jamk.fi

Information system for measurement data

Marek Polakovič

Bachelor's thesis
May 2015

Software Engineering
School of Technology, Communication and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Polakovič, Marek	Bachelor's thesis	29.05.2015
		Language of publication: English
	Number of pages: 68	Permission for web publication: x

Title of publication
Information system for measurement data

Degree programme
Software Engineering

Tutor(s)
Esa Salmikangas

Assigned by
Faculty of Electrical Engineering, University of Žilina

Abstract

The main goal of this bachelor's thesis was to analyse and develop an information system for storing, managing and analysing transformer measurement data. The thesis focused on designing and developing multiple parts of an information system, such as data processing, reporting and charting part.

The system is intended as a framework of the project APVV-0703-10 - analysis and diagnostic measurements of power transformers using a "Sweep Frequency Response Analysis" at the University of Žilina. The information system is used for storing, processing, managing and analysing data obtained from modern measurement devices. Additionally, it would be able to generate final report as a result of the analysis.

The system was written in Java under Java Enterprise Edition, with PrimeFaces defining the user interface for web application and with many other Java frameworks and libraries. The system also supports REST services for processing, managing and analysing data. This enables future development for additional platforms (like Android, Windows Phone, etc.), however, the implementation of the REST services is outside the scope of this thesis.

The information system is still being developed and it has been presented to various people working with power transformers. Currently the system works in majority of web browsers.

Keywords/tags
Information system, Java, Java Enterprise Edition, transformer, measurement data, analysis, JasperReports, Highcharts, PrimeFaces, WildFly

Miscellaneous

Contents

Figures

Tables

Acronyms and glossary

Ajax

Also AJAX, asynchronous JavaScript and XML, group of client technologies for creating asynchronous web application

API

Application programming interface, set of routines, protocols and tools for building software applications

CRUD

Create Read Update Delete – basic skeleton of GUI for managing data in database

CSV

Comma Separated Value – specific file that contains columns and rows, each column is separated from others by comma

EJB

Enterprise JavaBeans, application logic implementation

EAV

Entity-Attribute-Value model is a data model to describe entities with their attributes. The final set of all attributes that can be used to describe a certain entity is potentially vast, however, the number of attributes that will apply to a given entity is relatively modest. (Wikipedia, 2015)

FTP

File Transfer Protocol – this protocol is used for transferring data between computers over the Internet

Framework

Universal, reusable software environment. This environment provides particular functionality as part of a larger software platform

IS

Abbreviation for Information System

Java

Computer programming language that is concurrent, class-based, object – oriented

Java EE

Java Enterprise Edition (JEE) – enterprise Java computing platform that provides an API and runtime environment for running and developing enterprise software, usually used for more complex projects

JSF

JavaServer Faces – JEE specification for building component-based user interfaces for web application

JSP

JavaServer Pages – technology that helps developers to create dynamically generated web pages

POJO

Plain Old Java Object – this type of java object contains only constructors, attributes and its getters and setters

PrimeFaces

Ajax framework based on JSF, extended API for JSF

SFRA

Sweep Frequency Response Analysis

1 Introduction

1.1 Assigner

This project based on a real request from the Faculty of Electrical Engineering at University of Žilina. The majority of activities are oriented into monitoring reliability and quality, study reconfiguring circuits to computers, control of quality and reliability according to IEC standards, application of programmable logical arrays and lastly diagnostic and analysis of failures and destructive analysis.

Some of the employees of this faculty monitor and control different types of electronic systems, components and materials. These activities are mostly performed in the field of electronics and electro technology.

1.2 Topic introduction

Generally, there are several problems during the development of an information system. One of these problems is how to store data efficiently. Every second modern electronic devices produce hundreds of bytes of data. These data have to be processed and stored properly. The analysis of stored data can contain invaluable information provided that everything was performed properly. Big data is the term describing the problem of efficient storage and analysis of huge amounts of data. (Zikopoulos, et al., 2013). Some of the new techniques of data storage and processing were used to cope with this problem. The following chapter explains why this problem was mentioned.

1.3 Objective of thesis

The goal of this thesis was to design and develop new information system (some parts of it) that uses the obtained data for storing, managing, reporting and analysing. This bachelor’s thesis based on the project APVV-0703-10 - analysis and diagnostic measurements of power transformers using a "Sweep Frequency Response Analysis." As can be seen from the project name, the data are obtained as a measurement of power transformer. Measurement data were non-organized, not well stored and lastly, there was a need for creating a system that supports at least these activities: to process and store data, compare data among measurements and generate final report. So far all of equipment

operators of measurements and analysts have to store their measurement data manually in a folder structure, and also they are not able to process the analysis on them or share them easily. Therefore, the purpose of the thesis was to improve these processes.

1.4 Outline of thesis

The thesis is divided to the several parts or chapters. Chapter one contains brief information about the assigner, introduction to the project and analysis of current state.

Chapter two contains the analysis and gathering the requirements with the theoretical background. This analysis and theoretical background are necessary in order to properly understand the topic.

The third chapter describes the whole architecture of this information system with information about the used technologies. All content in this chapter is aimed at the design parts and for giving the answers for questions such as what was necessary to use, design, what were the decisions etc.

Chapter four is aimed at the implementation phase from a practical view. In this chapter all newly created processes are described from the user and developer point of view.

The last two chapters present confirmation of the results with the testing part, conclusions and a short self-assessment.

2 Analysis and requirements gathering

All necessary information needed for better understanding of the whole process of gathering requirements or analysis phase can be found in this chapter. During the whole project lifecycle each step was consulted with experts in electro-technology (technicians) and also these experts set up basic requirements that should be considered in the solution.

2.1 Gathering of requirements

The current process of managing transformer measurements does not have any dedicated software to support it. The process that is described below is the result of several meetings with the host company.

Firstly, it was necessary to identify all interested parties. During the analytical phase two main roles of the information system were identified. People who belong to these groups are included in obtaining, managing, analyzing and distributing measurements.

Concretely, these roles were identified:

1. Technicians
    a. measurement equipment operator – a person, who operates the measurement device during measurement
    b. analyst – a person, who views and reviews measurement data, performs data measurement analysis and finally generates reports that contain the analysis results
    c. administrator – a person, who takes care of the system by back-upping all data, database, etc., who has a right to perform administrative tasks e.g. user roles administration, roles administration etc.
2. customers – a person, who reads, archives and uses final reports.

The first group can be divided into three sub-groups, so a common name for these sub-groups is *Technicians*. These sub-groups represent more specific roles, as described before, in this whole process.

Measurement equipment operators are responsible for obtaining the data from the transformer. It means that they have to get to the transformer and obtain the data using measurement device. This way, they get data from each transformer is planned for measurement and they come back to the office. Then they download the data from the measuring device in CSV format to their own computers. Now the files are ready to be processed by an analyst. (Grondžák, Grigeľ, Polakovič, & Remenec, 2014)

Firstly all necessary files that are needed for analysis must be downloaded/sent to user's computer (for example via email, FTP or USB drive). These files are opened using a specified software able to perform the SFRA analysis and draw charts based on the input data. Checking the result and exporting the printed charts into some image format (JPEG, PNG, etc.) are carried out manually by the analyst. After this phase a final report can be created using some kind of word processor (for instance Word or TeX). After that this document is sent by e-mail to SSE (consumer of reports, from application scope it is customer). The whole process is sometimes time consuming and has several issues that could happen.

During this process several problems were found out. Data are stored in CSV files on multiple computers. File sets on different computers do not have to match, so certain data can be unreachable from one device. On the other hand, these files can be easily lost. Also it is almost impossible to search or filter this data. Furthermore, the creation of reports is manual and every time a report is created, the technician has to copy the data, results and charts into the text processor.

Therefore, the goal for this system is to centralize data storage and analysis and every technician and consumer can access the data easily from everywhere (no matter if using tablet, mobile phone, desktop or laptop). It is also important to automate the report creation and transformer measurement management.

Following sections describe and explain how these goals were achieved.

2.2 Analysis

During the whole project lifecycle each step was consulted with experts in electro-technology who work with transformers and perform the analysis and measurements on them. The host institution clarified us their ideas and hinted how the application should look like, what kind of features should be included and how the analysis should be performed.

One result of the analyse is based on the fact that the best solution for having a centralized data storage and analysis is to use a client server architecture with a centralized database server. This decision was made when by discussing and considering several facts, such as that the application should be accessible to everybody and the database should be available from all parts of the world. Also the same data should be shown to all users in real time.

The analysis of the requirements also resulted in an identification of several subsystems (user's activities) of use cases that are logically or functionally interconnected. Each subsystem represents an atomic task which the system should be able to perform. The use-case diagram of the proposed system design is shown in Figure 1. The assignment of roles to the activities is illustrated in the diagram. (Grondžák, Grigeľ, Polakovič, & Remenec, 2014). Seven subsystems were identified. Next these subsystems are described in more detail.

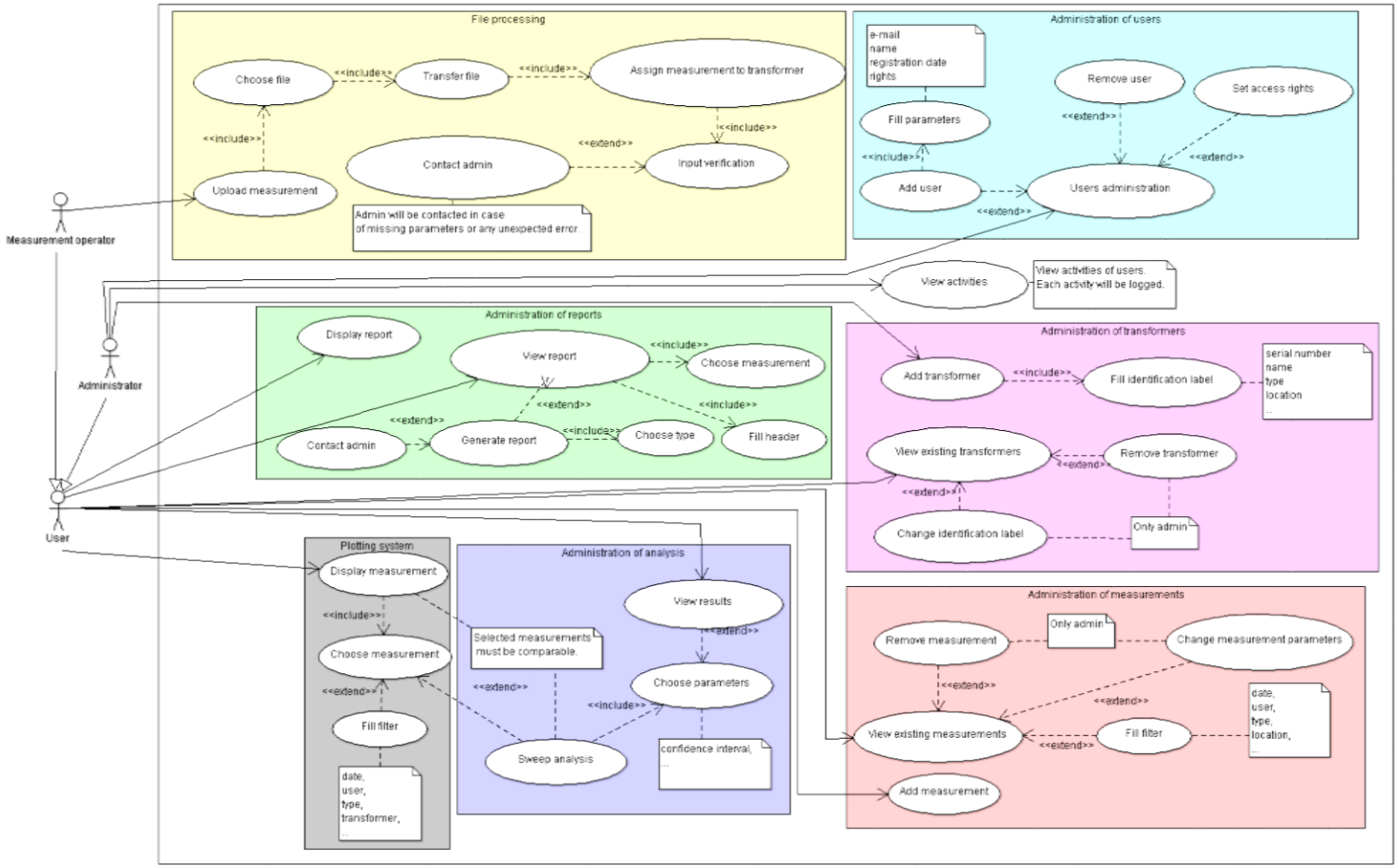


Figure 1. Use-case diagram of the information system design

Following subsystems were identificated:

- file processing – is the core subsystem, allows users to upload date in CSV files into the database,
- administration of measurement – the most important subsystem, allows the user to view, edit and delete measurement according to measured data,
- administration of analysis – provides an implementation of algorithms for data analysis, for instance SFRA,
- plotting subsystem – allows the user to view measured data in the graphical form,
- administration of reports – enables creation of different forms of reports,
- administration of transformers – allows to manage static information about tranformers like their charakteristics, location, type, etc., and
- administration of users – enables managing users (creating, deleteting, changing password, user role assignment, etc.).

