# GOOGLE WEB TOOLKIT AS MODERN AJAX FRAMEWORK FOR DEVELOPMENT OF RIAS.

Grzegorz Nowak

Bachelor's Thesis
April 2011

Degree Program in Information Technology
School of Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

<table>
<tr><td rowspan="3">Author(s)<br>NOWAK, Grzegorz</td><td>Type of publication<br>Bachelor´s Thesis</td><td>Date<br>13.05.2011</td></tr>
<tr><td>Pages<br>54</td><td>Language<br>English</td></tr>
<tr><td>Confidential<br>( ) Until</td><td>Permission for web publication<br>(X)</td></tr>
<tr><td colspan="3">Title<br>GOOGLE WEB TOOLKIT AS MODERN AJAX FRAMEWORK FOR DEVELOPMENT OF RIAS.</td></tr>
<tr><td colspan="3">Degree Program<br>Information Technology (Data Network Technology)</td></tr>
<tr><td colspan="3">Tutor(s)<br>Peltomäki, Juha</td></tr>
<tr><td colspan="3">Assigned by<br>Bitcomp Oy</td></tr>
<tr><td colspan="3">Abstract<br>Google Web Toolkit is an Ajax based framework for development of interactive web pages – Rich Internet Applications. With additional libraries that add support for handling geographical data it allowed creating a user-friendly experience comparable to most popular web applications in the net.<br><br>The main objective was to develop an application which allows publishing of spatial data over the web using modern technologies. The most important part was the map widget which should support full user interaction with presented data.<br><br>The project’s implementation was split into several phases. Analysis of requirements and architecture gave detailed knowledge about the project itself and the tools needed. Prepared development environment and project’s structure was a signal that the implementation of actual features might get started. Displaying data on the map was the subsequent step after implementing the application’s business logic.<br><br>The result of the process was a fully functional application, entirely compliant with placed requirements. Additionally, the documentation of the project was created to allow faster adaptation of the application to future needs.<br><br>Developed solution offers great functionality in its category. The products offered by competitive companies were comparable or worse than the presented application.</td></tr>
<tr><td colspan="3">Keywords<br>Google Web Toolkit, Ajax, RIA, Smart Client, GIS, GeoServer, GeoTools, OpenLayers, Maven</td></tr>
<tr><td colspan="3">Miscellaneous</td></tr>
</table>

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

# CONTENTS

# FIGURES



# TABLES

# TERMS

AJAX – **A**synchronous **J**avaScript **a**nd **X**ML

API – **A**pplication **P**rogramming **I**nterface

CI – **C**ontinuous **I**ntegration

CRS – **C**oordinate **R**eference **S**ystem

CRUD – **C**reate **R**ead **U**pdate **D**elete

CSS – **C**ascading **S**tyle **S**heets

DTO – **D**ata **T**ransfer **O**bject

ERM – **E**lectronic **R**esource **M**anagement

GET – type of HTTP request

GIS – **G**eographic **I**nformation **S**ystem

GML – **G**eography **M**arkup **L**anguage

GWT – **G**oogle **W**eb **T**oolkit

HTML – **H**yper**t**ext **M**arkup **L**anguage

HTTP – **H**yper**t**ext **T**ransfer **P**rotocol

IDE – **I**ntegrated **D**evelopment **E**nvironment

JAXB – **J**ava **A**rchitecture for **X**ML **B**inding

JNI – **J**ava **N**ative **I**nterfaces

JSNI – **J**ava**S**cript **N**ative **I**nterfaces

JSON – **J**ava**S**cript **O**bject **N**otation

JVM – **J**ava **V**irtual **M**achine

MVC – **M**odel **V**iew **C**ontroller

MVP – **M**odel **V**iew **P**resenter

OSM – **O**pen **S**treet **M**ap

PHP – **P**HP: **H**ypertext **P**reprocessor

POM – **P**roject **O**bject **M**odel

RIA – **R**ich **I**nternet **A**pplication

RMI – **R**emote **M**ethod **I**nvocation

RPC – **R**emote **P**rocedure **C**all

SCM – **S**ource **C**ode **M**anagement

SDK – **S**oftware **D**evelopment **K**it

TDD – **T**est **D**riven **D**evelopment

URL – **U**niform **R**esource **L**ocator

WFS – **W**eb **F**eature **S**ervice

WMS – **W**eb **M**ap **S**ervice

XLS – **X**ML for **L**ocation **S**ervices.

XML – E**x**tensible **M**arkup **L**anguage

# 1 INTRODUCTION

Rich Internet Application as the name suggests is a type of interactive and dynamic web page the functionality of which is usually comparable with features provided by standalone desktop applications. There are different technologies allowing development of such including Flash, Java, Silver Light and Ajax based frameworks. Ajax is the only methodology from those mentioned above, which does not require any additional plug-ins in most of modern browsers. Unfortunately developing applications for different browsers using Ajax is a laborious process. The main reasons for such situation are the differences in the implementations of technologies used in Ajax across browsers. To allow development without worrying about those differences, JavaScript frameworks were created: Prototype, jQuery, Dojo, Google Web Toolkit and many more.

The main objective of this thesis is to report development of RIA for publishing of spatial data over the web. The application allows visitors to view published data on the map, and also search it using provided tools. The overall usability should be as high as possible to support visitors with different levels of technical knowledge

The project is implemented using Google Web Toolkit and managed by Maven. The user interface is based on Smart GWT widget library, except the map component which is part of OpenLayers library. Spatial data is provided using WFS hosted by GeoServer. During the development process several tools were used to improve the quality of final result.

This thesis was created during the internship in Bitcomp Oy. The company creates a suite of tools to manage resources electronically. The whole process of forest management has been completely covered including mobile access. When one of the clients reported a need for an application which allows publishing of already stored data, the company decided to develop such solution using the technologies mentioned above.

The thesis was developed during a double degree program, and because of that, it was evaluated in two independent universities: JAMK University of Applied Sciences and Cracow University of Technology.

# 2 THEORETICAL BASIS

## 2.1 Ajax

### 2.1.1 General Information

Ajax is a set of technologies used to create dynamic web pages. It consists of following:

- Asynchronous calls using XMLHttpRequest object
- XML for data transfer
- HTML and CSS for displaying the data
- DOM for modifying the structure of the document
- JavaScript that ties together all above technologies. (Moore D, Budd R & Benson E, 2007, 35-36)

The XMLHttpRequest is a special object that allows downloading content in the background without forcing the user to wait or reload the page. XML is not the only possible carrier of information. It is possible to use others, for example plain text or JSON which in JavaScript environment is very handy. The JavaScript is not the only option but it is also widely integrated into many popular web browsers in contrast to Flash or Java.

### 2.1.2 Evaluation

There are many advantages and drawbacks of using Ajax.

The advantages with Ajax are listed below:

- Only changing content is transferred so the data transfer and time needed is lower than during full page reload
- User has the impression of an interactive web page – an application in which functionalities are sometimes same or even richer than standalone applications
- Most of the technologies used in Ajax are open source and easily accessible for developers.

The disadvantages are listed as follows:

- The browser history is unable to store a dynamically created webpage
- Bookmarking of such application is also limited to the state in which it starts
- In case of slow and unreliable internet connection the responsiveness of the application might be really low in comparison to a regular web page which will load longer but its runtime will not be interrupted
- To allow web crawlers to index the content there should be a way to access it without using Ajax
- Some web browsers might not support JavaScript or it may be turned off. Even in such case the user should be able to view the content
- Too many server requests with XMLHttpRequest may put the servers under bigger stress.

As it is clearly visible above, the advantages of Ajax are totally obscured by disadvantages. One of the reasons why so many frameworks based on Ajax were created was to overcome these drawbacks easily and to make the development faster.

## 2.2 Google Web Toolkit

GWT is a powerful Ajax framework that gives the ability to develop RIAs without worrying about the drawbacks of Ajax. The application is developed in Java language, but it is also possible to add fragments of code in JavaScript. The source might be launched in two ways:

- Development mode – Java classes are being compiled to byte code and then run on custom JVM which redirects the output into the browser (GWT Developer Plug-in is required)
- Production mode – GWT Compiler translates the Java source into highly optimized (adjustable) and obfuscated (optional) JavaScript, tailored for different browsers and languages.

The development mode is a very important part since it enables the developers to use standard Java tools to debug the code.

The application created in GWT consists of client side code (translated into JavaScript); however, also server side code might be included (in form of servlets). The communication between those two sides is based on custom protocol – GWT-RPC – which was designed to simplify the process. (GWT Dev Guide, 2011.)

### 2.2.1 Compatibility with Java and libraries

One of the reasons for creating GWT was to allow Java developers to use the language they know to create advanced web applications which run in most of the major browsers without forcing the users to install additional plug-ins. This goal in most cases was achieved with some exceptions:

- Java primitive types are all translated into corresponding ones except the numeric types. JavaScript has only one numeric type which is 64bit floating point, so Java numeric types are emulated on top of it. Another outcome is that strict floating point calculations cannot be supported
- Exception handling is supported except for the specific set of exceptions which do not occur in JavaScript. For handling JavaScript specific exceptions – JavaScriptException is thrown
- Java multithreading facilities are not supported since JavaScript is interpreted by single threaded engines. However, there are mechanisms which allow developers to execute code at specific time, or in special manner, where the time consuming task will not freeze the user interface
- Reflection is not supported
- Finalization is not supported as JavaScript does not support it.

Runtime libraries are supported only in case when the functionalities provided by them are accessible by the browser. There is a full list of all emulated libraries available to be found at the following website:
http://code.google.com/intl/en-US/webtoolkit/doc/latest/RefJreEmulation.html.
During launching the application in development mode the compiler checks if the libraries used in the project are translatable. If not, the developer is notified. Some of the libraries which are emulated might have small differences in behavior:

- Methods using regular expressions should get patterns which have the same meaning in Java and JavaScript
- Serialization is supported in a custom way. Until version 1.4 the IsSerializable interface was used to mark objects which are able to serialize
- DateTimeFormat, NumberFormat and Timer classes have their own implementations next to original Java ones. (GWT Dev Guide, 2011.)

### 2.2.2 JavaScript Native Interfaces

GWT Compiler transforms the Java source code into JavaScript. To allow the use of JavaScript and existing libraries, JSNI was introduced. The syntax is quite similar to the one in JNI, but instead of attaching libraries to native methods, JavaScript code is provided in special comment. In-lining of JavaScript is very easy; however, in case of creation GWT libraries source code classes are required in the archive. When JavaScript library provides a functionality which is unavailable in GWT and porting it into GWT is too expensive, creation of bindings is the only solution. There are many ways of accessing properties and methods of a JavaScript object. The simplest one is to create methods for every functionality and field. There are also Overlay types introduced in GWT 1.5. These are useful in some scenarios, but in case of standalone libraries the bindings should be as light as possible, and give direct access to class functionalities. (Google API for GWT, 2011.)

