, which contains the hyperlink to the InitServlet, which initializes values and forwards the request to the reports.jsp page. Reports.jsp was developed as a user interface for the Reporting module. The version used in MHG_ORIGINAL_APP_SURFILES web application is a downscaled version of the original MHG ERP reports.jsp file. For the convenience of testing all the parameters that are selected in this page are predefined in InitServlet. There is a button in reports.jsp file that is, when clicked, forwarding the request to the ReportServlet that is able to generate and return an Excel report to the client. The ReportServlet that is used in this application is a standard MHG ERP class that was sent to me as a part of development package described in section 4. The ReportServlet was configured in WAR file to intercept requests directed to .rsl URL extension. In order to enable ReportServlet JXLS functionality JXLS 0.9 libraries were added to the project as well as required underlying POI 3.6. libraries.

The MHG_ORIGINAL_APP_SURFILES was able to return all 13 types of Excel reports based on test database that I created. Figure 4.11 features a ChippingsByChipper report produced by the application.

	A	B	C	D	E	F	G	I	J	K
1	Jacob			Chippings by chipper						
2				02/03/2011						
3	Chipping date between: Fri Feb 01 18:56:22 EET 2008 - Wed Mar 02 18:56:22 EET 2011									
4										
5	Chipper: 12455									
6										
7	Chipping date	Load book num	Chipper	Land owner	Storage	Project name	Storage sort	loose-m³	m³	Groun
8	12/10/09	1	Andrew Smith	George Washington	124	planOne	1	1234	493.6	Yes
9	12/10/09	1	Andrew Smith	George Washington	124	planOne	2	1234	493.6	Yes
10	12/10/09	1	Andrew Smith	George Washington	124	planOne	3	1234	493.6	Yes
11	12/10/09	1	Andrew Smith	George Washington	124	planOne	4	1234	493.6	Yes
12	19/02/10	5	Andrew Smith	George Washington	124	planOne	1	1234	493.6	Yes
13	19/02/10	5	Andrew Smith	George Washington	124	planOne	2	1234	493.6	Yes
14	19/02/10	5	Andrew Smith	George Washington	124	planOne	3	1234	493.6	Yes
15	19/02/10	5	Andrew Smith	George Washington	124	planOne	4	1234	493.6	Yes
16										
17	Chipper:12455 total							1234	493.6	
18										
19	Report total							1234	493.6	

Figure 4.11. JXLS generated Excel report

The SQL query of this report was modified in order to get larger output, through an exclusion of the Storage sort limiting parameter. Except this modification, the template is a real life MHG ERP template. The features of the generated report include the ability to apply different colors to text and cell background as well as to add strait lines. A useful feature of the framework is the ability to include hidden formulas to the Excel report. There are several inconsistencies in the execution of Microsoft excel when observing JXLS generated reports. When report is opened, sometimes it influences other open instances of opened Excel reports. Sometimes reports can not be closed, and other reports can not be opened before the JXLS generated report is closed. Once the report is closed, the dialog pops up, asking to save changes to the report, even if no changes were done. When a report with a standard SQL query was generated the Microsoft Excel issued a circular reference warning, though the formulas were not modified in any way. Also when reports are opened in Windows 7, the list of the frequently visited folders is deleted. These framework inconsistencies were some of the reasons behind the decision to redesign the MHG ERP Reporting module.

The Excel report featured in Figure 4.11. is a ChippingsByChipper report. This type of report allows customers to see all the work done by chippers operating chipping machines over a selected period of time. Chipping machines are used in the forest industry to turn wood into chips that are used as a Biofuel to provide heat and electricity. The header of ChippingsByChipper report includes such features as the name of the client, the date of the report generation and the time interval of report data. Customer is able to see operations done by each chipping machine and view calculated totals for the work accomplished. The chipping machine operations are represented as a table consisting of 15 columns: Chipping date, Load book number, Chipper/Operator of the machine, Land owner, Storage identification number, Plan name, Storage sort, the volume of the chips represented by Loose

m3 and m3, Ground, Metal and Stone markers, Humidity percentage, planned energy output in MWh, and weight of the chips. ChippingsByChipper report was used as a pilot report template for the implementation of JasperReports functionality. ChippingsByChipper.xls template is a typical MHG ERP Reporting module template, featuring all the used methods and techniques used in other reports. It is also a medium sized template featuring a single sophisticated SQL query. Because of these characteristics it was chosen by me as a showcase of the JasperReports technologies.

The most significant features of JXLS templates include the ability to reference parameters in a manner similar to JSP expressions, the possibility to include “if ”, “then”, “for”, “for each” and other conditional expressions, and to embed dynamic SQL queries. The template is represented by an Excel document, so it is possible to define the values inside Excel cells, to hide values, to embed Excel formulas and to access individual values of the database returned row. The iteration through the rows is done in a manner similar to the JasperReports, and it is not possible to explicitly access a specified row or a value in a column. But, in general, due to the availability of the logical expressions and to more Java like coding JXLS allows to develop very dynamic and functional reports with a small amounts of code.

In order to provide functionality for the report.jsp page, the InitServlet needed to be upgraded to be able to send selection options for the report.jsp. This required creation of 14 ArrayList objects filled with String variables. The ArrayList objects are appended to specifically designed Request bean object, which is forwarded to the report.jsp page. The completed program with partially enabled report.jsp page was demonstrated to the company. During the demonstration I was also informed that the company is switching its presentation layer framework from JSF to a much more advanced IceFaces.

In order to develop IceFaces applications a pack of plug-ins was installed through Netbeans plug-in manager accessed through Tools>Plugins. I was able to find IceFaces plug in under Available Plugins tab and installed the plug-in with a help of a straight-forward wizard. The plug-in package included IceFaces Design-time and Run-time libraries, IceFaces project integration, Visual Web IceFaces plug-ins as well as Visual Web JSF Backwards compatibility kit. In the process of framework testing I have discovered its incompatibility with iReport plug in. To solve the problem I removed the iReport plug-in and installed it as a separate application. After framework was set up I have created a new project named MHG_ORIG_APP_ICEFACES with Visual Web IceFaces selected as a framework. After that I have copied most of the classes to the new Web Application. The initial goal was to

develop IceFaces based User Interface so the rest of the functionality of the application was left unchanged.