Next the focus in on the identificated subsystems. In the following paragraphs a deeper description of these subsystems can be found. The next part aims at describing the subsystems that are related to the goals of this paper.

Subsystem – File processing

This is the first core module of the system. This part of the application should allow users to upload the CSV file obtained from measurement device to the server. Firstly, the file that contains measurement data should be serialized and subsequently sent via network to the server. When the file is successfully uploaded it should be processed by the server. It means that application can parse all obtained data and get the rest of data that is necessary for the application (for example name of columns, numeric waveform data, transformer serial number and other related data). Finally, these parsed data will be persisted into database. Generally this part allows user to add new measurements. More details about this subsystem will be described later in this paper.

Administration of measurements

This is another core module, because main task of this system is to store, archive and manage all measurements in the database and also because all uploaded measurements have to be manage by technicians. This subsystem allows user to remove measurements, edit certain properties of them, filter and search measurement by using some of the measurement properties.

Administration of analysis

This module is responsible for performing some type of analysis. One of this analysis could be Sweep Frequency Response Analysis method with parameters specified by user. Main tasks of this subsystem are: to perform implemented type of analysis using given parameters, show user the analysis results and indicate whether the transformer is fine or it malfunctioned.

Plotting subsystem

When a certain measurements in measurement manager will be selected, the data according to the selected records should be displayed in a graphical form (at least line chart). One part of this data is also a chart with curve representing transformer's measurement data. User should be able to save actual graphical image, work with charts like changing selected curves (one curve represents one column in measurement data), manipulating with zoom and so on. Also charts should be able to plot whole data in logarithmic scale. More details about this subsystem will be described later in this paper.

Administration of reports

When the system will had all necessary data and charts of the measurement it should be able to create final report. The system should create protocols automatically using data in the database. Of these, the application should allow user to choose which parts the report will contain and which not. More details about this subsystem will be described later in this paper.

Administration of transformers

A user uploads measurement data of a certain transformer, therefore it is necessary to have some transformers in the evidence of the system. When user uploads data, the system should try to determine which transformer it belongs to. This subsystem should allow a technician to edit information about an existing transformer, to remove it or to add a new one.

Administration of users

As mentioned before, there are two basic groups of users – common users and administrators. User manager provides CRUD (Create Read Update Delete) operations above them. This the way how a new technician can be added to the system can be edited information about existing ones. The application also provides the possibility to add a new administrator.

3 Architecture and design of application

In the previous chapters the basic concepts that may be met up in this work were successively described. The analysis was also made and described together with the gathering of requirements. Therefore, now an implementation platform for the application itself and also the whole application design can be described. All technologies described below are used in the application. There is no deep description how the technologies are used, however, there are the reasons and descriptions why these technologies are needed, how each part is interconnected and what was necessary to study or to explore.

3.1 Implementation platform for application

3.1.1 Introduction to implementation platform

The chosen implementation platform depends on the gathered requirements and performed analysis. All decision were based on results of a prior analysis, consultation with the representatives of the host company and the author's own reflections and ideas how the application should look like, perform the task and help to the end-user.

Today, several operation systems exist and that is why it is necessary for the application to be platform independent and to be supported on major systems. By this requirement, the application is runnable almost on each computer. Also, there are plans that the application can be extended in the future with several modules, so it is necessary to design a flexible, easy to develop or extend application.

That is why this system is designed as a web application with client – server architecture. The architecture client – server is well known, thus it is not described in this paper. There can be some questions what the benefits of this architecture are and why this information system is designed as a web application. The answer is simple. The application has to share all available data to all users and all data should be available from all parts of the world, which is another reason for web application There is a high possibility that a technician uses this application outside during performing measurements via a mobile device (such as a tablet), therefore this is the next reason why the application is designed as a client-server or web app.

By analysing the requirements regarding to the above-mentioned plans and considerations, several requirements that may affect the choice of implementation platforms were formulated:

- possibility to create intuitive GUI
- possibility to create multiplatform application
- possibility for easy and fast way how to developing new features with minimal knowledge
- type safety
- common available language
- possibility to generate reports, printing charts and processing plenty of data.

3.1.2 Choice of programming language

The final set of available solutions – platforms and programming languages was minimized by these base requirements. These programming platforms meet most of the criteria: Java EE or ASP .NET.

Both platforms have plenty of common pluses or minuses for the future application. Both of them are commonly available, provides several frameworks and APIs that can be freely used, for example the framework for object relation mapping, - ORM, generating reports and printing charts, etc. The development phase is quite easy whether using ASP or Java platform and both these languages are supported by several development environments. Visual Studio can be used for developing application under ASP and this environment helps us with automation and making some tasks faster. (Microsoft, ASP .NET) On the other hand NetBeans, IntelliJ or Eclipse can be used for developing an application under Java EE. (Microsoft, ADO .NET Overview MSDN)

ASP.NET

The major disadvantages of ASP platform is the weak support of several database systems or mechanism. There are two common frameworks for database manipulation: Entity framework and NHibernate. The entity framework works very

well with Microsoft SQL Server, however, several problems has occurred when database system, like PostgreSQL, MySQL, are used (Microsoft, ASP .NET) On the other hand there is NHibernate. This framework has many disadvantages for the thesis project. One of these minuses is a missing mechanism such as reverse engineering (creating model classes that contain access methods to the existing relation database model) or forward engineering (opposite process of reverse engineering), annotations and more. (NHibernate Community, 2014)

The last of the disadvantages for developing the application under .NET platform is insufficient multiplatform for deploying application on servers with another operation system than Windows. (Microsoft, ADO .NET Overview MSDN)

The decision was not to use ASP. NET as an implementation platform, because this platform has serious disadvantages for the project. The next candidate was Java EE. No kind of limitation and serious disadvantages were found in Java platform, so the decision was to use Java EE. Also, this platform suits very well for this project with all related frameworks and APIs.

3.1.3 Java EE

Java EE is a set of technologies that enables the development of large-scale information systems (like .NET, ASP, WCF etc.). This platform also includes API for accessing databases (ORM), viewing web pages, web services implementation, and many other features. The great advantage of this platform is the amount of available open libraries for its expansion. Several technologies can be used in each layer for a specific problem. This platform is available with several ORM providers such as Hibernate, DataNucleus or Eclipse Link. In addition, working with the database is defined in Java Persistence API specification that state how to manage work with relational data (relational database) for the Java platform in general. Of these, Hibernate follows one of the referential specifications (EclipseLink JPA). The advantage of this platform is the support of different database systems, including NoSQL. (Gupta, 2013)

Nowadays, Java community is very strong, and a great deal of referential specifications to help the application to communicate better with several existing APIs

or components can be found. There are plenty of extending APIs that provides some useful technologies, frameworks and functionalities for certain tasks such as generating different types of documents, printing charts, ORM, or moreover, there is possibility to combine several additional libraries and components for creation of web GUI (PrimeFaces, GWT, Vaadin, etc.). Some of these frameworks, components and APIs are described later in this work.

The information about used technologies and frameworks that were used during implementation phase are described in the next chapters.

3.1.4 Apache Maven

Apache Maven is a tool for software project management and for automation building process. Maven was developed under Java platform and it is mostly used for Java projects. Maven is based on the concept of project object model (POM). All configuration are stored in one file, called pom.xml as was mentioned before. (Sonatype Company, 2008)

There are several goals that Maven is able to achieve. Maven is mainly used for these three tasks: defining dependencies, identifying project and defining the way how to build it. Another task was to introduce certain rules (best practices) for developing software, for example, it defines the directory structure of project, and to control the migration phase for new features or projects. (Tutorials Point, 2014) (Sonatype Company, 2008)

So far the mission of Maven was described, however, there is still one unanswered question: why should Maven be used for defining dependencies or for building process, if these possibilities are also included in several developers' environments. The answer for this question in described with a practical example below.

A situation is imagined that the user is going to develop a larger project (enterprise application, information system or expert system) with using many libraries for generating reports, using several types of charts, creating GUI and more.

If the user is not using Maven all these libraries (.jar files) should be added manually. This is possible on a small-scale project with less libraries (up to 5-10), however, what

about a project that will contain plenty of them (for example 20 and more). It is necessary to mention that the number of libraries can increase extremely for a few reasons. One of these reasons is also mentioned previously (need for a new library for some new feature). The other one is that libraries can have different versions and can have some other dependencies, thus, finally it is necessary to include these additional libraries also. Now it can be noted that these libraries need to be included only once and it is done, however, what about the situation when the project is developed by many more developers? Every library has to be delivered to each developer, then this library has to be linked/included into each copy of the project. As can be seen, there is still much work, even when this process can be fully automatized. It can be mentioned that all libraries can be stored with the code in some versioning system (e.g. Mercurial, GIT), however, this idea has also one major disadvantage, which is that if all libraries are pushed to the version control it can take a lot of storage. Now the reasons why it is necessary or good to use Maven are known, so now it can be described how these problems were solved using Apache Maven.

Maven solves all problems that were mentioned before. Maven is able to define all dependencies in one configuration file - pom.xml. In Maven terminology a more generic word for each dependency is artifact. The artifact is the output of the Maven build, generally it can be a jar, war or another executable file. Maven uses the artifacts for identifying certain dependencies for building and running the code. The design of the artifact can be seen in the Code example 1.

```
<project>
      ...
      <dependencies>
       ...
       <dependency>
                <groupId>net.sf.jasperreports</groupId>
                <artifactId>jasperreports</artifactId>
                <version>5.6.0</version>
       </dependency>
       …
      </dependencies>
      ...
</project>
```

Code example 1. Definition of Maven artifact

All artifacts are identified by three fields (fields are a part of maven coordinate system): *groupId*, *artifactId* and *version*. (Gupta, 2013)

All dependencies are defined between XML element dependencies. Element *groupId* contains the unique name of the artifact among all projects stored in the Maven repository. This name should follow at least one convention – it should be named in reverse domain name notation. The element *artifactId* is used to uniquely identify project name. The project name can be freely chosen by developers. The last element that was not mentioned yet is element version. This element contains the chosen version of the project. Any conventions for the version can be used, in this case there are no limitations. The defined dependency for the project that is named *jasperreports* can be seen in Code example 1. This dependency is identified by *net.sf.jasperreports* and version 5.6.0 is required. (Tutorials Point, 2014) This was an example of one dependency that is used in this project.

The process of obtaining dependencies needed to defined next. During the building process Maven is checking if the library can be obtained from the local Maven repository, if not this library is going to be downloaded from the remote (external, global) repository to the local repository. After this phase Maven copies all needed libraries to the project. If a library is already located in the local repository, Maven skips downloading the library and it just copies it to the project.

As mentioned before there are two types of repositories – local and remote. The local repository is stored in the computer in the working directory of Maven. The remote repository is stored on a server that provides this service. Global Maven repository can be found on webpage http://mvnrepository.com/, however, there are also some smaller repositories set up by a third parties. In this project one global repository and also some of the smaller ones are used. The definition of remote (not global) repository is shown in the Code example 2.

```
…
<repositories>
        <repository>
                <id>prime-repo</id>
                <name>PrimeFaces Maven Repository</name>
                 <url>http://repository.primefaces.org</url>
      </repository>
</repositories>
…
```

Code example 2. Definition of remote repository in Maven

The definition of repository contains three fields: *id*, *name* and *URL*. The field id is used for unique identification of a repository in the project. The name is used as an information field for developers. It helps developers to better identify what kind of repositories are used. The last field is URL. This field contains a direct link to the remote repository. The example illustrates that one remote repository is used and can be reached via repository.primefaces.org and it was named as PrimeFaces Maven Repository. (The Apache Software Foundation)

There is a possibility to include the project into remote Maven repository. To do this extra credentials to the remote repository are needed. This process is not described in this paper, however, it was mentioned before because it is necessary to identify the project, if Maven is used. The project is identified by filling in four elements into the project element. These elements are: *groupId*, *artifactId*, *version* and *packaging*. The purpose of these elements is the same as was described in dependency part. The undescribed element is packaging. This element can currently contain these values: pom, jar, maven-plugin, ejb, war, ear, rar or par. These values stand for packaging type. (The Apache Software Foundation)

So far, two main missions were described – defining dependencies and the identification of the project. Next, the last important mission of Maven can be described. This mission is about building and deploying. Maven enables to define several profiles for different situations. One profile can be used for building and deploying application on a local server, another one can be used for the same tasks on

a remote server, another can be used for running a test (e.g. Arquillian). Different scenarios, plugins, dependencies and configurations for each profile can be defined, if Maven is used. Also, Maven tries to make the deployment phase easier, because it enables to configure exec-maven-plugin for defining additional commands. These commands are then executed during the deployment phase.