### 2.2.3 Modularity

GWT provides a special way of splitting functionalities and allowing developers to configure them separately – modules. They are defined in a separate XML file which should be located in the root package of the module. The module descriptor consists of information about other inherited modules, entry point classes, compiler options, locations of resources intended for automatic inclusion such as JavaScript source files or CSS style sheets. Inside of the module there are separate packages for client side code, server side code and shared code which might be used on both sides as well. Public package is used as location for files that should be available for the application during the runtime. GWT library structure, as well as the bindings for JavaScript libraries, are based on module template (In addition, Java source files are included for the compiler to resolve native method definitions).

### 2.2.4 Deferred binding

GWT during the compilation creates multiple versions of the application depending on specific criteria. The main one is the browser. Since JavaScript APIs differs a lot among browsers, GWT creates separate versions for the chosen ones. This minimizes the amount of data needed to be transferred, since only the code for a specific browser is downloaded by the client. Another case is internationalization of the application. If the developer specifies that he/she wants to support two languages, the compiler creates separate versions of the application for every browser and for every language. If in some rare cases there is need to use deferred binding, the developer may define it in module descriptor.

### 2.2.5 Threads and timers

JavaScript does not support multiple threads, and neither does GWT; however; there are utilities provided allowing developers to mimic multithreading environment. ScheduledCommand interface is similar to Java Runnable as it provides one method which is executed. Developers may schedule it as deferred command, which means that it will be executed in near future when the control will be returned to the main JavaScript loop. Such behavior is very helpful in cases when the action should be performed when the application has done all the work assigned by the developer and is waiting for events. Other cases where multithreading is needed are time consuming tasks. Web browsers have a defense mechanism that protects them from freezing if the JavaScript engine froze – the browser asks the user if it should kill the unresponsive script. To avoid such situations, IncrementalCommand was introduced. It allows splitting the task into smaller pieces and allowing the event loop control when those pieces are executed. The last utility in this section is Timer class. It allows developers to perform specific actions at some point in the future or do the task periodically. One example use of the timer is making periodic calls to the server to check if state of some property did not change. (GWT Dev Guide, 2011.)

### 2.2.6 GWT-RPC

GWT provides an easy and efficient way to communicate with the server side. The provided mechanism invokes methods remotely (RPC or RMI), and uses custom serialization to

transfer method parameters and results. The figure below shows the class diagram of the mechanism:

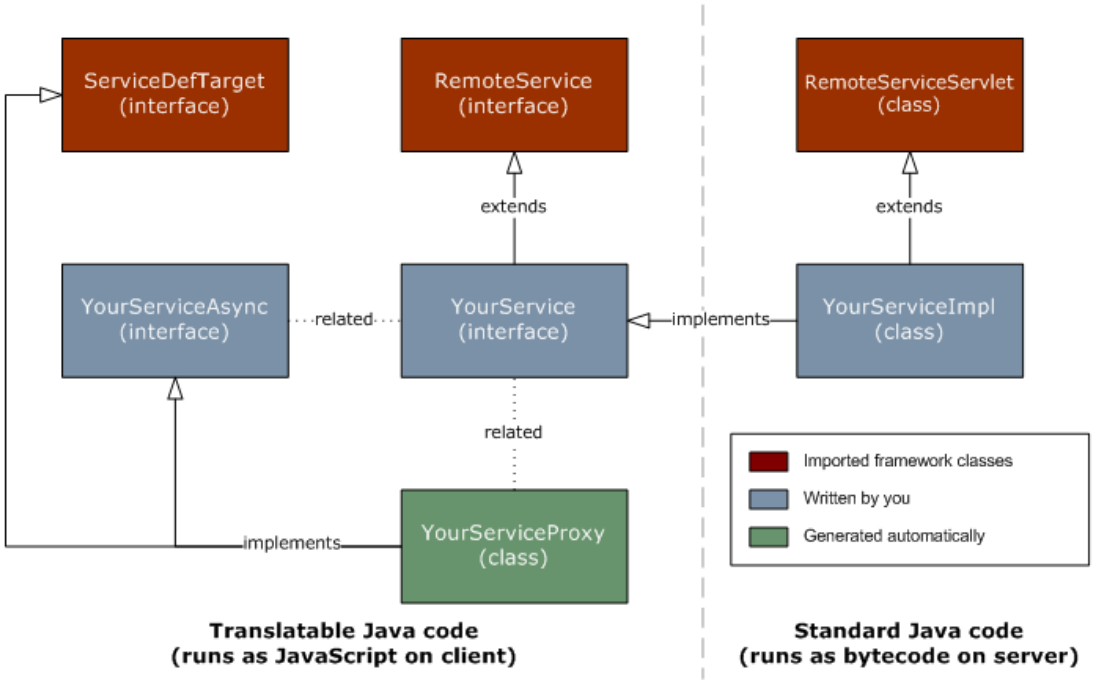


FIGURE 1 GWT-RPC class diagram (GWT Dev Guide, 2011.)

To make use of the GWT-RPC the developer has to complete five steps:

1. Create interface of the servlet
2. Generate or create asynchronous version of that interface
3. Implement the service on the server side and add the servlet into web descriptor file of Java web application
4. Create instance of the servlet on client side using GWT.create() method
5. Invoke specified methods.

The mechanism allows sending parameters and receiving results or exceptions using serialization mechanism. Serialization in GWT differs from the one known in Java but still allows processing of several data types:

- Primitive data types and their respective wrappers
- String, Date classes
- Enum Java types, but only the names
- Arrays of types on this list
- User defined classes implementing IsSerializable or Serializable interface
- Collections of Serializable type.

### 2.2.7 Logging

Until GWT 2.1, logging capabilities had a lower priority, and thus the functionality was quite poor. To enrich it, developers might use GWT Log open source library. It provides many useful features including logging to different places like Firebug console or to div container on top of the application. (GWT Logging library, 2011) In GWT 2.1 similar functionality was introduced. Both of the libraries (internal and third-party) use deferred binding to allow the compiler to create different versions of the application depending on the status of the logging, which might be enabled using HTTP GET parameters.

### 2.2.8 Localization

In GWT localization is handled in a different way than on other platforms. Instead of performing localization during the runtime, deferred binding is used to provide specialized versions for different locales. Triggering between locales is made using HTTP GET parameter or as a “meta” tag in the host page. During application bootstrapping the process localized version is downloaded and launched. Supported locales should be added to module descriptor. In rare cases when the developer wants to provide different versions of the same locale for different parts of the world without forcing the compiler to create too many permutations, he/she can use runtime localization. In such situation the translations are the same but currency, number, time and date formatting are different.

There are two major types of providing the translations:

- Static string internalization – done using properties files for separate languages
- Dynamic string internalization–fully dynamic, based on Directory class.

The fastest way to add different translations is done by using Messages interface. Developer has to provide different properties files for different translations. Those files are part of the compilation process where the result is set of localized permutations of application.

For other places where localization is needed GWT provides own implementations of DateTimeFormat and NumberFormat classes.

### 2.2.9 History support

As mentioned in the chapter about Ajax, web applications have problems with handling history since the content of the page is changed dynamically and the browser only records a page reload. GWT solves this problem in a similar way to other Ajax frameworks: URL fragment identifier tracking. It is the most reliable way of handling application state.

The mechanism for handling history is included in a single class. It provides the interfaces for generating new “history tokens”, and allows responding to their changes using a single callback interface. History token is in a form of URL fragment identifier (string attached at the end of the URL with “#” character at the beginning). It is up to developers how to form the history token, but often key-value pair encoding with comma or semicolon as a separator is used. When the history token changes, an event is fired with the token provided for processing. The parsing process should be efficient and error prone since the token might be typed by the user of the application. (GWT Dev Guide, 2011.)

### 2.2.10 Patterns

Depending on the way how the project is developed different architectural patterns might be used. Model View Controller pattern is used quite often on many different platforms including Java. It gives possibility to split parts of application logic and interface. In case of the GWT application, Model is the part responsible for business logic including calls to the server using GWT-RPC. It is able to notify its observers about the changes in underlying data. The View is the user interface build up using GWT widgets or other libraries like GWT-Ext or Smart GWT. It reacts to changes in the Model and dispatches user actions to the Controller. The Controller is responsible for handling events sent from the View and according to them modify the Model.

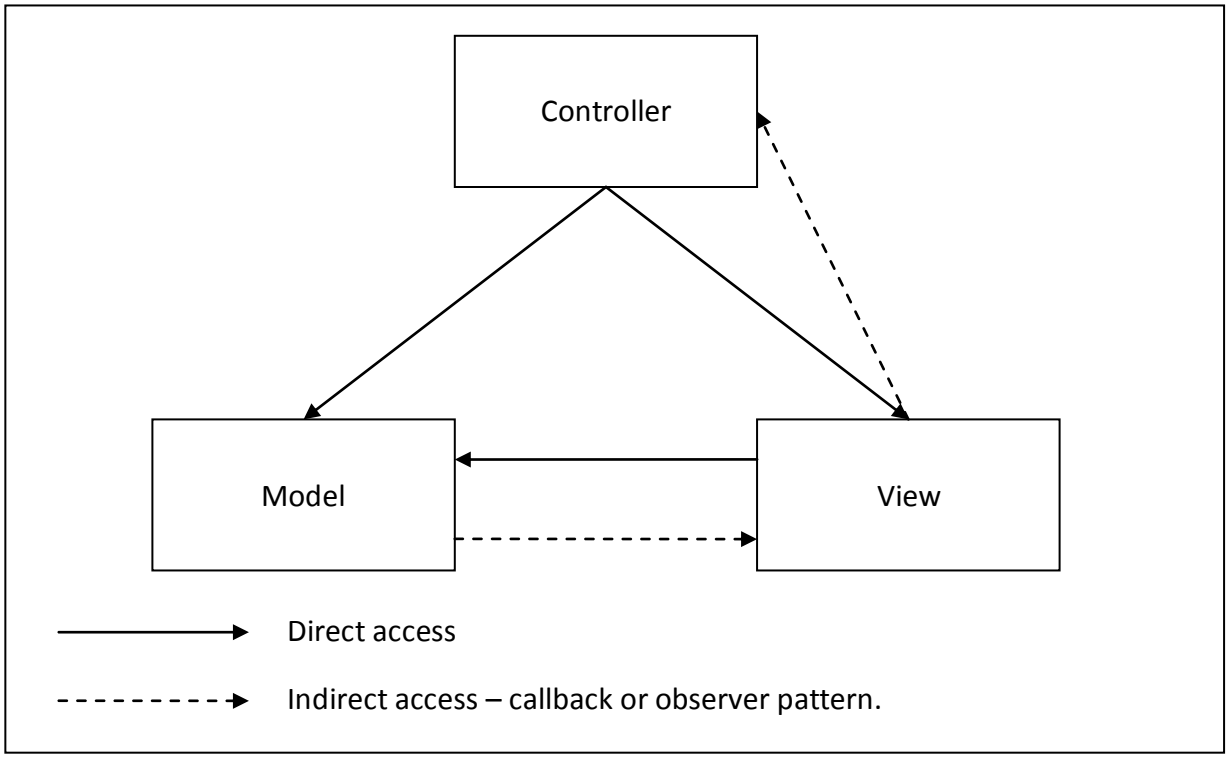


FIGURE 2 Model View Controller diagram

Another architectural pattern which is recommended for developers eager to create applications according to test driven development methodology is Model View Presenter pattern. It has been created by modification of MVC. Instead of Controller, the Presenter is the middle man between Model and View. The Presenter gets events from the View, performs actions on the Model, and updates the View according to changes in the Model. This way the application interface is more separated from the application logic, which gives better testing possibilities.

