Prakash Sapkota

Action-Based Study and Development of a Web Service Application in Java for METLA

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

30 January 2014

Helsinki Metropolia University of Applied Sciences

Author	Prakash Sapkota
Title	Action-Based Study and Development of a Web Service Application in Java for METLA
Number of Pages	38 pages + 4 appendices
Date	30 January 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Mika Galkin, Senior System Analyst Sami Sainio, Lecturer

The primary purpose of the thesis project was to carry out an action-based study of web services by developing a forestry related web service application for MetINFO.

MetINFO is an information division of the Finnish Forest Research Institute (METLA). It provides various forest-related information services and tools in order to make forest-related information more visible and useful. The goal of the project was to develop a web service application which could be used by Finnish sawmills to upload their roundwood sales data to MetINFO. The uploaded data is used to calculate statistics about roundwood sales in Finland by different forestry centers and price areas.

The development of the project involved various steps. Initially, the requirements of the application were analyzed. Based on the requirements, the application was designed and developed using feature-driven development methodology. As the outcome, fully functioning web services for uploading roundwood sales data and a web based application for administering uploaded data were created.

The developed application was tested in a test environment and all the known bugs were fixed. Technical documentation of the application and a user manual were created and the application was deployed in the production environment. Finally, the application was demonstrated to MetINFO and the Finnish sawmills were informed about the service by MetINFO.

Keywords	METLA, SOA, Web services, RESTful services, Java, Spring Framework, Apache CXF, roundwood sales data

Contents

Appendices

Abbreviations and Terms

AOP	Aspect-oriented Programming
API	Application Programming Interface
ASP	Active Server Pages
CORBA	Common Object Request Broker Architecture
DOM	Document Object Model
FDD	Feature Driven Development
FTP	File Transfer Protocol
GML	Geography Markup Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JAX	Java API for XML
JAXB	Java API for XML Binding
JAXP	Java API for XML Processing
JDK	Java Development Kit
JEE	Java Enterprise Edition
JME	Java Mobile Edition
JRE	Java Runtime Environment
JSE	Java Standard Edition
JSON	JavaScript Object Notation
JSP	Java Server Page
JVM	Java Virtual Machine
METLA	Metsäntutkimuslaitos (Finnish Forest Research Institute)
MVC	Model View Controller
OASIS	Advancing Open Standards for the Information Technology
ORM	Object Relational Mapping
PHP	Hypertext Preprocessor
POJO	Plain Old Java Object
REST	Representational State Transfer
RDF	Resource Description Frameworks
ROA	Resource Oriented Architecture
RPC	Remote Procedure Call
RSS	Rich Site Summary
R&D	Research and Development

SAAJ	SOAP with Attachments API for Java
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SVG	Scalable Vector Graphics
UC	University of California
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WS	Web Service
WSDL	Web Service Description Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

1 Introduction

Service-Oriented Architecture (SOA) is a software architecture design pattern that supports self-contained and loosely coupled interaction between various software components. The main aim of SOA is to enable easy mutual data exchange between software components without having to worry about the platform in which the software components are developed or deployed. Web service is the technology which can be used to implement applications and services in SOA. Because of interoperability needs between different platforms and the availability of development tools and technologies, web service is widely adopted in versatile applications and services use cases. For example, large vendors such as Google, Amazon, IBM, Oracle, and Microsoft have implemented their various applications and services using web service. In this context, the primary motivation toward the project was to explore service-oriented architecture and its implementation using web services.

The company, METLA, in which I carried out my thesis project, is a forest research institute in Finland. MetINFO is one of the METLA divisions which provides information and statistics services to METLA. As forestry is one significant part of the Finnish economy, knowledge and research data about forestry becomes crucial for forestry stakeholders such as forest owners, investors, and paper industries.

In the project, MetINFO wanted to create a service which could be used by Finnish sawmills to submit their roundwood sales data to MetINFO. This data would be then used to produce statistics about roundwood sales in Finland. The service had to be developed in such a way that a software client to the service could be developed in any computer platform or in any computer programming language. In other words, the service had to be implemented in a service-oriented architecture. In addition to the main service, MetINFO wanted to create a web application that would have different features to monitor and administer the service, incoming data, and application users.

Analyzing the requirements from MetINFO, two different kinds of applications had to be developed. The first was a web service application. For the web service application, both REST and SOAP-based web services were used. In order to implement a REST based or RESTful service, Spring MVC REST was used. In order to implement a SOAP-based service, Apache CXF was used. The second kind of application was a

normal web application with a user interface. For this kind of application, Java EE and Spring MVC were used.

The project had two goals. The first goal was to provide a solution for METLA, which itself has significance to various forestry stakeholders. The second was to gain an understanding of different software development technologies such as web services, Java, Spring Framework, and other software development tools and methodologies by using them in a real world application.

2 Theoretical Background of the Technologies Used

2.1 Markup Languages and XML

A markup language is a system or a computer language for annotating a document in such a way that will be syntactically different from the text. Generally, in markup languages, documents are annotated by tags. GML, SGML, HTML, XHTML, XML are some of the markup languages. HTML is one of the most widely used markup languages as it is used in document formats for the World Wide Web. Similarly, XML is also another widely used markup language which is the foundation of various other technologies.

XML stands for Extensible Markup Language. It is a platform-independent way to represent data. In other words, any data encoded in XML can be read by any application on any platform. The specification for XML is defined by W3C. The specification contains a set of rules for encoding a document in a format that can be both machine- and human-readable. XML does not have a fixed collection of markup tags, but it defines rules to create tags. As users can define their own tags in a logical structure, XML can be used as the foundation of other various document formats and markup languages. The main design goals of XML focus on usability, extensibility, simplicity and generality over the Internet. [1]

An XML document is an ordered, labeled tree. XML is composed of character data type or leaf nodes that contain the actual data (text strings) and element nodes. The element nodes are also called element types. The element nodes can have a set of attributes, each consisting of a name and a value. They can also have child nodes. [2]

Due to the extensibility nature, XML has been used to develop several technologies such as RSS, Atom, SVG, and RDF. The web service is also a technology whose building block is XML. Several components of the web service technology, such as SOAP, WSDL, and UDDI, are built based on XML. [2]

2.2 SOA and Web Services

In today's era of the Internet, the majority of software applications are distributed, meaning that they are required to communicate between applications over the network. SOA is an architectural style for building software applications in a network environment, which provides platform-independent communication between interacting parties. It is a flexible and highly dynamic architecture. Moreover, it is a process-driven model allowing businesses to adapt quickly to changing market conditions. In SOA, applications and services are loosely coupled and highly interoperable. [4]

Figure 1 shows the basic architecture of SOA.

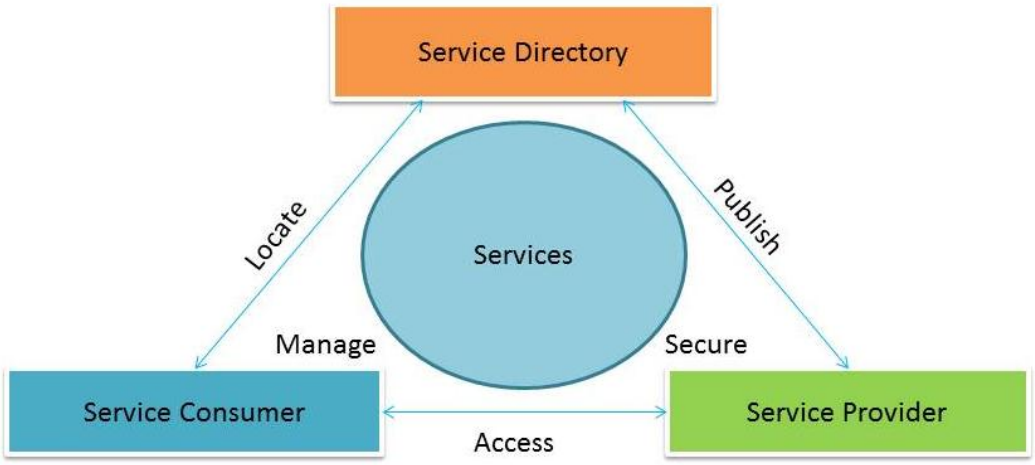

Figure 1. Basic service-oriented architecture.

As shown in figure 1, SOA contains three components, namely service provider, service directory, and service consumer. The service provider publishes a service to the service directory following the service standards. The service consumer then locates

the service provider from the service directory, accesses the service from the service provider, and finally binds to the service. [3]

The SOA model has several benefits over the traditional model. Firstly, as software components can be reused in the SOA model, there will be a lower software development cost. Similarly, because of the model-driven implementation in the SOA model, it becomes easy to develop new functions rapidly. Moreover, data confidentiality and integrity, organizational flexibility, ability to expose internal functionality, ability to integrate existing assets are some other benefits of the SOA model to list a few. [5]

As mentioned previously in the introduction section, web services are the building block of the SOA model implementation. Especially during recent years, the imperative to connect processes, systems, applications and people has changed the way applications are being developed. Successful software systems require interoperability across platforms and flexible service that can be easily maintained, modified and evolved over time. This has led to the acceptance of XML as a universal language for formatting, representing, storing and transmitting machine-readable structured data which is independent of the hardware, software platform, and programming language. Building on the wide acceptance of XML, web services are computer applications which use standard protocols to exchange data over a network. [6]

W3C defines a web service as a software system designed to support interoperable machine-to-machine interaction over a network. The web service has an interface described in a machine-readable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP-messages, typically using HTTP with an XML serialization in conjunction with other web-related standards. In other words, web service is a method or system of data exchange between two different systems or processes over the World Wide Web. A web service can be defined as a software functionality or feature provided on a network address which can be utilized by other computer systems over the Internet regardless of the computer language, software platform, and hardware used in that system. [7]

Generally, there are two major classes of web services. One of them is a SOAP-based web service which is simply known as ‘web service’. Another kind of web service is a REST-compliant web service. It is also known as RESTful web service. In the following

sections, only the SOAP-based web service will be discussed. The RESTful web service will be discussed in section 2.3. [8]

Web services provide many advantages for companies and software developers. Some of the key advantages are listed below:

- Interoperability - because web services follow the use of standard-based communications methods, all web services are interoperable with each other. This means that a web service developed in any hardware, software platform or programming language can communicate to another web service developed in any hardware, software platform or programming language. This is one of the key advantages of web services. In this sense, web services are virtually platform-independent. [9]

- Reusability - loose coupling, autonomy, statelessness, granularity and discoverability are the core features of web services. Application of these features enables web services to be reusable. [10]

- Usability - web services use open standards such as SOAP, WSDL, UDDI, and XML. Not only that, but also web services are enabled and supported by some of the widely adopted software platforms such as Java, .NET, and PHP. Because of these reasons, web services are useable for all kinds of companies and also for a variety of application use cases.

- Based on SOA - web services are specified and standardized for the entire stack of service-oriented architecture.