The default building phase has eight phases. These phases are illustrated in Table 1.

Table 1. Default building phase

1	Validate	validate if all information is available (includes dependencies) and if the project is correct (no syntax errors, no missing dependencies, no errors in configuration files, …)
2	Compile	compiles the whole source code
3	Test	runs test for compiled source code by using a suitable unit testing plugin/framework
4	Package	takes all compiled code and packages it to the distributable format (defined in packaging element)
5	Integration test	deploys and runs integration test into environment where test can be run
6	Verify	runs checks if the package is valid
7	Install	installs the package into the local repository, for using as a dependency in other projects
8	Deploy	copies the final package to the server environment (The Apache Software Foundation)

The last thing that is necessary to state about Maven is that it is not a just tool for developers, it is also a tool for a person who is deploying (installing) the final project to the customer. What is necessary to do is to download and install Maven on the deploy machine, copy the code and run several commands, like mvn deploy. All necessary configuration files will be configured and libraries will be downloaded during this phase.

3.1.5 Manage bean

This is component architecture on the server side for creating modules. Each module encapsulates business logic and takes care of security and transactions. Several integrated APIs are found in container, such as dependency injection, component life cycle, JDBC, Java Persistent API (JPA), Remote Method Invocation (RMI), Java naming and Directory Interfaces (JNDI) and more. This is why these kind of Java Beans are used to build business layers (See Figure 2).

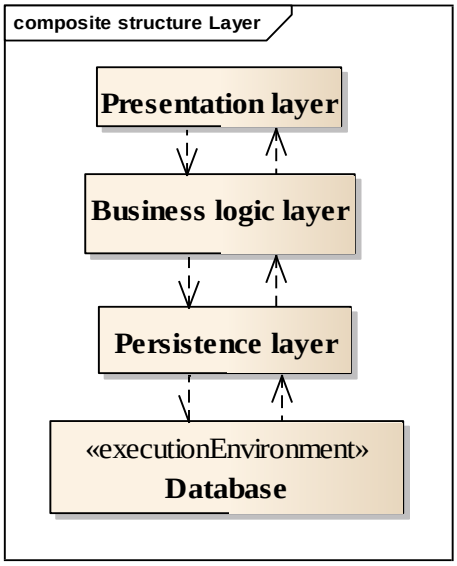


Figure 2. Layer architecture

Manage beans sit between the persistent layer and the presentation layer, which is an entry point for presentation-tier technologies like JSF (described later in the thesis). Nowadays manage beans are a very simple server-side development model for Java. They also bring scalability and reusability for an enterprise application. All this comes from annotating a single Plain Old Java Object (POJO) that is deployed in a container. Generally the bean is represented as Java class with additional annotations. (Goncalves, 2009)

Several types of containers containing beans (JSF, EJB, CDI…) can be found. These containers work independently. JSF, CDI and EJB Managed beans can be distinguished by the way how they are used and managed in a container. The main role of the container is to deal with all the technical plumbing, which means that

developers of business logic can concentrate on implementing the business logic. (Gupta, 2013) (Goncalves, 2009)

This project was developed by using CDI managed beans, which is why the next part is aimed at the CDI and not at the other ones.

3.1.6 Weld CDI

CDI Managed Beans contain a full and better-defined technology of managed bean components (EJB). Apart from the ability to inject and manage these components offers it offers more scopes, implementation of the various design patterns (Observer, interceptor, stereotype, decorator etc.), type-safe injection, events and similar. (Jboss by RedHat)

Weld CDI is a referential implementation of CDI (Context and Dependency Injection) for Java language to support dependency injection, management of beans and management of object's life cycle. CDI is just a specification and Weld CDI is the concrete implementation as mentioned earlier. Weld CDI was developed by RedHat during project JBoss. (Jboss by RedHat) (Gibson).

The main features of Weld CDI are implementation of several design patterns (e.g. Observer or Singleton), interceptors, events and several conversation scopes. For better understanding the whole application process it is necessary to explain several notations. If the Weld CDI bean from a JSF view is to be used, this can be achieved by

```
@Named("measurementUpload")
@ViewScoped
public class MeasurementUploadController implements
Serializable{
    @Inject
    private TransformerMatchService transformerMatchService;

    @PostConstruct
    public void initConversation(){
       // this code is executed after creation of this bean,
       // e.g. for initialize transformerMatchService and
for obtaining values
       }
}
```

Code example 3. Example of annotations in Weld CDI Bean

tagging the class with the *javax.inject.Named* annotation. If a bean is to be injected into another bean, the field is to be annotated with *javax.inject.Inject* annotation. On the other hand, all types of scopes used in the system should be familiar to the user. A base description can be found in the next paragraphs. A general example of using this annotation is shown in the Code example 3. (Gibson)

RequestScoped – exists until a request is not finished, it exists only for a very short period. All local attributes are initialized by calling a certain method. After finishing the request this bean is destroyed. It is mostly used for implementation of algorithms – a bean is created, input parameters are entered, processing is waited for, results are obtained and finally the bean is destroyed, because it is not needed anymore. (Finnigan, 2013)

SessionScoped – exists during the client session, each client has his own object, user's interaction across multiple HTTP request. (Rubinger, Lee, & Burke, 2010)

ApplicationScoped – exists during the whole life-cycle of an application, which means that an instance is created only once for the duration of the whole application. (Rubinger, Lee, & Burke, 2010)

ViewScoped – exists during viewing certain JSF views in the web browser. Usually it is used as a backing component for JSF front-end page. (Finnigan, 2013)

ConversationScoped – The developer can control the life-cycle of the object. This scope is very useful, when backing component is needed for more JSF views. (Gupta, 2013)

3.1.7 PrimeFaces (JSF)

JavaServer Faces (JSF) is a Java a specification for creating and building user interfaces for web applications, in other words, it is MVC web application framework. This technology is based on reusable UI components, so if creating a new web page several components can be used. JSF uses templating system and the default systems for JSF 2 are called **Facelets view**. The reference implementation of JSF is Mojarra. There are many other implementations of web frameworks such as MyFaces, Wicket and more. The JSF specification defines several predefined UI components, e.g.

selectOneMenu, selectManyMenu, dataTable, validation component, command button, etc., and provides a facility to interconnect widgets (client side) with data sources (server side) (Tutorials Point)

The process of creating a new webpage is described next. At first XML files that containing JSF tags for defining the webpage should be created. So the whole webpage is defined in one of several XML files. These XML files contain JSF components with other HTML tags or JavaScript. **Facelets** are mostly used for transformation of input XML files into JSF component tree. Basically the FacesServlet processes all requests, loads related view templates, and after that the component tree can be built and the response can be created (rendered) and sent to the client. Typically the response is in HTML language, however JavaScript can be also included, which means that user is able to see just HTML output and is not able to see JSF XML. (Tutorials Point)

Web view can be created dynamically, which can be proceeded using JSF or JSTL (JavaServer pages Standard Tag Library) tags. Several useful tags can be found in JSTL, for example tags for conditions (if), loops (foreach), catching exceptions (catch), formatting output (formatNumber, formatDate…), defining local variables (set, remove) and many others. Thus, finally a more complex web view can be created using these modern technologies. (Tutorials Point)

PrimeFaces is a web application framework for JSF based application, in other words, it is open source UI component library for JSF technology. This library contains many components mostly used in building GUI. Also, there is a possibility of using some of the predefined themes and templates. The webpage view can be built like a puzzle. This effect can be proceeded by using templates, because each part of webpage can be defined in separate files (e.g. one file for header, another for body and another for footer), so each component can be defined once and used through the whole application. At the end these files can be merged into one by using include tag. An example code is illustrated in the Code example 4. (PrimeTek)

Several development environments (e.g. NetBeans) provide some extra features, such as generating CRUD GUI interface with corresponding facades for managing certain entities (entities are stored in database system).

```
<p:dataTable var="car" value="#{dtBasicView.cars}">
    <p:column headerText="Id">
        <h:outputText value="#{car.id}" />
    </p:column>

    <p:column headerText="Year" sortBy="#{car.year}">
        <h:outputText value="#{car.year}" />
    </p:column>

    <p:column headerText="Brand" filterBy="#{car.brand}">
        <h:outputText value="#{car.brand}" />
    </p:column>

    <p:column headerText="Color">
        <h:outputText value="#{car.color}" />
    </p:column>
</p:dataTable>
```

Code example 4. PrineFaces example – DataTable (PrimeTek)

There are several web application frameworks for JSF based application like ICEFacec, RichFaces, Tobago, Oracle ADF. Each of these web application frameworks provides several JSF components. Finally, PrimeFaces is used in the application, because it has very good support, product documentation, showcase and many UI components in comparison with others. (Mastertheboss.com)

3.1.8 Chart library

The application should be able to show charts with data obtained from the measurement. These charts have to meet all requirements – they must be interacted (zoom, print selected data/lines), have logarithm axis, present the current state of chart (actual view) downloadable, support dynamically added data and be compatible with PrimeFaces.

Before starting the implementation phase of this feature, it was necessary to make a research. Many chart libraries can be found on the market, however only some of them satisfy all customer requirements. After research following libraries were found:

WickedCharts, PrimeFaces charts, JFreeChart, HighCharts and jqPlot. So the final set was minimized and now these libraries could be compared and fully tested. All the found libraries were tested and implemented with a great precision in the web application, which is the best way to choose the best one. During this phase several of the pluses and minuses were found. For better understanding only some key features were chosen for comparison. The comparison is presented in Table 2. Sign '+' stands for has an ability or if a library is able to provide a certain feature, otherwise sign '-' is the opposite of sign plus.

Table 2. Features comparison of chart libraries

	WicketCharts	PrimeFaces	JFreeChart	HighCharts	jqPlot
Zooming	+	+	-	+	+
Saving current state (image)	+	-	+	+	-
Logarithm axis	+	-	+	+	-
Un/selection certain series in realtime	+	+	-	+	-
Compatible with JSF	+/- certain problems with adding data on request	+	+/- by generating images on backend	+	+
Interacted	+	+	-	+	+
Code modification	+	+/- difficult to modify	-	+	+
Easy to use	+	+	+	+	-
Documentation and examples	+	+	+ (buyable)	+	+

Only two libraries (WicketCharts and HighCharts) meet almost all criteria and the rest of them not as can be read from the table. Firstly, the reasons why the rest of the libraries are insufficient is explained and then the chosen library is described.

To sum up, the charts implemented in **PrimeFaces** (uses jqPlot) are good and suitable, however, there are some crucial disadvantages. This library does not support logarithm axis and it is difficult to make some changes in this library (e.g. implementing logarithm axis, change legend etc.).

The next library not sufficient for the task is **JFreeChart**. This library is easy to use, however, the generated charts are not interactive in the web view (just in desktop view using Java SE). This library is only able to generate images on the server side and send them to the client.

The last insufficient library is **jqPlot**. The major minus of this library is that it is not able to add data dynamically on request, print the current state of graph to the image and it is quite difficult to use.

