Mid Sweden University
The Department of Information Technology and Media (ITM)
Author: Lena Sundin
E-mail address: lesu0901@student.miun.se
Study program: Programvaruteknik, 180 points
Examiner: Ulf Jennehag, ulf.jennehag@miun.se
Tutor: Johan Timrén, johan.timren@miun.se
Scope: 17358 words inclusive of appendices
Date: 2012-09-08

B.A. Thesis within Computer Science
DT099G, 15 points

Saving Lives With Geo-Spatial Web Standards

A Test Architecture for Evaluating the Possibility of Sharing Heterogeneous Data Among the Emergency Services

Lena Sundin

Abstract

The emergency services in neighboring Jämtland, Sweden and Nord-Sør Trøndelag, Norway have identified a need to share data across department, municipality and country borders. A project is started to evaluate the possibility of sharing information through a common Web Geographic Information System (GIS).

The data about resources at the various departments suffers from a high level of heterogeneity, fragmentation and protocol incompatibility. The Open Geo-Spatial Consortium (OGC) issue Web standards to harmonize the processing of geo-spatial data and promote interoperability between GIS systems. A test model based on the Thin Thread Model, emulating a potential final solution, is built to evaluate the usability of these Web standards in the situation. Successful test cases including CRUD operations and relatively smooth swapping of layer modules show that using well-established standards can be beneficial. A proposed architecture extending the test model presents the idea of a centralized proxy node and a meta data catalog.

The study highlights the issue of responsibility and question of which authority should maintain centralized nodes. In order to successfully implement a Web portal, the project participants must from a technical point of view investigate how to access all desired data, agree on protocols for communication and ensure that each owner of data provides an API in agreement with the protocol. The OGC Web standards are proven a good option and focal point.

Keywords: Geo-spatiality, OGC, WMS, WFS, GIS

Contributions

My deepest appreciation, admiration and gratitude goes towards Johan Timrén, the tutor at Mid Sweden University. Battling a life-threatening disease, you keep your head up, spirits strong and inspiration generously flowing. You have shown me a whole new world of knowledge and drastically improved my skills as a programmer. This study would not exist without you.

Börje Hansson, thank you for suggesting me as a candidate for the project. And thank you for invaluable wisdom, guidance and patience.

Simon Keskitalo, project leader of GGI2, thank you for a spot in the project and a trust in my skills. Time has not been on our side, yet we did the best with what we had.

Thank you to Cleber Arruda, Malmö municipality and Malsor Limani, Jönköping municipality, for providing hands-on help when the programming became stagnant. Special thanks to Cleber for suggesting the proxy HTTP request circumvention in OpenLayers.

Table of Contents

Abbreviations

CRS	Coordinate Reference System
CRUD	Create, Read, Update, Delete
EPSG	European Petroleum Survey Group
GGI2	Gränslös Geografisk Information, projekt 2
GIS	Geographic Information System
GML	Geography Markup Language
OGC	Open Geo-Spatial Consortium
SFA	Simple Feature Access
SOAP	Simple Object Access Protocol
SOSI	Samordnet opplegg for stedfestet informasjon
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TMS	Tile Map Service
WKB	Well-Known Binary
WFS	Web Feature Service
WFS-T	Web Feature Service Transactional
WKT	Well-Known Text
WMS	Web Map Service

1. Introduction

Few government services are more important than those saving our lives. Proper preparation, documentation of resources, cooperation and useful tools are vital to the operations of the fire, health and police departments. As geographical distances seem to shrink with the help of ever-faster transportation, the span of responsibility of the emergency services grows larger. Unfortunately the IT systems are often lagging behind due to restrictive budgets or lack of technical competence. This report will investigate how the latest standards in geographical Web application technology can be used to build a Web interface for sharing data as a step towards better efficiency.

1.1. Background

The fire, health and police departments on each side of the Swedish-Norwegian border in the region of Jämtland and Nord-Sør Trøndelag have identified operational shortcomings. Each unit, department, county (Swedish *län*, Norwegian *fylke*) and country functions as an isolated entity without regard to the others. There are no common IT systems for sharing data about resources which manifests itself as confusion and frustration during emergency response actions. The County Administrative Boards of Sweden and Norway (Sw. *Länsstyrelsen*, No. *Fylkesmannen*) have agreed that a common method of sharing data is needed throughout the region (depicted in Illustration 1).