Development of the IceFaces interface turned out to be a bigger challenge then creation and modification of the JSF user interfaces. The UI IceFaces editor, though being an extension to Visual web JSF editor is much less logical and more complicated. It has complete incompatibility with the standard JSF components. During the process of IceFaces framework learning, I was provided with an IceFaces interface in a form of report.jspx file that was developed by the company as part to the IceFaces framework migration. Jspx file extension is used for JSP files that represent a part UI design and can be included in the other JSP files. For the development convenience I have created my own report.jsp file featuring an extra links to SecondInit and ThirdInit Servlets created in the MHG_ORIG_APP_ICEFACES4 version of the application. The extra Servlets were designed to pass different Session bean parameters to the ReportServlet, allowing simulation of different users accessing the Reporting module. Figure 4.12. features the report.jsp page in the Visual Web IceFaces UI designer. In fact, due to the poor performance of UI designer the interface was mostly developed by adding code to the underlying JSP file. Unlike JSF, IceFaces are not able to utilize ArrayLists of Strings for specifying selection options for the dropdown menus, but are using Arrays of SelectItem objects or DefaultSelectedItems objects containing SelectItem objects instead. In order to provide the report.jsp with selection options the InitServlet class was modified to append 14 DefaultSelectedItems objects to the Request bean.

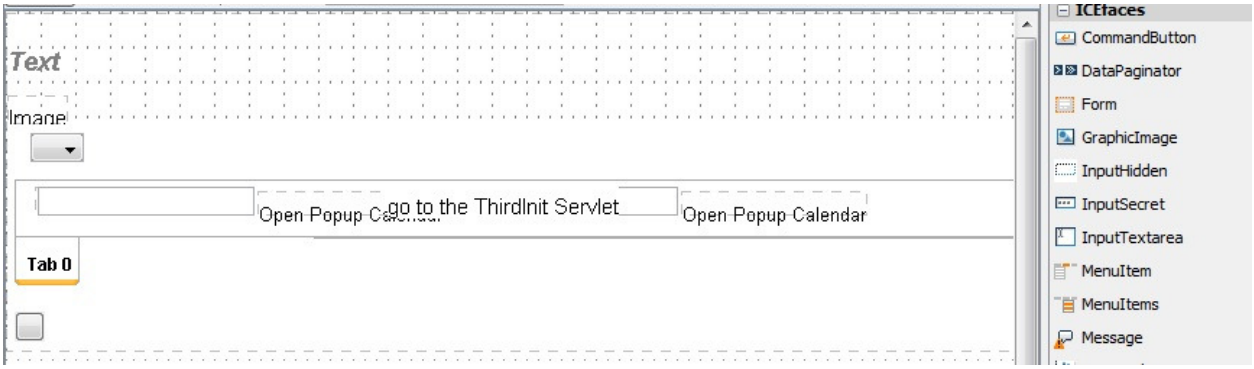


Figure 4.12. report.jsp as seen in Visual Web IceFaces UI designer

The next step in implementation of the project was substitution of JXLS framework with JasperReports framework to generate the report. As a first step, I have added ReportGenerator class to the application. The ReportGenerator is the same class that is discussed in section 4.2, though its *returnExcelReport* and *returnPDFReport* methods return

an output stream containing generated reports, just like methods in ExcelReportGenerator class discussed in the same section. So, the ReportServlet is the class responsible for returning the ready reports to the client. Methods of ReportGenerator class are being called from the ReportServlet method. ReportServlet is also the class that is creating a database connection through DB class. In order to be compatible with JasperReports framework ReportServlet class was modified. The main difference is the way the HashMap is built. When JXLS framework is used, the HashMap contains data objects represented by model classes of MHG ERP. This is convenient because in JXLS templates there are no restrictions over the place where methods can be called on Java objects. In JasperReports templates Java code can be executed only inside expression tags. In jrxml templates expression tags can not be placed before the SQL query and field definition tags, so there is no way to manipulate data and to get values from the Java objects, that can be used in dynamic SQL query. So, in order to provide functionality similar to JXLS report, the database parameter logic had to be taken out of the report template and was placed in the ReportServlet.
In MHG_ORIGINAL_APP_ICEFACES4 application only ChippingsByChipper reports were enabled for JasperReports generation. The ReportServlet has specific methods for each of the 13 report types, so the *doChippingsByChipperReport* method was modified to be compatible with JasperReports framework. The modification was the interaction with the HashMap, similar to the modifications in other ReportServlet methods.

Figure 4.13. shows a ChippingsByChipper .jrxml template. This template is a further development of report3.jrxml template. The major improvements of the template include the ability to use data from MHG_APP_DATABASE in its main table, the dynamically generated charts and a barcode. The template was developed to resemble the functionality of the ChippingsByChipper JXSL template. It features similar information in its header: the name of the report, the name of the client, the data time interval and the chipping machine which activity the table describes. Also the date and the time of the report generation are included in the header. The sum of specific fields is shown in the summary part of the table. JXLS templates are featuring Excel formulas and Java code embedded into them to calculate some report specific data. For example, the ChippingsByChipper report is featuring formulas to convert the value of the chips volume represented in square meters, to reference the resulting value, or to perform Excel mathematical operations such as calculating the sum of a all values in a column. For the most part, the presentation level textFieldExpressions were enough to implement the needed functionality like conversion of a number in a different format or performing some calculations. But in order to create report wide calculations like to

sum all the values of one specific column JasperReports variables were used. Variables are declared in the beginning of the JRXML template and their evaluation time and mathematical function can be specified in a form of attribute. Unlike the fields and parameters, inside each variable declaration there is a variable expression that allows to reference and to manipulate fields and parameters.