On the other hand, web services in some use cases have some disadvantages. Some of the main disadvantages of web services are listed as follows:

- Heavyweight - as a web service uses only XML, all of the operations of the web service such as discovering a web service, binding a web service, authentication, and exchanging core data to another service require XML processing. Because of that web services are too heavyweight for most of the real-world applications on the web.

- Difficult to a create client in the browser - even though many tools and technologies are available for several software platforms to create a client of the web service in an easy way, creating a web service client in browsers generally requires enormous work. Therefore, web services do not seem appealing for use with web-based client side applications.

To sum up the advantages and disadvantages, web services seem to be weakly suited for client-to-business solutions, whereas best suited for legacy enterprise systems involving business-to-business communication.

In order for any software technology to be a feasible solution, it has to be supported by available software platforms. The web service is one of the software technologies which is best supported by major software platforms such as Java, .NET, and PHP. For example, Java standard and enterprise edition provide different specifications for developing web services. JAX-WS, JAXB, JAX-RPC, and SAAJ are some of the specifications related to web services and XML processing by Java. Java also provides implementations of those specifications in various distributions such as Java Standard Edition (JSE) distribution, Java Enterprise Edition (JEE) distribution and Glassfish application server.

The web service technology itself is a combination of various specifications. We can view the web service technology as a stack of various components. Each components of the web service plays a certain role in the web service. The web service stack can be visually represented by figure 2.

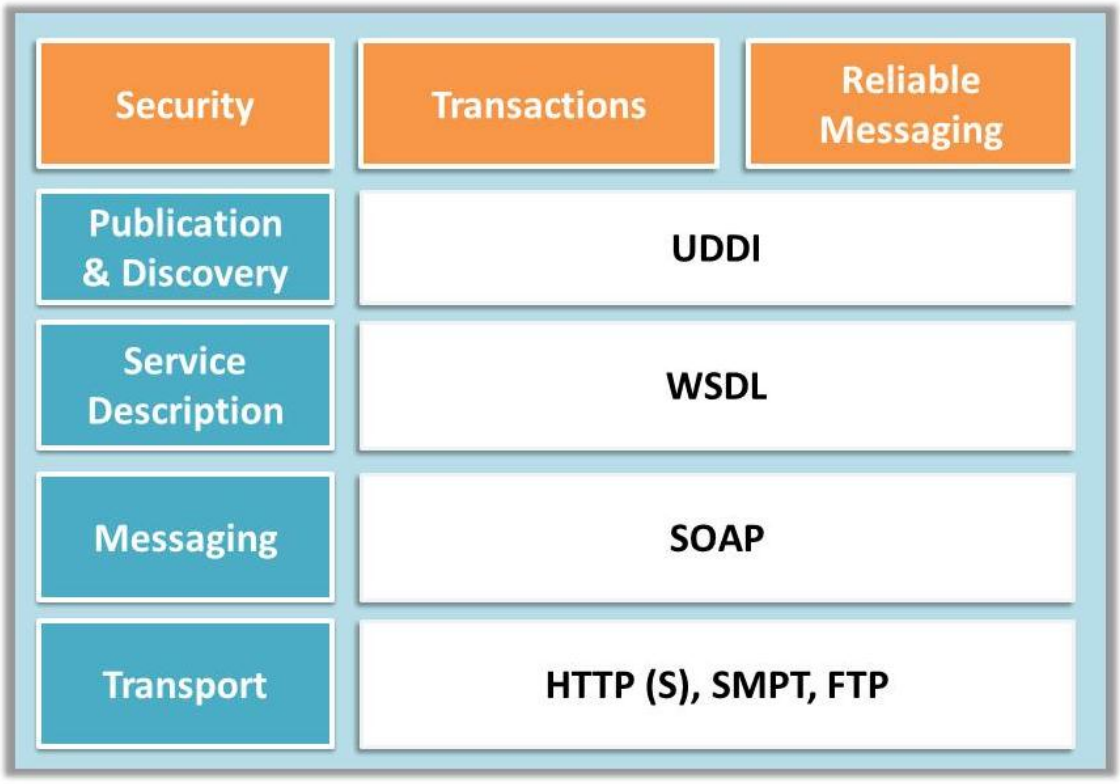

Figure 2. An example web service stack showing the relationship between web services standards. Reprinted from MC Press online [11].

From figure 2, the bottommost component is a transport layer. On top of the transport layer, a messaging layer exists. Similarly, on top of the messaging layer, service description and on top of the service description, publication and discovery layer exists. Finally, the web service quality layer exists on top of the publication and discovery layer. Web service quality includes specifications related to security, transactions, and reliable messaging. More details about each web service stack are given below.

2.2.1 Web Services Transport

The web service transport layer is the lowest layer stack on the web services. It is responsible for the communication of data between machines over a network. Web services must be network-accessible to be invoked by a service client or service requestor. Web services which are publicly available on the Internet use commonly used network protocols. Because of widespread use, HTTP is the de facto standard network protocol for web services that are Internet-available. Other Internet protocols including

SMTP and FTP can also be used by web services as a network protocol. Additionally, some Intranet domains can use proprietary or platform and vendor specific protocols such as CORBA and MQSeries. The specific use of the network protocol used in a given use case mainly depends upon the requirements of application such as availability, reliability, security, and performance. This enables web services to take advantages of the existing network infrastructures and message-oriented middleware infrastructure, such as MQSeries. [8]

Web services provide a unified programming model for the development and consumption of a private Intranet as well as public Internet services. This is one of the benefits of web services. If an enterprise uses multiple types of network infrastructures, HTTP can still be used as a common, interoperable bridge to connect disparate systems. As a result, the decision of network technology can be totally transparent to the developer and consumer of the service. [8]

2.2.2 Web Services Messaging

The web services messaging layer works on top of the transport layer. It is the core layer of web services. It defines the rule or syntax of the data to be communicated. Similarly, this layer contains the real data to be communicated.

The web services messaging layer uses XML-based SOAP protocol. Based on W3C's definition, SOAP is an XML-based lightweight protocol for exchange of information in a decentralized, distributed environment. It consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. The current version of the SOAP is 1.2. [13]

As mentioned in previous paragraph, SOAP consists of three parts which are listed below in more detail:

- The SOAP envelope - this is the core part of SOAP. It defines an overall system for expressing **what a** message contains, **who** should deal with it, and whether it is mandatory or optional [13].

- The SOAP encoding rules - this defines a serialization mechanism that can be used to exchange instances of application-defined data types [13].

- The SOAP RPC representation - this defines a convention that can be used to represent remote procedure calls and responses [13].

A visual representation of SOAP envelope is shown in figure 3 below.

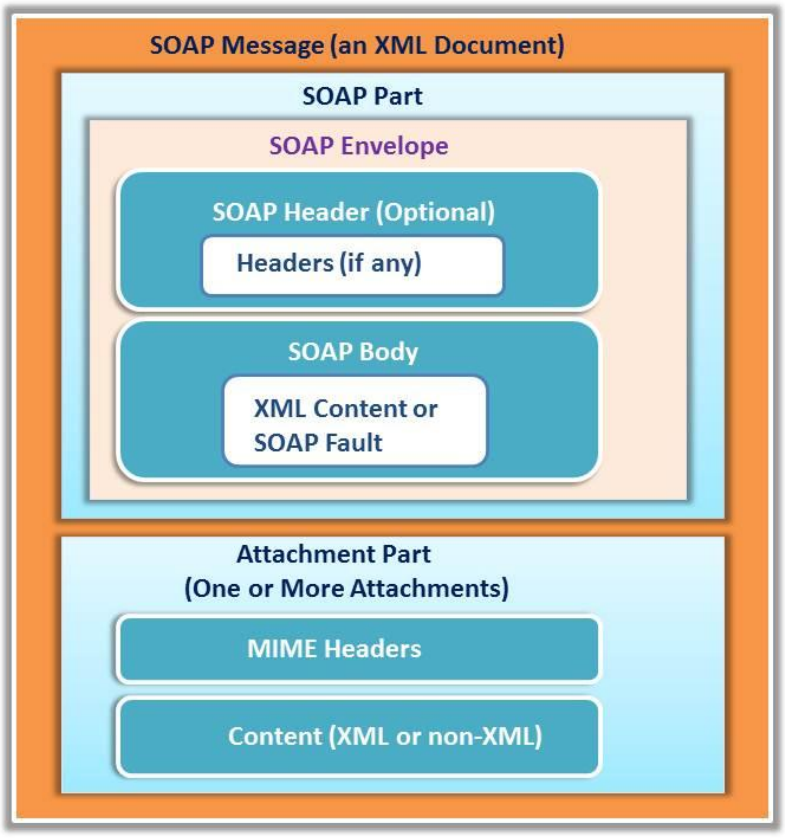

Figure 3. SOAP message with an attachment object. Reprinted from Oracle Java tutorials website [12].

Based on figure 3, the SOAP envelope part contains two parts, namely header and body. The header part contains information such as control information and quality of service. This part is an optional part of the SOAP envelope. Header blocks are targeted at receivers in the SOAP message path. The SOAP body contains application (business) data. Also, it may contain fault information if some error occurs in the system. It is a mandatory part of the SOAP envelope. The body part is targeted at the ultimate

receiver in the SOAP message path. Besides the SOAP envelope, the SOAP message may also contain additional XML or non-XML attachments.

2.2.3 Web Services Description

Web service description is the layer on top of the web service messaging layer in the web service stack. It provides information about the service such as what information to send to the service, what information the service is going to send back, and where to find the service in the first place. Web services use WSDL as a description language.

WSDL is the core technology of web services description. It is based on XML. W3C defines WSDL as an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The current version of WSDL is 2.0. [14]

A WSDL document contains five main segments namely types, messages, port types, bindings, and services. The types segment contains information about the type definitions of the data being involved in the service using some type system such as XSD. Similarly, the messages segment contains information about the abstract and typed definition of data being communicated. The port types segment contains information about abstract set of operations supported by one or more service endpoints. Again, the bindings segment contains information about the concrete protocol and data specification for a port type. Finally, the services segment contains information about the concrete specification of service endpoints. [14]

The main benefit of WSDL is that it is based on a standard and it is machine-readable. Without WSDL, a service calling syntax must be determined from a documentation that is provided, or by examining wire messages. With the use of WSDL, it is possible to generate proxies of web services in an automated way in a truly language- and platform-independent way. This makes any software platform to create tools or libraries for generating a web service client in an automated way.

2.2.4 Web Services Publication and Discovery

This is another layer of the web services stack. This layer provides specification about ways to publish a web service in a network and ways to search for those published services. In most common use cases, there is a web service provider which publishes a service and a service consumer which uses this service. Both the service provider and service consumer use standards to publish and search for web services.

2.2.5 Web Services Quality

Specifications of the web services quality layer are generally related to non-functional aspects of web services such as reliability, transaction, and security. WS-Reliability, WS-Transaction, and WS-Security are the three most common specifications for this layer.