The final set was reduced to two candidates. Generally **WicketCharts** library is written in Java. This library uses Apache Wicket and HighCharts on the backed side. Thus, this library provides similar functionality as HighCharts. The most crucial reasons why WicketCharts are not used in the application are that it is necessary to use another web framework (Apache Wicket) along JSF, this web framework is much slower and less powerful then JSF, and finally, new data cannot be added dynamically to the existing charts. (Uribe)

Finally **HighCharts** are used in this application. This library is written in pure JavaScript (requires one of this libraries jQuery, MooTools or Prototype). It offers an easy way of adding interactive charts to the web application. This library supports 19 types of charts with great deal of additional functionality, such as adding multiple axes, tooltips, dynamically adding and remove certain series and more. On this web page (http://jsfiddle.net/hvv37dcg/2/) an example of using charts can be seen. (Highcharts)

3.1.9 JasperReports

It is an open source tool used for creating reports in Java. JasperReports is written in Java. This tool can produce pixel-perfect documents that can be printed, viewed or exported in a variety of files types, such as HTML, PDF, Microsoft Word or Excel, RTF, ODT and more. (Jaspersoft Community)

This library can be used in applications that contain Java, including Java SE, Java EE or web application. It reads all instructions for generating reports from an XML file or .jasper file. Jasper uses their own XML files called **JRXML**. The whole structure of output document is defined in this jrxml. This file can be hand-coded, generated or designed by using a tool (iReport, Jaspersoft studio). JasperReports predefines several components that can be used in output documents like tables, graphs (using JFreeChart), text fields or images. Files **.jasper** are compiled classes of JRXML. Each JRXML is compiled before producing the output document. A short example of two elements of JRXML file is illustrated in the Code example 5.

```
<textField isBlankWhenNull="true">
       <reportElement x="155" y="0" width="25" height="11"/>
       <textElement/>
       <textFieldExpression class="java.lang.String">
              <![CDATA[$F{ShipRegion}]]>
     </textFieldExpression>
</textField>
<textField pattern="dd/MM/yyyy">
       <reportElement x="185" y="0" width="50" height="11"/>
       <textElement/>
       <textFieldExpression class="java.sql.Timestamp">
              <![CDATA[$F{OrderDate}]]>
       </textFieldExpression>
</textField>
```

Code example 5. Example of elements in JRXML (Jaspersoft Community)

Other more complex examples can be found on the webpage http://community.jaspersoft.com/. (Heffelfinger)

Generally working with JasperReports has three phases. The first phase is to define output document into XML file (.jrxml). After that these files should be compiled for .jasper files to be generated (compiled by certain tool). The second phase can be

skipped, because it is also possible to compile these files at runtime by using the JasperCompileManager class, however, this is not a very good way to do it in everyday life, because the application is pushed to compile the JRXML file each time. These files can be compiled only once by using related tools. The last phase is to use compiled file and to send the data in. After that the output document can proceed. This process can be better described via Java code (See Code example 6). The goal in this example is to create a basic report containing only two fields - ship region and date. Part of the JRXML file was shown earlier in the Code example 5. (JavaWorld)

```
// First, load document definition from JRXML into
JasperDesign class; can be skipped
JasperDesign jasperDesign =
JasperManager.loadXmlDesign("Delivery.jrxml");

// Second, take created object and compile it into
JasperRepor, can be skipped
JasperReport jasperReport =
JasperManager.compileReport(jasperDesign);

// Third, create a map of parameters to pass to the report.
HashMap params = new HashMap();
params.put("ShipRegion", "Slovakia");
params.put("OrderDate", new Date(2014-1900, 2, 25);

// Create JasperPrint object by using fillReport() method,
JasperFillManager tries to find all entries of HashMap in
the jasper document and put the related value to the found
place.
JasperPrint jasperPrint =
JasperFillManager.fillReport(jasperReport, params);

// Last step – generate output PDF document from JasperPrint
object (contain whole document with filled in data)
JRPdfExporter pdfExp = new JRPdfExporter();      // Can be
used JRDocxExporter for docx
pdfExp.setExporterInput(jasperPrint);
pdfExp.exportReport();
```

Code example 6. Generating PDF file using JasperReports

3.2 Database technologies

3.2.1 Choice of database system

The system should archive two different types of data. The first type is storing information about power transformers, locations, manufactures, theirs measurements and all other related information. The second type stores system logs.

The first data types are related to the relations, so it would be good to store this data in a relation database system. There is also another reason for using relation database, which is that the mostly used operations in the system are about searching transformers with all additional information based on certain filters (name, date, location…). The relation database systems are optimised for select operation. Thus, that is why the relation database system is used instead of NoSQL systems. PostgreSQL was chosen as a relation database system, because this database system is open source, does not have high hardware requirements, is fast enough, appears to be more flexible and comfortable in comparison with MySQL and finally it follows ACID (Atomicity, Consistency, Isolation, Durability) in a better level in comparison with MySQL. (Matiaško, Vajsová, Zábovský, & Chochlík, 2008)

The last requirement for the database system was the ability to store logs from the application. The reason why logs are to be stored into the database is the availability of obtaining logs in a structured format and the possibility of searching or filtering these logs. Working with the logs has certain peculiarities, such as inserting to the database is a relatively common and frequent operation. On the other hand reading is used rarely or less frequently. Over time there can be another requirement for recording some extra information in the log entry, however, all old logs should also be still available. A log entry usually contains just text information like timestamp, severity (fine, info, warning, error, fatal), related class (class that error has happened). (Matiaško, Vajsová, Zábovský, & Chochlík, 2008)

As mentioned before relation database systems are optimized for select operations, however, the logging system requires a frequent insert operation. Also, there is the possibility of changing a log entry (changed table columns by adding or removing) that may cause several problems in the existing relationship model. That is why this

type of database systems are not good for this requirement. It was necessary to find another type of database system that would be suitable for this problem. One solution was found by using document-oriented database (MongoDB). Logging system is not within the scope of this thesis, however, it was necessary to be mentioned, because all logs that are generated by newly created parts are stored in this database.

3.2.2 Hibernate

Hibernate ORM, developed by JBoss Community was used for the communication with database. It is a library that implements object relation mapping (ORM). Two types of Hibernate can be distinguished - ORM and OGM. Hibernate OGM is used for mapping NoSQL database into Java object. This technology is not used in the application, therefore these lines are aimed at Hibernate ORM. (The Hibernate Team)

Hibernate ORM maps relation database model into Java objects with attributes corresponding to the columns in data tables. It is more common and natural to work with objects rather with values that are obtained directly from the database by using direct SQL statements (select, insert, update, delete). (Ottinger, Linwood, & Minter, 2014)

For correct use Hibernate ORM technology it is necessary to define the way how a certain object is mapped to the database table. This mapping can be proceeded by using Java annotations. Hibernate uses JPA (Java Persistence API) specification so JPA annotations can be used for this mapping.

For a better understanding how to map certain objects to the database table a short example has been provided (See Code example 7). In the Code example 7 POJO class can be seen mapped to the database table. Concretely, the name of the table is *label*. This table contains two attributes – *idLabel* and *name*. For the creation of a new POJO class these steps need to be followed: create an attribute corresponding to the column names, add standard get, set, default constructor, hashCode and equal methods. These methods have to be added, because all entries should be uniquely distinguished. All relationships (both sides) are mapped as lists (*java.util.List*), set (*java.util.Set*) or generally collections (*java.util.Collection*). All classed created by this way are called POJO. When all definition steps have been done then JPA annotations can be used for

marking a class to the certain table (*@Entity* and *@Table*), attributes (their standard get methods) to the certain columns (*@Column* and *@Id* if related column is and primary key) and relationships between classes (tables) (*@OneToMany*). (Tutorials Point)

```
@Entity
@Table(name="label",schema="public")
public class Label  implements java.io.Serializable {
     private int idLabel;
     private String name;
     private Set<Transformer> transformers = new
HashSet<Transformer>(0);

    public Label() { }

    public Label(int idLabel, String name) {
        this.idLabel = idLabel;
        this.name = name;
    }

    public Label(int idLabel, String name, Set<Transformer>
transformers) {
       this.idLabel = idLabel;
       this.name = name;
       this.transformers = transformers;}

    @Id
    @Column(name="id_label", unique=true, nullable=false)
    public int getIdLabel() { return this.idLabel; }

    public void setIdLabel(int idLabel) { this.idLabel =
idLabel; }

    @Column(name="name", nullable=false)
    public String getName() { return this.name; }

    public void setName(String name) { this.name = name; }

    @OneToMany(fetch=FetchType.LAZY, mappedBy="label")
    public Set<Transformer> getTransformers() { return
this.transformers; }

    public void setTransformers(Set<Transformer>
transformers) {
          this.transformers = transformers;
    }
```

Code example 7. Example of POJO class with JPA annotation

The most important configuration file for Hibernate is hibernate.cfg.xml. In this file detail information about connection, like host, name, database name, password are defined, however, also some other configurations are included (e.g. session timeout, batch size and so on). (The Hibernate Team)

If working with database model containing plenty of entities it can be quite laborious to manually create all POJO classes with related annotations or form the scope of view to manually write SQL statements for creation database tables, in case POJO class has been changed in Java, which is a reason why Hibernate is able to provide reverse and forward engineering. **Reverse engineering** is the process when Hibernate generates POJO classes with all related annotations from the database model (stored in database). **Forward engineering** is an opposite process – POJO classes have been done with JPA annotations, and a database model needs to be created into database. These process are done automatically and can also be used during the migration phase. During the developing phase, the data model (was made in ER editor) was imported into the PostgreSQL database system and then the reverse engineering was used for generating POJO classes. (The Hibernate Team)

Sometimes there is a need to customize more complicated SQL statements. Hibernate enables us to write these statements in two ways. The first way is to use **HQL** (Hibernate Query Language). This language is very similar to SQL, however, it works with the mapped Java POJO classes. The second way Hibernate enables is to use **native SQL** queries for certain database systems. There is one disadvantage of using native SQL queries, it is a portability, because this statements needs certain DB system for running correctly in comparison with HQL. HQL and POJO classes are independent of the database system. Hibernate supports all most popular database systems, e.g. Oracle DB, IBM DB2, PostgreSQL, MS SQL, MySQL and others. So this system can be switched to use different database systems on the condition that native SQL queries were not used. (Ottinger, Linwood, & Minter, 2014)

To sum up, Hibernate offers Object Relational Mapping, reverse and forward engineering and independence of the database queries from used database system. This is the way how the work with database can proceed in object oriented style independently of the platform. If some changes were made in database server or in

POJO classes, forward or reverse engineer could be used to move on all changes to the database model or to the model classes without the need for doing the same process twice (in the database model and also in the model classes).

4 Implementation

As mentioned before this solution is based on Java EE and the described technologies (Weld CDI, Hibernate, HighCharts, JasperReports…). All necessary information, such as basic terms, requirement gathering, analysis phase, design phase, decisions and used technologies, have already been provided and described earlier in this paper, therefore now the concrete implementation phase with description of interconnection can be described.

4.1 Project structure

Information about the project structure is provided in this chapter. Project structure is described via text and using images. The whole information system is quite extensive, therefore it is divided into the several packages. A package is a mechanism for organizing all files (Java classes, resources, other files) into some namespaces. Only some packages, the most important and directly or indirectly used in the scope of this thesis, are described.

All packages are stored under one namespace - *sk.fri*. Java package naming conventions were used for creating unique namespaces. The list off necessary packages contains this packages:

- controller
- controller.util
- highcharts
- jasper
- jasper.cust
- model
- model.types
- parser
- parser.interfaces
- service
- util
- resources.

It is not enough to name all used packages, but also it is necessary to provide some short information about them, like what can be found in this package, why this package is necessary and what is the purpose.

Controller – all controllers are stored in this package, these controllers are used by PrimeFaces to communicate with database, application services and resources. Therefore all controller classes located in these packages interconnected with *resource* and *service* package.

Controller.util – only one class can be found in this package so far, methods for showing status messages are stored in this class. Status messages contains information, e.g. successfully proceeded, some error has occurred, printing exceptions and more

Highcharts – In this package classes are stored that are used among HighCharts library, for example, in this package class *series* was created. All necessary information about one series (name of the series, data, boolean value if series is shown and more) can be found in this class. These classes are described later more in detail.

Jasper – all classes closely used with JasperReports library can be found here. These classes are sent to the JasperReports library during the report generating phase. These classes are described later more in detail.

Jasper.util – some utilities are stored in this package that are used in Jasper package. So far there is only one utility stored for setting logarithmic axis with minimal and maximal value for axis x.

Model – contains classes mapped from database by hibernate. These classes are simple POJOs classes and they are also annotated by JPA annotations. Basic principles were described before in chapter 3.2.2.

Model.types – in this package one class containing the logic for newly implemented database type – array.

Parser – Only one class can be found in this package. Data parsing logic for one certain type of CSV file (produced by measurement device from Doble company) is implemented in this class. These classes are described later more in detail.

Parser.interfaces – This system processes and obtains all necessary data from any CSV file. For creation a support for new CSV file type it is necessary to create a new class to be implemented for all interfaces stored in this package. Methods, like read file, process file, create measurement can be found in these interfaces.

Service – is the main service package. This package's classes are used by sk.fri.controller and sk.fri.ws (in other words GUI controllers and web services). Generally service package classes are main database stubs, which means they provide the main bridge between the world and the database.

Util - Definitions and initializations of frequently used objects like EntityManager, Logger, LanguageBean and more other can be found here.

Resource – all resources that are used in this project are stored in this package. These resources can be images, CSS files, compiled files for jasper reports, JavaScript files and other.

The project structure should be clear for everybody at this point, therefore now the next paragraphs focus on the implementation phase. The questions such as what was the problem, how was the problem solved, what was necessary to do, what is the final solution and any other questions of certain parts were answered in the next paragraphs.

4.2 Data processing

Several important facts were revealed by analysis of the CSV file structure. One of this fact was that each CSV file can contain different measured values (measurement columns), denoted in the header of the CSV file. To be able to import the CSV file correctly it is necessary to store the header information as well. Only then it is lately possible to identify, which values were measured in some particular measurement.