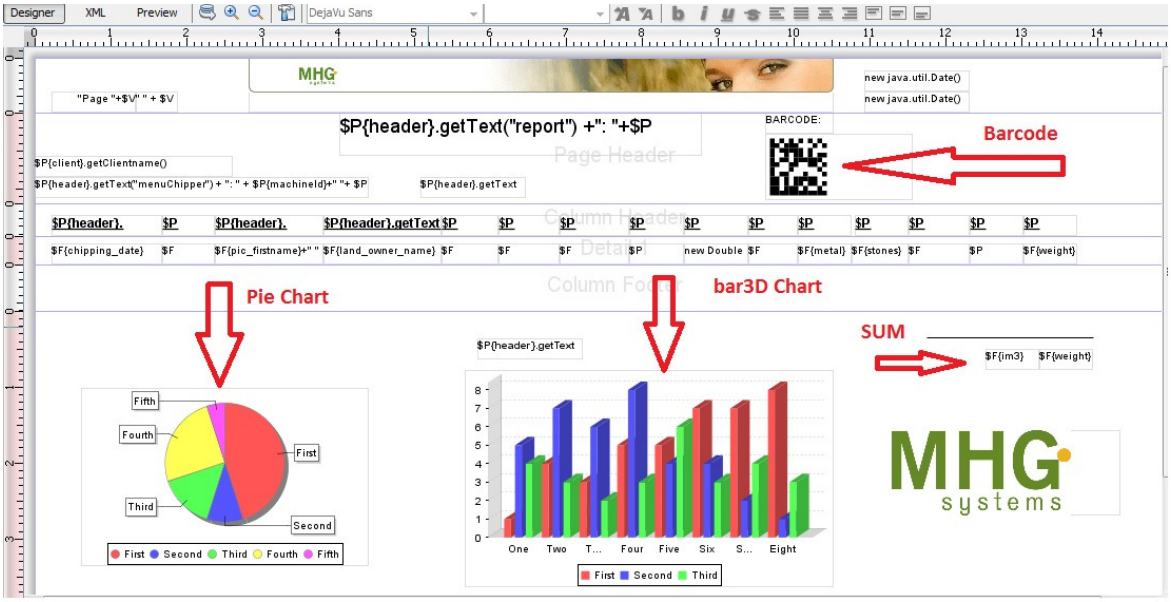


Figure 4.13. ChippingsByChipper template viewed in iReport

Important differences from the same report type generated by JXLS framework include not just images but also special dynamically generated elements. The Pie and bar3D charts are used to graphically display the most important parameters for each entry. Pie chart is using two parameters: *key* expression and *value* expression. Key expression defines the column of the table that will define the amount of portions of the chart and value expression defines the column that holds the values to be represented in the chart. The bar3D chart is different from the Pie chart because the columns are shown in 3D. Bar3D chart requires three parameters to be specified: *series* expression, *category* expression and *value* expression. The series expression defines the objects that are being compared. Each object is assigned a specified color. The category expression defines the x-axis of the chart comparison coordinate system. It is used to compare values for each category. The value expression simply represents a value of the specified column. Both charts can be drag-and-dropped from the iReport palette and the required parameters can be assigned through a chart wizard or by manually coding them in the jrxml file. The charts are implemented using jFreeChart libraries. In order to enable chart functionality, the libraries need to be added to the project.

jFreeChart library consists of four jar files: jcommon 1.0.16, jFreeChart 1.0.13, jFreeChart 1.0.13-swt and swtgraphics2d. It is also possible to append barcodes from the iReport palette. When the barcode icon is drag-and-dropped in the template area a Barcode wizard is launched. Barcode can be provided by one of two available frameworks: Barbecue and Barcode4J. After testing several barcodes from both frameworks I can conclude that the Barcode 4J is a preferred framework, because the barcode types that it features are more widely used. To enable Barcode4J functionality, a framework specific libraries need to be appended. It is worth to mention that when the required libraries are downloaded and added to the project the libraries folder in the project tree is featuring 39 Barcode4J specific jar files. It is also worth to mention that both jFreeChart and Barcode4J require AppacheCommons framework to be installed to function.

When executed, the MHG_ORIG_APP_ICEFACES web application generated a report shown in the figure 4.14. In that version of the reports the available database data didn't allow to create a 3D chart featuring several columns, though the pie chart works perfectly. After modification of the test database this issue was fixed. It is also worth of notice that this version of template featured full internationalization.

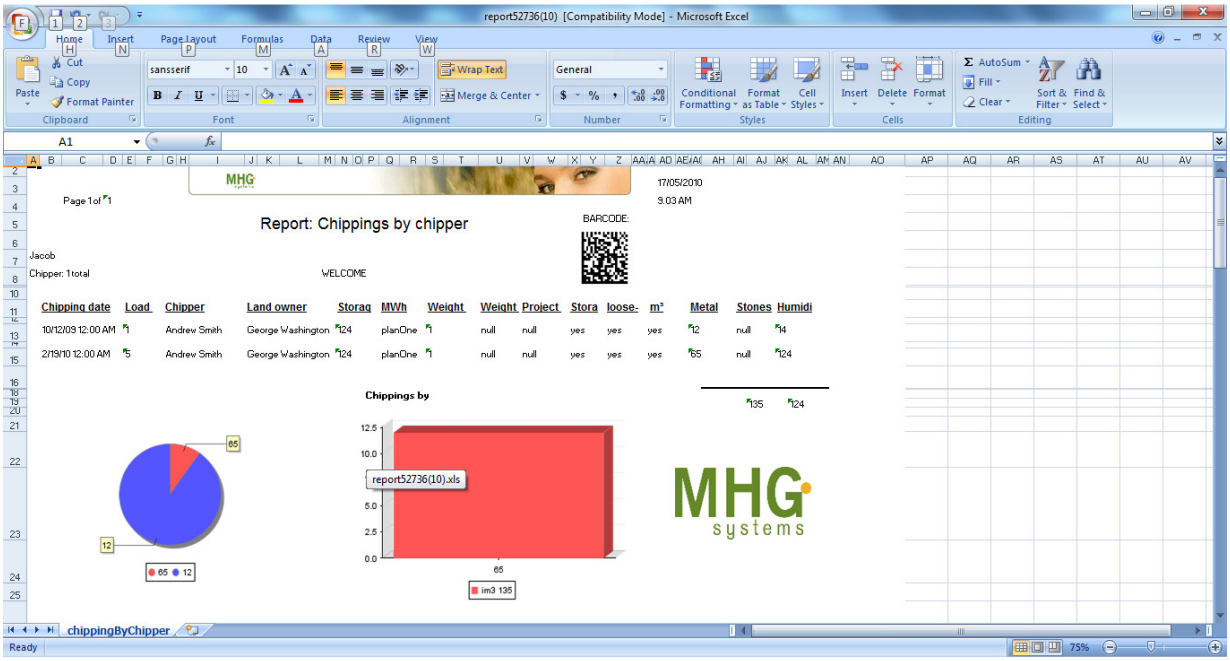


Figure 4.14. ChippingsByChipper report viewed in Microsoft Excel

4.4. Enabling advanced functionality

Though the returned report shown in figure 4.13. resembles the structure and the table design of the initial JXLS report there is a large difference between them. The difference is discussed in the context of the ChippingsByChipper report. In the original report the list of the chipping machines is provided to the report via an ArrayList and evaluated by the template. JXLS template is using conditional operators to navigate through the list. The fields of each Machine JavaBean object are used in dynamic SQL query and other parts of a report. In this way for each machine there is a separate table created. The final report can feature as many tables as there are machine objects provided.

JasperReports framework features several technologies to deal with the need to extend the template functionality. Three of the technologies: grouping, Scriptlets and Subreports were evaluated as the means for enabling the multiple tables functionality.