WS-Reliability-related specifications define ways to exchange a SOAP message with guaranteed delivery and guaranteed message ordering. They are defined as SOAP header extensions and are independent of the underlying protocols [15]. Similarly, WS-Transaction defines mechanisms for the transactional interoperability between web services and provides transactional qualities. Mainly, it defines mechanisms which enable all the parties of web service to achieve mutually agreed outcome. Web services transaction related issues are supported by three different specifications, namely WS-Coordination, WS-AtomicTransaction, WS-BusinessActivity. [16]

Finally, regarding WS-Security, it is enhancement to SOAP messaging to provide quality of protection through integrity, confidentiality and single authentication. WS-Security specification provides support for end-to-end message security. It is implemented by adding a security header in SOAP messages. The security header is composed of three parts: security token, signature directives, and encryption directives. In the security header, the security token is for providing authentication information, signature directives are for digital signature information, and encryption directives are for encrypted data. [17]

2.3 Resource-Oriented Architectures and RESTful Services

A resource-oriented architecture (ROA) in software engineering is an architectural style of the software design and programming paradigm for developing software in the form of resources [19]. In the context of ROA, resources may refer to any entity that can be identified and assigned a uniform resource identifier (URI). Those resources within the ROA concept include not only IT infrastructure elements such as servers, computers and other devices, but also web pages, scripts, JSP/ASP/PHP pages, and other entities such as database tables. [20]

In other words, ROA is based upon the concept of resource and each resource is a directly distributed component which is handled through a well-defined standard, common interface [18]. Roy Thomas Fielding's doctoral dissertation 'Architectural Styles and the Design of Network-based Software Architectures' [21] outlines four essential concepts underlying the resource-oriented architecture, which are given as follow:

- Resource - Anything which can be identified and assigned a URI
- Resource name - Unique identification of the resource
- Resource representation - Useful information about the current state of resource
- Resource interface - uniform interface for accessing the resource and manipulating its state [21]

Similarly, the same doctoral dissertation of Roy Thomas Fielding summarizes four properties of ROA, which are given as follow:

- Addressability
- Statelessness
- Connectedness
- A uniform interface [21]

In ROA, the resource interfaces are based on the HTTP operations. Table 1 summarizes the resource methods and how they could be implemented using the HTTP protocol.

Table 1. Resource interfaces of ROA, their short descriptions and their corresponding HTTP operations.

Resource method	**Description**	**HTTP operation**
Create	Creates a new resource and the corresponding unique identifier	PUT
Retrieve	Retrieves the representation of the resource	GET
Update	Modifies the resource	POST
Delete	Deletes the resource (and, optionally linked resources)	DELETE (only referred resource)
Get meta information	Obtains meta information about the resource	HEAD

Based on the principles of ROA, Representational State Transfer (REST) is a lightweight architectural style for exchanging structured information in a decentralized and distributed environment. The term '*representational state transfer'* was initially introduced by Fielding [21]. The initial term '*representational'* refers that the service manipulates the client state by sending resource representations. The remaining term '*state transfer'* refers that a client manipulates service resources by sending state transferring representations. [22]

As in SOA-based web services, the RESTful service is also composed a stack of different layers. A visual representation of the RESTful service stack is shown in figure 4.

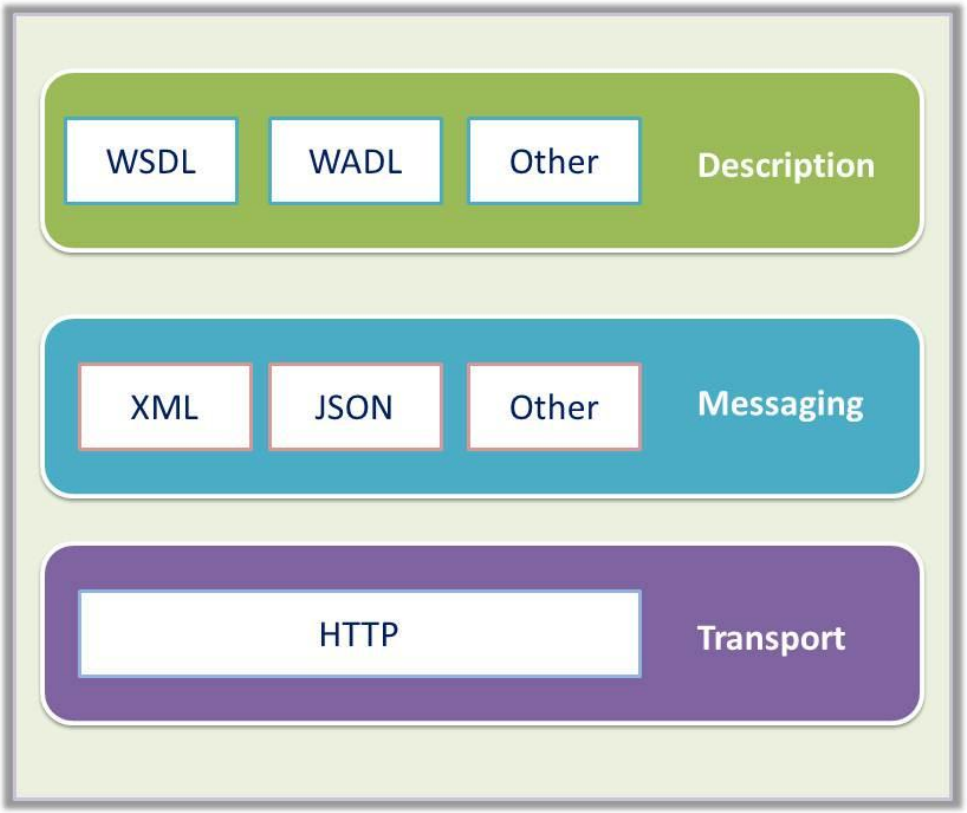

Figure 4. RESTful service stack. Reprinted from a lecture slide by Peeter Kitsnik [22].

As shown in figure 4, the bottommost layer of the RESTful service stack is the transport layer. The transport layer is responsible for data communication on the network. The HTTP protocol is used for the data communication on the network in the RESTful service. Similarly, on top of the transport layer, the messaging layer exists. Messaging layer is responsible for application data communication. Any data formats such as XML, JSON can be used in this layer for data communication. Finally, the description layer is responsible for the description of the RESTful service. WADL is a widely used protocol for this layer. However, some other technologies such as WSDL or even normal documentation may be used for the description of the RESTful service. [22]

Considering the advantages of the RESTful service, a lightweight alternative for service-oriented-architecture may be the top advantage. Similarly, the REST approach reduces complexity in development and deployment of services. Additionally, the RESTful service is naturally oriented to web resources. Last but not least, another advantage of RESTful service is that it can be easily consumed from web-based client applications. For example the Ajax technology can be perfectly used as a REST service client. [22]

Now considering the drawbacks, the RESTful service is not an industry standard; it is just an architectural style. Similarly, quality and composition of services are not directly considered in the RESTful service. Moreover, it is not suitable for exposing an existing procedure-oriented enterprise system. [22]

Analyzing the advantages and disadvantages of the RESTful service, it can be concluded that the RESTful service is best suited for user-to-business solutions on the web. However, it is less suited for business-to-business solutions.

2.4 Java Platform

Java Platform is one of the software development and deployment platforms. It is a set of several computer software products providing a system for developing application software and deploying it into cross-platform computing environment. As of today, the Java platform is used in wide variety of computing platforms from low-end devices such as smart phones, embedded devices to high-end devices such as supercomputers and enterprise servers. [23]

The Java platform consists of three components: Java Runtime Environment (JRE), Java language compiler and libraries. The JRE provides the Java Virtual Machine (JVM), basic language libraries, and other components required to run applets and other components in the Java programming language. In order for programs to run in the JVM, first they must be compiled into Java bytecode, a standard portable binary format which typically is in form of *.class* file (Java class file). A Java program may consist of many such Java class files. Java provides a standard system for packaging those multiple class files into a single archive file. Depending on the type of a Java program, the archive file extension may be .jar, .war, .ear, or some other extension.

Similarly, the Java compiler is a compiler for the Java programming language. The Java compiler compiles Java source files and produces platform independent byte code. Then the bytecode is run on the JRE. As part of the Java platform, there exist several other JVM languages such as Scala, Clojure, Groovy, and Jython. However, as of today, the Java programming language is the most commonly used programming language in the Java platform.

Another important part of the Java platform is Java libraries. In addition to the core language libraries, Java provides several other libraries required for different kinds of devices and applications. Similarly, in addition to the official libraries provided from Oracle, Java also has very large base of open sources libraries for variety of devices and applications.

Java, as a language, is a general purpose concurrent and class based object oriented computer programming language. The main design goal of Java was to have as few implementation dependencies as possible. One of the main principles of Java is “Write once and run anywhere”. As of today, Java is also considered as one of the most popular computer programming languages in the world. Initially, Java was specifically designed for consumer electronic devices programming. However, after several years, Java has been adopted in all kinds of devices such as enterprise servers, personal computers, mobiles, tablets and other embedded devices. In order to address the specific needs of different kinds of devices and applications, Java releases different distributions of Java which are listed below: [24, 23.]

- JavaFX **-** it is targeted for creating and delivering rich internet applications that can run across wide variety of devices.

- Java Card **-** it is the tiniest of the Java platforms targeted for embedded devices. It is mainly used to run Java applications securely on smart cards and similar other small memory footprint devices.

- Java Mobile Edition (JME) - it is targeted for low resource electronic devices such as embedded devices, mobile phones etc.

- Java Standard Edition (Java SE) - it is targeted for developing applications for general purpose computers and servers.

- Java Enterprise Edition (JEE) - it is targeted for developing enterprise scale applications running in servers. It uses the Java SE as the foundation for its APIs. It provides several specifications related to such as web applications, XML processing, web services, and the RESTful services.

2.4.1 JAX-WS and Apache CXF

Java API for XML Web Services (JAX-WS) is a part of the JEE specifications which defines programming APIs for building web services and clients that communicate using XML. Using JAX-WS, it is possible to develop both the message-oriented as well as the Remote Procedure Call-oriented (RPC-oriented) web services. JAX-WS uses the technologies defined by the W3C such as HTTP, SOAP, and WSDL. [25]

The main benefit of JAX-WS is that it hides most of the complexity related to a SOAP message and a WSDL document from application developer and provides an out-of-box service for creating a web services and a web service client. On the server-side, application developer specifies web service operations by defining methods in an interface written in the Java Programming language and the developer also codes one or more implementations for those interfaces. The JAX-WS runtime and the libraries automatically generate a WSDL file and create necessary parts of the web service to handle a request from a client. On the client-side, it is also possible to automatically generate a proxy class (a local object representing service) using JAX-WS related Java tools and then the web services can be called just by invoking methods on the proxy. This way, the developer does not have to generate or parse the SOAP message. Similarly, neither the developer has to parse or generate the WSDL file. [25]

JAX-WS itself is built on the foundation of several other Java specifications. In addition to the Java core libraries, JAXP and JAXB are two main specifications on foundation of which JAX-WS is built. JAXP is a Java API for XML processing. It provides APIs for programming several XML technologies such as SAX, DOM, XSLT, XPath, XQuery, and XML validation. JAXB is a Java API for mapping and binding XML to Java objects and generating XML from Java objects.