Illustration 1:
Jämtland and Nord-Sør Trøndelag

1.2. Lack of Interoperability

A study issued by the U.S National Institute of Standards and Technology estimated the costs of inadequate interoperability in the American construction industry to surmount $15.8 billion per year [1]. The inadequacy was explained as the product of 1) the fragmented

nature of the industry with its long history and many big, as well as small, actors and 2) manual re-entry of data, duplication of business functions and paper-based business models. Number 1) includes issues internally (lack of system interoperability) as well as externally (towards other companies). Number 2) consumes significant amounts of time which translates into increased costs. These circumstances are not unique to the American construction industry - the emergency departments of Jämtland/Nord-Sør Trøndelag report working under the exact same conditions.

1) Fragmentation

The structure of the Swedish and Norwegian emergency services is devised to give each department autonomy down to the unit level. For example, the fire stations in Järpen, Jämtland and Östersund, Jämtland are located in the same county roughly 70 km apart. Yet they run separate agendas and use different means of data-tracking. In the event of an emergency, the response coordinators often find themselves wondering which vehicles and tools could be fetched from nearby. The current solution consists of inquiry via phone. Needless to say, cooperative tools between countries and departments are just as scarce. One exception is the police, which on a national level keeps criminal registries accessible to all units. Furthermore, the RAKEL mobile communication system is used nationally by all police units, increasingly gaining momentum also with the health and fire services [2]. However, the RAKEL tool is not a means of sharing data through a Web interface, but rather a communication channel.

2) Time-Consuming Redundancy and Paper Work

Many of the involved parties do not have efficient tools for working with data, for example one of the fire stations uses a flat file document. The concept of data modelling is foreign to the key consumers and producers of data (concepts such as using UML diagrams, grouping data into entities, understanding attributes etc.) The systems are very prone to redundancy errors and can be difficult to search and query.

1.3. Organizational Structure

The initiative is started, funded and supported by a group of decision-makers consisting of government officials from the County Administrative Boards of Sweden and Norway. They give funding to projects that produce results and reports which guide and legitimize the decision-making process. The Boards weigh economical, political and societal factors against each other and often employ experts to make reasonable estimations. Gränslös Geografisk Information, projekt 2 (GGI2, En. Border-less Geographical Information, project 2) is a project established to handle the problem presented in this report. It consists of

representatives from the emergency departments, Geographical Information System (GIS) experts, web developers and project coordinators. The goals for the project are:

- Geographical information shall be available to the users.
- The users shall have enough competence to use the geographical information.
- Common practice scenarios increase the understanding of how to use geographical systems and improve the possibilities for efficient cooperation across the border.

The group coordinators' task is to research potential solutions, deliver an estimation of future progression and answer the question *Where do we go from here?* The ambition is to find an economically, legally and practically plausible solution. A further aspiration is to attract attention on a national level and perhaps involve or influence higher Administrative Boards.

Through GGI2, the system end users gather in meetings to determine what data they have on hand and what they need to share. The process involves surveys and discussions with GIS and Web experts. The results are logged and utilized in the progression of GGI2 and this report. See Appendix A for one of the surveys taken by the users (NB. the survey is authored by GGI2).

The project is not expected to result in a final product, but rather an estimation of the resources required to solve the task at hand. If deemed manageable, the possibility for further funding and development is high and more projects will probably be initiated to materialize the proposed ideas.

1.4. High-Level Aim

This study will focus on the technical aspect of the problem with the overall aim to answer how the desired geographical data practically can be accumulated and displayed. Since time and cost are important factors, this will be done by looking at already existing standards within the field of geo-spatial Web technology. Following already existing standards has two major advantages: 1) Promotion of interoperability and 2) Reducing programming time and effort. The low-level aims will be defined in chapter 2. *Problem Formulation*.

1.5. Target Audience

The report aim is to produce an estimation of the technical possibilities which will act as guidance and input/output in the GGI2 project. It is written for project coordinators and decision makers who may not have

knowledge of common IT terms. The theoretical section must however keep a certain degree of technical depth and will be of more interest to an audience of Web developers.

1.6. Scope

Limiting the scope is an important issue since the task at hand is so massive. The report focuses solely on the question whether geo-spatial Web standards are a possible technology to use in the future software that will solve the problem.

Many different standards exist to complete the given task. The study will zone in on a couple of the most prevalent standards and divert attention towards a few of these, including Web Feature Service (WFS), Web Map Service (WMS) and Tile Map Service (TMS).

The participating departments and their data fall under various categories of information classification. Details such as references by name and organizational routines will be left out. The report writer and GGI2 project coordinators may be subject to legal ramifications in the case of information leakage. The tendency of the report will, as a precaution, lean more towards general descriptions of concepts, as opposed to exemplification by fine-grained example.

The report will not present a solution to the over-all problem. Every actor involved in GGI2 understands that a final product is outside the frames of reason given the time and resources. The issue is so large and complex that it might ultimately only be solved on a national level.