JasperReports allows to dynamically group the data returned from SQL query in a logical manner. To enable this functionality *group* element needs to be added to the JRXML template. Group element includes the groupExpression element that holds the expression used to logically group the data. Group element can also contain two additional elements: groupHeader and groupFooter to add some text or images in the beginning and at the end of each group. The main idea behind data grouping as it is implemented in JasperReports is that SQL query can return tables where several rows have the same value in one of the columns. This can happen when database employs one-to-many table relationships, so one value in a table can be associated with several values in another table. The grouping functionality of JasperReports allows to group the rows based on the value of such columns. GroupExpression is used to evaluate the need to add a new group to the report. The new group is created only when the value of the groupExpression changes and groupExpression needs to include a reference to a field returned from the execution of the query. The field reference is needed because JasperReports framework needs to decide which rows to include to a specific group by matching the field value to the according value in the returned row. The field value is iterated automatically, in the same way as rows are iterated in the detail column, so there is no means to control the order of the groups.

The major elements in JRXML template can be assigned a printWhenExpression. This expression returns Boolean true or false, and, based on the outcome, can either include or exclude the element from the final filled report. Combined with the grouping and several modifications of the SQL query I was able to implement ChippingsByChipper report

functionality. The problem is that such reports would be completely inefficient. If the user wants to see a report about, for example, three machines, the SQL query will return a large table containing all the machines that this user is able to access and the rows corresponding to the machines that were not selected by the user will not be added to the filled report based on the evaluation of printWhenExpression. Such sort of situation would be acceptable if I was doing a test project, but the aim of the thesis was to show the ability of JasperReports framework to replace JXLS as a reporting framework for real life ERP system. In some situations user can be able to access thousands of machines, so the database will have to return enormous tables, and each row would have to be evaluated and added to a specific group during the process of template filling and only groups corresponding to the machines that were selected by the user will be added to the ready report, with the rest of the groups discarded. Though, it would be possible to increase the efficiency by supplying large chunks of the SQL query generated in, for example, ReportServlet to the template via the parameters. But such approach of the template generation was dismissed, because JRXML template can include only one SQL query per template, several of the required reports can not be implemented with such approach.

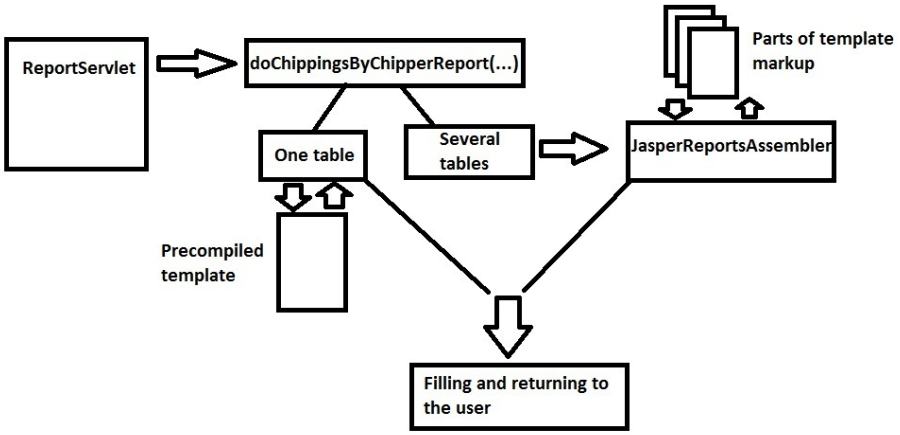
In JasperReports framework Scriptlets are chunks of Java code implemented in a separate Java class that can be executed at specific moments during the filling process of the reports. Scriptlets can not interfere or change the sequence of template filling. Scriptlets are able to access fields and parameters of the report, but are not able to modify them, though Scriptlets are able to access and modify variables. Because of such limited functionality Scriptlets were not used in the project.

JRXML template can be included into another template as a subreport element. Such approach was designed to simplify the design of complicated reports, so such reports can be divided into several templates combined with each other at the filling time. Every JRXML template can be included as a subreport element in another template if all the required parameters are provided. If subreport is a dynamic template HashMap parameters and datasource need to be provided to it in order to be filled. The subreport tag in JRXML contains several tags in order to pass enough information to subreport so it can be filled. The parameters can be passed to the subreport by subreportParameter tag which also includes subreportParameterExpression tag where, for example superreport parameters can be referenced. There need to be as many subreportParameter tags as there are parameters defined in subreport. In order to pass the datasource the connectionExpression needs to be defined, and the location of the subreport is specified in subreportExpression.

The subreports functionality seemed to be a suitable option for the project. To test the subreport functionality, the SQL query and the table generation markup was transferred to the subreport template named CBCsub.jrxml. The superreport ChippingsByChipper.jrxml contained only header and summary parts of the report. During the execution of the template an exception was thrown because there was no SQL query in the superreport. So, in order to execute the report the simple SQL query was added to the superreport though returned fields it returned were not used.

The generated report, though implemented in a different way looked the same as the normal report it was derived from, shown on image 4.14. So, the subreports functionality allows execution of several SQL queries during the generation process of the report. The only issue left to solve was the dynamic addition of variable amount of subreports to the generated report. The first approach was to modify subreport to recursively include itself if some condition would be met. But, because there is no way to include any conditional operators to the template such approach can not be implemented. There were no information about such JasperReports functionality available in either of existing books dedicated to JasperReports framework: JasperReports for Java developers and Definitive guide to JasperReports. There were also no information about how to deal with the need to dynamically control the number of subreports available on the internet, so I was solving this problem by reading the specified books and experimenting with the framework.

The first most obvious way to solve the problem was to generate the superreport templates dynamically, so that the evaluation logic can be moved to Java class. This approach was developed in a serious of projects named MHG_Jasper in versions from 0.1 to 0.3. This projects were had similar classes to the MHG_ORIG_APP_ICEFACES4 that was the latest and fully functional project able to generate ChippingsByChipper reports. In order to programmatically generate the superreport, the JasperReportAssembler class was created. This class was designed to load pre-defined parts of the ChippingsByChipper reports template from three text files, and to add code to the file as an output stream. The code was generated and refined in a form of strings. The figure 4.15 shows the structure of MHG_JASPER V0.3 Application.



.