Regarding Apache CXF, it is an open source Java based web services framework by the Apache Software Foundation. It provides implementation of the Java web service specifications JAX-WS and JAX-RS. The main feature of Apache CXF is that it supports a variety of the web service standards including SOAP, WS-I Basic Profile, WSDL, WS-Addressing, WS-Policy, WS-ReliableMessaging, WS-Security, WS-SecurityPolicy, WS-SecureConversation, and WS-Trust. Similarly, it supports a variety of frontend programming models. [26]

2.4.2 Spring Framework

Spring Framework is an open source application development framework for the Java platform. It provides a comprehensive programming and configuration model for modern java based enterprise applications. It enables to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to the POJOs. This capability applies to the Java SE programming model and to full and partial Java EE. The core features of the Spring Framework can be used in any kind of Java application. [27]

The Spring Framework has a layered architecture, which allows becoming selective about which of its components to use while also providing a cohesive framework for the JEE application development. The layered architecture of the Spring Framework is shown in figure 5 below:

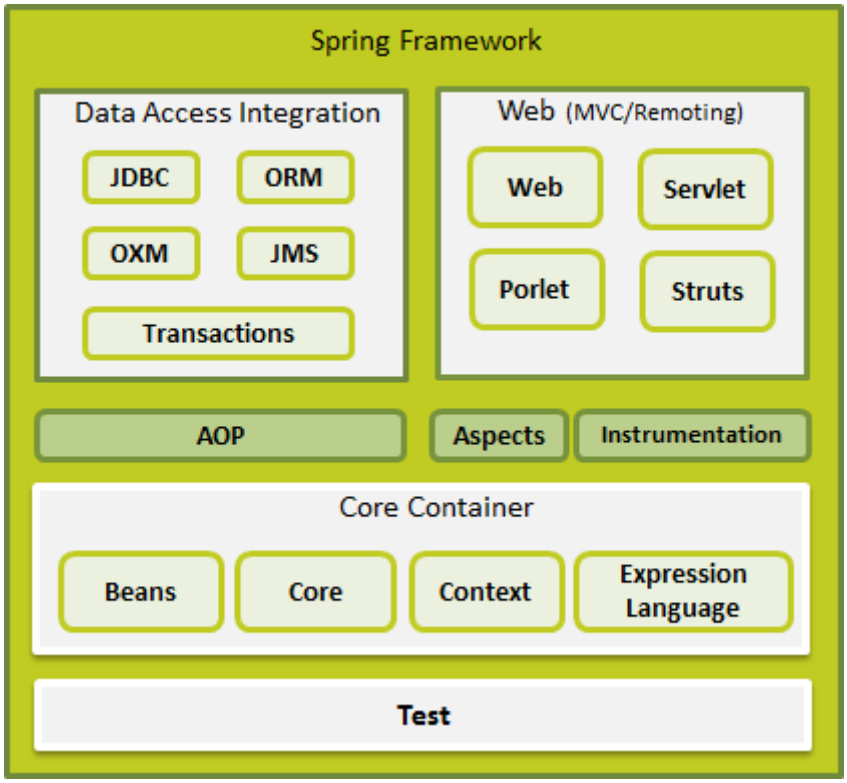

Figure 5. The Spring Framework layered architecture. Reprinted from Tutorialspoint [28].

As shown in figure 5, the component ‘Core Container’ is the core of the Spring Framework. It is Spring Framework’s inversion of control container. It is designed in such a way that it can be used with other several Java application development frameworks such as JSF, Struts, and Wicket. The web layer of the Spring Framework provides simplified programming model for developing Java web applications. One of the important parts of the web layer is Spring MVC framework. It provides the MVC model for developing web applications. Some other modules of the Spring Framework are Spring Test Framework, Spring AOP, Spring Data, and Spring Security.

2.5 Development Tools

Software development tools are the programs or the software applications that software developers use to create, debug, maintain, or otherwise support other programs and applications. IDEs, build tools, version control systems, debugging tools etc. are generally considered as development tools. There are many alternatives for development tools available in market. A short introduction of some of the commonly used development tools is given below:

Eclipse IDE

Eclipse is a multi-language software development environment. It consists of a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written in Java. It can be used to write applications in Java and, using other various plugins, other programming languages including Scala, C, C++, PHP, Perl, JavaScript and some other languages.

Apache Maven

Apache Maven is a build automation tool used primarily for Java projects, but it can also be used to build and manage the projects written in Scala, Ruby, C# and other languages. It uses a plugin-based architecture which enables the project (software) to be controllable using standard input. One important part of Maven is the dependency management. Based on XML declarations, Maven downloads required dependencies from different repositories.

Version Control System and Git

Version control, also known as revision control and source control is a management process of changes made to computer programs, documents, web sites, and other

kinds of information. In the version control, every change is usually identified by a revision number or revision code and each revision is associated with a timestamp and a person making the change. Based on the revision identifier, the changes can be reverted back, or committed to the repository, or merged to the changes made by other persons. There are many applications which implement the version control system. Git is one of the version control applications. It is free, open source, distributed and lightweight version control system.

The working principle of a normal version control system is shown in figure 6 below.

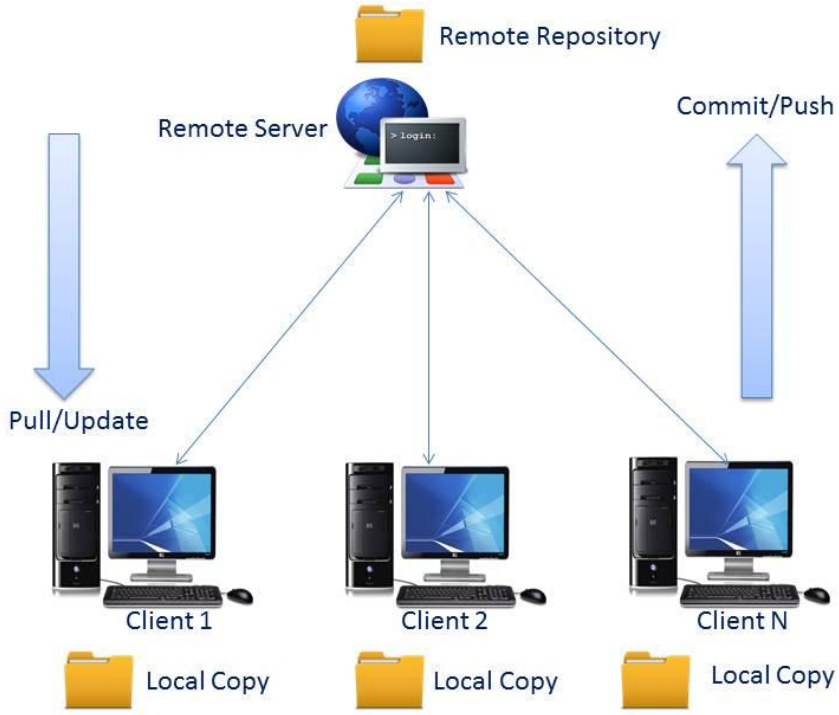

Figure 6. The version control system. Reprinted from WebHostingBillBoard [29].

From figure 6, it is seen that many persons can take the same copy of the document in different local machines from the central repository. Every person works on different copy in a local machine, but all of them commit changes to the same remote copy (repository).

3 Analysis, Development and Deployment

3.1 Requirements Analysis

As mentioned in the introduction section, the project was carried out in the MetINFO division of METLA. The MetINFO Statistics Service provides timely and comprehensive statistics on the Finnish forest sector activities. The primary goal of the project was to create a web based application which has a service where the Finnish sawmills can upload their roundwood sales data in a regular basis and thereafter MetINFO uses those data to calculate statistics which represent roundwood sales in Finland from different forestry centers and by price areas.

Analyzing the requirements of the application in surface level, the application has four basic requirements which are given in the list below:

1. Web service where the Finnish sawmills can upload their roundwood sales data in an automated way using a computer program.
2. Web based interface from where some sawmills which do not have a computer based system can upload the roundwood sales data manually.
3. Web based interfaces from which METLA data manager can monitor and administrator the application and the uploaded data.
4. Parsing the uploaded raw data and storing them in a relational database.

Analyzing the **first** requirement of the application in more detail, the web service has to be protected. Only authenticated users can upload data. Similarly, the web service can accept data only in XML format. After ensuring that the data is in XML format, the application has to check whether or not the uploaded data is well formed XML data. If the data is not well-formed XML, the request to upload data should be rejected. If the data is well-formed, it should be again checked against the XML schema provided by METLA. If the data is validated successfully against the schema, then it should be parsed and validated in the application level to check whether it contains some unexpected data. Finally, if the XML contains valid data, then the system should return a success message to the uploader or the client.

For the data analysis, one sample valid XML data which the web service could accept was provided by METLA. It is shown in appendix 1.

Analyzing the **second** requirement of the application in more detail, the web interface should have a feature to upload a file where the users can upload the roundwood sales data in XML format manually. The validation and handling process of the uploaded file should be same as in uploaded data from the web service. But, only the difference is that in the web interface, the error or success message has to be displayed in the web interface.

Now, analyzing the **third** requirement in more detail, the application should have a number of features which the data manager could use to monitor the application and the uploaded data. The features required to be developed in this part of the requirements are given below in the list below.

1. A web interface based user management system where data manager can add a new account, edit the account and change password for the account. Have two different roles *sawmill* and *admin.* Additionally, have a form based authentication and authorization system where the *admin* role is able to view all the features, whereas the *sawmill* role is able to view only the upload roundwood sales data feature.
2. Admin can view the data uploads as a list and the list contains a link to view the uploaded raw data in details.
3. Admin can remove data uploads, but system should show confirmation dialog to confirm the remove action.
4. Search data uploads by a company business ID.

Finally, analyzing the **fourth** requirement in more detail, the application has to parse the successfully uploaded and validated XML data, and the parsed data has to be stored in relational database tables in respective relational structure.

In addition to the functional requirements, the application has some technical requirements which are given below:

1. The application should use Java as programming language.
2. The application should be deployable on the Tomcat 6 or later versions.
3. The application should use HTTPS for the data communication.
4. All the data being involved in the application, the application files, and the web pages should be encoded in UTF-8.

5. The application should use MySQL database for storing the data.

3.2 Design of the Application Architecture

After the analysis of the functional requirements, the following application components were needed to be developed:

- Web service module
- RESTful service module
- MVC application module having different web interface based features
- Form based and basic HTTP authentication and authorization module
- Other helper components such as domain model, database related services, validation module, utility classes, etc.