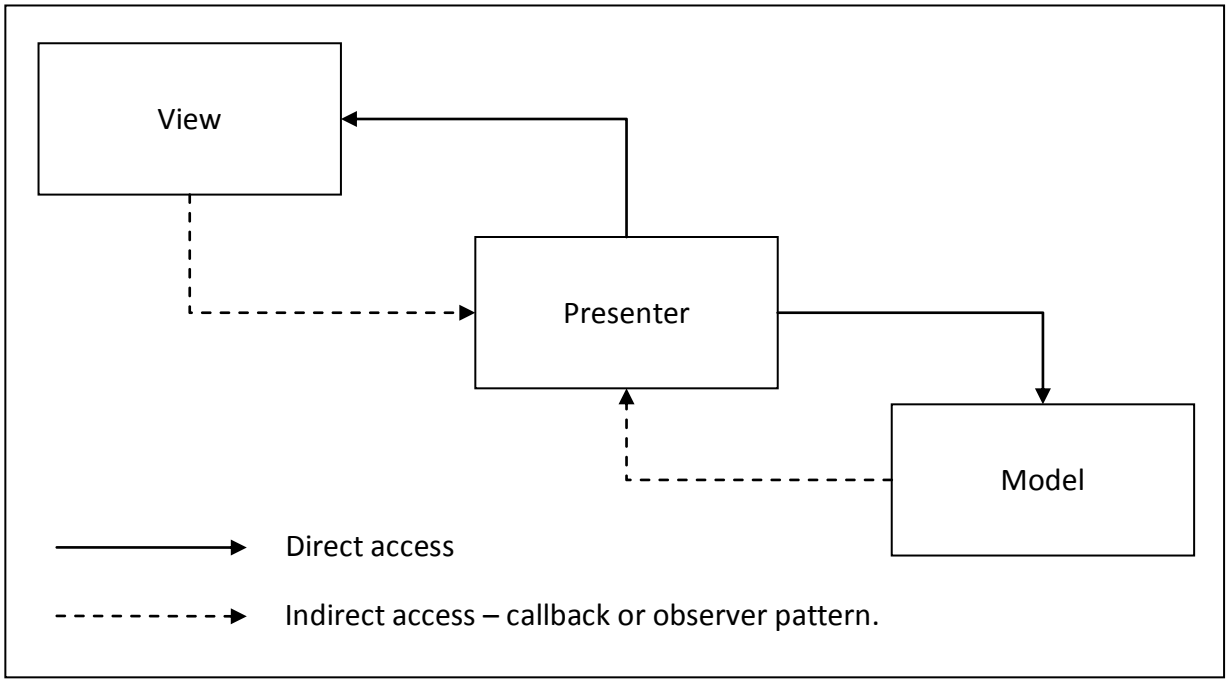


FIGURE 3 Model View Presenter diagram

Design patterns are used in some common problematic situations. In GWT they might also be used in some specific scenarios.

### 2.2.11 Testing

GWT makes use of Java language syntax and some of the libraries, but developers should always remember that the final output is in JavaScript. Testing applications is performed on two levels:

- Unit testing
- Integration testing.

In GWT there are two possibilities to execute unit tests. Methods which can be compiled into byte code with regular Java compiler can be tested with TestCase class from JUnit framework. (JUnit, 2011) In case when there are parts of functionalities or libraries based on JSNI, the GWTTestCase must be used. The difference between those two is that GWTTestCase is a specialized version which uses HtmlUnit. HtmlUnit is a headless browser

that allows interaction with websites which were not rendered or displayed. The browser provides set of interfaces instead, to allow different operations similar to those performed in regular browsers. (HtmlUnit, 2011)

Integration testing might be done using open source testing platform – Selenium. There are different ways of creating tests. The simplest one is to use Firefox plug-in – Selenium IDE – to create test cases and then automatically convert them to Java code (other languages are also supported). (Selenium, 2011) The output is in a form of JUnit TestCase which connects to the instance of Selenium Server. During the execution, the chosen browser is launched using the instance of the server, the application is executed and tested as if the regular user would perform it.

### 2.2.12 Launching and compilation details

GWT applications as mentioned earlier are able to run in two modes: development and production.

The development mode allows to skip the compilation phase, and check how application runs in a browser. The server side code is compiled into byte code and published in embedded instance of Jetty server. The client side code is also compiled to byte code, but the output is sent to the browser. Additionally, when the application is launched using Eclipse with GWT plug-in installed, the developer is able to debug the code line by line. In case when the server side should have access to some external resources like database or other application modules, “-noServer” option is used. In this mode the server side code and static resources like host page, style sheets, images and scripts have to be uploaded into standalone application server.

To run the application in production mode, the whole application must be compiled: server side code into byte code and client side code into JavaScript. During the compilation there are several files created which take part in the bootstrapping process:

1. Launching of the application is made by entering the host page
2. The host page may consist of style sheet definitions scripts, title, optional history handling system “iframe” tag and script tag with bootstrapping code

3. When the bootstrapping code is loaded it checks, in what type of the browser it runs, the locale, and optionally other parameters for deferred binding
4. After reading all the parameters it downloads and launches the proper version of the application from the set of available permutations
5. During the launching of the application, resources included in module descriptors are loaded so they are available during runtime
6. When all the resources are ready EntryPoint.onModuleLoad() methods of all modules are invoked.

### 2.2.13 Smart GWT

GWT comes with a set of styled widgets for common use. For simple applications those components should fit perfectly. For creating rich applications advanced libraries were created:

- GWT-Ext – GWT bindings on top of ExtJS library – project deprecated
- Smart GWT – wrapper for Smart Client JavaScript library.

The Smart GWT library consists of many advanced widgets, whose functionality, look and feel is very similar to widgets used in application frameworks for desktop applications.

Basic types of widgets in the library are containers: VLayout, HLayout, VStack, and HStack. These components are used to split the available space for other widgets. The developer is able to control the size, alignment and many other parameters.

For data input use cases there is DynamicForm class which is a form of a container for other components – FormItems:

- TextItem – regular text input
- SelectItem – drop down menu
- ComboBoxItem – combination of two above
- CheckboxItem.

There are many ways of presenting the data. For tabular data the best choice is ListGrid widget. It may be used not only for showing the data but also for manipulation. It provides

many features including displaying different types of elements such as images, hyperlinks and others. Grouping and sorting of items using different criteria is also possible. (Smart GWT, 2011)

Toolbar is the most often used way of modifying data in desktop applications. Smart GWT library also provides such functionality. The toolbar may have different types of controls including text and image buttons, select items and many others.

### 2.2.14 Evaluation

Google Web Toolkit solves most of the problems of Ajax applications. There are several tools for developing and testing. Additionally all major browsers have JavaScript engines included, and problems with slow networking are getting lower priority since the speed is constantly improving.

## 2.3 Geographic Information System

### 2.3.1 Open Geospatial Consortium

The Open Geospatial Consortium is an international industry consortium of around four hundred companies, government agencies and universities working together to develop free, easily available interface standards.OGC Standards are a base for different IT solutions including Web and mobile platform.(OGC Website, 2011)

In this thesis the most important part of the OGC work are 2 standards: Web Feature Service and Web Map Service.

### 2.3.2 Web Feature Service

WFS is a platform independent protocol over HTTP that allows transferring spatial data through the web. As in the name, feature is the basic unit in WFS. It is an object that contains geographic localization in form of GML definition (shape file definition is also possible) and additional optional attributes related to that instance. WFS defines operations that can be performed on the server implementing the standard:

- GetCapabilities – returns supported versions of the standard and other options
- DescribeFeatureType – returns XML schema to allow client parse returned features in proper way
- GetFeature – returns a set of features which match the criteria of the query.

Basic implementation of the standard allows only querying of features. In transactional implementations a full set of CRUD operations is supported. (OGC, 04-094, 3)

### 2.3.3 Web Map Service

WMS is another protocol created on top of HTTP. It allows transferring spatial images generated by the server from GIS data. WMS defines a set of operations and parameters that are required to return a result in form of an image:

- GetCapabilities – returns information about parameters of the service and layers which are served
- GetMap – returns map image according to input parameters
- GetFeatureInfo – optional command (supported only by "queryable" services) that allows retrieving the feature according to localization provided as one of the parameters. (OGC, 06-042, 6)

### 2.3.4 GeoServer

GeoServer is an open source reference implementation of WFS, which additionally supports other OGC standards. The data for GeoServer might be provided in several ways. The most popular is PostgreSQL database with PostGIS add-on which provides special utilities for handling spatial data. Of course it is possible to use other databases like MySQL or Oracle, using custom drivers. The server has embedded OpenLayers client for previewing hosted data. (GeoServer, 2011)

### 2.3.5 GeoTools

GeoTools is a set of libraries written in Java that encapsulate the communication with WFS or WMS server. (GeoTools, 2011) These libraries are very useful when access to the spatial data in data base is not provided directly but by WFS server acting as a provider.

There are different implementations of interfaces defined by the library. Currently GeoServer project uses those libraries for handling WFS communication.

### 2.3.6 GWT OpenLayers

OpenLayers is an open source mapping JavaScript library. It allows web developers to embed a map into their websites. (OpenLayers, 2011) It is developed using object oriented techniques, and has a modular structure. It supports OGC standards as well as it gives possibility to use different map providers like Google Maps and Open Street Map.

GWT wrapper development started on September 2007[1]. The project is also open source, and provides most of the functionalities of the original library.

## 2.4 Development environment

### 2.4.1 Integrated development environment

IDE is an application that helps developers in the process of creation of software. Usually basic components which are included in modern IDEs are:

- Source code editor
- Compiler, interpreter, linker
- Tools for build automation
- Debugger.

For object oriented languages there are also tools like class browser which helps in viewing and editing the source.

There are several IDEs with support for Java language but for this thesis Eclipse was chosen as an example because it is open source solution.