Figure 4.15. MHG_JASPER V0.3 application workflow

There were done some considerable modifications to the ReportGenerator class. So, the method that would generate the ChippingsByChipper reports would now evaluate the number of parameters. If there was a need to be only a single table, the method would load predefined JRXML template and then it will be filled and returned to the user. If there was a need to generate several tables in the report, the method would be calling the generateCBCTemplate method in JasperReportAssembler which is calling the internal methods to generate the template. In general, though this approach is fully functional, it is an example of an overly complex implementation. The JasperReportAssembler is has a lot of embedded jrxml markup, so it is extremely hard to maintain or modify. In general this class is a suitable way to embed the Java logic into a jrxml template, but though it was able to generate only ChippingsByChipper templates, the class already included 200 lines of code. So, if the other 12 templates were implemented in a similar manner the class would be extremely complicated, unless every template will have its own class implementing the required logic.

There were several other possibilities evaluated, but none of them was suitable. In order to implement the required functionality, conditional expressions were needed to be used. Though there are no conditional expressions available in JasperReports templates, the underlying framework is probably using them in order to perform such functionality as iteration through the fields, addition and removal of variables and so on. The main idea of the final implementation of the program was to indirectly manipulate the conditional expressions that are hidden inside the JasperReports framework. This could be accomplished by manipulating the data and providing each part of the report with a separate amount of parameters and a special datasource. There are two features of the reporting framework that

made such approach possible: the ability to pass the *unused datasource* to the subreport and the JasperReports support of *different types of datasources*. It is possible to pass an SQL connection to the subreport, if it was declared as a parameter in the superreport and even if no query was executed on it in the superreport. This feature provided subreport with the required datasource. JasperReports framework is able to support multiple datasources. The idea behind a datasource for the superreport was the need to provide the superreport template with a datasource that will define the number of tables to be generated and extra values for each table, so the report will be able to iterate through the datasource automatically, adding subreports as rows are added to the detail element. So, two Hash Map Objects are passed to the template: Hash Map of objects as a datasource for the superreport, and a HashMap of parameters with Connection object included to be passed to the subreport during filling process. The both Hash Maps are created in the ReportServlet class. So when superreport is iterating through the Datasource Map, each time iteration happens the subreport is added as part of the detail element. Subreport code is placed in the detail element, because detail elements and markup that it includes are added as a result of datasource iteration. The current value of the Map is passed to the subreport as one of the initial parameters, so it is able to use this value in creation of a dynamic SQL query. When the query is executed, the table is generated and added to the report. This approach allows to use pre-defined templates and to allow the use of the same templates no matter how much of the tables the report will include, so every type of the 13 report types required by the ERP can be implemented. Such approach was designed during the implementation of the final thesis and is the most practical way to implement the advanced JXLS functionality in JasperReports framework. This method would be helpful to anyone who is trying to implement extremely dynamic and complicated reports in JasperReports framework.

This approach can be implemented following a set of strict rules and specific logic. First of all, each part of the report plays one of the two available roles: either it is used to allow the addition of extra subreports or it is used to query and represent data. Second, the addition of new subreports is accomplished by the standard iteration of the specific datasources. And third, all datasources, except the superreport datasource are passed to the template as parameters of the HashMap. The development of this approach is the main finding of this final thesis. From my current knowledge, this method was not described in any publicly available sources or used in any available examples. The developed method is a workaround addressing the problem of the lack of conditional expressions in JasperReports templates. The method was designed based on publicly available information and can be used

when there is a need to convert advanced JXLS templates to the JRXML templates. Figure 4.16 shows the use of the method when several levels of tables need to be dynamically added in the report. The *Add Subreports* subreport is including the subreports used to represent data into the final report. If the *Data Representation* subreports need to include some additional tables, they should first reference the Add Subreports subreport which will perform the addition of the subreports. The figure also shows the difference of the datasources for two types of the subreports.

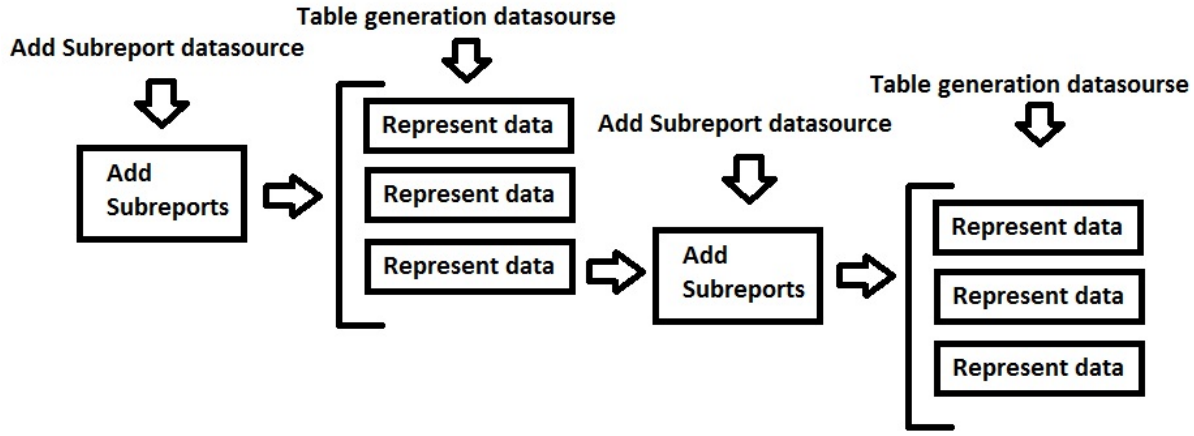


Figure 4.16. The dynamic addition of variable amount of subreports.

In the final implementation of the application named MHG_JASPER V0.5 several changes were made compared to the previous implementations of the MHG ERP Reporting module. First of all, there was a drop down menu added to the reports.jsp file, so the user is able to choose the format of the generated report. The available formats include XLS, PDF and HTML. In order to generate reports in HTML format the returnHtmlReport was added to the ReportGenerator class. This method is using JasperReports framework to convert the Jasper Print object into an HTML markup. Once selected, the HTML version of the report can be viewed in a separate window of the browser. To allow the representation of the images, several JasperReports standard Servlets were declared in web.xml file of the application. The structure of the final MHG_JASPER V0.5 Web Application is shown on the Figure 4.17. There are a few changes compared to the initial MHG ERP Reporting module architecture. The InitServlet is used to initialize value, the IceFaces JSP page is used to provide application with extra parameters, ReportServlet is loading the .jasper templates and forwarding it to the ReportGenerator class, which fills the reports using database and forwards the ready XLS, PDF or HTML report to the ReportServlet which sends the ready report to the user.