1.7. Outline

1 Introduction has presented an initial picture, problem domain and high-level task.

2 Problem Formulation further investigates the issue and defines low-level, verifiable goals.

3 Model explains the thin thread model that will be used in building a test architecture for the study.

4 Theory investigates the science behind geo-spatial Web standards, explaining concepts reoccurring in program code that are important for deeper understanding.

5 Implementation describes the test model to provide the greatest degree of replication possible.

6 Results anchors the results of test cases in relation to the research goals.

7 Discussion evaluates the relevance and importance of following geo-spatial Web standards, proposes an architecture for future Web portal implementations and finally gives a few pointers as to where the project should focus next.

2. Problem Formulation

This section describes more thoroughly the implications of the task at hand and defines concrete goals that will result in a verification or falsification of a hypothesis.

2.1. Concepts

The report will pivot around a few central concepts that can be interpreted differently depending on context. The concepts used shall be interpreted as follows:

Resource Data The term 'data' can take on many meanings. In this study - the information about resources that the emergency departments wish to share among each other. This information will be treated and also referred to as *geo-spatial* data, a term described thoroughly in the theoretical section of this report.

Meta data Data about resource data, ie. information about how and where to fetch data, on what format to expect data and so on.

Object May refer to any single component as contained in a data source, such as a tuple representing a fire truck.

Decentralization Distribution of governance and administration over several units, in this case leading to semi-autonomy.

Fragmentation A consequence of decentralization, where units operate autonomously within different systems.

Esotericism Generally referring to ideas and beliefs only understood by a group of select people. In this context, referring to non-standard data storage methods invented and supported locally, such as Word documents.

Interoperability Attribute of systems functioning together and communicating without major re-hauls. It is a desirable property since it enables separate modules to function together in a plug-and-play fashion, promoting sharing of resources. The development of standards is highly driven by the urge for interoperability.

Heterogeneity The state of data that comes in different formats or from various types of sources. The consequence of heterogeneous data is a lack of interoperability. The opposite, homogeneity, refers to interoperable data.

Harmonization The process of restructuring data with the purpose of making it homogeneous/harmonizing.

Washing Converting data to a format appropriate for a given situation.

2.2. Outlining the problem

It has been determined that the data contained at the emergency departments is heterogeneous and the systems are not interoperable. A deeper understanding of the current state is obtained via meetings with representatives and studying the output of GGI2 surveys. Appendix A contains one of the surveys taken by the users within the project. These surveys and discussions allow the current situation to be approached from a technical angle. The results can be summarized in a tree-like structure with 3 main branches: Resource Data, Meta data and Demands on Data.

- Resource Data

  .1 Objects are represented with varying parameters and attributes, e.g. the information associated with a fire truck at one department may not be identical to other representations.

  .2 Varying methods of data storage; relational databases from different vendors, commercial products concealed behind proprietary APIs and licenses, flat files (Microsoft Word and Excel) and staff knowledge

  .3 Objects are not always associated with geo-spatial data.

  .4 Decentralization and autonomy have caused many *ad hoc* solutions and varying methods of retrieval.

- Meta data

  .1 Information about how and where to fetch resource data is often lacking.

  .2 Existing meta data is decentralized and heterogeneous.

- Demands on data

  .1 Abstraction levels vary from unit to unit, for example the specific details of an ambulance's contents may be more interesting to ambulance operators than police officers - a police officer may only know if the ambulance exists or not.

2.3. Three Examples

Three examples taken from the current situation will demonstrate the breadth and variation in data storage methods. The examples taken span from 'most usable to 'least usable'. 'Usable' in this context refers to how ready the data is for use as a means to solve the problem at hand, using geo-spatial Web standards.

2.3.1. Norwegian Directory of Common Safety

This could be considered the most organized data source as far as the author has been able to investigate. The Norwegian side (*Direktoratet for samfunnssikkerhet*) has produced geo-spatial data and conducted projects more extensively than other participants. This has resulted in an OGC standard Web Map Service (WMS) node in the EPSG:4326 projection. The service offers raster layers of fire stations, district offices and related facilities. Ideally, the data would be presented in the WFS format to allow sharing of attributes. A WMS service only produces points on a map which may not be sufficient in an emergency situation. Illustration 2 shows an example of the rendered output of the service.