Based on the required application components, the following application frameworks were used:

- Spring Framework core as bean container of the whole application
- Spring Security Framework for form based and basic HTTP authentication and authorization
- Spring Framework MVC REST support for RESTful service
- Spring MVC for MVC application module
- Apache CXF implementation of JAX-WS for web service module
- Hibernate for database related services

Following the recommended programming model and the best practices for the frameworks used, an overall architecture of the application was designed which is shown in figure 7.

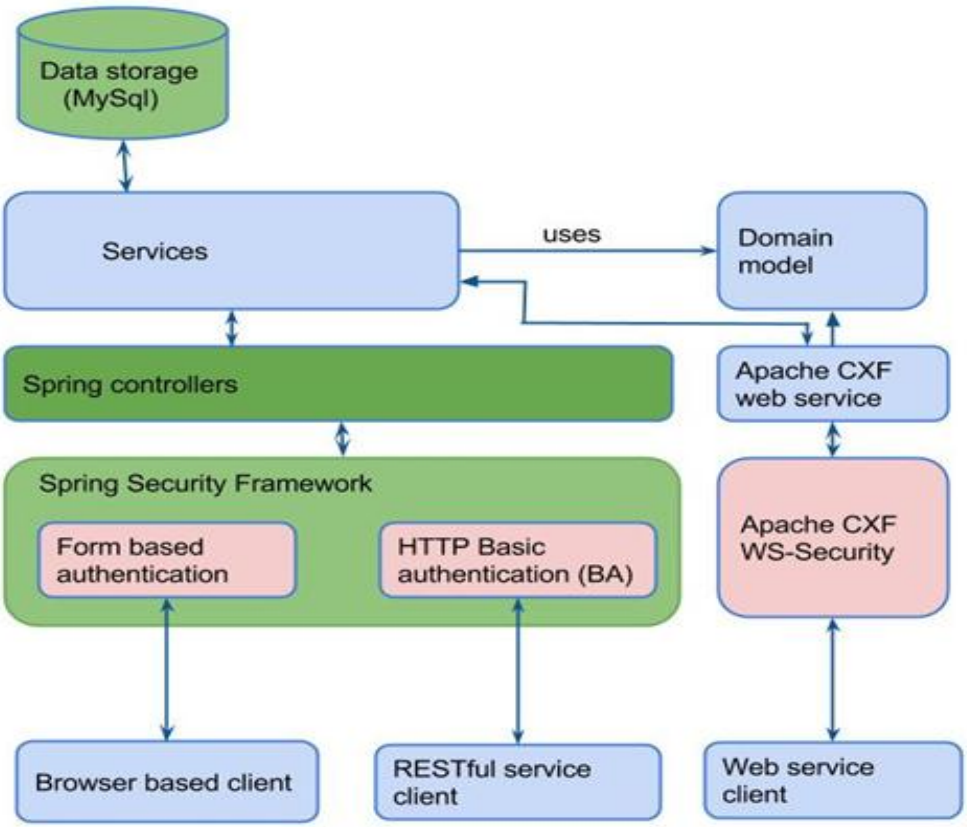

Figure 7. Overall architecture of the application.

As shown in figure 7, the bottommost layer of the application is client layer. In the application, there are three different kinds of clients. The first is a browser based web client. It contains user interfaces for several application features required for the sawmills and the data managers. The other two types of the clients are RESTful service client and web service client. Those clients are the programs to be created by the sawmills.

The browser based client programs interact to the Spring controller classes. Each interaction is secured by the form based authentication of the Spring Security Framework. Similarly, the RESTful service client interacts to the Spring REST controller. In this interaction, security is handled by the HTTP basic authentication (BA) of the Spring Security Framework. The web service client interacts to the Apache CXF web service. The security is handled by the Apache CXF WS-Security implementation.

The Spring controller and the Apache CXF web service modules handle requests by the respective clients. They use the service module and the domain model for various

operations such as reading data from the database, inserting data to the database, data processing, and data binding. The database operations are handled by Hibernate ORM. The domain model works as an entity model for Hibernate for the database related operations. The same domain model also works as the model for Java and XML binding (for JAXB). Using the JAXB, the XML file uploaded by clients can be automatically bound to the Java object model without having to parse XML manually by the application.

3.3 Development Environment and Project Setup

After analyzing the frameworks and the technologies to be used in the project, the following programs were installed in the development machine as development tools and test runtime environment:

- Java Development Kit (JDK)
- Eclipse IDE for Java EE Developers
- Apache Maven
- Git
- Maven and Git plugin for Eclipse
- MySQL server
- Apache Tomcat server

After installing the development tools and the test runtime environment, creation of the base project was started. Initially, a *maven project* type was created by opening Eclipse and going to create new project option. The Eclipse ‘New Maven Project’ window is shown in figure 8.

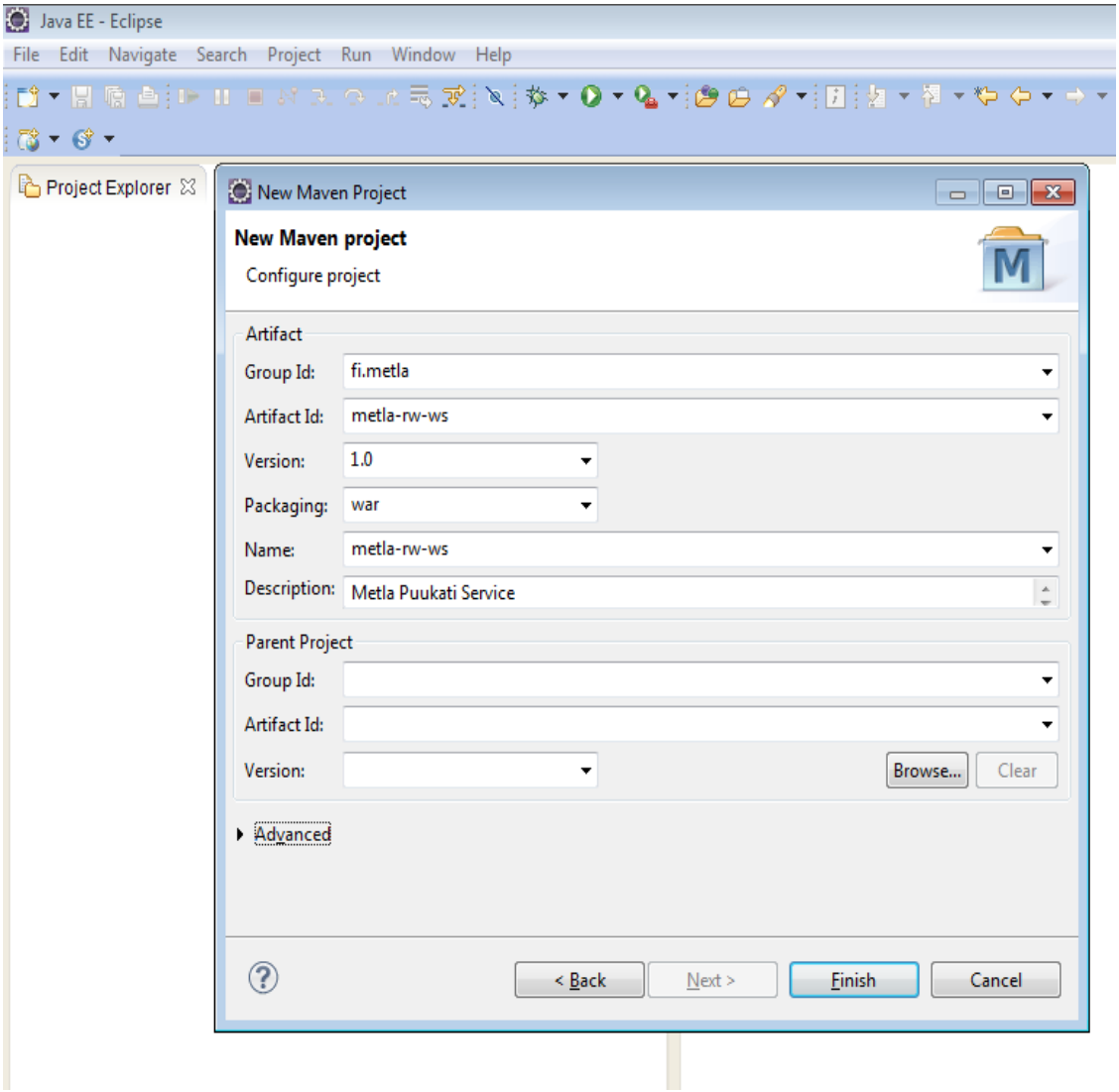

Figure 8. Snapshot of the Eclipse 'New Maven Project' window used to create the new maven project.

As seen in figure 8, the *Group Id* field refers to the base java package (base folder) for the java source files. Similarly, the *Artifact Id* refers to the name of the project or the root folder. It is similar to the *Name* field.

After creating the basic *Maven project*, some configuration files, language files, resources files etc. were added and the necessary folder structures were created. Then, a remote Git repository was created and all the local files and the folders were committed and pushed to the remote repository. The view of the folder structure and the Maven project file (pom.xml) is shown in figure 9.

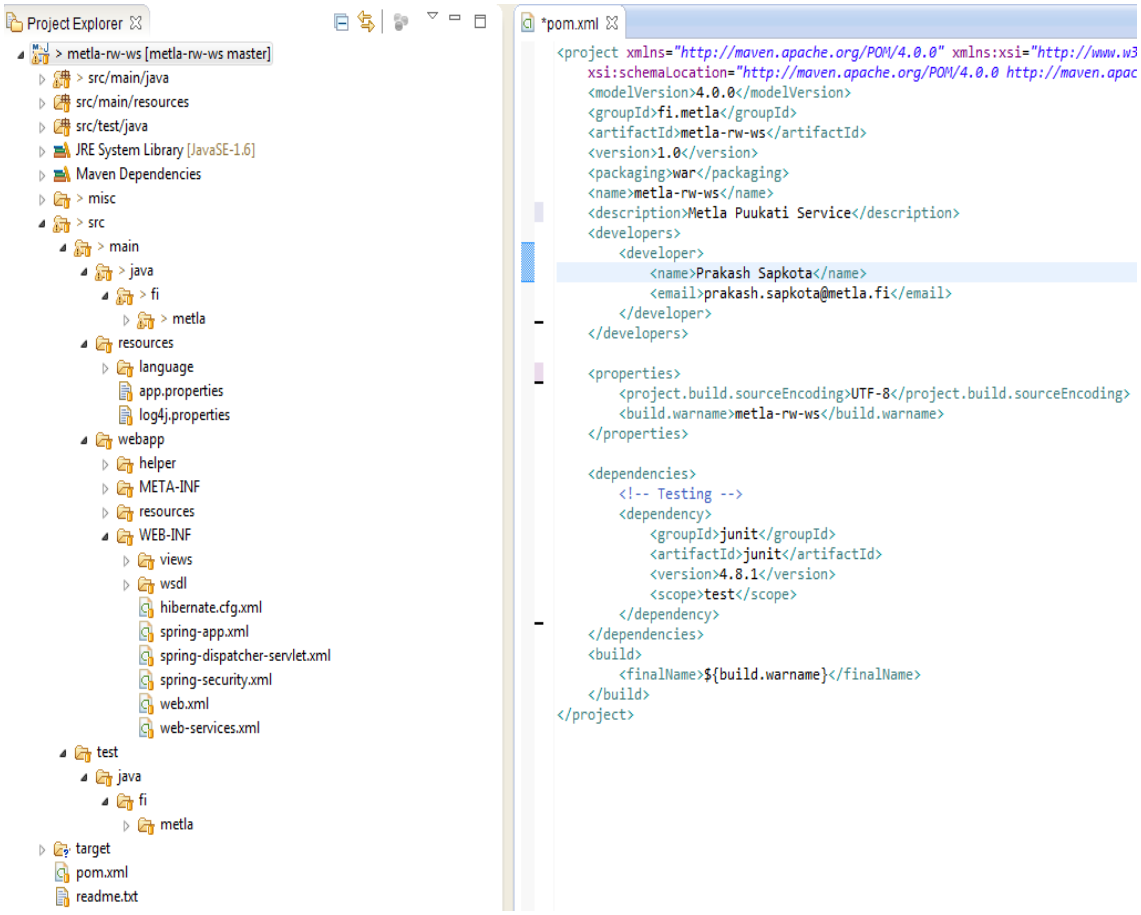

Figure 9. Snapshot of the folder structure, configuration files and Maven project file (pom.xml) of the created base project.