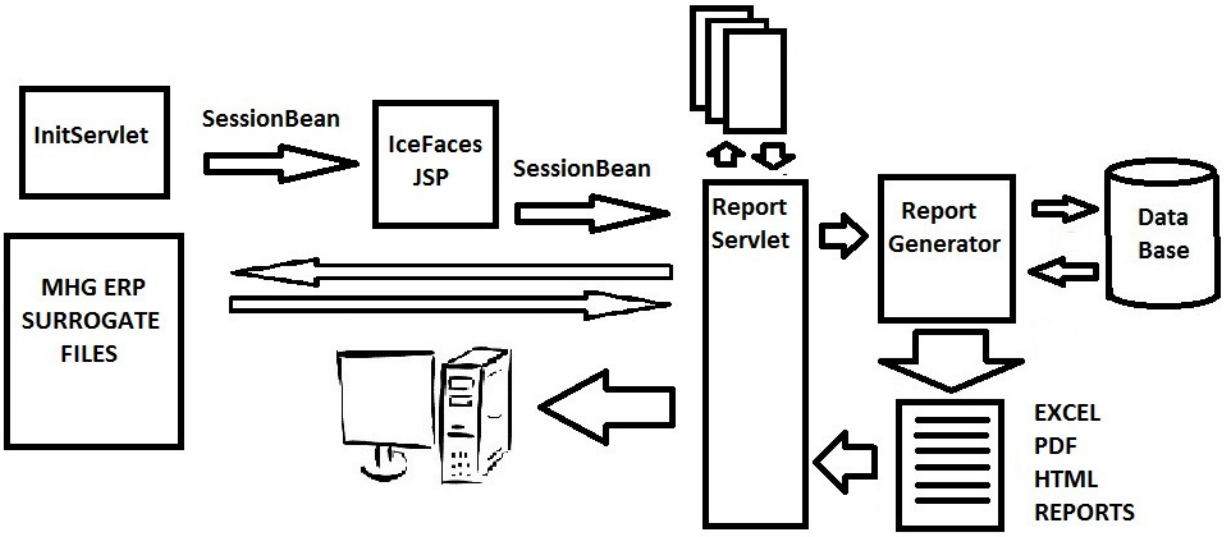


Figure 4.17. Structure of MHG_JASPER V0.5.

Except the extended functionality the templates are featuring some changes in the design. The design concept is based on several JasperReports standard templates that come bundled with iReport UI editor. The new design looks more modern and attractive. Figure 4.18 is featuring the latest, fully functional version of ChippingsByChipper Excel report. There was a small issue concerning the colors of the report, because Excel does support only limited amount of colors. In order to improve Excel compatibility the standard green was used as a main color of the report design. The template can be used to generate reports with variable amount of tables. Before the final integration to the MHG ERP System, the template will be tested with real MHG ERP Database.

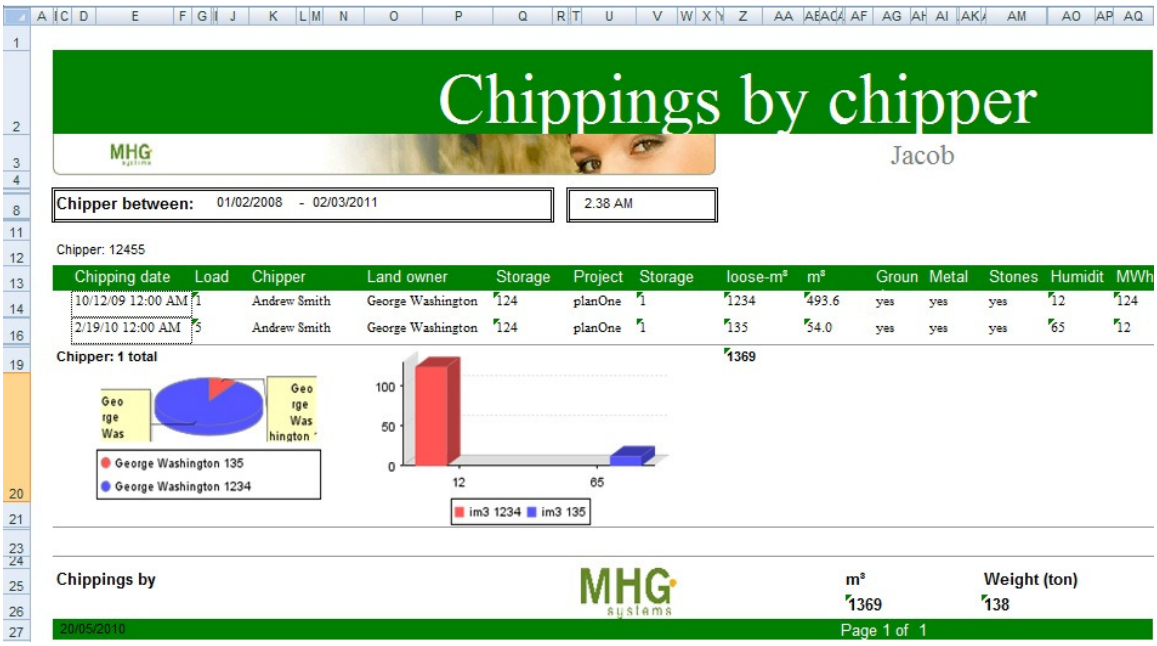


Figure 4.19. Final version of ChippingsByChipper Excel report

5 CONCLUSION

The aim of the thesis was to showcase the possibility to build fully functional MHG ERP Reporting module based on JasperReports framework. The original module, based on JXLS framework had some reliability and usability issues and the change of the framework was seen as a way to fix those problems of the module. During the development of the module I have discovered and implemented several extra enhancements to the original reports like the improved design featuring images and the ability to graphically represent the data through using charts.

When I started implementation of the project, I had no previous experience with server side technologies. During the development process, I have learned the types of computer architecture, the types of clients and servers, ways to build and use databases. Also I have learned to develop Java Enterprise Edition applications featuring Servlets, JSP, JSF and IceFaces frameworks. An important part of the learning process was the learning of the Java Excel report generation frameworks: the JXLS and JasperReports. The most advanced features of both frameworks were used, to build scalable, highly dynamic and sophisticated reports. In general this was the largest and most complicated project that I have undertaken during my programming experience. And the most important experience was to deal with real life server side applications, and the tough requirements in terms of functionality, reliability and efficiency.

The MHG_JASPER V0.5 is by far the largest application that I have developed. The code is over 3000 lines long, so if it was appended to this thesis, the number of pages required to print it would more than double. If all the other versions of the program developed during the project implementation were counted, the number of lines would also easily double. An important part of the thesis was the development of the test database that holds 24 tables.