Because of this feature, special attention was paid to the design of the structure of the database. (Grondžák, Grigeľ, Polakovič, & Remenec, 2014)

To store such polymorph data it was necessary to use dynamic database model, also called Entity Attribute Value (EAV) database model. It is not necessary to give a general description of this type of database model, however, what is the necessary is to define how the data are recorded. (Grondžák, Grigeľ, Polakovič, & Remenec, 2014) Data is recorded in three columns:

- The entity: the item that contains several attributes and going to be described (table measurement).
- The attribute: a foreign key into a table of attribute definitions. The attribute definitions table contains the following columns: an attribute ID, attribute name, description, data type, and columns (table *column_data*).
- The value of the attribute (table *measurement_column* and *measurand*).

The measurement part of the database scope can be seen in the Figure 3. This database part will also be used later during the description of the generating and charting part.

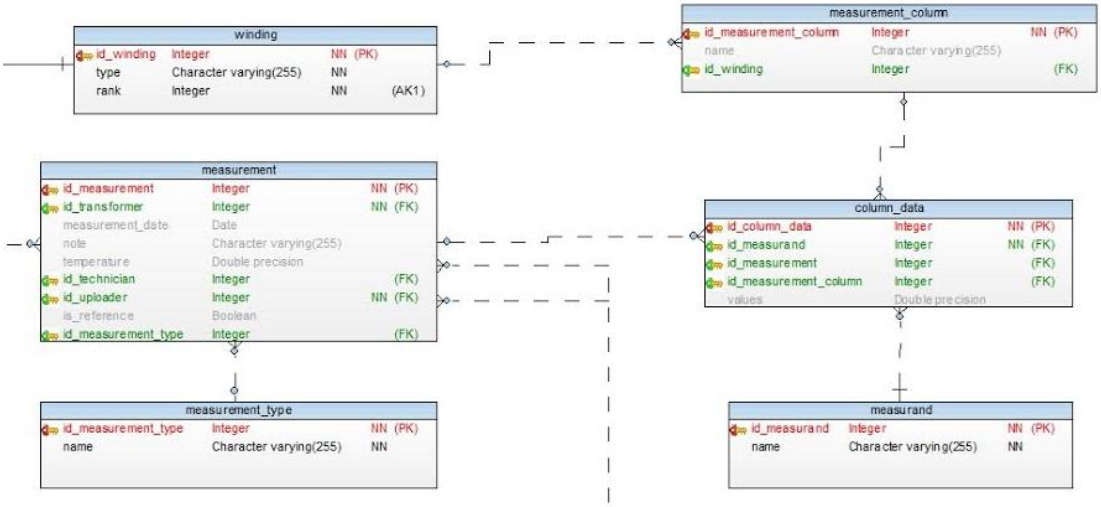


Figure 3. Measurement part database model (EAV)

Applied to the described case, each measurement column name is stored in a row of a single common table. This kind of model allows easy extensibility and maintenance. It does not require any modification of the application code, when new types of measured values are imported from CVS file. Thus, when new kind of data is stored in the database, it is not necessary to modify the application. (Grondžák, Grigeľ, Polakovič, & Remenec, 2014)

Related information about used database model with EAV database model was provided, therefore, now the whole process of implementation phase can be described.

Even prior to describing the whole process it is necessary to acquaint the reader with one type of input CSV file structure. CSV file structure, generated by measurement device created by Doble company can be seen in the Code example 8.

```
Line 1: [FRA Data v2.0],,,,,,,,,,,,,,,,,,,,,,,,
Line 2:
temp,NaN,RH,NaN,temp,NaN,RH,NaN,temp,NaN,RH,NaN,temp,NaN,RH,
NaN,temp,NaN,RH,NaN,temp,NaN,RH,NaN
Line 3: A-Nssek freq,A-Nssek RMS,A-Nssek Peaks,A-Nssek
Phase,B-Nssek freq,B-Nssek RMS,B-Nssek Peaks,B-Nssek
Phase,C-Nssek freq,C-Nssek RMS,C-Nssek Peaks,C-Nssek
Phase,a-nsprim freq,a-nsprim RMS,a-nsprim Peaks,a-nsprim
Phase,b-nsprim freq,b-nsprim RMS,b-nsprim Peaks,b-nsprim
Phase,c-nsprim freq,c-nsprim RMS,c-nsprim Peaks,c-nsprim
Phase
Line 4: 20.074,-79.737,-79.759,97.306,20.074,-0.302,-0.302,-
4.337,20.074,-0.978,-0.978,-4.076,20.074.……
Line 1255: [nameplate ZU_BB T XX nakratko_Skoda_xxxxx_24-12-
2014.csv]
Line 1256: Company, Žilinská univerzita
Line 1282: [nameplate end]
Line 1283: [Plot Names]
Line 1293: end of file
```

Code example 8. Example of CSV structure

Before starting the implementation of CSV parser it was necessary to explore the CSV structure. On the first look this CSV structure does not follow any of the commonly used definitions, however the structure of this file is defined by the measurement device, is always the same (no changes) and is divided into six parts. These parts are shown in the previous figure and are highlighted using numbered lines. It is possible to find the type of the CSV file in the first part/line. In the second row some units are defined, however, these units are not used anywhere. The next two parts are very important. Line number 3 contains the header of the CSV file. Header contains the names of the measured variables. The values of these variables are stored as a record (one record per line) in several lines (approximately from line 4 to line 1255).

At this point all measured data were obtained and now it is necessary to obtain the other values, such as nameplate data. Nameplate data are stored between starting and ending tag – [nameplate] (Line 1255 - 1282) and they are stored as name – value pair (one example is shown in line 1256). Nameplate data could be a company that owns the power transformer, location, serial number, manufacturer, manufacturer year, number of windings/phases with their values and units and so on.

The last part of this file is plot names. This section contains information what the names of the measured columns are. Each name of a column is stored in a new line as a new record.

As soon as the file structure was known the implementation phase could begin. The implementation phase for obtaining all data from CSV file was quite easy if the file structure was already known. For parsing CSV file Opencsv library is used. This library is very simple CSV parser for Java. It supports arbitrary numbers of values per record, ignoring commas that are quoted, reading gradually line per line or reading all the entries at once, and finally creating an array of a string that contains the obtained values for each line. (Opencsv)

Now all data are parsed and stored in *MeasurementParsed* object (from package parser). This object contains all obtained information that are stored in several collections (hash map or arraylist). This object implements *ParsedMeasurment* the interface that is stored in *parser.interfaces.* At this point the newly created part of the

application is able to process the input file, so now the whole process can be described from the user's scope of view.

Processing the new file via Graphical User Interface (GUI) has four steps. Each step should be done and user is pushed to do the whole process step by step. This approach was chosen because all necessary information is to be filled in without any complications (forget to fill in/check some values and so on). The first step is to upload the measured CSV files to the server (See Figure 4). This can be done with drag-and-drop feature or by choosing file component. The file should be dropped to the white space on step 1. After choosing the data file, this file is checked, if it contains the correct data and it is a CSV file.

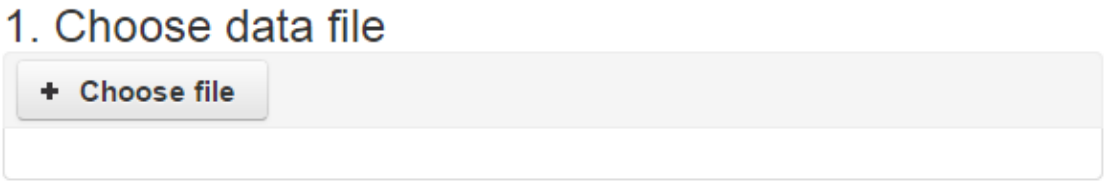


Figure 4. Upload - Upload data file

If the first step was done correctly (e.g. file is not corrupted, contains needed information, file is CSV file), step 2 will be shown. On the next figure (See Figure 5) are shown 2 block of data. The first block contains information that file was successfully uploaded and handled. The second block contains the table with the list of founded transformers. In this case the application was able to identify the correct transformer directly, however if the application is not able to identify the correct transformer directly it offers a bigger set of transformers. The application is always trying to minimize this set – it tries to select transformers with certain information (e.g. city, location, serial number or company) obtained from the CSV file. If no matches found the whole database set of transformers is shown and the user is pushed to find the correct transformer manually.

Figure 5. Upload - Choose related transformer

After successfully choosing transformer in step 2, step 3 is shown. In this step is necessary to select measurement type, measurement operator and if the measurement is reference (See Figure 6). For choosing measurement type or operator select one menu element was used. There is a possibility to create a new measurement operator, if the measurement operator was not found in the select one menu element, by choosing the value *other* from the combo box. Reference measurement is such as measurement that all other measurements are compared against this one. Usually it is the first measurement of the power transformer.

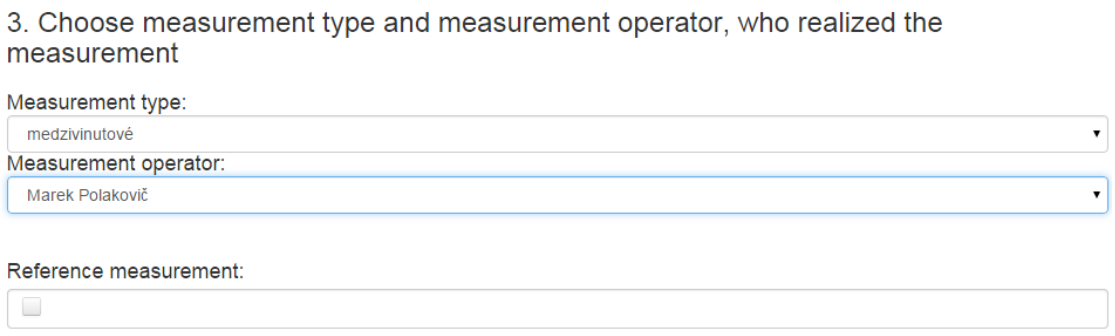


Figure 6. Upload - Choose measurement type and operator

Now, in this state, when all these described steps were done there is the possibility to save the measurement to the database. This can be proceeded by pressing the upload button. This process take a few of seconds, because plenty of data should be stored

into database, so that is why the loading bar is shown during this process. When the process is ended the final dialog is shown (See Figure 7). This dialog contains information whether the import was successful or unsuccessful and it also contains two extra buttons. One button is used for uploading another measurement and the other one is used for showing imported data in another view. More detailed information about view for showing all uploaded data is described in Chapter 4.3.

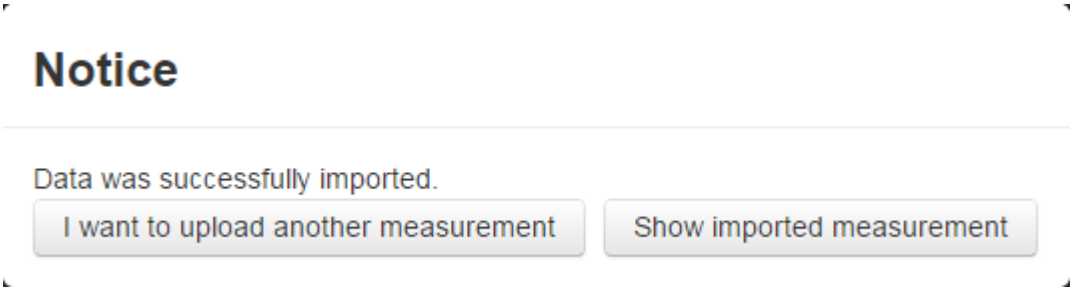


Figure 7. Upload - Final dialog

When the final notice is shown this file process ends and the user can continue with other tasks.

4.3 Charting library – HighCharts

In this situation some measurements are already stored in the database, so it is possible to see and draw uploaded data. To show correct data it is necessary to provide two or three steps.

The first step is to choose power transformer. The power transformer can be chosen by using table (See Figure 8). It was necessary to implement some filters for certain columns, because of the fact that the number of power transformers can be very high. Two types of filters are used. One type uses a combo box element and the other one input text element. There is only a small set of cities, companies and manufactures, so that is why this filter uses combo box in comparison with the other fields. For implementing filters for other fields, such as locations, labels or serial numbers input text was used. This element is better for these fields, because of the fact that these

fields can contain various distinct item values, e.g. the serial number should be unique among all transformers. Filter method assigned to the columns containing filter with input box is set to *contains*, which means that as soon as the value in the filter has changed this method starts to search for entered value in certain column.