Shown in figure 9, metla-rw-ws is name of the root folder of the project. Under the root folder, the file pom.xml is Maven Project description file. Build related information such as project properties, plugins, dependencies, and library repositories are configured in this file. Similarly, the folder *target* under the root folder contains all the compiled source files and the *war* file that is built. The folder *src* under the root folder contains all the application source files, configuration files and resource files. The *webapp* folder under the *src* > *main* folder contains web application related source files such as JSP, HTML, and JavaScript files arranged and put under different folders. The XML files under the folder WEB-INF are application configuration files required for the JEE and the other frameworks Hibernate, Spring Security, Spring Core, and Apache CXF.

3.4 Web Service Development

After setting up the development environment and the application base project, the actual implementations of the requirements was started. The implementation of the features was done following feature-driven development methodology. Feature-driven development (FDD) is an interactive and incremental software development methodology. It is one of the agile methods for developing software. This methodology is based on functionality (feature) perspective. The main purpose of this methodology is to produce and deliver working software functionalities repeatedly in a timely manner. Following this methodology, the first feature to be developed was the SOAP based web service.

In order to develop the web service, firstly web service specification or interface was created. The source code of the interface is given below in listing 1.

```
@WebService(name = "PuukatiService")
public interface PuukatiService {
   @WebMethod
   @WebResult(name = "Message", targetNamespace = "")
   public String uploadWoodSalesData
          (@WebParam(name =   "RoundWoodSalesData")
          @XmlElement(required = true)
          RoundWoodSalesData roundWoodSalesData);
}
```

Listing 1. The web service specification/interface source code.

As illustrated in listing 1, the annotation *@WebService* tells the JAX-WS runtime that it is the class or the interface for the web service. Similarly, the annotation *@WebMethod* tells the runtime that it is the web service operation. By using the *@WebResult* annotation, it is possible to specify the element name of response message. In this web service, the web service has one parameter, which is the whole roundwood sales XML data. This data is represented by the Java class *RoundWoodSalesData*. The *@WebParam* of the *RoundWoodSalesData* type annotated by *@XmlElement* tells the runtime that inside the SOAP envelope, there should be respective XML data represented by class the *RoundWoodSalesData.* With this declaration, web service runtime

parses the XML inside the SOAP envelope, and if it is valid roundwood sales data, it binds the XML to Java object model and passes the instance of *RoundWoodSalesData* class as method parameter in the implementation of the web service.

After creating the interface of the web service, another essential part of the web service is implementation of the interface. The source code of the implementation class is shown in appendix 2.

In addition to the interface and the implementation class for the web service, some other configurations were made for the web service in various configuration files. Similarly, WS-Security based on username token was enabled by configuring the Apache CXF. After running the program on the local tomcat server, the service endpoint was available on the URL: 'http://localhost:8080/secured/services/PuukatiService?wsdl'. A sample client for the web service was created and the web service and the database operations were tested.

The WSDL file generated and available on the endpoint of the web service is given in appendix 3.

3.5 RESTful Service Development

After developing the SOAP based web service, the next feature to be developed was the RESTful service for uploading the roundwood sales data. The main purpose of developing this service was to be used with the web interface based client in the application. However, it was expected to be used by the sawmills who prefer to create RESTful service client over creating SOAP based web service client.

In order to develop the RESTful service, the Spring REST controller was created and the service was implemented in "*/secured/public/services/upload-data*" path of the application. The method implementation of the RESTful service in the Spring controller is shown in appendix 4. In addition to the service implementation, HTTP basic authentication was configured in the Spring Security configuration file for protecting the RESTful service. A configuration segment for the RESTful service of that configuration file is given in listing 2.

```
<!-- For REST service -->
<http pattern="/secured/public/services/**"
          useexpressions="true"
          create-session="stateless">
          <intercept-url pattern="/**"
          access="hasRole('WOOD_COMPANY') or hasRole('ADMIN')"/>
          <http-basic/>
</http>
```

Listing 2. Security configuration for the RESTful services.

The configuration shown in listing 2 forces all the HTTP requests having URL starting with "*/secured/public/services/**" for HTTP basic authentication. After the implementation of RESTful service, the service was tested and finally, a short documentation was created describing the service and how to create a service client.

3.6 Development of Other Features

Besides the SOAP based web service and the RESTful service, all other features to be developed are similar in nature. All of those features are the web interface based functionalities. For implementing those features, MVC architecture was used. To implement the MVC architecture, the Spring MVC framework was used.

A similar approach was used to create all the web interface based functionalities. For example, one can create the view for the application first and then implement controller and model later. In this application, each feature was implemented using step by step approach as given below:

1. Create View
In this step, all the necessary views such as web forms, tables, lists, and menu items are created. In addition to that, some client side scripts or style sheets may be created.

2. Create Controller
In this step, a controller for handling requests is created. The controller is responsible for handling requests and preparing necessary responses by using the model.

3. Create Model

In this step, model part of the MVC architecture is created. The model part is responsible for various actions such as processing data, reading, writing, or updating data to database.

The following functionalities were developed using the step by step approach mentioned above:

- Add new user account to system
- Edit user account
- Remove user account
- Change user account password
- View all data uploads
- Remove certain data upload
- View raw data uploaded from data upload
- Search data upload by business ID of company
- Upload roundwood sales XML data from user interface

After the development of the required web interface based features, form based authentication and authorization was configured in the Spring Security to protect the developed features.

3.7 Deployment

After developing the required features following the FDD development methodology, the application was deployed in Apache Tomcat in Linux server. Next, the application was made available on the URL: https://services.metla.fi/. In the end, the application was demonstrated to the METLA data managers and other stakeholders.

4 Results

Comparing the concrete results gained in the project with respect to the requirements, all of the requirements of the application were fulfilled. The main visible results in comparison with the requirements are illustrated below in this section.

Firstly, creating a web service to enable sawmills to submit the roundwood sales data to MetINFO was the first requirement of the application. To fulfill this requirement, a SOAP-based web service was created. A snapshot of the service endpoint visited in the browser and a part of the WSDL file of the service is illustrated in figure 10 below.

Figure 10. Snapshot of a part of the WSDL file and service endpoint URL visited in the browser.

Similarly, creating a web-based interface to upload the roundwood sales data manually was the second requirement of the project. For this requirement, the view illustrated in figure 11 was created and an upload functionality was implemented in the backend.

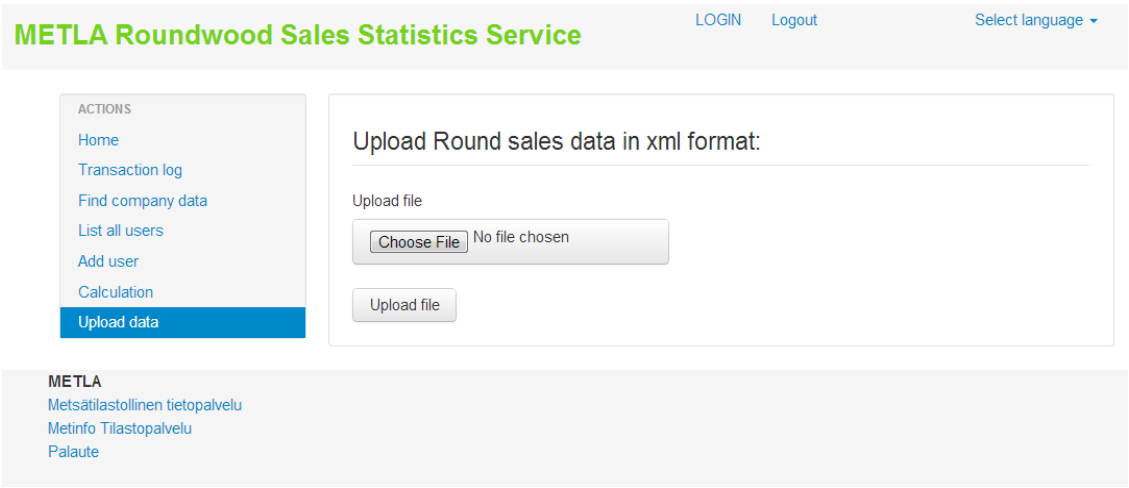

Figure 11. Snapshot of upload roundwood sales data manually through web interface feature.

Creating web-interface-based functionalities for the administration of the data and application was the third requirement of the project. For this requirement, a number of web-interface-based features were developed. The front page of the web-interface-based features is shown in figure 12 below.