Another important issue of the project was the time management and implementation planning. I have been following the Spiral Development paradigm, and the influence of this method can be clearly seen in the development sequence. I was researching, implementing new functionality in prototypes, learning and rethinking the concepts and then prototyping again. During the course of the project the 12 Web Application projects were developed in the NetBeans 6.5.1. IDE.

An important method of building JasperReports templates featuring variable amount of tables was developed during this project. This method is the most significant finding of the project. It can be useful for anyone who is facing the task of migrating from JXLS to JasperReports framework, especially when there is a need to keep the advanced functionality of the JXLS framework. The main principles of the conversion were defined to present the method in structural and easy to understand manner.

One of the important characteristics of the project is the confidentiality of the source code. In order to keep the confidentiality of the code that I got from MHG Systems and the code that was developed for the company I was describing the final implementation of the module in an abstract way. But in section 4.2. the underlying technologies such as JSPs and Servlets as well as methods of creation of different objects were discussed in great detail, so reader can have a general understanding of the principles on which this project is built. Though it is always a challenge to keep a line between the confidential and useful information, I was trying to write the final thesis in an easy to follow and clear to understand way, though the reader definitely needs some Object Oriented programming experience in order to fully understand the solutions to the problems, introduced by the task.

During the implementation of the project, I was closely cooperating with the company that this project was developed for. Every major iteration of the program was presented to the company staff, so I could get some feedback and the company would be aware of my progress. The members of the team are really satisfied with the success and some features of the implementation such as the possibility to add images and to create charts have exceeded expectations of the employer.

This final thesis is just a first step in the implementation of the MHG ERP Reporting module. After the first ChippingsByChipper report was fully functional, I gave the source code of the project to the company for further evaluation. Based on my implementation the final decision to switch to JasperReports framework was made. Right now I am working to implement the rest of the templates, and because of all the experience that I got with the JasperReports framework during this project, right now the full implementation of the Reporting module is just a matter of time. All the required technologies and methods have been learned and tested on the ChippingsByChipper template. In general, the JasperReports framework proved to be reliable, well designed, and, in the context of conditional operators, overly restrictive. But in general this framework is much more advanced and better implemented when the JXLS.

After the rest of the templates are implemented, module will undergo intensive testing, which is going to include the testing with the real database, as well as some extra black box and load testing. After the testing is complete, the module will be added to the MHG ERP systems and will help enterprises in Bioenergy field to be more effective, efficient and productive.

BIBLIOGRAPHY

Bergsten, Hans 2004, JavaServer Faces, O'Reilly, Sebastopol

Bidgoli, Hossein, 2004 The Internet Encyclopedia, Volume 1, John Wiley & Sons, Indianapolis

Converse, Tim el al, 2004 PHP5 and MySQL Bible, Willey Publishing, Indianapolis

DuBois, Paul, 2002 MySQL Cookbook**,** O'Reilly, Sebastopol

Englander, Robert, 1997 Developing Java Beans, O'Reilly, Sebastopol

Eschen, Rainer 2009, ICEfaces 1.8 Next Generation Enterprise Web Development, Packt publishing, Birmingham

Farley, Jim & Crawford, William 2004, Java Enterprise in a Nutshell, O'Reilly, Sebastopol

Heffelfinger, David R. 2006 , JasperReports for Java Developers, Packt publishing, Birmingham

Kurzweil, Ray 2005, Singularity is near. When humans transcend biology, Viking Penguin, London

Laurie, Ben & Laurie, Peter, 2002 Apache: The Definitive Guide, O'Reilly, Sebastopol

Matthews, Mark el al, **2003** MySQL and Java Developer's Guide, Willey Publishing, Indianapolis

McGovern, James el al, 2003 Java 2 Enterprise Edition 1.4 Bible, Wiley Publishing, Inc., Indianapolis

Mukhar, Kevin & Zelenak, Chris 2004, Beginning Java EE 5: From Novice to Professional, Apress, New York

O'Donahue, John 2002, Java Database Programming Bible, John Wiley & Sons, Indianapolis

Oaks, Scott & Wong, Henry 2004, Java Threads, Third Edition, O'Reilly Media

Reese, George 2001, Database Programming with JDBC and Java, Second Edition, O'Reilly & Associates, Sebastopol

Thomas M., Todd 2002, Java Data Access—JDBC, JNDI, and JAXP, M&T Books, New York

Topley, Kim 2002, J2ME in a Nutshell, O'Reilly, Sebastopol

Wadia, Zabin el al, 2008, The Definitive Guide to Apache MyFaces and Facelets, Apress, New York

Walkenbach, John 2007, Excel 2007 Bible, Wiley Publishing, Indianapolis

Yadav, Subhash Chandra & Singh, Sanjay Kumer, 2009 An Introduction to Client/Server Computing, New Age International, New Delhi

Electronic sources

Apache POI project by Andrew C. Oliver, el al. [referred 30.04.2010]
Available in www-format:
<URL: http://poi.apache.org/>.

April 2010 Web Server Survey by NetCraft Updated 07.01.2010 [referred 17.3.2010].
Available in www-format:
<URL: http://news.netcraft.com/archives/web_server_survey.html>.

Ajax push by IceSoft [referred 30.4.2010]
Available in www-format:
<URL:http://www.icefaces.org/main/ajax-java/ajaxpush.iface>

Boehm, Barry 1988, A Spiral Model of Software Development and Enhancement, Computer May 1988, 61-72
Available in pdf-format:
<URL: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/spiral.pdf>

Compaq Active Answers, 2000 Linux Web Solution with Apache, PHP, MySQL, and ht://Dig, Available in pdf-format:
web.deu.edu.tr/doc/misc/Linux_Web_Solution_php_mySql_Apache.pdf

Connecting to a MySQL Database [referred on 16.05.2010]
Available in www-format
<URL http://netbeans.org/kb/docs/ide/mysql.html>

Java EE at Glance by Oracle Corporation [referred 24.4.2010].
Available in www-format:
<URL: http://java.sun.com/javaee/>.

Java Excel API - A Java API to read, write, and modify Excel spreadsheets [referred 30.04.2010]

Available in www-format:

<URL: http://jexcelapi.sourceforge.net/>.

JXLS by JXLS team [referred 30.04.2010]

Available in www-format:

<URL: http://jxls.sourceforge.net/>.

Netbeans 6.8 features [referred 13.05.2010]

Avaliable in www-format

URL< http://netbeans.org/features/all.html>

Pelegri-Llopart, Eduardo el al, 2007 GlassFish Overview, Updated 06.2007

Available in pdf-format:

<URL: https://glassfish.dev.java.net/faq/v2/GlassFishOverview.pdf>

Spreadsheet tutorial [referred 30.04.2010].