Eclipse comes in a variety of versions for different purposes, since it supports many languages and methodologies (programming, modeling). The bundle which is the most

[1] First version of the library was uploaded to SourceForge repository (available to be found at the following location: http://gwt-openlayers.hg.sourceforge.net/hgweb/gwt-openlayers/) on September 10th, 2007

appropriate for developing RIAs is Eclipse IDE for Java EE developers. It comes with built- in support for CVS, and technologies related to web development: JavaScript, HTML, CSS and many more. (Eclipse, 2011)

GWT SDK might be used as a set of tools for development of web applications, but there is also Eclipse plug-in available that adds support for GWT projects. For smaller projects the plug-in functionality is perfect, but in case of enterprise applications with several modules using different libraries it is not sufficient. The most important feature of the plug-in which is used even if the project structure is managed in a different way is running GWT applications in development mode with debugger attached. It allows debugging code of the dynamic web page in the same manner as in regular Java application.

For managing enterprise scale projects Maven with additional plug-ins is often used. Maven functionalities will be discussed in the following chapter, so in this one Maven integration with Eclipse is explained. To add support of Maven project to Eclipse, M2Eclipse plug-in should be installed. At the time of writing this thesis, plug-in allowed using embedded version of Maven 3.0 or supplying a path to standalone version. Plug-in adds support of:

- Single, and multi module Maven projects
- Executing Maven phases, goals with specified profiles
- Tools for editing project's POMs with custom editor
- Many other tools for Maven related tasks. (M2Eclipse, 2011)

Maven support in Eclipse is very useful in case of multi module projects, since the plug-in integrates with Eclipse class path, so for building workspace dependencies are used, instead of artifacts from local or remote repository.

### 2.4.2 Source configuration management

SCM allows tracking of changes applied to source files, documentation and other software project related information. It is most often used during development of software to allow members of the team to work simultaneously with the same files. Tools for SCM come as standalone applications or as plug-ins embedded into IDE's. An example of such is the plug-in for CVS in Eclipse.

Most commonly used operations are:

- Check-in – process of putting the file into the repository
- Check-out – pulling the specific or newest version of the file – working copy
- Commit – applying changes made on working copy on repository version
- Update – applying changes made in the repository on working copy
- Merge – applying several changes into a single file – requires developer's supervision. (CVS, 2011)

Source code management is an important topic because it allows collaborative work and gives the possibility to revert a resource's state in case changes applied are wrong.

### 2.4.3 Source quality management

Quality management in general is the process of keeping high quality of the product or service as well as means and tools to achieve that level. In case of software development, the process concerns quality of the source code which is produced by the developers. Of course, there are different ways and tools for testing the quality but here one will be reviewed – Sonar.

Sonar is an open source solution allowing developers to manage the quality of code they produce. It uses reports generated by different tools like FindBugs, PMD, and Checkstyle to create extensive report about the code quality. There are seven categories in which Sonar operates to generate a report:

- Architecture, design
- Duplications
- Unit tests
- Complexity
- Potential bugs
- Coding rules
- Comments.

The report is structured to allow as efficient use as possible so development is not interrupted. (Sonar, 2011)

### 2.4.4 Continuous integration

CI is the process of building, testing, and quality scanning of the software project in very short intervals which depend on the speed of development. It improves the quality of the results by allowing developers to detect problems almost immediately, instead of in the last phase. To automate the process, specialized tools were created, which perform scheduled tasks related to software projects. One example is Hudson. It offers integration with different source repositories, build, testing and quality management tools, which support is provided using plug-ins. (Hudson, 2011)

### 2.4.5 Component repository management

Component repositories act as storage facilities for final products of development. The major role is similar to source repositories, but instead of managing sources of applications and documents, components repositories manage the results of building process. The main advantage of such approach in case of modular solutions is availability of different versions of modules for use in other projects.

Maven component repositories include all the advantages of standard component repositories, but also may act as a proxy between Maven central repository, and local repository of every developer. Caching of artifacts makes the time need to build the projects shorter and in continuous integration the time needed for single build is very important.

## 2.5 Maven

“Maven is a project management tool which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system, and logic for executing plug-in goals at defined phases in a lifecycle.”(O’Brien T, Casey J & Fox B, 2010.)

Convention over configuration in Maven has many advantages:

- Creating and launching new project takes only few commands

- Common interfaces for build, helps in moving between projects
- All the defaults are customizable so if a need arises and the developer needs to change some parameters – he/she can easily do it
- Modular structure of Maven allows using different versions of plug-ins
- Project model as a standardized place for all information about the project as a developed structure
- Bigger functionalities without a huge amount of work as for example in Ant.

In this chapter Maven features are described more in detail, especially for GWT projects.

### 2.5.1 Project Object Model

Every Maven project contains its object model. The file which contains it is called pom.xml, and until Maven version 3.0 it was always XML file (in Maven 3.0 it is possible to use other languages for POM; however, it requires additional plug-ins). It consists of all the project related information.

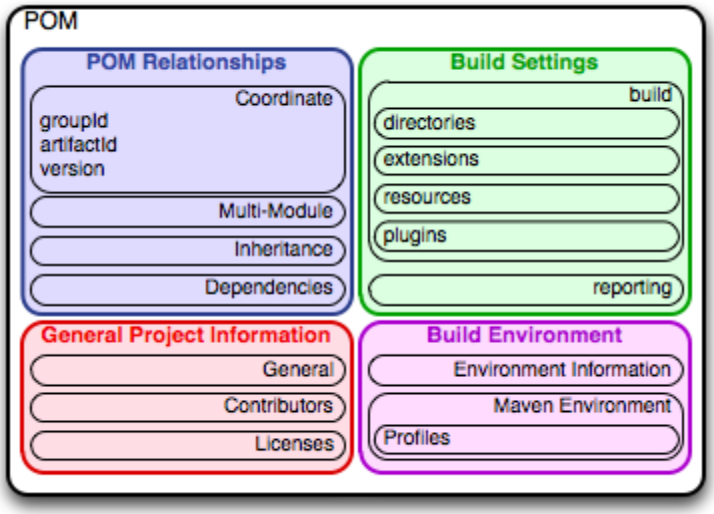


FIGURE 4 Maven POM structure (O'Brien T, Casey J & Fox B, 2010.)

As mentioned previously, the “convention over configuration” is also visible here. POM file inherits all values from its parent. If the parent is not specified the default parent called Super POM is provided.

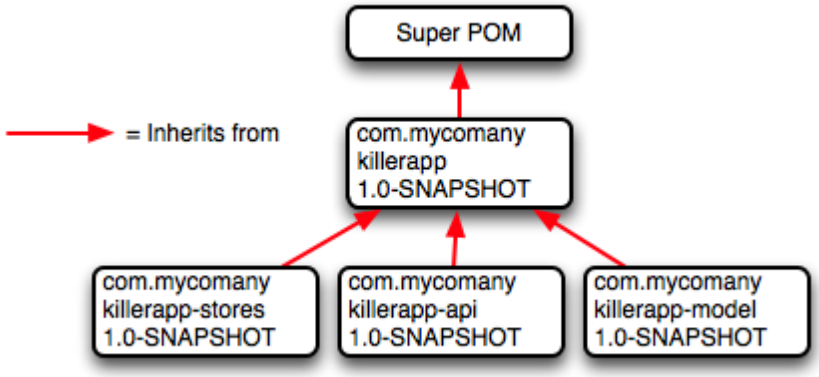


FIGURE 5 Maven POM inheritance diagram (O’Brien T, Casey J & Fox B, 2010.)

As in the above figure, the effective POM can be quite complicated, that is why a tool for viewing it is provided- “help:effective-pom”.