-

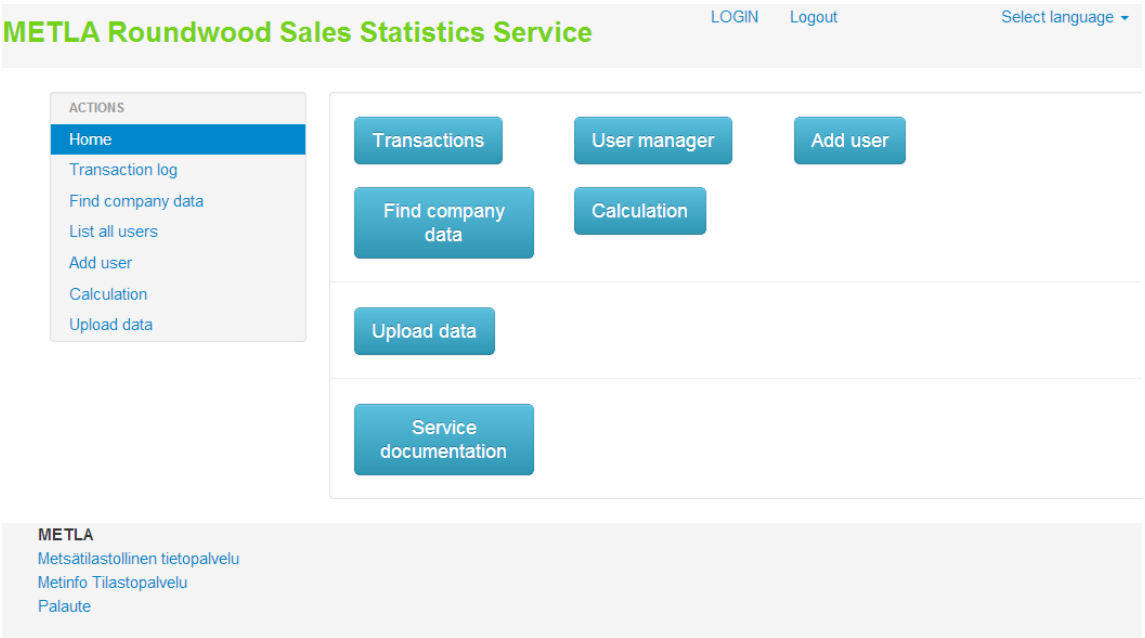

Figure 12. Front page of the web-interface-based features.

As shown in figure 12, the application front page has a menu for several features. The view shown above is the front page view for a user with the *ADMIN* role. A user with the WOOD_COMPANY role will be able to view only the *upload data* and *service documentation* menus.

Additionally, a view from *Transaction log* is shown below in figure 13.

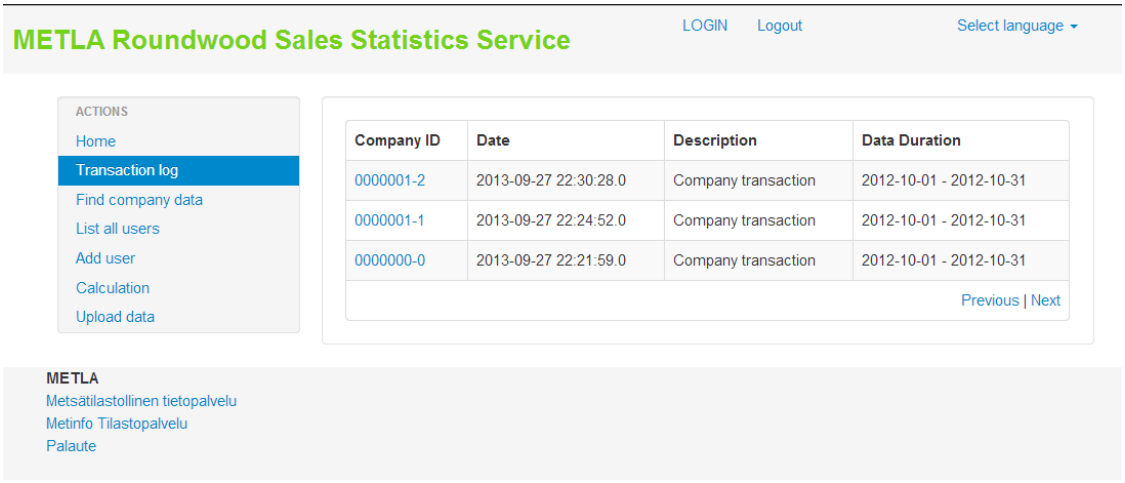

Figure 13. Transaction log or list of data uploads view. The view shown above is a view of the test data. The company IDs shown are not real company business IDs.

Finally, the fourth requirement of the project was to save all the unique uploads in the relational database. For this requirement, four different tables were created. Also, in the web services implementation, the successfully uploaded XML was bounded to the Java object model using JAXB and again the Java object model was persisted to the created tables using Hibernate. The table names and list of their attributes created are shown in table 2.

Table 2. Tables and their list of attributes created for storing roundwood sales data.

Tables	Attributes
roundwood_sales_data	roundwood_sales_data_id(PK), company_id, start_date, end_date
roundwood_sales_row	roundwood_sales_row_id(PK), area_code, area_type, purchase_mode_code, roundwood_sales_data_id(FK)
assortment_compact_class	assortment_compact_class_id(PK), assortment_class_code, roundwood_sales_row_id(FK)
assortment_compact	assortment_compact_id(PK), assortment_info, currency, quantity, quantity_unit, stem_type, total_price, tree_species, unit_price, assortment_compact_class_id(FK)

Additionally, the view of the uploaded roundwood sales data, loaded from the database is shown below in figure 14.

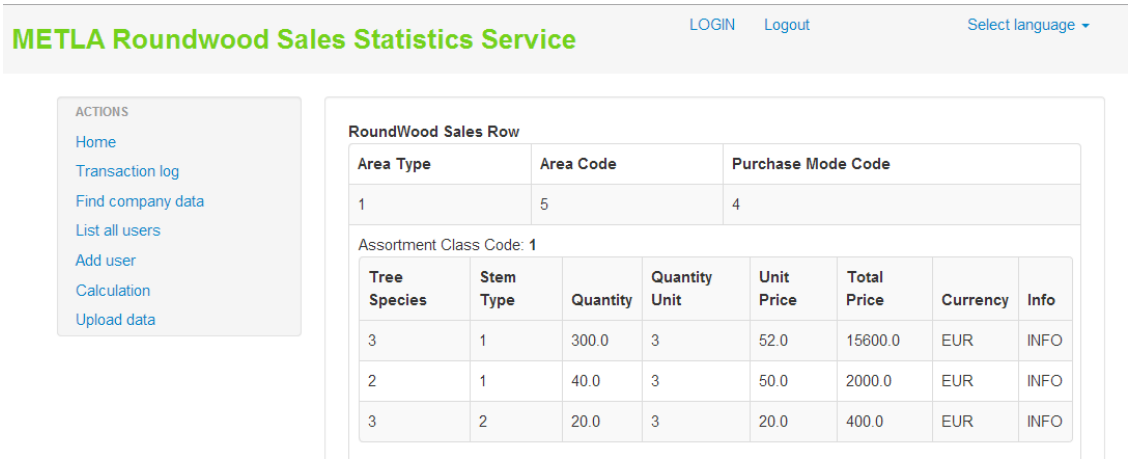

Figure 14. Details view of roundwood sales data loaded from the database. Note that the data shown in figure above is test data and does not represent the sales data of any real company.

As shown in figure 14, using the feature to view roundwood sales data details, it is possible to view the raw data uploaded by different companies.

5 Evaluation of Results

5.1 Benefits

The forestry sector plays a significant role in the economy of Finland. Information and statistics about various sectors of forestry can help the forestry stakeholders such as government, forest owners, and paper mills to make better decisions to enhance the competitiveness. Having a web service to collect data about roundwood sales was an important step to calculate the roundwood sales statistics in Finland. This was the main benefit of the project.

In addition to the concrete benefit mentioned above, the project provided me several other benefits for my career and experience in my field of interest or study. The course module in our school was mainly based on the theoretical aspect of the web services and service-oriented architecture. The project offered a chance to explore practical aspects of use cases and implementation of service-oriented architecture and web services. Gaining a practical understanding of various Java platform libraries, development tools, and design patterns was another benefit of the project. Furthermore, being practically involved in the lifecycle of software development such as requirement analysis, design, implementation, testing, and deployment was another crucial benefit of the project for my future career.

5.2 Challenges

Even though the project was successful in the end, some challenges were faced during the execution of the project. The first challenge faced during the project execution was regarding the decision of using the right web service implementation framework. The Java platform provides multiple open source frameworks for web service implementation such as Apache CXF, Apache Axis2, and Glassfish Metro. Choosing the right framework from the list of available frameworks was confusing and challenging. Initially Apache Axis2 was chosen. Due to the lack of good integration of that framework with

other application modules, it was found that it was not the right selection for the project. For this reason, the Apache CXF framework was finally chosen over Apache Axis2.

Lack of prior practical experience in web service development was another major challenge for the project. Because of this, web service development took more time than expected and overall the initial schedule was affected.

5.3 Future Improvement

In addition to the benefits of the project mentioned above, providing some lessons for future improvement was another important benefit of the project. It was realized that choosing the right frameworks or libraries for specific kind of project needs was crucial for the project success. In future projects, allocating more time for this and doing more study would be one vital ingredient for smooth execution of a project.

After the development of the web service and RESTful service, sawmills were informed about the services. It was found that all of the sawmills preferred RESTful services over the SOAP-based web service because of the lightweight nature and ease of client creation in the RESTful service. Even though it was initially concluded that the SOAP-based web services were best suited for business-to-business communication, it was realized that depending the nature of the service, the RESTful service can also be suited for business-to-business communication.

Initially it was assumed that everything in the project would go normally and as expected. Mainly in the software development project, even some minor issues in development can take much more time than allocated. Additionally, some unexpected issues may arise in the project. Hence, it was believed that allocating some extra time in the project for unexpected delays and unknown issues could be a wise decision during project planning.

6 Conclusion

The primary motivation toward the project was based on two goals. The first goal was to create an application with a service for collecting roundwood sales data from Finnish sawmills which had significance to various forestry stakeholders. Another goal was to gain an understanding of the different software development technologies such as web services, Java, Spring Framework, and other software development tools and methodologies by using them in a real world application.

Having those goals in mind, the project was started by analyzing the requirements from MetINFO. After the analysis of the requirements, the overall architecture for the application was designed. For the implementation of the application, feature-driven development methodology was used. Initially, the web service and RESTful service to collect roundwood sales data were developed and afterwards other web-interface-based features were developed following the MVC architecture design pattern. Subsequently, security rules for the application were configured and the application was deployed in the production environment.

In addition to the design and implementation of the application, an extensive study was carried out side by side for various web service technologies and development frameworks. In this way, with an extensive study and with action-based research, a practical understanding of the different software development technologies was gained.

To sum up, comparing the initial goals of the project with the results, the main aim of the project was achieved successfully. Moreover, some valuable lessons about various aspects of software development such as use of the right framework for a project, the right use case of SOAP-based and RESTful web services, project planning, and time allocation were learnt.

References

1. Extensible Markup Language (XML) 1.0 (Fifth Edition) [Online]. W3C. URL: http://www.w3.org/TR/REC-xml/. Accessed 21 September 2013.

2. New to XML [Online]. Developer Works, IBM. URL: http://www.ibm.com/developerworks/xml/newto. Accessed 21 September 2013.

3. Introduction to SOA and Web services for IT [Online]. InterTech Solutions Inc. URL: http://www.intertechinc.com/soa_it.html. Accessed 21 September 2013.

4. Understanding Service-Oriented architecture [Online]. Developer Network, Microsoft. URL: http://msdn.microsoft.com/en-us/library/aa480021.aspx. Accessed 22 September 2013.

5. SOA features and Benefits [Online]. The open group. URL: http://www.opengroup.org/soa/source-book/soa/soa_features.htm. Accessed 22 September 2013.