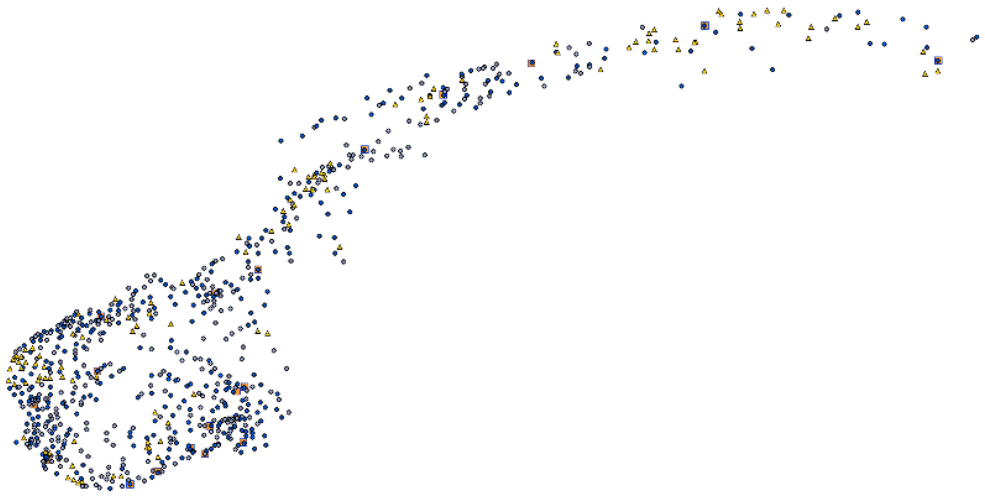

Illustration 2: *Norwegian fire stations and regional offices*

2.3.2. Norwegian Central Town Registry

The Norwegian Federal Map Unit (*Statens kartverk*) offers a service for searching towns through the Central Town Registry (*Sentrale stedsnavnsregister*). It is consumed as a Web Service via SOAP/XML. The service is available to the public at https://ws.geonorge.no/SKWS52/. This may come across as convenient, but it presents a number of issues. First of all, the service does not follow any OGC standard. Furthermore, studying the documentation reveals that attributes delivered are spatially referenced via the Samordnet Opplegg for Stedfestet Informasjon (SOSI) data format which was

crafted in 1987 for usage in Norway exclusively [35]. The shape of the earth and science of geodetics is the reason countries invent their own, unique Coordinate Reference Systems.

2.3.3. A Small Fire Station in Jämtland

A fire station, that shall go unnamed for security reasons, serving a small community in Jämtland has only 3 employees. Their funding is minimal and investing in proper data storage has been a lesser priority. The current method of storing data is via flat files (Word documents) where information about existing fire vehicles, important phone numbers etc. is entered manually. There is no given structure, but it is structured enough to provide the employees with the bare minimum thus far. This method is the least organized out of all the project participants' methods, and suffers from a high degree of esotericism.

2.4. Outlining Desired Functionality

The meetings resulting in the outlining of problems also result in discussions regarding desired functionality of a common Web portal. Representatives from each department have identified a set of stipulations for the potential solution and decided that a meaningful common interface will contain at least the following:

- Detailed maps of the region
- Resources at all departments such as vehicles, tools, knowledge etc.
- Addresses and phone numbers to all departments and resource pools
- Information about buildings, estates and populations

Some further functionality is desired but not a requirement. The departments are open to future solutions enabling the following:

- Ability to create dynamic events in an interface corresponding to real and on-going events. These should be linked to data about resources and function as a tool for administration and communication.
- Ability to watch vehicles move live on a map.
- Saving 'views', ie. bookmarking a set of interesting resources to quickly pull them up with one click.

A test model will not be able to cover a full solution, and the study makes no claim to do so. The test model of this study will stretch as far as

involving all possible technologies and tools, to the extent that their functionality can be evaluated. It will be performed via test cases which, upon success, will show that the fundamentals are covered and hopefully ready for expansion.

2.5. Establishing Goals

The emergency departments have decided that they need a common gateway, a Web map application, in order to share their data. It needs to solve the problems outlined in the previous sections, preferably within reasonable time and cost margins. The aim of the study is to evaluate existing geo-spatial Web standards as an option in the situation. Technologies sharing standards have the advantage that modules within an architecture can communicate without the need for interpreters or tedious translation. A whole architecture could be built if there was a common way to communicate geo-spatial data. Furthermore, the final system may be more interoperable with other systems implementing the same standards. There is also a time aspect to consider - using existing standards and techniques can save time and subsequently money. Questions that must be answered are:

- What are the latest technical standards offered for developing Web applications of this sort?
- How do these standards measure as a solution in the given situation?
- If all data could be translated into a standard format and transmitted via standard protocols, would that be sufficient to build a final implementation?
- Evaluating these standards, can anything be said about time and cost associated with the proposed solution?

These questions will be answered via building a test model based on geo-spatial Web standards through the means of frameworks and tools offered freely by the open programming community.

This report has two goals. The first goal achieves an environment for an empirical study to be conducted. The second goal performs the empirical study.