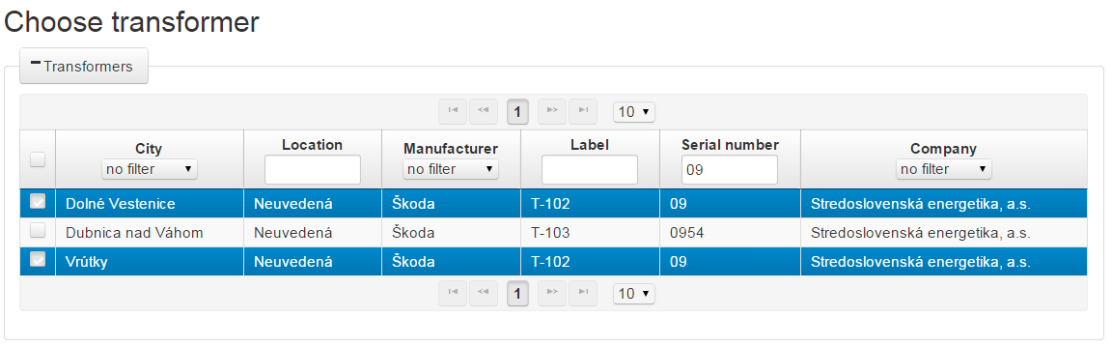


Figure 8. Charting - Choose transformer

It is possible to select more transformers as shown in Figure 8, because sometimes it is necessary to compare their measurements. If one or more transformers are selected it is possible to select theirs measurements – step two (See Figure 9). To see related measurements is necessary to choose certain transformer from the tab. Each tab represents data of one transformer that was chosen before in step 1. Also, only measurements belonging to the selected transformer are shown.

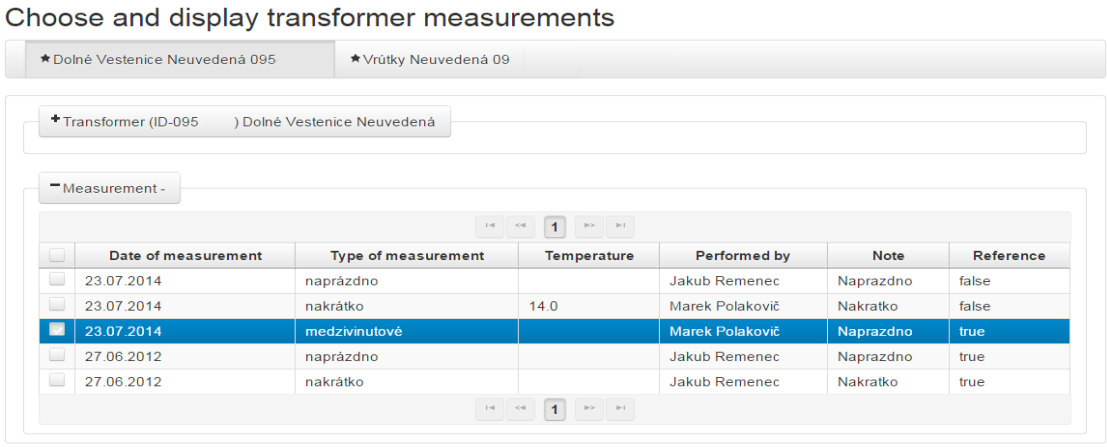


Figure 9. Charting - Choose measurement

If there is some misunderstanding of which transformer is chosen, it is possible to display all its details. These details can be shown by clicking on the button that is located below the tab with the transformers. The rolled up details can be seen in Figure 10.

Serial number:		**Frequency:**	0.0
City:	Dolné Vestenice	**Hookup:**	YNyn0/d
Location:	Neuvedená	**Type:**	
Manufacturer:	Škoda	**Switch position:**	14
Label:	T-102	**Company:**	
Year:		**Registry**	Valid

Windings:	▪ 110.0 kV, 16.0 kW,primary ▪ 23.0 kV, 16.0 kW,secondary ▪ 6.3 kV, 5.0 kW,tertiary

Figure 10. Charting - Transfomer details

As can be seen in Figure 9 only one measurement was selected, which means for charting library that it should be able to print all its data. In this example only one measurement was selected. This measurement contains seven columns that can be printed out (See Figure 11). If the measurement data was downloaded suddenly, this data is cached. In this case it means that the bean that stores this data is annotated as a *ViewScoped*.

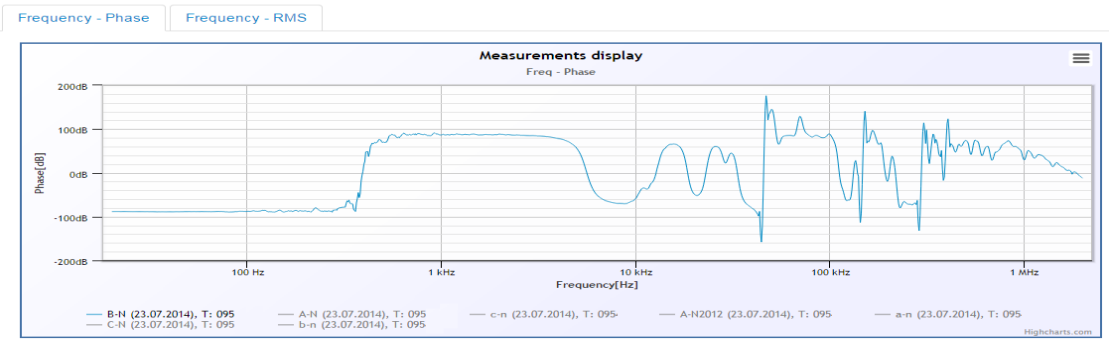


Figure 11. Charting - Chart example

There are two tabs: *Frequency – Phase* and *Frequency – RMS* above the chart, see Figure 11. These tabs represent what type of chart is shown. When the first tab is selected the chart displays the frequency (in Hz) plotted against the phase (in dB). On the other hand, when the second tab is selected the chart displays the frequency (in Hz) plotted against the RMS.

All series loaded into chart are not visible by default, so they should be selected manually by clicking on theirs names. The loaded data related to the certain measurements are stored with other attributes in class - *Series.* This approach was chosen, because many series can be loaded there and after adding all these series into the chart, the chart becomes complicated and disarranged.

Because of the fact that each chart contains much data, it was necessary to arrange axis x. Axis x was set up to logarithmic axis and also only some key values (100 Hz, 1 kHz, 10 kHz…) are shown. There is also a possibility to zoom certain areas of the chart and to export the current state of the chart as image in several formats (at least png and svg).

Finally, if the chart is plotted correctly and all values are checked by measurement analyst it is possible to generate final report. This part is described in the next chapter.

4.4 Reporting – JasperReports

The first step in generation of the final report was to design JRML files. These files were designed in JasperSoft studio. Jasper library enables to use several sub-reports, so the final report includes several smaller sub-reports. Each part of the final report is a sub-report. Using sub-reports is very useful, because it is much easier to design sub-reports and all related Java classes that contain all data. The final JRXML report contains seven sub-reports. One example of a generated report is shown in Appendix (See Appendix 1).

All final report data are allusive in one object – the instance of class *TransformerDetailReport*. (See Figure 12)

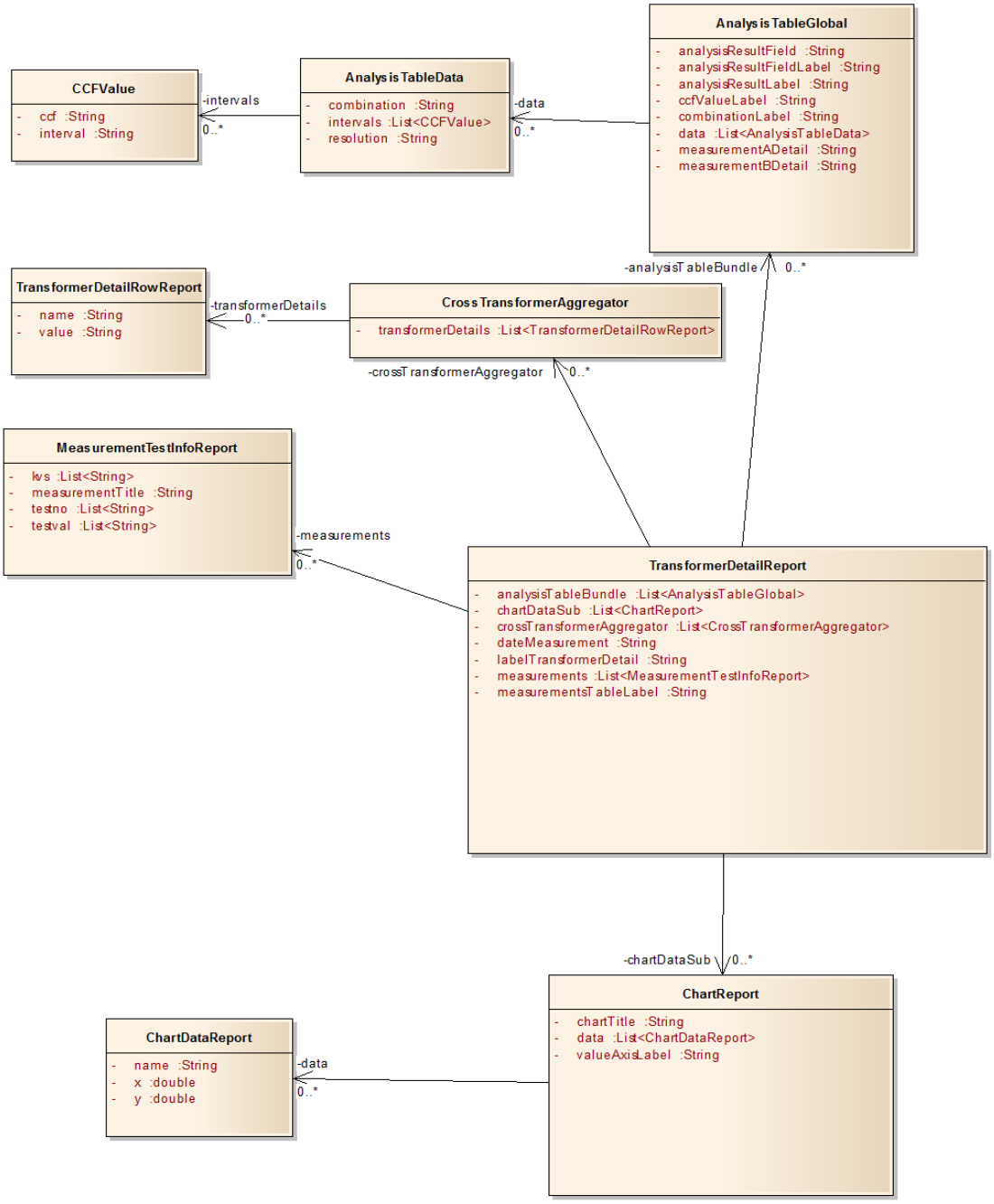


Figure 12. Objects used for Jasper 1

The first sub-report is designed for the title page. All necessary information (like subject, related company, logos and name of the person who elaborated this protocol) that are shown in the report should be stored in the *IntroPageReport* class (See Figure 13).

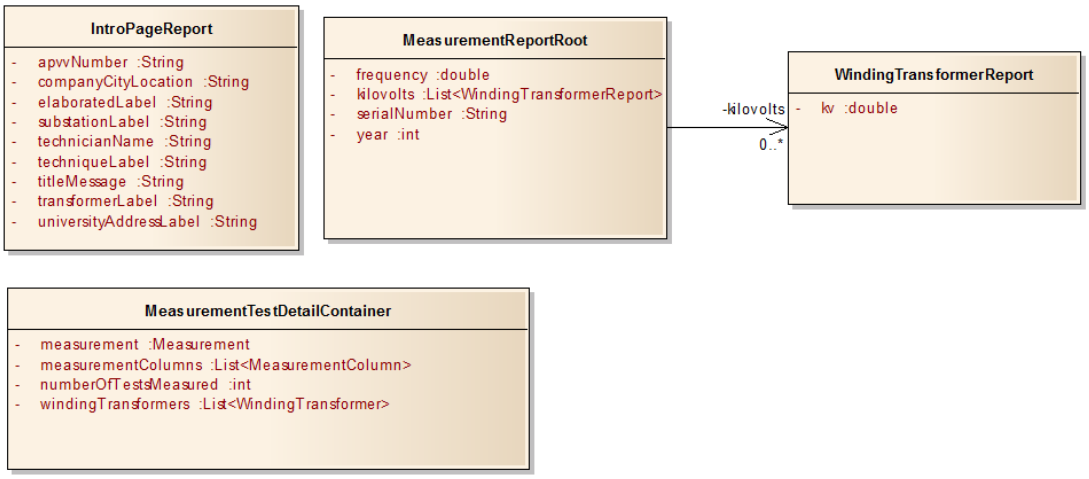


Figure 13. Objects used for Jasper 2

The second sub-report is used for printing nameplate data. This data is stored in *TransformerDetailRowReport* as a key-value pair (See Figure 14). Some data were deleted because of the security issue.