Available in www-format:

<URL: http://people.usd.edu/~bwjames/tut/excel/1.html>.

Top 10 Largest Databases in the World by Business intelligence lowdown [referred 30.04.2010]

Avaliable in www-format

<URL: http://www.businessintelligencelowdown.com/2007/02/top_10_largest_.html>

APPENDIX A

JRXML file used in MHG_JASPER_TEST technology demonstrator application

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="MHG TEST REPOR"
pageWidth="1024" pageHeight="570" columnWidth="984" leftMargin="20"
rightMargin="20" topMargin="20" bottomMargin="20">
        <property name="ireport.zoom" value="1.0"/>
        <property name="ireport.x" value="0"/>
        <property name="ireport.y" value="0"/>
        <style name="report3T" isDefault="false" hAlign="Center"/>
        <queryString>
                <![CDATA[SELECT NUM, NAME, SURNAME,CITY, STREET,
ADDRESS FROM DATATABLE]]></queryString>
<field name="NUM" class="java.lang.Integer">
                <fieldDescription><![CDATA[]]></fieldDescription></field>
<field name="NAME" class="java.lang.String">
                <fieldDescription><![CDATA[]]></fieldDescription></field>
<field name="SURNAME" class="java.lang.String">
                <fieldDescription><![CDATA[]]></fieldDescription></field>
<field name="CITY" class="java.lang.String">
                <fieldDescription><![CDATA[]]></fieldDescription></field>
<field name="STREET" class="java.lang.String">
                <fieldDescription><![CDATA[]]></fieldDescription></field>
<field name="ADDRESS" class="java.lang.String">
                <fieldDescription><![CDATA[]]></fieldDescription></field>
<background><band splitType="Stretch"/></background>
<title>
<band height="130" splitType="Stretch">
                <staticText>
                <reportElement mode="Opaque" x="50" y="109" width="69" height="20"
backcolor="#CCCCCC"/>
<textElement textAlignment="Right"/><text><![CDATA[NUM]]></text></staticText>
```

<staticText><reportElement x="0" y="55" width="621" height="35"/><textElement
textAlignment="Center"><font size="26" isBold="true"/></textElement>
<text><![CDATA[MHG ERP JASPER REPORTS TEST REPORT]]></text></staticText>
<staticText><reportElement mode="Opaque" x="119" y="109" width="100" height="20"
backcolor="#CCCCCC"/><textElement textAlignment="Right"/>
<text><![CDATA[NAME]]></text></staticText><staticText>
<reportElement mode="Opaque" x="219" y="109" width="80" height="20"
backcolor="#CCCCCC"/><textElement textAlignment="Right"/>
<text><![CDATA[SURNAME]]></text></staticText>
<staticText><reportElement mode="Opaque" x="299" y="109" width="110" height="20"
backcolor="#CCCCCC"/><textElement textAlignment="Right"/>
<text><![CDATA[CITY]]></text></staticText><staticText>
<reportElement mode="Opaque" x="409" y="109" width="101" height="20"
backcolor="#CCCCCC"/><textElement textAlignment="Right"/>
<text><![CDATA[STREET]]></text></staticText><staticText>
<reportElement mode="Opaque" x="510" y="109" width="122" height="20"
backcolor="#CCCCCC"/><textElement textAlignment="Right"/>
<text><![CDATA[ADDRESS]]></text></staticText><image>
<reportElement x="0" y="0" width="950" height="55"/>
<imageExpression
class="java.lang.String"><![CDATA["C:\\Users\\Ruslan\\Desktop\\MHG_IMAGES\\up.gif"]
]></imageExpression></image>
<staticText><reportElement x="621" y="55" width="100" height="20"/><textElement
textAlignment="Right"><font isBold="true"/></textElement>
<text><![CDATA[DATE:]]></text></staticText>
<staticText><reportElement x="621" y="75" width="100" height="15"/>
<textElement textAlignment="Right"><font isBold="true"/></textElement>
<text><![CDATA[TIME:]]></text></staticText>
<textField pattern="K:mm a, z"><reportElement x="721" y="75" width="110"
height="15"/><textElement/>
<textFieldExpression class="java.util.Date"><![CDATA[new
java.util.Date()]]></textFieldExpression></textField><textField pattern="EEEEE dd
MMMMM yyyy"><reportElement x="721" y="55" width="157" height="20"/>

```
<textElement/><textFieldExpression class="java.util.Date"><![CDATA[new
java.util.Date()]]></textFieldExpression></textField>
</band></title>
<pageHeader>
<band splitType="Stretch"/></pageHeader>
<columnHeader><band splitType="Stretch"/></columnHeader>
<detail><band height="20" splitType="Stretch">
<textField><reportElement x="219" y="0" width="80" height="20"/>
<textElement textAlignment="Right"/>
<textFieldExpression
class="java.lang.String"><![CDATA[$F{SURNAME}]]></textFieldExpression>
</textField>
<textField><reportElement x="299" y="0" width="110" height="20"/>
<textElement textAlignment="Right"/>
<textFieldExpression
class="java.lang.String"><![CDATA[$F{CITY}]]></textFieldExpression></textField>
<textField><reportElement x="409" y="0" width="101" height="20"/>
<textElement textAlignment="Right"/>
<textFieldExpression
class="java.lang.String"><![CDATA[$F{STREET}]]></textFieldExpression></textField>
<textField><reportElement x="510" y="0" width="122" height="20"/>
<textElement textAlignment="Right"/>
<textFieldExpression
class="java.lang.String"><![CDATA[$F{ADDRESS}]]></textFieldExpression></textField>
<textField><reportElement x="119" y="0" width="100" height="20"/>
<textElement textAlignment="Right"/>
<textFieldExpression
class="java.lang.String"><![CDATA[$F{NAME}]]></textFieldExpression></textField>
<textField><reportElement x="50" y="0" width="69" height="20" backcolor="#FFFFFF"/>
<textElement textAlignment="Right"/>
<textFieldExpression
class="java.lang.Integer"><![CDATA[$F{NUM}]]></textFieldExpression></textField>
</band></detail>
<columnFooter><band splitType="Stretch"/></columnFooter>
```

<pageFooter><band height="101" splitType="Stretch">
<image><reportElement x="409" y="13" width="130" height="52"/>
<imageExpression
class="java.lang.String"><![CDATA["C:\\Users\\Ruslan\\Desktop\\MHG_IMAGES\\back.jpg"]]></imageExpression></image></band></pageFooter>
<summary><band splitType="Stretch"/></summary>
</jasperReport>