6. Web Services and Microsoft Platform [Online]. Developer Network, MSDN, Microsoft. URL: http://msdn.microsoft.com/en-us/library/aa480728.aspx. Accessed 22 September 2013.

7. Web Services Glossary [Online]. W3C. URL: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/. Accessed 22 September 2013.

8. Web Services architecture [Online]. W3C. URL: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. Accessed 22 September 2013.

9. Web services benefits [Online]. Developer Network, Microsoft. URL: http://msdn.microsoft.com/en-us/library/cc508708.aspx. Accessed 22 September 2013.

10. Gabriel Bechara. What is Reuseable Service [Online]. Oracle; March 2009. URL: http://www.oracle.com/technetwork/articles/bechara-reusable-service-087796.html. Accessed 22 September 2013.

11. Rahim Lalani. State-of-the-art Web services [Online]. MC Press online; 22 March 2006. URL: http://www.mcpressonline.com/internet/general/state-of-the-art-web-services.html. Accessed 22 September 2013.

12. Overview of SAAJ [Online]. J2EE 1.4 Tutorials, Oracle. URL: http://docs.oracle.com/javaee/1.4/tutorial/doc/SAAJ2.html. Accessed 22 September 2013.

13. Simple Object Access Protocol (SOAP) 1.1 [Online]. W3C Note, W3C; 08 May 2000.

URL: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Accessed 22 September 2013.

14. Web Services Description Language (WSDL) 1.1 [online]. W3C note, W3C; 14 March 2001.
URL: http://www.w3.org/TR/wsdl. Accessed 23 September 2013.

15. Web Services Reliable messaging TC WS-Reliability 1.1 [Online]. OASIS Standard, OASIS; 15 November 2004.
URL: http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf. Accessed 23 2013.

16. WS-Transaction specification by Microsoft [Online]. Developer Network, Microsoft.
URL: http://msdn.microsoft.com/en-us/library/ms951262.aspx. Accessed 23 September 2013.

17. Web Services Security SOAP Message Security 1.0 (WS-Security 2004) [Online]. OASIS Standard 2004, OASIS; 1 March 2004.
URL: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf. Accessed 23 September 2013.

18. Christian Elfers, Roberto Lucchi, and Michel Millot. Resource Oriented Architecture and REST [Online]. Joint Research Centre, European Commission.
URL:
http://inspire.jrc.ec.europa.eu/reports/ImplementingRules/network/Resource_orientated_architecture_and_REST.pdf. Accessed 24 September 2013.

19. Jørgen Thelin. A Comparison of Service-oriented, Resource-oriented, and Object-oriented Architecture Styles [Online]. Cape Clear Software Inc.
URL: http://research.microsoft.com/pubs/117710/3-arch-styles.pdf. Accessed 24 September 2013.

20. Ivy Wigmore. Resource-oriented architecture (ROA) [Online]. WhatIs.com; July 2012.
URL: http://whatis.techtarget.com/definition/resource-oriented-architecture-ROA. Accessed 24 September 2013.

21. Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures [Online]. Dissertation, University of California, Irvine; 2000.
URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Accessed 24 September 2013.

22. Peeter Kitsnik. Introduction to REST Services. Lecture slide for course *Java Enterprise Technology* for Bachelor of Engineering, Helsinki Metropolia University of Applied Sciences.

23. Learn About Java Technology [Online]. Oracle.
URL: http://www.java.com/en/about/. Accessed 24 September 2013.

24. Java Platform, Standard Edition (Java SE) Technical Documentation [Online]. Oracle Technology Networks, Oracle Corporation; March 19 2013.

URL: http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf. Accessed 24 September 2013.

25. Building Web Services with JAX-WS [Online]. The Java EE 6 Tutorial, Oracle Corporation.
URL: http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html. Accessed 25 September 2013.

26. Apache CXF: An Open-Source Services Framework [Online]. Apache CXF, Apache Software Foundation.
URL: http://cxf.apache.org/. Accessed 25 September 2013.

27. Spring Framework Introduction [Online]. Spring, GoPivotal.
URL: http://projects.spring.io/spring-framework/. Accessed 25 September 2013.

28. Spring Quick Guide [Online]. Tutorialspoint.
URL: http://www.tutorialspoint.com/spring/spring_quick_guide.htm. Accessed 25 September 2013.

29. Installing Subversion on Ubuntu [Online]. WebHostingBillBoard.
URL: http://www.webhostingbillboard.com/development/installing-subversion-on-ubuntu/. Accessed 25 September 2013.

Appendix 1: A Sample Valid Roundwood Sales Data in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<RoundWoodSalesData  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ac="http://standardit.tapio.fi/schemas/workingsitetrade/assortment/2011
/02/14"
xsi:schemaLocation="http://standardit.tapio.fi/schemas/workingsitetrade/woodp
urchasestatistics/2012/05/08 Working-
SiteTrade_WoodPurchaseStatistics_20120508.xsd"
xmlns="http://standardit.tapio.fi/schemas/workingsitetrade/woodpurchasestatis
tics/2012/05/08">
            <CompanyID>1000000-0</CompanyID>
            <StartDate>2012-10-01</StartDate>
            <EndDate>2012-10-31</EndDate>
            <RoundWoodSalesRows>
               <RoundWoodSalesRow>
                        <AreaType>1</AreaType>
                        <AreaCode>5</AreaCode>
                        <PurchaseModeCode>4</PurchaseModeCode>
                        <ac:AssortmentCompactClasses>
                        <!--1 or more repetitions:-->
                        <ac:AssortmentCompactClass>
                        <ac:AssortmentClassCode>1</ac:AssortmentClassCode>
                        <ac:AssortmentsCompact>
                              <!--1 or more repetitions:-->
                              <ac:AssortmentCompact>
                              <ac:AssortmentInfo>2</ac:AssortmentInfo>
                              <ac:Currency>EUR</ac:Currency>
                              <ac:Quantity>20</ac:Quantity>
                              <ac:QuantityUnit>3</ac:QuantityUnit>
                              <ac:StemType>3</ac:StemType>
                              <ac:TotalPrice>200</ac:TotalPrice>
                              <ac:TreeSpecies>3</ac:TreeSpecies>
                              <ac:UnitPrice>20</ac:UnitPrice>
                        </ac:AssortmentCompact>
                        </ac:AssortmentsCompact>
                        </ac:AssortmentCompactClass>
                        </ac:AssortmentCompactClasses>
               </RoundWoodSalesRow>
            </RoundWoodSalesRows>
</RoundWoodSalesData>
```

Analyzing the given sample data shown above, the raw roundwood sales data contains the sales data of certain wood type, its unit price, and volume sold by certain assortment class, by certain area, for given company, and for a given period of time.

Appendix 2: Web Service Implementation Class

```
@WebService(wsdlLocation = "WEB-INF/wsdl/PuukatiService.xml")
@Component("puukatiServiceImpl")
public class PuukatiServiceImpl extends ServiceSupport implements
PuukatiService {

    RoundWoodSalesDataValidator rwsdv = new RoundWoodSalesDataValidator();

    public String uploadWoodSalesData(RoundWoodSalesData roundWoodSalesData)
            {
            //Validate data
            rwsdv.validateRoundWoodSalesData(roundWoodSalesData);

            //Find if the data is already uploaded by a company for the given
            period
            RoundWoodSalesData rwsd = getRaw
            DataDao().findCompanyRoundSalesDataForDuration(roundWoodSalesData
            .getCompanyId(), roundWoodSalesData.getStartDate(), roundWood
            SalesData.getEndDate());

            //If data is already uploaded then nofity message
            if (rwsd != null) {
                        return "Sales data from " + roundWoodSalesDa
                        ta.getStartDate() + "   to " + roundWoodSalesDa
                        ta.getEndDate() + " has been already up        load
                        ed!";
            }

            //If it is unique data, then save it to database
            getRawDataDao().insertRoundWoodSalesData(roundWoodSalesData);

            return "Success! your data has been uploaded!";
    }
```

Appendix 3: Generated WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="PuukatiServiceImplService" target-
Namespace="http://ws.puukati.metla.fi/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:mrw="http://ws.puukati.metla.fi/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns:rw="http://standardit.tapio.fi/schemas/workingsitetrade/woodpurchasesta
tistics/2012/05/08"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <wsdl:types>
        <xs:schema elementFormDefault="qualified" target-
Namespace="http://ws.puukati.metla.fi/" version="1.0" >
            <xs:element name="uploadWoodSalesData"
type="mrw:uploadWoodSalesData"/>
            <xs:element name="uploadWoodSalesDataResponse"
type="mrw:uploadWoodSalesDataResponse"/>
            <xs:complexType name="uploadWoodSalesData">
                <xs:sequence>
                    <xs:element ref="rw:RoundWoodSalesData"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="uploadWoodSalesDataResponse">
                <xs:sequence>
                    <xs:element minOccurs="0" name="Message"
type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
    </wsdl:types>

    <wsdl:message name="uploadWoodSalesDataResponse">
        <wsdl:part element="mrw:uploadWoodSalesDataResponse"
name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="uploadWoodSalesData">
        <wsdl:part element="mrw:uploadWoodSalesData" name="parameters">
        </wsdl:part>
    </wsdl:message>

    <wsdl:portType name="PuukatiService">
        <wsdl:operation name="uploadWoodSalesData">
            <wsdl:input message="mrw:uploadWoodSalesData"
name="uploadWoodSalesData">
            </wsdl:input>
            <wsdl:output message="mrw:uploadWoodSalesDataResponse"
name="uploadWoodSalesDataResponse">
            </wsdl:output>
        </wsdl:operation>
    </wsdl:portType>
```

```
    <wsdl:binding name="PuukatiServiceImplServiceSoapBinding"
type="mrw:PuukatiService">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="uploadWoodSalesData">
            <soap:operation soapAction="" style="document"/>
            <wsdl:input name="uploadWoodSalesData">
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="uploadWoodSalesDataResponse">
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="PuukatiServiceImplService">
        <wsdl:port binding="mrw:PuukatiServiceImplServiceSoapBinding"
name="PuukatiServiceImplPort">
            <soap:address loca-
tion="https://localhost:8080/secured/services/PuukatiService?wsdl"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Appendix 4: Implementation of RESTful Service

```
@RequestMapping(value = "/services/upload-data", method = RequestMethod.PUT)
@ResponseBody
public void doPostConsumeFile(HttpServletRequest req,
            Model map, HttpServletResponse response) throws IOException {

            RoundWoodSalesData rwsd = null;
            String message = "";
            PrintWriter out = response.getWriter();
            response.setContentType("text/xml");
            try {
            rwsd = unMarshalFromIn
            putStream(req.getInputStream());
            rwsdv.validateRoundWoodSalesData(rwsd);
            saveIfUnique(rwsd);
            } catch (Exception e) {
                        out.write("<error>" + e.getMessage() +
                        "</error>");
                        return;
            }
            out.write("<message>We have received your file successful
            ly</message>");
}
```