Transformer nameplate data:

Substation, label	
Manufacturer	Škoda, s.n.:
Manufacturer year	1984
Type, hookup	6 ERH 29M-0, YNyn0/d
Frequency	0.0 Hz
Windings	110.0/23.0/110.0 kV
Power	16.0/16.0/16.0 MVA
Switch position	14

Figure 14 Report - nameplate data

The third and fourth sub-report were used for creating a table that contains information about the measurement's columns, windings and frequency (See Figure 15). This table is quite complex and it was quite difficult to design it. The most

difficult was to ensure that this table should always has the same view, however, all data that are shown are dynamical. For example, it means that the measurement can have only two windings and four tests. The full dimension of this table is to have three windings and each winding has three tests. These information are stored in *MeasurementTestInfoReport*.

Measurement: 23.07.2014 short circuit. Transformer s.n.: xxxxx								
110.0 kV primary			23.0 kV secondary			6.3 kV tertiary		
Test 8	Test 9	Test 10	Test 11	Test 12	Test 13	Test 14		
A-N	B-N	C-N	a-n	b-n	c-n	A-N2012		

Figure 15. Report - measurement table

The next sub-report was used for plotting charts into a final report. This report contains only data for printing charts and the related title (See Figure 16). All data are stored in two objects – *ChartReport* and *ChartDataReport*. (See Figure 12) One problem was found during printing charts. This problem was about the quality of the image. These printed images were unreadable and within poor quality, so it was necessary to increase the dpi (dots per inch) of image. This approach was done by setting the value for *net.sf.jasperreports.image.dpi* parameter to 600 dpi (default value was set to 20 dpi).

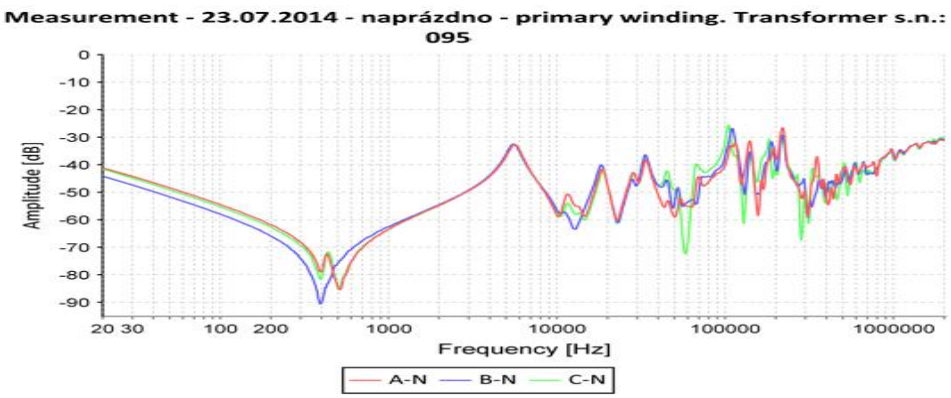


Figure 16. Report - chart

The last three sub-reports are used for printing analysis results. One example of the analysis result can be seen in Figure 17. The first of these sub-reports is used for printing the header of the table, the second for printing the combination with analysis result and the last one is used for printing certain intervals with CCF values. The reason why a new sub-report is used for intervals is that the number of intervals is not static (usually there are three or four intervals, however, there can be even more or less). The structure of data object has been already mentioned and described in the paragraphs above.

Analysis result			
Combination	**CCF value on interval**		**Analysis result**
A-N - B-N	0 Hz - 2000 Hz	0.9986	**Movement of the main and tap windings, ground impedances variations, axial displacement**
	2000 Hz - 20000 Hz	0.998	
	20000 Hz - 400000 Hz	0.994	
	400000 Hz - 2000000 Hz	0.9852	

Figure 17. Report – analysis

After all JRXML files were defined they were moved to the related place in the project structure. Subsequently, all these files were compiled and copied to the public directory (directory containing web pages in this application).

If all files were in right place the application is able to fill in correct and actual data to the related objects. This step is done every time when the user calls an action for generating a report (Create measurement report) by clicking on menu item PDF or DOCX (See Figure 18). If a user clicks on the PDF item the document will be generated as a PDF document, otherwise the document will be generated as a DOCX document.

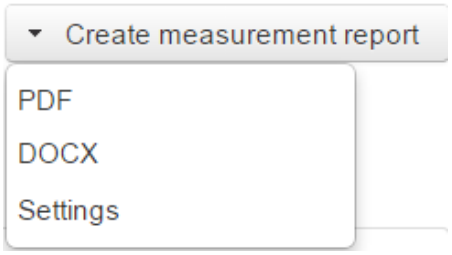


Figure 18. Report - menu

There is one extra menu item as shown on the figure above. After clicking on this menu item the settings window is shown (See Figure 19). It is possible to choose what parts are to be generated in the final report. The user can then decide whether the analysis results are to be included or which series are to be included in the report.

Figure 19. Report – settings

This process took some time, because much data should be processed by jasper library, so it was necessary to implement the loading bar. Generating the whole report took approximately 10 seconds, mainly because of the fact that the final report is quite complicated and contains a lot of data (e.g. measurement information, data for plotting, analysis and so on).

Therefore, the whole newly implemented parts were done and the testing phase could be started. To see how the results were confirmed and how the application was tested the reader is invited to continue reading.

5 Confirmation of results and testing

The host company was also involved in the development phase. All fully developed parts were step by step deployed at the host company for the host company to be able to test them. It was necessary to communicate with the representative of the host company frequently in this phase because of the fact that the application should be validated, confirmed, robust and bug free.

Several tests were done by the host company. One of these tests was to upload several sample data (correct data and also wrong data with same errors) from the measured power transformers. If all sample data were uploaded it was necessary to check them. The first check was done by reviewing measurement management parts. If all data was shown correctly, the second phase could be started. The second phase was to check the column data. These data could be checked by using charts. Several uploaded columns from measurement were selected and printed out into charts. After that this chart could be reviewed and compared to the old ones (created by using the old process that means not using this IS) by an analyst. If no errors had occurred during these phases the last test could be started. The last test was to generate a sample report into docx or PDF file. Subsequently, these files were checked and compared to the old protocols created manually. It was necessary to test the generating phase by using several combinations, such as to generate report with one measurement, with more measurements, with the chart part, in English/Slovak language and further.

All reported issues, bugs and suggestions were fixed. By using this process it was achieved that the application was confirmed, usable and valuable for them.

6 Discussion

The main goal of this bachelor's thesis was to analyse and develop an information system for storing, managing and analysing transformer measurement data. Basically, these goals were necessary to achieve: design and develop multiple parts of an information system, such as data processing, reporting and charting part. The requirements for this information system were presented in Chapter 2.1.

The parts of the information system that are the output of this paper were to facilitate the work of technicians and to improve the outcome of the power transformer analysis, powered by the technician. These parts of information system allow the technician to upload new measurements including mapping this measurement to the certain power transformer with other additional data (note, temperature, outdoor conditions during measuring) and are described in Chapter 4. All uploaded data are processed and stored in common database, therefore the measurement data are shared easily. Subsequently, it offers technicians a possibility to review this measurement data considering the related power transformer. The technician is able to compare the selected measurement along the others if more measurements were performed and uploaded into the information system. This comparison was done via interactive charts, which means that the measurement expert can focus only on parts that look different. After this phase the technician is able to generate the final report that will contain power transformer plate information with the measurements details and the result of the analyses. The measurement details contains several information, e.g. date of performing measurement, measurement combinations and finally printed out measurement data via charts.

These parts of system do not substitute any technician or measurement operators, however, each of these parts offers to them a wider range of information for making decisions and analysis. These parts are able to show theoretically an unlimited number of measurements in one chart, therefore it allows to explore much more measurements in a short time. From an analytics point of view, the software makes the technician's work easier and faster. It stores a whole database of power transformers with related

measurements and allows to share this data without any extra work (e.g. copy and send all needed files to another person).

Generally, the creation of an information system is quite common topic. This information system is used for storing, processing, managing and analysing data obtained from modern measurement devices. The implementation of this type of system is not so common, however, it is very rare. The information system is still being developed and it has been presented to various people working with power transformers. Currently the system works in majority of web browsers.

By using this software the technician is able to obtain and produce all needed information in a short time and that is why all technician's work time can be aimed at analysing all stored data, thus it saves a large amount of working hours of the host company. From another perspective these results can be also used for designing and implementing similar information systems or modules.

Generally, there was only one limitation. This limitations was related to the process of obtaining the correct data from the CSV files and information about the power transformers. Currently, the system supports only one structure of the input file. This is one of the areas that can be used for further development in the future. Another area can be developed in the future is implementation of 3D simulation with 3D analysis of power transformers. These methods are currently under development and test by the Faculty of Electrical Engineering.

The most difficult part of this thesis was to gather and properly understand all requirements. This was carried out in many sessions with the representative of the host company and it took plenty of time. Another quite difficult part was to explore all possibilities how set up goals can be achieved. On the other hand during the whole process I gained plenty new information regarding to the developing process, and in that sense increased my professional knowledge and experience.

Many technologies were used during the implementation phase of the solution. The reader has the opportunity to find basic principles, information and use of these technologies in this paper. The justification of their use, a description of ways how

they are used and advantages / disadvantages of using certain technology can also be found in this thesis (See Chapter 3). The technologies included in the solution are Java EE 7, EJB 3.x (implementation of business logic), WeldCDI (Context and Dependency Injection), Hibernate ORM (manipulating with database systems), PrimeFaces / JSF (developing the UI), Apache Maven (dependency and build management), JasperReports (generating text reports), HighCharts (working with interactive charts), OpenCSV (parsing CSV files) and other.

Considering the fact that all implemented parts solved all the defined objects – creation of plugins/modules for processing and storing measurement data, comparing obtained data among several measurements and generating final reports, this application and paper meet all goals that were specified by the host company, and finally, it is very helpful for the company.

References

Finnigan, Ken. 2013. JBoss Weld CDI for Java Platform. Birmingham: Pack Publishing.

The Apache Software Foundation. n.d. Maven. Accessed on 12.04.2015, retrieved from https://maven.apache.org/.

Gibson, Andy. n.d. Comparing JSF Beans, CDI Beans and EJBs. Accessed on 26.04.2015, retrieved from http://www.andygibson.net/blog/article/comparing-jsf-beans-cdi-beans-and-ejbs/.

Goncalves, Antonio. 2009. Beginning Java EE 6 platform with GlassFish 3: From Novice to Professional. USA: Apress.

Grondžák, Karol, Rudolf Grigeľ, Marek Polakovič, and Jakub Remenec. 2014. "The architecture of an information system for storing the transformer measurement data." DESAM 2014. Žilina: University of Zilina. 12-15.

Gupta, Arun. 2013. Java EE7 Essentials. United States of America: O'Reilly Media.

Heffelfinger, David R. n.d. Getting Started With JasperReports. Accessed on 27.04.2015, retrieved from http://ensode.net/jasperreports_intro.html.

Highcharts. n.d. Highcharts. Accessed on 27.04.2015, retrieved from http://www.highcharts.com/.

Jaspersoft Community. n.d. JasperReports Library - Samples. Accessed on 26.04.2015, retrieved from http://community.jaspersoft.com/wiki/jasperreports-library-samples#Jasper_Sample.

Jaspersoft Community. n.d. JasperReports Library. Accessed on 26.04.2015, retrieved from http://community.jaspersoft.com/project/jasperreports-library.

JavaWorld. n.d. Reports made easy with JasperReports. Accessed on 26.04.2015, retrieved from http://www.javaworld.com/article/2074594/java-security/reports-made-easy-with-jasperreports.html.

Mastertheboss.com. n.d. PrimeFaces vs RichFaces vs IceFaces. Accessed on 21.04.2015, retrieved from http://www.mastertheboss.com/jboss-web/richfaces/primefaces-vs-richfaces-vs-icefaces.

Matiaško, Karol, Monika Vajsová, Michal Zábovský, and Matúš Chochlík. 208. Databázové systémy - Databázové technológie a aplikácie. Žilina: EDIS.

Microsoft. n.d. ADO .NET Overview MSDN. Accessed on 05.04.2015, retrieved from https://msdn.microsoft.com/en-us/library/h43ks021%28v=vs.110%29.aspx.

Microsoft. n.d. ASP .NET. Accessed on 04 06 2015, retrieved from http://www.asp.net/.

NHibernate Community. 2014. NHibernate Official Site. Accessed on 23.04.2015, retrieved from http://nhibernate.info/.

Opencsv. n.d. opencsv - General. Accessed on 03.05.2015, retrieved from http://opencsv.sourceforge.net/.

Ottinger, Joseph, Jeff Linwood, and Dave Minter. 2014. Beginning Hibernate. New York: Apress.

PrimeTek. n.d. PrimeFaces. Accessed on 19.04.2015, retrieved from http://www.primefaces.org/.

RedHat, Jboss by. n.d. Weld - CDI Reference Implementation. Accessed on 04 13 2015, retrieved from http://docs.jboss.org/weld/reference/latest-2.2/en-US/html/.