Here is the example of the simplest POM:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.nowaklabs.maven</groupId>
  <artifactId>project</artifactId>
  <version>1-SNAPSHOT</version>
</project>
```

As in the above example Maven uses a special way of versioning projects. Releases have regular version numbers like “1.0”. In case of the projects under active development phrase “SNAPSHOT” is added. During the build Maven treats the resulting artifact as a snapshot of the project in a specific time. The phrase is changed into timestamp formatted to be easy readable for human.

During creation of the POM it is possible to use different types of properties connected with the environment, project or define new ones, which are then accessible through the POM.

One of the features in Maven is dependency resolution. Instead of getting proper versions of libraries and putting them in proper places of the project, Maven takes care of libraries automatically. There are different scopes of the dependencies: compile, provided, test, runtime etc. Also the dependencies of the parent POM are inherited by its children.

### 2.5.2 Build lifecycle

Build lifecycle of the Maven project is a sequence of phases that are executed during the build. There are sets of phases for building different types of projects, cleaning them and also creating the website of the project using Maven site generation plug-in.

In case of JAR and WAR projects the lifecycle is quite similar:

TABLE 1 Maven default lifecycle for WAR and JAR packages

| # | Phase name | Plug-in : goal | Description |
|---|---|---|---|
| 1 | Process-resources | resources:resources | Plug-in copies resources files, if the filters where applied then replaces tokens in the files with proper values. |
| 2 | Compile | compiler:compile | Compiler plug-in compiles the Java source files into class files. |
| 3 | Process-test-resources | resources:testResources | Plug-in does the same job with test resources files. |
| 4 | Test-compile | compiler:testCompile | Compiles the test classes. |
| 5 | Test | surfire:test | Launches compiled test classes. |
| 6 | Package | war:war or jar:jar | Performs packaging into proper archive type. |
| 7 | Install | install:install | Installs the packaged artifact into local repository. |
| 8 | Deploy | deploy:deploy | Deploys the package into standalone remote repository. |

### 2.5.3 Profiles

The POM of the project consists of the information about it. In case when some information is dependent on the environment in which the project is built, Maven profiles come in handy. Using the profile it is possible to customize different parameters of the project. During the build Maven takes those environment- specific options from the active profile. The activation can be made in several ways both automatic and manual. In general the use of profiles is not encouraged because of breaking the project portability, but in some situations it is the only way to allow building without manual modifications of the POM. Such case for example might be the location of the application server for deployment of the artifact.

### 2.5.4 Archetype

The archetype Maven plug-in allows developers to easily create project structure of the specific type. Instead of manually creating folders and POM from scratch, Maven users can just invoke the archetype plug-in with the project type and other needed options. There is a long list of available archetypes in Maven central repository. This feature is another example of the convention over configuration.

### 2.5.5 Other plug-ins

GWT projects might be handled using tools included in SDK or using plug-in for Eclipse. The third option is GWT Maven plug-in. It is very useful because it allows performing GWT related tasks using Maven interface:

- Running application in development mode but without attaching Java debugger
- Compiling the project into JavaScript
- Automatic internalization interfaces generation
- Automatic generation of asynchronous GWT-RPC interfaces.

Plug-in also provides archetype for easy generation of basic project structure. (GWT Maven, 2011)

Cargo plug-in is a multipurpose solution, however, the main functionalities used this thesis are:

- Deployment of packages into Java EE containers
- Merging of Java EE modules.

The first functionality is useful in case of automated builds where the artifact is automatically deployed to different environments: “development” or “testing”.

Merging is provided using special Maven project type – “uberwar”. Modules defined in such project are merged including web deployment descriptor files whose merge instructions can be provided by the developer. (Cargo, 2011)

# 3 PRACTICAL IMPLEMENTATION

## 3.1 Bitcomp Oy

Bitcomp Oy is an international company with agencies in Finland and Poland. The main office is located in Jyvaskyla in Central Finland. The company was founded in 1998. The main tasks of Bitcomp are IT consulting and development of IT solutions for partner companies. The projects are based on latest technologies connected with:

- Geographic Information Systems
- Rich Internet Applications
- Mobile Solutions.

Bitcomp's flagship product – BitApps – is a combination of all products mentioned above. BitApps is a versatile ERM solution that, thanks to fully featured mobile client, is accessible anywhere and anytime. Remote access removes the need for managing documents since all the operations are supported even outside the client's premises. (Bitcomp, 2011)

## 3.2 Analysis of the requirements

The requirements of the application were acquired from the client. Functional requirements define what features the final application should contain and they are gathered in Table 2.

TABLE 2 Functional requirements

| N | P | Description of requirement |
|---|---|---|
| 1 | 4 | Map component with different background map sources: Maanmittauslaitos, Open Street Maps and other WMS compatible providers. |
| 2 | 4 | Tool for searching addresses. The results should be visible on the map. |
| 3 | 5 | Tool for searching selling places using different criteria including the area bound by map view port. The results should be shown as a list and also on the map. |
| 4 | 4 | Showing selling places on the map with different styles depending on its state: default, found (search tool returned it as a result), selected (user highlighted the result to perform some action). |
| 5 | 2 | Showing selling places on the map depending on the zoom level – in case when the geometry of the place is too small, more visible replacement should be provided. |
| 6 | 3 | Providing set of operations on search results: adding, removing, and viewing. The user should be able to select the features for the operation using map or list. |

Non functional requirements define the overall behavior of the system and are provided in Table 3.

TABLE 3 Non functional requirements

| N | P | Description of requirement |
|---|---|---|
| 1 | 5 | Cross-browser compatibility – the application will be accessible globally, so the possible users should not have difficulties in using the application in favorite browser. |
| 2 | 4 | Stability – the output is a dynamic web application and it should not crash or force the user to restart the browser. |
| 3 | 3 | Security – only authorized users should have the access to the data. |
| 4 | 5 | Simplicity – application should be as simple to use and intuitive as possible so even older people will not have problems in using it. |
| 5 | 5 | Inviting – user should have the irresistible temptation to use the application (this requirement is directed to the person in charge of the final look of the application - engraver). |
| 6 | 4 | Internationalization – application user interface should be translated to Finnish and English. |

Based on functional requirements a use case diagram was created, see Figure 6.

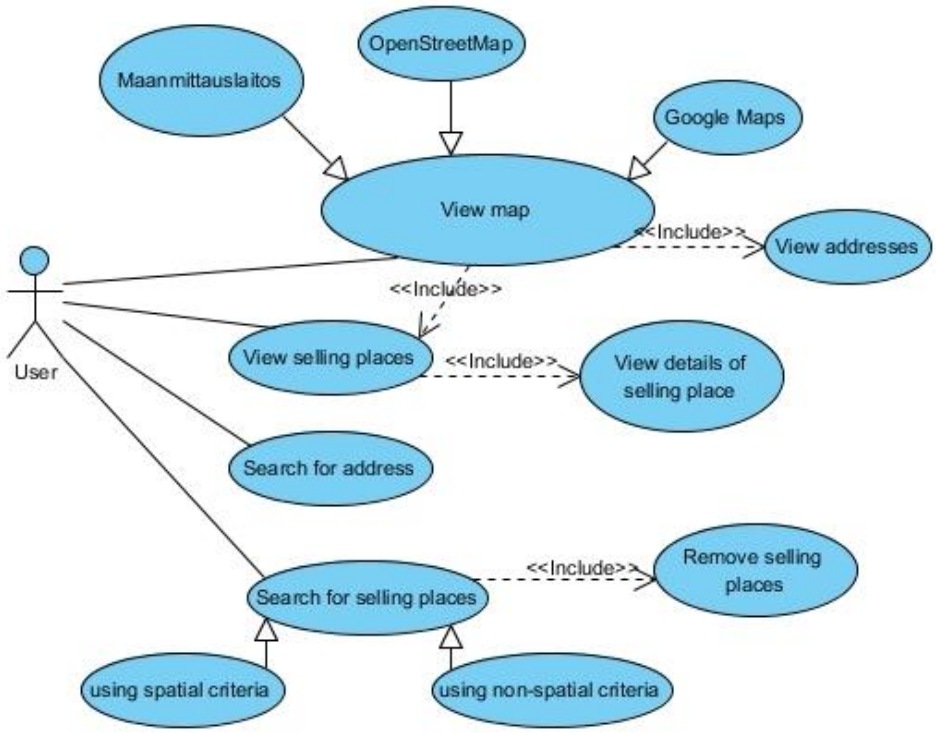


FIGURE 6 Use case diagram

## 3.3 Analysis of the architecture

The project is a solution that makes use of several parts:

- Client which is an application available to the users
- Service which consists of several servlets used by the Client to retrieve information from different sources
- GeoServer – map server that is a web application
- GeoCoder – another web application providing geo-coding service based on the XLS and JAXB standards.

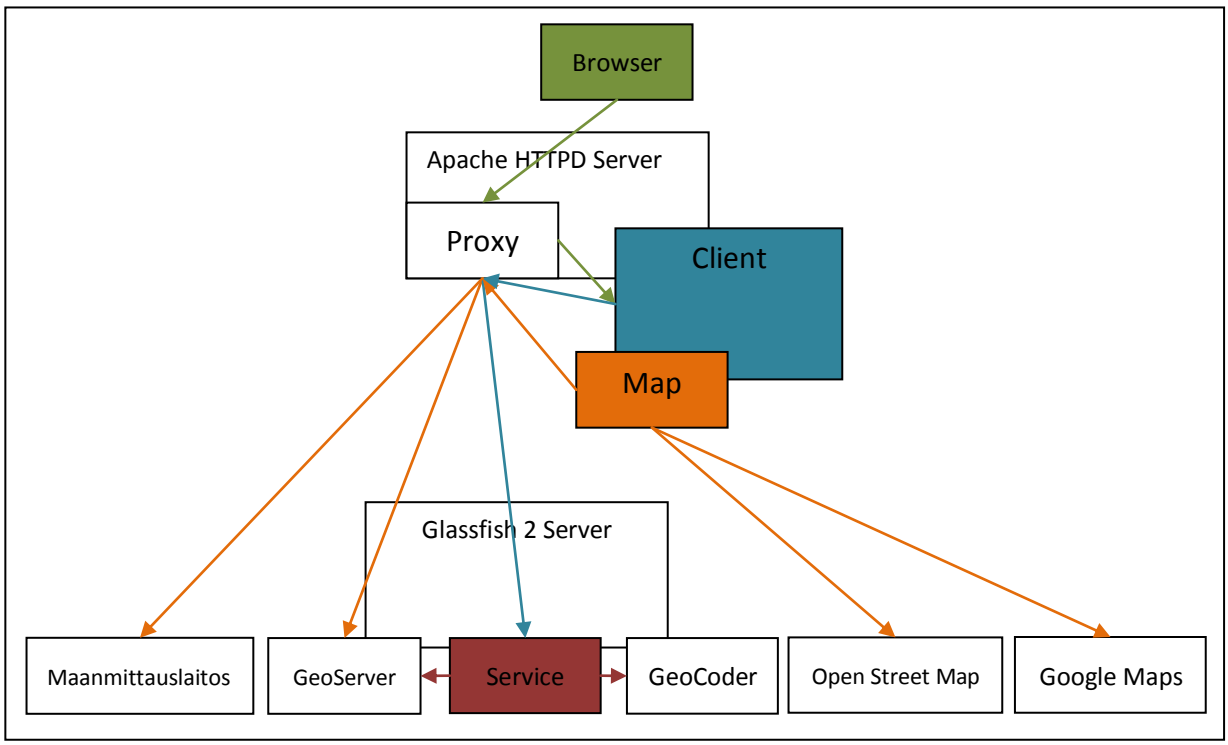


FIGURE 7 Architecture diagram

GeoCoder and GeoServer instances should be deployed on Java EE application server. In addition, GeoServer should have access to the data which will be provided to the Client – mainly map related content.

The Service part is also a web application which uses the GeoCoder and GeoServer resources for different purposes. Mainly it processes the data so it can be easily accessed by the Client.

The Client part is a GWT application with embedded map component compiled to JavaScript. The compiled code should be deployed to a highly efficient HTTP server.

To avoid the violation of SOP access most of those parts should be accessed using a proxy server. It is required so that all the requests from the browser will be directed to the same host, port, and using same protocol every time.

Access to Maanmittauslaitos map provider requires simple authentication. To allow visitors of the application to use it without providing the authentication data (and mostly likely they do not have it) the proxy was used again. In this case, instead of only redirecting calls to

different server, the Authentication header is added to every request, which goes through the proxy.

## 3.4 Workspace setup

During development of a software project there are several tools that improve the process. The first one needed is IDE which may be used for editing different kinds of source and configuration files, debugging and testing. In this project Eclipse Helios JEE version was used with additional plug-ins for GWT and Maven. CVS built- in plug-in was configured to allow synchronization with the company's repository.

The built versions of the project are automatically deployed to the application server – in this case Glassfish 2 was used as other projects in the company were deployed on such server.

As mentioned in the previous chapter, access to some project parts should be provided through proxy. For this project a free and open source solution was chosen: Apache HTTP Server 2.2. The required configuration was limited to setting up the proxies to applications deployed to the Glassfish server, and also proxies to GeoServer instance which was deployed on one of company's servers.

There are many different web browsers available to the users. That is why during the creation of the web application it is so important to test the solution on as many of them as possible. At the time of writing this thesis the most popular browsers are visible in Figure 8.

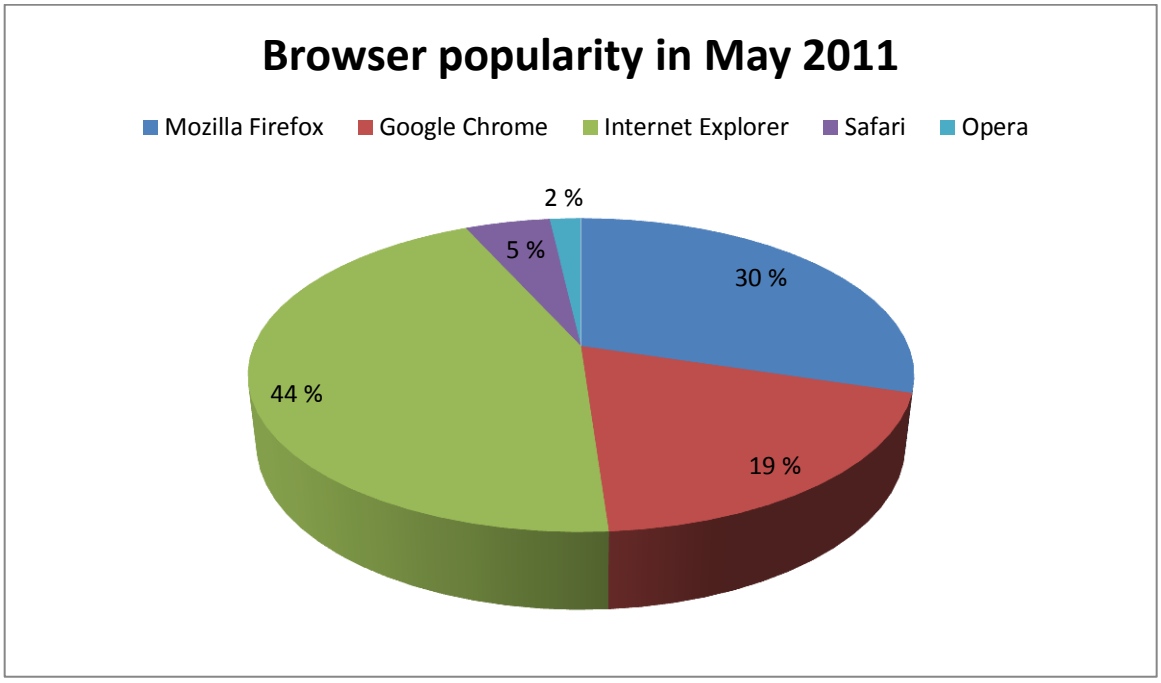


FIGURE 8 Browser popularity diagram (StatCounter, 2011)

The top three have a plug-in which gives the ability to run the GWT application in development mode. It is very handy, because it is possible to debug the Java code and the compilation to JavaScript takes time.

## 3.5 Project structure

After setting up the environment it was time to create the structure of the project. It should be divided into several sub-modules:

1. Common – subproject containing common interfaces between Service and Client
2. GwtOpenLayersWrapper – module for adding functionalities to GWT-OpenLayers
3. Service – module with server side code
4. Client – client Java and JavaScript code
5. Uberwar – special project for merging Client and Service project output into one deployable war.

The common module should contain all the source files used both on Client and Service side:

- Data Transfer Objects
- Servlets' interfaces and their asynchronous versions for GWT-RPC mechanism
- Libraries which may be used on both, server and client side.

The module output should be a simple Java JAR library with source files, unit tests and GWT module files. For that reason the structure of Maven "Simple Java project" archetype was customized. Additional folders for resources and test resources were added for storing all non-Java files. In those folders module descriptor files are placed with definitions required by GWT compiler. During the process-resources phase Maven resource plug-in copies all resource files into target directory, including the Java source files needed for resolving JSNI native methods.

There are two options to allow sending of objects using GWT-RPC:

- Implementing IsSerializable interface created just for GWT
- Implementing standard Java Serializable interface which requires special files (generated by GWT compiler) with permissions which types of objects can be serialized.

The first option was chosen because of the development requirements – deployment of Client and Service module should be independent, and only GWT compiler during the compilation is able to create the needed permission files.

Testing of this module's classes may be done using regular JUnit TestCase or special GWTTestCase if the class contains JSNI native methods. Using GWTTestCase requires custom configuration of Maven Surefire plug-in. Instead of using default class loader, providing paths to source and compiled files was required.

GWT-OpenLayers library is an open source project whose major role was to provide GWT bindings for OpenLayers. Unfortunately, at the time of creating the project, the library was not upgraded for a long time. That was the main reason to create the wrapper that will consist of all the necessary improvements and features not available in the GWT-OpenLayers. Splitting the code related to GWT-OpenLayers separates the source and the test classes from

the main project and also gives the possibility to reuse added functionalities in other projects.

GwtOpenLayersWrapper project was created in the same manner as the Common module. Additionally module's "public" folder was created to store the content that should be available for the application during the runtime. In this case the public folder was filled with libraries in JavaScript:

- OpenLayers.js – map library including theme files
- OpenStreetMap.js – definitions of OSM layers
- Proj4js.js – Projection library with additional definition and projection files.

These libraries after adding them to the module descriptor file are automatically added to the main page as script tags, so the browser loads them (with proper version of the main application) during the bootstrapping process.

Service module is a web application that contains implementation of GWT-RPC interfaces from Common module. To make use of all features of Eclipse connected with web application development the Dynamic Web Project was created and customized by Maven Eclipse plug-in. The project's dependencies had to consist of additional libraries:

- Log4j library to enrich logging capabilities
- Gwt-servlet.jar and gwt-dev.jar libraries to allow implementation of GWT-RPC mechanism
- GeoTools libraries
- GeoCoder libraries
- JAXB implementations.

The module uses different types of servers. To allow easier configuration of the application for different deployment environments, profiles and resources filtering was used. Depending on the chosen profile, configuration files are filtered with proper values. This gives the ability to automate the build and deployment process which can be performed by every developer without knowing the details of the specific environment.

Every web application has deployment descriptor file that describes the application internal parts – in this case servlets. Additionally, there are application server specific files (in case of Glassfish 2 – “sun-web.xml”) which describe the context root of the application that should be set up properly during the application automatic deployment.

Client module is the application frontend created using GWT. The project structure was generated using the archetype for GWT projects. As in the previous projects some modifications were required. First of all, “test” and “generateAsync" goals were disabled. Testing of the application was handled the same way as in the Common and GwtOpenLayersWrapper sub module. The goal “generateAsync” created asynchronous versions of interfaces for GWT-RPC but the results were not satisfying (generated interfaces did not have information about types of input parameters in case when generic types were nested).

The module is the actual web application used by the users. To perform integration tests Selenium plug-in for Maven was chosen. It gives the opportunity to run test cases first recorded using the IDE plug-in for Firefox. Since the project uses the Smart GWT library which assigns IDs to elements dynamically, it was necessary to add an extension file which adds support for “scLocator”. It allows distinguishing document elements of Smart GWT widgets by Selenium. The file is copied from resources folder to target folder during prepare test resources phase using Ant copy command. The embedded Selenium Server is also being launched before and stopped after the integration test phase. The integration tests have the same structure as the unit tests. To avoid running them too early they were excluded from test phase and included in the integration-test phase.

Since the output of the Client project is pure JavaScript, Maven resource filtering plug-in was useless. For copying separate configuration files Ant task plug-in for Maven was used.

Unfortunately, at the time of creating the project GWT archetype did not create the Eclipse launch file which would run the development mode connected to the Eclipse debugger. The solution to this problem was to trick Eclipse into thinking that the project is actually GWT native project created using Eclipse wizard. After editing the project file and adding the GWT project nature it was possible to create launch file using Eclipse tools.

For handling history tokens, singleton class was created. The tokens can be added parsed and removed using this class, and also during the application startup when history handler event is not fired, this class takes care of the input parameters.

The whole application uses built-in GWT static string internalization. Translations are in separate files and GWT compiler during the compilation creates separate versions of the application depending on the chosen locale.

The project in this state is only a demonstration of possibilities. To be able to switch between different implementations like between map providers or strategies, HTTP GET parameters were used. The OpenLayers library in the version that was used in the project (latest stable release – 2.10) was not designed to handle dynamic changes of some parameters, so at that stage the simplest solution was chosen – launching application with startup parameters.

Uberwar is a special type of the Maven project introduced by Cargo plug-in. It is used to merge few WAR packages into one “uberwar” which is sometimes very handy. In this case Uberwar is created from Client and Service modules.

The parent project in this thesis is the container for all the sub modules. The POM consists of several items as follows:

- Module declarations
- Reporting plug-ins declarations which reports are analyzed by Sonar
- Profile definitions for different development environments.

The whole project after testing all the functionalities and dependencies was checked in into CVS repository. Hudson job was created which used development profile for building and deployment to development environment.

For some time all the artifacts were being deployed to the company’s Artifactory but during the remote work the time needed for such operation is much longer, so it was disabled at some point.

## 3.6 Application layout

The first part of the time reserved for implementation was dedicated to creation of the layout. The application has a simple static layout with separate places for functionalities. The skeleton is very common: header content and footer part, plus content split into smaller panels, one for search form and results list and the second one for address search field and map component. Figure 9 below illustrates this:

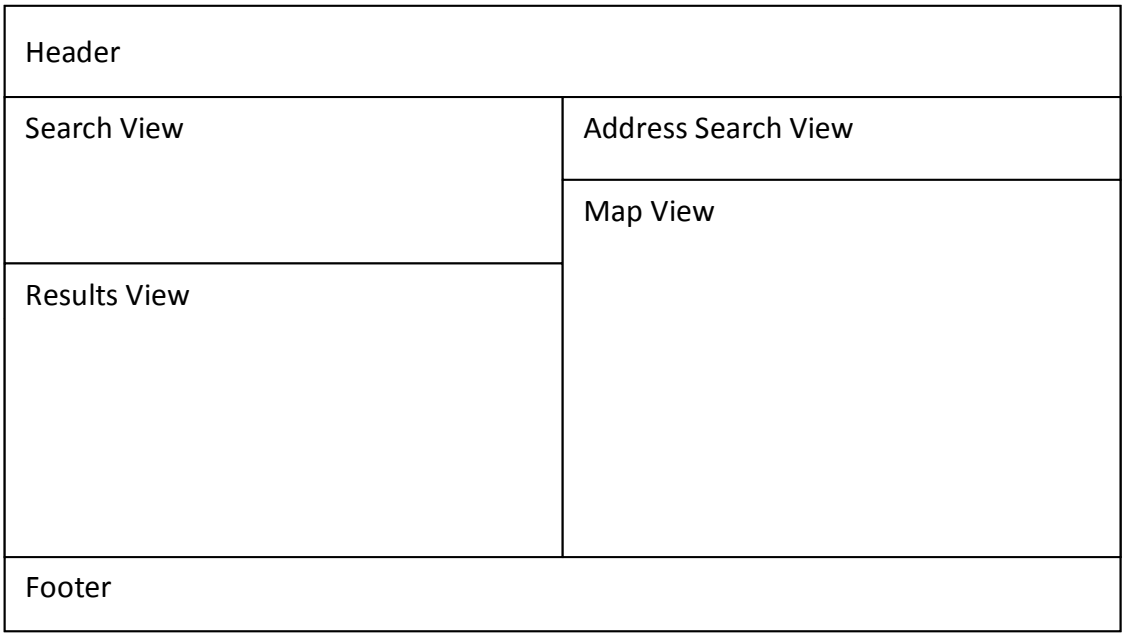


FIGURE 9 Skeleton of the application layout

Almost all used components are from the SmartGWT library. That is an advantage because they were designed to cooperate with each other to make the layouts dynamic and resizable.

The major problem during the creation of the layout was embedding of the map component, which is the OpenLayers JavaScript object wrapped with GWT native widget. It did not resize as the other components. The only solution to this problem was to manually control the sizing of the map. During the resize event the map was sized to 0 by 0 pixel dimension to

allow other components take as much space as needed and after small delay resizing the map to the available space. Unfortunately, this workaround did not work in Firefox. After closer look into the document structure it turned out that one of enclosing div elements was not changing the size. The solution was simple: during changing the size of the map, also the enclosing div was adjusted. This workaround works until the layout changes since there is no way to determine by traversing the DOM structure, which div should be resized. During the creation of the layout there were many other problems. Another one was that dynamic form widget, switched to group mode was overleaped by the map component but this only occurred in Internet Explorer. The only way to remove this “feature” was to remove the grouping.

Smart GWT showcase and Gmail have an interesting feature –full screen loading animation. It is a simple trick to show progress of the application until it is fully loaded. Small modification of Smart GWT loader gave satisfying results. It was possible to change the message during the time the application loads so the user has at least small knowledge about the state of the application.

OpenLayers component comes with a simple and very old theme. It is really easy to change it to a more modern one. The only thing is to change the folder with the old theme to the folder with a new one.

Embedding other components went quite smoothly. To allow the users to change the size of the map from default 50%, or to hide the whole search form and results grid the resize bar in layout widget was enabled.

The list grid also consists of the toolbar for manipulating the data in the list grid. Unfortunately, this component’s resizing capabilities are really poor – if the toolbar has not enough space to hold all added elements it blocks the downsizing instead of hiding items and adding “more” button.

## 3.7 Address search

The address search functionality allows users to enter an address into text field and get pins on the map pointing to the location of that address.

Model of this functionality should provide several interfaces for:

- setting the search phrase
- running the search
- registering observers – views, other models and notifying them about changes in data
- getting the search results in form of a list of addresses with exact coordinates
- Clearing stored data.