- Enable testing of a few of the most common geo-spatial Web standards by using technologies for accessing and performing simple operations such as fetching data about a fire station, modifying it and writing it back to the store. This should be done through a user interface.

- The test model must emulate a potential solution by implementing software layers. Each layer must be implemented through at least two different technologies to portray the scenario of a heterogeneous architecture. The modularity of layers and usability of Web standards shall be tested by swapping between technologies and estimating the degree of effort required to perform a swap. Each of the following software components (layers) must at least be involved

  .1) Data Source - An underlying database containing geographical information about resources.

  .2) Data Access - Methods for accessing a data source

  .3) User Interface - Methods for presenting the data to the user

2.6. Tools

Frameworks will be used to avoid the time-consuming effort involved in building Web applications, such as opening connections, designing a harmonic user interface, handling click actions etc. The chosen framework stack (PostGIS, MapServer, OpenLayers) consists of a set of open-source tools built around current geo-spatial Web standards. The technologies are all free of charge, although costs may be associated with server and bandwidth housing.

Proprietary programs are not an option due to budget restraints. The tools have been chosen for this study because they are very plausible candidates for a real, future solution.They have been vigorously tested, are constantly developed by the open-source community and are not associated with license costs. From a programmer's point of view, there is also more flexibility and possibilities to alter the frameworks as desired.

3. Model

The concept used is the *thin thread model* in which a system's components are implemented fully from top to bottom layer [3 p 289-290]. 'Layers' in the context of software engineering refers to the separate instances of a system, from user interface to underlying data source and everything in between. Any instance that may operate separately from others can be considered a layer. The purpose of dividing software into layers is the *separation of concerns* that inherently follows. This means decoupling modules and making them independent from each other which is important in large applications [4]. It allows modules to be switched out or changed without effecting other modules. Separating concerns improves code maintainability and alleviates the distribution of nodes.

Small test cases are sent to run through the architecture like 'threads'. The objective is to put focus on systemic issues. The concept of the thin thread model is depicted in Illustration 3 where the vertically stacked segments represent system layers. The red thread is a test case. This shows how the model may not encompass the full width of a system, but on the other hand it penetrates the full vertical axis.

The model involves at least three layers; Data Source, Data Access and User Interface. It must be functional enough to offer CRUD (Create, Read, Update, Delete) operations that propagate all the way from front to back-end. Each layer must involve the use of geo-spatial Web standards. This will practically be implemented as asoftware architecture built on a stack of frameworks (PostGIS, MapServer/TinyOWS and OpenLayers).

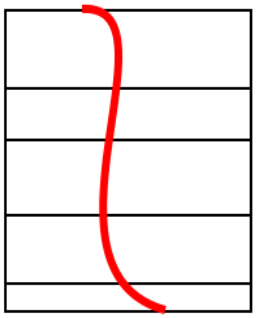

Illustration 3: *The thin thread model*

The choice of model is motivated by the fact that a real, final interface is going to be very large-scale and contain many layers. Resources are going to be distributed and probably delivered in a Web Services-

oriented manner. A holistic test case will thus be more meaningful than a fine-grained one. The thin thread model is suitable for testing Web services structures.

3.1. Empirically Evaluating the Model

The empirical study will involve swapping out modules in the architecture. It is of interest to achieve a feel for the interoperability between different technologies implementing the same standards. The degree of human involvement – refactoring of code, reconfiguration of servers etc. – will be used as a measure to evaluate the flexibility of the model. Complete lack of need for reconfiguration will be considered a best case.

This is not a true empirical study since no hard values are measured. The measurement will consist of an estimation made by the report writer, relative to the best case scenario. This is a known issue, but the results will hopefully still be meaningful and produce an estimation for the GGI2 project.

3.2. Criticism of the Model

The model does not fully depict a whole solution and that may effect the results. What works well in a test environment may not function in a production environment and the simplicity of the model does not properly represent the complexity of the final system - but this is not the point. The study is an effort to start investigating existing technologies and standards and how they can possibly be woven together and utilized. The research will also result in a grasp of the latest advancements of geo-spatial Web technology.

Further criticism of the model presents itself when scrutinizing the stack of tools. The frameworks are explicitly designed to be interoperable which may skew the results. However, they are not dependent on each other and theoretically could be swapped out with any other framework supporting geo-spatial standards.

4. Theory

This section is divided into four chapters. The first chapter 4.1 will give an introduction to geodetics and the science behind geo-spatial technology. Concepts discussed will be used extensively throughout the report and are crucial in the understanding of geo-spatial Web standards. Programming an application with the chosen framework stack is next to impossible without some knowledge of the underlying science. The second chapter 4.2 explains the most common methods for storing geo-spatial data. The third chapter 4.3 introduces the Open Geo-Spatial Consortium and their most widely used Web standards. The final chapter is a case study of a software architecture project battling the issue of data heterogeneity.