Rubinger, Andrew Lee, and Bill Burke. 2010. Enterprise JavaBeans 3.1. United States of America: O'Reilly Media.

Sonatype Company. 2008. Maven: The Definitive Guide. Fulton: Sonatype Company.

The Hibernate Team. n.d. Hibernate Developer Guide. Accessed on 22.04.2015, retrieved from http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/.

Tutorials Point. n.d. Hibernate Tutorial. Accessed on 25.04.2015, retrieved from http://www.tutorialspoint.com/hibernate/index.htm.

Tutorials Point. n.d. JSP Standard Tag Library (JSTL) Tutorial. Accessed on 18.04.2015, retrieved from http://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm.

Tutorials Point. 2014. Maven Tutorial. Accessed on 21.04.2015, retrieved from http://www.tutorialspoint.com/maven/.

Uribe, Leonardo. n.d. JSF and MyFaces Hints and Tips. Accessed on 26.04.2015, retrieved from http://lu4242.blogspot.fi/2012/05/understandingjsf-2-and-wicket.html.

Wikipedia. n.d. Entity-attribute-value mode: Accessed on 11.05.2015, retrieved from http://en.wikipedia.org/wiki/Entity%E2%80%93attribute%E2%80%93value_model.

Zikopoulos, Paul, Dirk Deroo, Krishnan Parasuraman, Thomas Deutsch, David Corrigan, and James Giles. 2013. Harness the Power of Big Data. USA: McGraw Gill.

Appendices

Appendix 1 - Example of final report.

University of Žilina
Faculty of Electrical Engineering
Univerzitná 8215/1, 010 26 Žilina

Protocol of measurement transformer T-102 in substation Stredoslovenská energetika, a.s. Dolné Vestenice using SFRA technique APVV - 0703 – 10

Elaborated by: Marek Polakovic

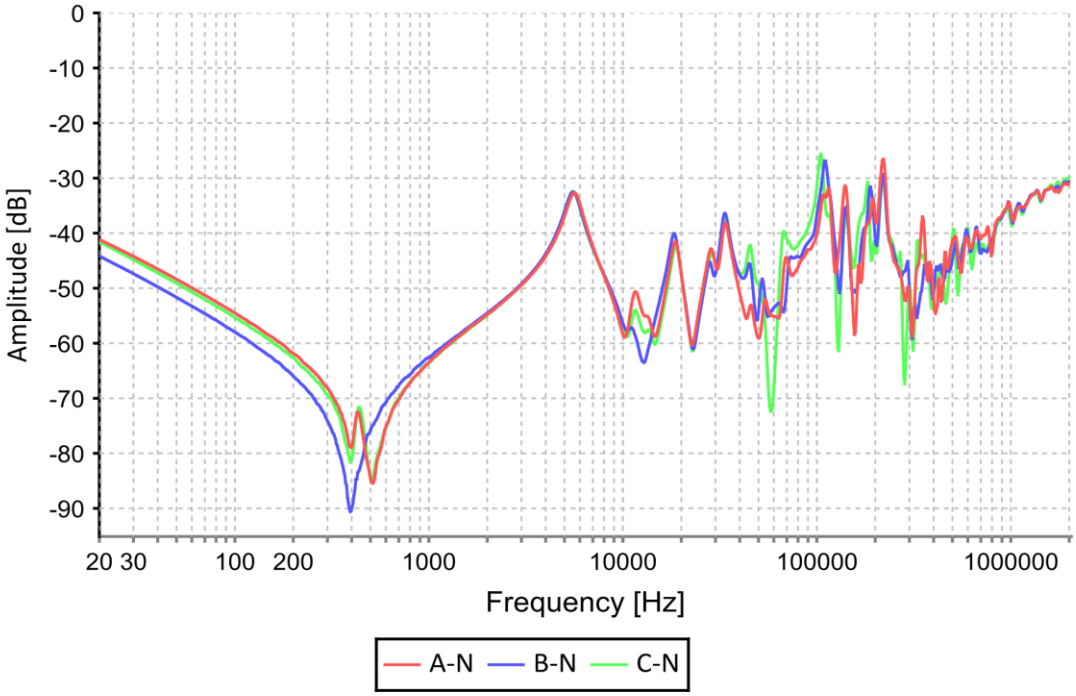
Transformer nameplate data:

Substation, label	Dolné Vestenice
Manufacturer	Škoda, s.n.: 095xxx
Manufacturer year	1984
Type, hookup	6 ERH 29M-0, YNyn0/d
Frequency	0.0 Hz
Windings	110.0/23.0/110.0 kV
Power	16.0/16.0/16.0 MVA
Switch position	14

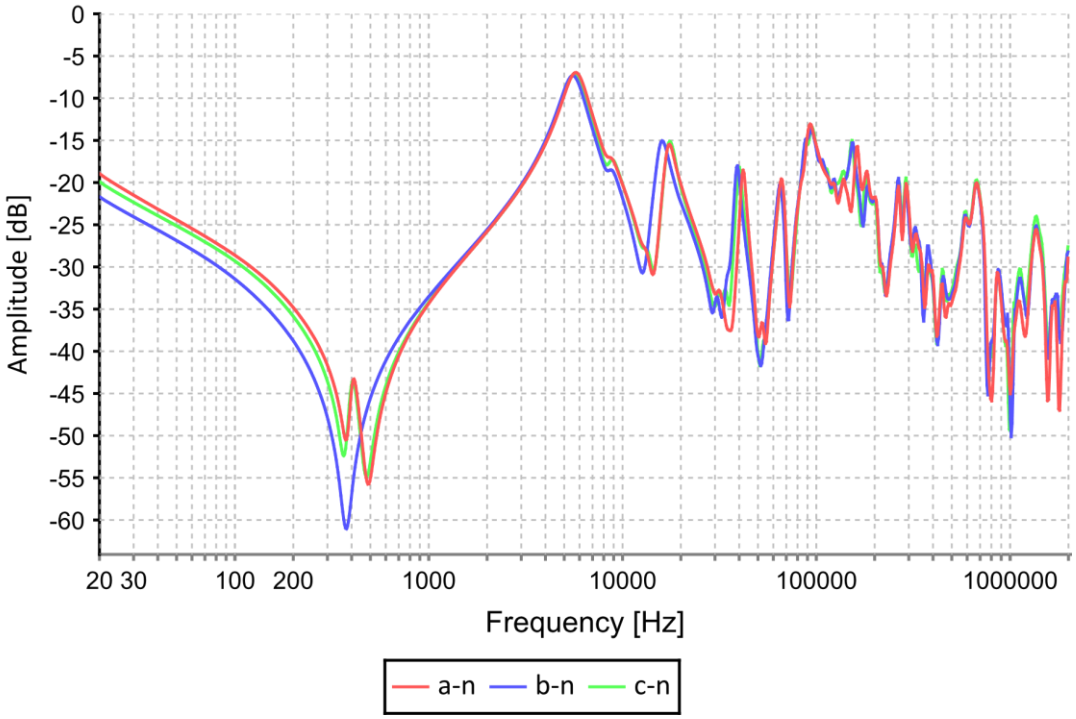
Measurements table

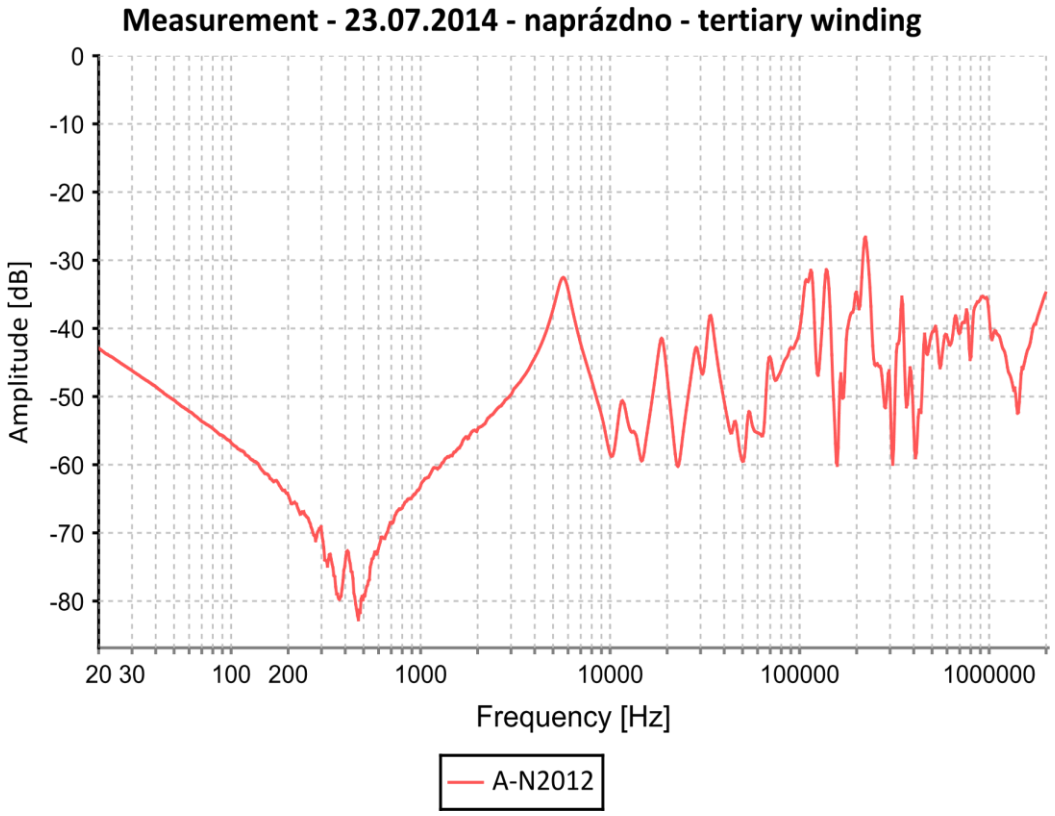
Measurement: 23.07.2014 open circuit. Transformer s.n.: 095xxx								
110.0 kV primary			23.0 kV secondary			6.3 kV tertiary		
Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7		
A-N	B-N	C-N	a-n	b-n	c-n	A-N2012		

Measurement - 23.07.2014 - naprázdno - primary winding. Transformer s.n.: 095



Measurement - 23.07.2014 - naprázdno - secondary winding. Transformer s.n.: 095




Measurement - 23.07.2014 - naprázdno - tertiary winding
Amplitude [dB]
0
-10
-20
-30
-40
-50
-60
-70
-80
20 30
100
200
1000
10000
100000
1000000
Frequency [Hz]
A-N2012

Analysis result			
Combination	CCF value on interval		Analysis result
A-N	0 Hz - 2000 Hz	0.9988	Movement of the main and tap windings, ground impedances variations, axial displacement
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 400000 Hz	0.9993	
	400000 Hz - 2000000 Hz	0.9875	
B-N	0 Hz - 2000 Hz	0.9994	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 400000 Hz	0.999	
	400000 Hz - 2000000 Hz	0.9917	
C-N	0 Hz - 2000 Hz	0.9992	Movement of the main and tap windings, ground impedances variations, axial displacement
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 400000 Hz	0.999	
	400000 Hz - 2000000 Hz	0.9889	
a-n	0 Hz - 2000 Hz	0.9979	Transformer is without error
	2000 Hz - 20000 Hz	0.9999	
	20000 Hz - 400000 Hz	0.9999	
	400000 Hz - 2000000 Hz	0.9998	

b-n	0 Hz - 2000 Hz	0.9996	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 400000 Hz	0.9999	
	400000 Hz - 2000000 Hz	0.9999	
c-n	0 Hz - 2000 Hz	0.9983	Transformer is without error
	2000 Hz - 20000 Hz	0.9999	
	20000 Hz - 400000 Hz	0.9999	
	400000 Hz - 2000000 Hz	0.9999	

Measurement 27.06.2012 open circuit (transformer ID: 095xxx)

Measurement 23.07.2014 open circuit (transformer ID: 095xxx)

Analysis result			
Combination	CCF value on interval		Analysis result
A-Nssek	20 Hz - 2000 Hz	1.0	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 200000 Hz	0.9999	
B-Nssek	20 Hz - 2000 Hz	1.0	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 200000 Hz	0.9998	
C-Nssek	20 Hz - 2000 Hz	1.0	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 200000 Hz	0.9997	
a-nsprim	20 Hz - 2000 Hz	1.0	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 20000 Hz	1.0	
b-nsprim	20 Hz - 2000 Hz	1.0	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 200000 Hz	1.0	
c-nsprim	20 Hz - 2000 Hz	1.0	Transformer is without error
	2000 Hz - 20000 Hz	1.0	
	20000 Hz - 200000 Hz	1.0	

Measurement 27.06.2012 nakrátko (transformer ID: 095xxx)

Measurement 23.07.2014 nakrátko (transformer ID: 095xxx)