Implementation of most elements from above list is very straight forward except for one: running the search. GeoCoder service uses XLS format for data exchange which is sent in the body of HTTP POST request. It requires providing exactly three parameters: municipality name, street name and street number. As one of the requirements states that the final application should be simple, the decision was made that user should not be constraint to one schema when providing the address search phrase. Parsing of string which may take different forms is a laborious process, so it was moved to the server side, together with the call to GeoCoder service. The client side search functionality is cut to calling the server side using GWT-RPC mechanism with search phrase as parameter, and getting the results as a list of addresses.

The servlet responsible for searching splits the task into three phases: parsing search phrase, calling the GeoCoder service, and parsing the response into a list of addresses.

The parsing process should be as flexible as possible to accept the majority of schemas. The first assumption was that the order of words in the phrase is not fixed, so processing of every word should be separated. Words in this case are sequences of characters which were split using some common separators: space, semicolon. Unfortunately, in Finland municipality or street names may consist of few words which are separated with space, for example, of a dash. Such case should also be considered. The algorithm of parsing consists of several steps to get trio of parameters needed to call GeoCoder:

- Split words are filtered and assigned to two groups: numeric (words starting with number e.g. 4C) and non-numeric (Helsinki)
- By iterating through the list of non-numeric words, possible post office names are found using a fixed list of all known in Finland
- For every possible post office match:
    - Other parts of the post office name are filtered (in case of multi word name) as well as the name of municipality in which the post office exists (by using post office to municipality mappings)
    - From the rest of the words the street name is formed
    - Permutations of municipality name, street name and street number are created and added to the list of possible addresses.

It is clearly visible that the main weakness of this algorithm is the impossibility to check if the street name combined from the rest of the words is a real street name which exists in the specific municipality. The only way to find out is to check all permutations one by one by calling GeoCoder. In case of the search phrase which – when processing - gives many possible post office matches, and additionally many street numbers, the final count of permutations to check might be overwhelming. That was the main reason to limit the overall time needed for calling GeoCoder service to five seconds. Subjective tests proved that such amount of time gives results with almost correct address phrase, but in case of very “wide” range of possible hits to check, waiting does not bother the user.

Every request to GeoCoder has to be built and every response parsed accordingly to XLS specification. There are different Java tools for manipulating XML data:

- DOM parsers
- SAX parsers
- JAXB.

DOM parsers are considered a very heavy solution. SAX is a lightweight parser but requires a lot of coding especially in case when the schema of the document is complicated. In this project JAXB was chosen as it provides the possibility of working with XML document in a similar way as it would be a Java object.

Preparing the request was limited to set three values in the XLS structure: municipality, street name and number. Parsing the response was also very easy and straight forward – reading the type of the response (exceptional or valid) and in case of valid response, reading the coordinates assigned to the address.

The model of this functionality has at least two observers:

- View which is part of the user interface and dispatches user actions to the Controller: entering the search phrase, hitting the search and clear buttons
- Model of the map component which observes changes of address search results.

## 3.8 Search

Search functionality enables users to perform search of selling places using spatial and non-spatial criteria and viewing the results in a form of a list and as geometries on the map.

As in the previous chapter, the Model of this functionality should provide a set of methods for:

- Getting possible values of specific criteria which may be used during the search
- Setting the chosen values which should be used for searching (including spatial criteria)
- Performing a search
- Getting the list of results and highlighted results
- Adding, removing and highlighting of search results
- Fetching features with specified ids
- Registering observers.

Users looking for selling places using a form with different fields should get hints about possible values which may be used for searching. To provide such functionality, contents of the database have to be checked and unique values for different criteria retrieved. The database contents rarely change in comparison to frequency of users using the application. Due to that fact the decision was made that possible values will be loaded on model initialization. The whole process can be presented in few steps:

- Calling server side using GWT-RPC
- On server side:
    - Querying WFS provided by GeoServer using GeoTools for unique values of specific feature attributes
    - Processing data (sorting, etc.)
    - Returning the result using custom DTO to Client
- Saving the values and notifying observers about changes.

Searching started by the user is done in the similar manner as in the previous chapter. The whole functionality was moved to server side, where criteria entered by the user are sent altogether. From those criteria using GeoTools, filter instance is created and used in query to WFS. The result is a set of features which are converted into DTO compatible with JavaScript. The client side after getting the results saves them and notifies observers.

The model in this functionality has at least three observers:

- View which allows entering or choosing values used for searching
- View showing the results in form of a list, and allowing to perform different operations on them
- Model of map component.

The view responsible for showing and interaction with results should dispatch all events into the Controller which will modify Model accordingly or will communicate with other controllers to perform actions which are out of scope of this functionality (e.g. zooming map to specified bounding box, getting map bounding box for spatial query).

## 3.9 Map

The map view in this project plays a very important role. It allows users to view selling places and results of address search and selling place search functionality.

The map component is a part of OpenLayers JavaScript library. GWT wrapper of the library gives the ability to use it in GWT applications. It is open source project with very small test coverage. To avoid a situation that the application may not work because of the wrapper

malfunction, the map setup was first tested in clean JavaScript environment. A separate project for that purpose was created, and used every time when a new functionality of OpenLayers library had to be used. It was a simple static web application with a single main page, all the libraries needed and single JavaScript source file responsible for configuration of the application. Separation of the source code from HTML document is very helpful because it enables the syntax check and auto completion feature of Eclipse for JavaScript code.

The model of this functionality should provide many more interfaces than in previous chapters. There should be methods for:

- Setting search and address search model. Those should be observed for changes of results
- Getting initialized layers with map images from different providers
- Getting initialized layer of address search results which content depends on results of address search model
- Getting initialized overlay layers with selling places visible in different styles depending on the state:
    - Default
    - Found – when the feature is in the list of results of search model
    - Selected – when the feature from the list of results is highlighted by the user
- Setting and getting a bubble popup window anchored to a place on the map.

Layers with map images are often called base layers because all other components like layers and popups are laid over them. Usually map images are provided using WMS but other protocols are used as well. At this stage of the project, two providers should be supported: Open Street Map and Maanmittauslaitos. Unfortunately, those maps are provided using different CRS, and OpenLayers can work with only one CRS at the time. Re-initialization of the map component during runtime to switch between CRSs is not possible in the currently used version (the problem is solved in upcoming version 3) At this point the decision was made that the application should launch with a specified map provider as a startup parameter.

Open Street Map is a map created by open community. People around the world are improving the quality of the map for free and anyone can use it (commercial use is allowed).

(OSM, 2011)Support of OSM is added using extension library, which defines how OpenLayers should get tiles from the server. Configuration of such layer is reduced to instantiation of the layer. The overview map control can also use this layer as a source of miniature map images used only for navigation.

Maanmittauslaitos is a leading Finnish map provider. Maps are sold in different packages, and the price is based on the amount of request to WMS. There are several maps offered which mainly differ in the level of detail. For this project a demo package was acquired which has all levels of details included. Unfortunately, WMS used by Maanmittauslaitos does not support dynamic changing of level of details depending on the zoom level. For that reason a fake background layer was created just to setup maximum and minimum scale of the map component. The background maps were added as overlays, which were automatically switching depending on the scale the map was displaying. This way when the user was zooming in, the map was providing more detailed background. Determining the scale boundaries between which proper maps should be shown was based on the subjective tests. The problem with switching the layers was not the only one in case of Maanmittauslaitos WMS. Because of creating fake base layer and the CRS in which map is provided, the overview map did not work out of the box. An additional layer was needed with custom configuration.

As the specification states, the results of the address search should be visible as points on the map with additional information available in a popup bubble. To display the results of the address search, the data stored in its model must be converted in to a type which the map component can understand (the operation is performed every time data in the model changes). There are two possibilities of displaying points on the map. Old Marker layer which is discouraged since it has been deprecated and Vector layer which supports features. Translation from a custom address object (with coordinates included) into a feature requires creating geometry of the future (in this case point with coordinates) and adding attributes which consist of information about the address. Styling features will be discussed later in this chapter.