4.1. Geo-Spatial Concepts

4.1.1. Geographic Information Systems

Geo-spatial data is information that can be referenced as a point or object within our earth's scope [5]. Geo-spatiality involves three dimensions (longitude, latitude, elevation) plus additional data and/or dimensions such as time, then referred to as *spatio-temporal data*. The term geo-spatiality is used in reference to our earth, thus the 'geo' in geo-spatial, but the underlying concepts are built upon mathematics which in theory can be applied to any physical shape in the universe.

Geo-spatial knowledge can be extremely powerful. It enables governments and companies to analyze resources, calculate consumer patterns, predict events and better understand the connection between past, present and future.

A Geographic Information System (GIS) stores, captures, processes, analyzes and displays geo-spatial data and makes it meaningful to humans. According to this definition a paper map with points marked out for mathematical analysis could be considered a GIS but more commonly the systems come in the form of software.

The evolution of GIS software has been marked by heterogeneity [6]. The first legacy systems were developed internally to suit particular needs within an organization. Limited capabilities resulted in systems tightly coupled with local file systems and data structures, and sharing data with other systems became next to impossible. The general technological advancements of the last few decades resulted in new possibilities for GIS developers. A wide number of new standards and protocols equated to an equally wide number of GIS software. Some major achievements with notable influence were the desktop PC and its increased usage, the growth of the Internet, mobile and wireless

technology and lately distributed services with data warehousing. Advancements in the field of data harvesting contributed to even further granularity with larger, more complex datasets being accrued.

The linear development of GISs can be described as convexing. Today there is a general need for GIS software to become more service-oriented which has given rise to a number of attempts at standardization and converging technologies.

Working with a GIS requires knowledge of a few terms and concepts that will be discussed in the next sections of this chapter.

4.1.2. Spatial Reference System

A Spatial Reference System (SRS) is any set of methods used in referring to points in a geo-spatial context [5 152 pp]. The earth is not perfectly spherical - it has a multitude of different elevations and craters. Modelling the earth is best achieved with the *geoid*, a shape taking into account the gravitational forces. If the earth was in perfect equilibrium, the geoid would have an equipotential surface [5]. The *ellipsoid* is a geometric shape with 3 radii - 2 equatorial and 1 polar. Varying the diameter of each will result in different shapes of the ellipsoid, but it will always have a perfectly smooth surface. Illustration 4 juxtaposes the two shapes [7][8].

Using the geoid as a depiction of the earth to present data through a SRS to a human user is too complicated due to all the computational work involved. No ordinary computer would have the capacity to handle such heavy amounts of data. Using an ellipsoid is easier but also very inaccurate if the same ellipsoid is applied to all of earth. The variations in elevation and depression are massive and will cause local deviations (known as *undulations*). No one ellipsoid could possibly capture the features of the entire world, at least given today's hardware and computational restrictions. This has given rise to a large collection of local reference systems. Each system uses its own ellipsoid that best reflects the local geographic circumstance. These are the many different Spatial Reference Systems. The science of conceptualizing the earth as geoids and ellipsoids is known as *geodetics*.

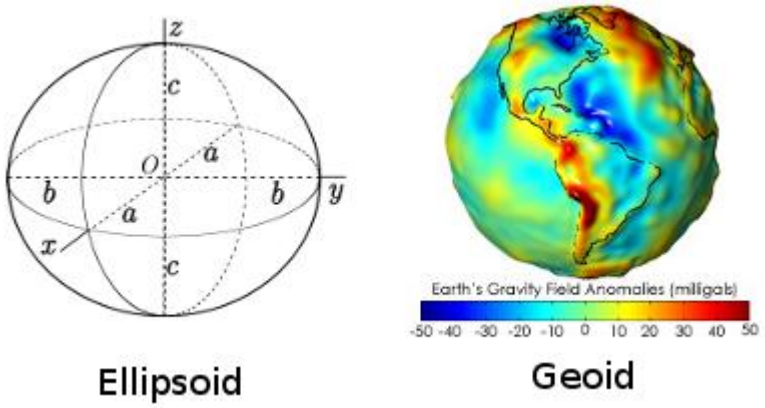

Illustration 4: *Ellipsoid and geoid*

4.1.3. Coordinate Reference System

To become useful, a SRS must somehow be measured and points on the surface must be referenced. The task is achieved by anchoring the ellipsoid relative to some point, known as *datum*. This point can be any arbitrary spot and is often chosen due to local proximity. Points can then be referred to as an offset to the datum. Each variation of point referral is known as a Coordinate Reference System (CRS) or sometimes Coordinate System (CS). A coordinate is "one of a sequence of n numbers designating the position of a point in an n-dimensional space" [10]. Inter-system translation is possible if the involved ellipsoids and data are known. [5 p 158]

4.1.4. Projection

Projection is the representation of a SRS in a flat, Cartesian plane. Most commonly this is needed when visualizing the earth on a map. The difficulty lies in the fact that the earth is round and can't be laid out as a two-dimensional map without some distortion occurring. The projection dictates exactly how the conversion is performed. One common example among the many existing methods is the Mercator projection in which an imagined cylinder wraps the earth, touches the equator, imprints the surface and then is rolled out flat. The method suffers from distortion that is larger towards the poles and lesser towards the equator. On the other hand it offers a consistent coordinate system in which longitudes and latitudes are equally incremented in a grid-like manner, appropriate for human interpretation. [5 158 pp]

4.1.5. The EPSG Geodetic Dataset

The International Association for Oil and Gas Producers (OGP) is an advocate organization for standards, cooperation and exchange within the oil and gas industry. The association is in charge of the European Petroleum Survey Group Geodetic Parameter Dataset, a structured repository of data used in referencing coordinates. Even though the OGP domain includes only oil and gas, the EPSG systems have been adopted

in other interest areas involving coordinates. The different EPSG definitions are different Coordinate Reference Systems [9].

Establishing a EPSG dataset involves a series of considerations and decisions regarding ellipsoids, projections and datum. The dataset must be optimized for the concerned region to avoid undulations. The most well-used system is EPSG:4326, also known as World Geodetic System 1984 (WGS 84) [5 158 pp]. There are many others, for instance EPSG:3006 which covers Sweden and places origo on the Swedish Mid-East Coast. The CRS of EPSG:3006 is optimized for referencing the Swedish domain while it is virtually useless in referring to any other area. Illustration 5 shows the EPSG:3006 projection area. Geodetic systems are referenced by their Spatial Reference Identifier (SRID), e.g. 3006 for EPSG:3006.

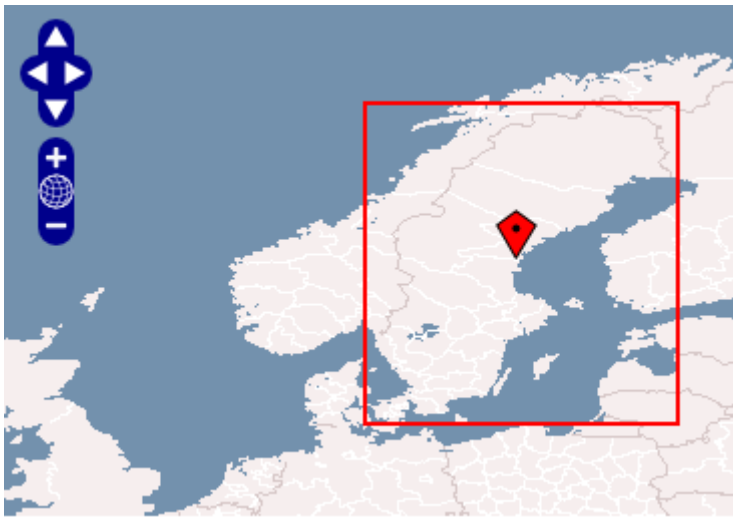

Illustration 5: *EPSG:3006*

4.2. Data Representation

Visualizing a geo-spatial context can be achieved in different ways out of which raster data and vector data are the most common. An understanding of these techniques is essential when working with a Web-enabled GIS which will undoubtedly involve visual representation.

4.2.1. Raster Data

A raster is a two-dimensional grid of pixels (also known as cells). Each pixel contains numeric values that can be rendered as a color. Every pixel represents the same distance in the map according to the given projection [5 371 pp]. A raster image will, when zoomed in, become 'pixelated' which is one of the limitations of the technique. The image can never show more fine-grained detail than the initial configuration

offers. Such an action can only be accomplished via again querying the server and receiving another raster image for a smaller area. However, no matter the zoom level, a raster image will always suffer from certain deviation from reality. Raster data is thus not appropriate when accuracy is a concern. Another downside of raster data is a large demand for storage space and resources demanded for rendering.

On the positive side, raster data is well-fitted for other kinds of analysis involving large volumes of data. Machine-produced samplings will often be in the format of matrices that are easily turned into rasters; aerial snapshots, information-gathering sensors and so on. Analysis of a raster grid can result in a set of vector data, a concept discussed in the next section.