Selling places should be visible to the user so he/she can view them directly from the map. Those which are found using search tool, or highlighted should differ in appearance. In addition, if geometry of a selling place is hardly visible in the specific scale, it should be replaced with a different one – more visible (e.g. points are not rescaled during zooming). There are different ways to achieve such effect but one additionally gives the user the ability to switch the visibility of everyone of those groups. There should be two sets of vector layers, one which is active when geometries of layers are clearly visible and the second one which provides points instead of original geometries and is visible in other cases. Every set should have a separate layer with different styles for regular selling places, those which were found or added using a map and those which were highlighted.

Vector layers in OpenLayers might be automatically filled with features using different strategies. In this project two of them were used: fixed and bounding box.

Fixed strategy is one of the simplest ones in the library. The layer which uses it, makes a single call to the providing service, and gets all features in the format and CRS specified during the request. There several advantages of such solution:

- CRS of returned features can be specified
- All the features are downloaded at once so after initial fetch the application will work faster – no waiting times for downloading more features.

On the other hand, in case of a huge amount of features in database the initial fetch may take a long time.

Supporting of showing the features on different layers with main layer using fixed strategy is also very simple. Since all the features are available from the beginning the only thing needed is switching the layer they are attached to on user interaction. The data needed for showing popups over the geometries can also be retrieved easily from the feature.

Bounding box strategy has a radically different approach. Instead of downloading all the features, only those which can be visible on the map are downloaded. The request to WFS contains information about viewport of the map, so the service knows which features it should return. This solution lowers the traffic used (only features which the user wants to see

are downloaded), but on the other hand, the returned features must be translated into correct CRS and the final user experience may suffer (because of the time need for fetching and translation of features).

Translation of coordinates in OpenLayers is done using Projection class, which is only a wrapper for Proj4js library. Proj4js is a port of a very popular projection library written in C – "proj.4". It uses custom definition and projection files of most popular coordinate reference systems. To be able to use the "EPSG:2393" which is specific for Finland an additional projection file has to be added to the library.

In case of bounding box, switching features between layers is not an option. The layer with this strategy dynamically changes its content, so the contents of the other two layers also had to be dynamic. The content of the main layer (the one which fetches the features from the service) is copied to other two layers depending on the values in the model of search functionality. Such operation is performed every time the content of the model changes, or when the user moves the map, and the layer fetches the additional features.

Interaction of the user with the content displayed on the map is provided by "SelectFeature" control. It is an opaque layer on top of all base layers and overlays that checks what feature was under the cursor at the time of making the mouse click, and launches the registered handlers for specific actions. In the older versions of OpenLayers library the control supported only a single layer, but in the version used for this project (2.10) "SelectFeature" supports multiply layers at the same time.

Style of the feature depends on the style applied to the layer to which the feature is attached. There are different style attributes that can be adjusted. Styles are applied to every type of geometry, but in case of point there are two possibilities:

- Showing point as a circle with adjustable radius
- Showing point as an image with specified size and position relative to the place on the map (possibility to create different kinds of pins and pointers).

Additionally, OpenLayers supports several internal states of the feature. Depending on that state a different style may be applied (e.g. hovering over the feature may change the style in which it is displayed).

## 3.10 Documentation

The documentation is a very important part of every software project and especially in a project that will be a foundation for other similar projects. The main purpose of making documentation is to allow other developers working on the project easily get to know the project structure and details. This way the time needed for familiarizing with the project is a lot shorter.

The first and a very important matter was to document the way the whole project structure was created and what tools were used. At the time of creation of the project mainly Ant build files were used for different projects in the company. Such solution works well, but requires a great deal of configuration. In case of Maven and its main policy – convention over configuration – the setup of the project is very easy.

The second useful feature for future development of the project was an overview map of the layout with descriptions which classes provide what functionalities. Such diagram helps in cases when only a small part of the application needs to be improved – developers do not have to familiarize themselves with the whole project.

The main and the most important part of the documentation in case of Java programming language is Javadoc. The documentation of functions and classes is instantly available in Eclipse or other IDE, so developers can see what the function does without looking into a separate documentation file or to the source code.

# 4 RESULTS

The main result of the process reported in this thesis is an interactive web application which allows publishing of geographical data (See FIGURE 10). The solution from the beginning was created according to specified requirements. It allows visitors to interact with published data in an intuitive manner. There are facilities provided for searching places on the map, as well as browsing and searching through the published collection.

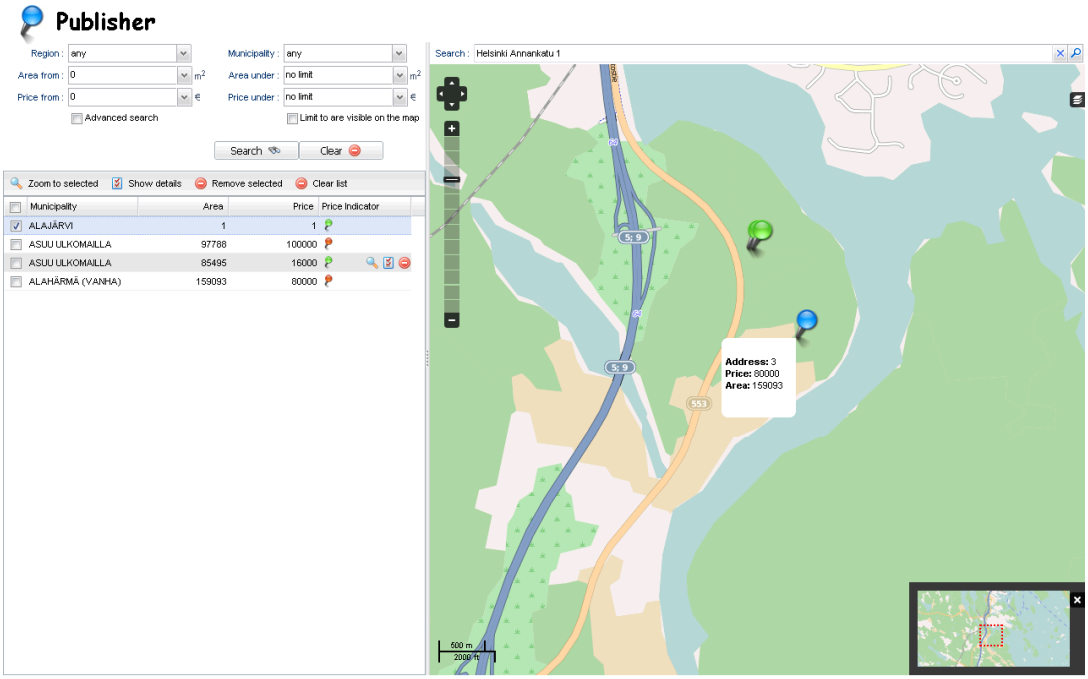


FIGURE 10 Final look of the application.

The development of the application using GWT made fulfilling most of the non-functional requirements a lot easier. GWT by default supports several browsers, and the application is more stable as development using Java language allows faster error detection. Translating the application to different languages is simplified. It only requires creating of translation file for every language. On the other hand the use of the GWT wrappers built on top of JavaScript libraries undermines the efforts of GWT designers as the compiler does not have a chance to analyze and optimize those wrapped libraries.

# 5 DISCUSSION

This thesis documents development process of a rich internet application which is able to present geographical data to the users. The entire process consists of several stages.

Analysis of requirements and architecture demanded the author to familiarize himself with terms and technologies used in the specification. The correct setup of development environment allowed uninterrupted progress through further stages. Modular structure of the solution ensured easy application of future modifications and improvements Simple and ergonomic layout design hides the complicated implementation and allows users with different level of technical knowledge to make full use of latent functionality. Embedded, interactive map is the most natural way of working with spatial data. Complete technical documentation of the project will allow faster adaptation of new developers and easier modification of the current version of the application.

The final result is a fully featured application ready for release. It provides similar or better functionalities in comparison to competitive solutions. As every other software project it may be further improved in many aspects:

- Look – more eye-catching appearance
- Custom controls – for better usability and user experience
- Accessibility – to make the use of application easier for disabled visitors or those with special needs
- Functionality
    - Possibility to add, modify and remove presented data – turning the publisher application into fully functional administrative solution
    - Authentication of users visiting the website and presenting different sets of data depending on the user's role.

During implementation of the presented application the author of this thesis went through the entire development process in corporate environment full of enterprise standards. The knowledge and experience gained will be extremely useful and valuable in the future carrier as a software developer.

# REFERENCES


Artifactory, 2011, About Artifactory. Accessed on May 11th, 2011, http://www.jfrog.com/products.php

Bitcomp, 2011, Bitcomp Oy. Accessed on May 11th, 2011, https://www.bitcomp.fi/fi/index.php?lang=en

Cargo, 2011, Cargo – Home. Accessed on May 11th, 2011, http://cargo.codehaus.org/

CVS, 2011, CVS – Open Source Version Control. Accessed on May 11th, 2011, http://www.nongnu.org/cvs/

Eclipse, 2011, The Eclipse Foundation open source community software. Accessed on May 11th, 2011, http://www.eclipse.org/

GeoServer, 2011, GeoServer. Accessed on May 11th, 2011, http://geoserver.org

GeoTools, 2011, GeoTools The Open Source Java GIS Toolkit. Accessed on May 11th, 2011, http://www.geotools.org/

Google API for GWT, 2011, Making Google API Library for GWT better. Accessed on May 11th, 2011, http://code.google.com/p/gwt-google-apis/wiki/MakingGALGWTBetter

GWT Dev Guide, 2011, Google Web Toolkit, Developer's Guide. Accessed on May 11th, 2011. http://code.google.com/intl/en-US/webtoolkit/doc/2.3/

GWT Logging library, 2011, Logging Library for Google Web Toolkit (GWT) with Deferred Binding. Accessed on May 11th, 2011, http://code.google.com/p/gwt-log/

GWT Maven, 2011, Maven GWT Plug-in. Accessed on May 11th, 2011, http://mojo.codehaus.org/gwt-maven-plugin/

Horstmann Cay S. and Cornell G, 2007, Core Java Volume I – Fundamentals – 8th edition, Indianapolis

HtmlUnit, 2011, HtmlUnit. Accessed on May 11th, 2011, http://htmlunit.sourceforge.net/

Hudson, 2011, Hudson Continuous Integration. Accessed on May 11th, 2011, http://hudson-ci.org/

JUnit, 2011, Resources for Test Driven Development. Accessed on May 11th, 2011, http://www.junit.org/

M2Eclipse, 2011, M2Eclipse | Sonatype. Accessed on May 11th, 2011, http://m2eclipse.sonatype.org/

Moore D, Budd R and Benson E, 2007, Professional Rich Internet Applications: AJAX and Beyond, Indianapolis

O'Brien T, Casey J and Fox B, 2010, Maven: The complete reference, version 0.8. Accessed on May 11th, 2011, http://www.sonatype.com/books/mvnref-book/reference/public-book.html

OGC 04-094, 1995, Web Feature Service Implementation Specification. Open Geospatial Consortium

OGC 06-042, 2006, Web Map Service Implementation Specification. Open Geospatial Consortium

OGC Website, 2011, OGC Website. Accessed on May 11th, 2011, http://www.opengeospatial.org/

OpenLayers, 2011, OpenLayers: Free Maps for the Web. Accessed on May 11th, 2011, http://openlayers.org/

OSM, 2011, Open Street Map. Accessed on May 11th, 2011, http://www.openstreetmap.org/

Selenium, 2011, Selenium web application testing system. Accessed on May 11th, 2011, http://seleniumhq.org/

Smart GWT, 2011, Isomorphic Software | Smart GWT. Accessed on May 11th, 2011, http://www.smartclient.com/product/sgwtOverview.jsp

Sonar, 2011, Sonar. Accessed on May 11th, 2011, http://www.sonarsource.org/

StatCounter, 2011, Top 5 Browsers on May 11th. Accessed on May 11th, 2011, http://gs.statcounter.com/