Some common formats for visually outputting raster data are JPEG, GIF, PNG and TIFF.

4.2.2. Vector Data

Vector data is made up of points, polygons, lines and curves represented as coordinates in a CRS. For example, a straight line can be represented with 3 values: 1 value indicating the type of shape (line) plus 2 coordinates - the start and the end of the line. If the projection is known that is all the information needed for visually rendering the shape. Vector graphics have exact precision regardless of the zoom level. In other words, a zoomed in vector graphic will not become pixelated like a raster image because it does not consist of pixels. Instead the underlying data will be rendered to a more fine-grained level of detail. [5 3 pp]

Points in vector format can be stored together with additional, non-geospatial data known as *discrete* or *attribute* data. For instance, a polygon representing a wildlife habitat could be linked with information about animal populations, responsible authorities and so on. The data can be partitioned into a *layer* which will be treated as a unit during fetching and rendering, avoiding the laborious task of retrieving each attribute in a separate call. Associating this data with a cluster of pixels in a raster image would not be possible in the same way. Furthermore, spatial analysis is easier when using points of reference rather than a grid of millions of random pixels. This makes vectors more appropriate for calculating and measuring geo-spatial data in situations where accuracy is important. The accuracy on the other hand can result in heavy use of processor power for very complex calculations.

Visually presenting vector data is commonly performed with Scalable Vector Graphics (SVG) [11]. The method presents graphics as XML, placing a condition on the recipient (often a browser) to support the SVG syntax. Vector graphics consist of paths between points.

Illustration 6 demonstrates the difference between raster and vector images, how raster images suffer from loss of precision and the way vector data can be associated with attributes. [12]

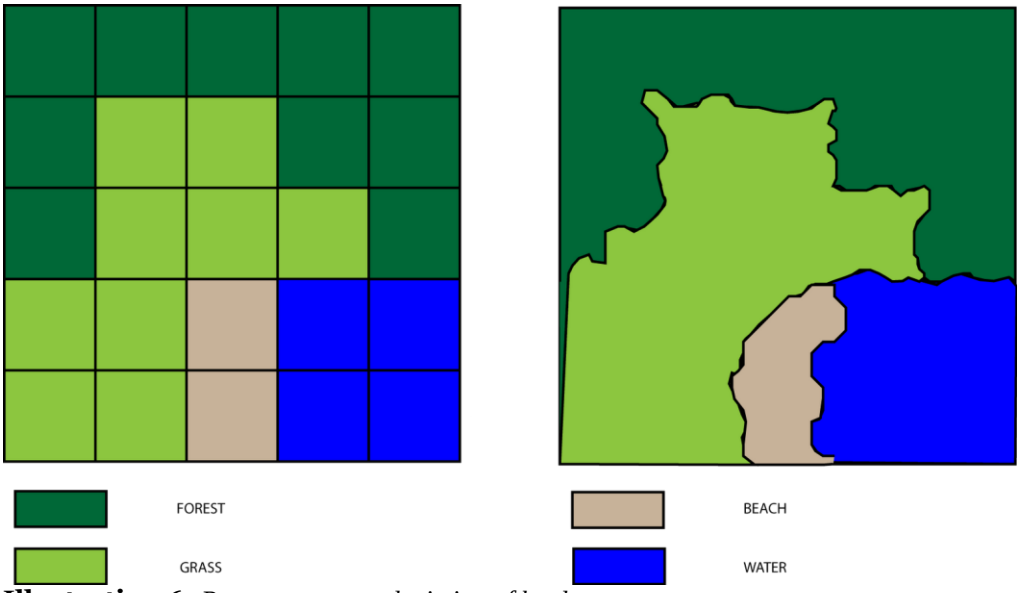

Illustration 6: *Raster vs. vector depiction of land area*

4.3. Open Geo-Spatial Consortium and Their Standards

The Open Geo-Spatial Consortium (OGC) is an international collaborating conglomeration of organizations, companies, government agencies and academic institutions with the aim to develop and establish geo-spatial standards [13]. The ambition is to benefit all users of geo-spatial tools through promoting interoperability, advancing development and providing free standards. The OGC define their goals as:

> Goal 1 - Provide free and openly available standards to the market, tangible value to Members, and measurable benefits to users.
>
> Goal 2 - Lead worldwide in the creation and establishment of standards that allow geospatial content and services to be seamlessly integrated into business and civic processes, the spatial web and enterprise computing.
>
> Goal 3 - Facilitate the adoption of open, spatially enabled reference architectures in enterprise environments worldwide.
>
> Goal 4 - Advance standards in support of the formation of new and innovative markets and applications for geo-spatial technologies.
>
> Goal 5 - Accelerate market assimilation of interoperability research through collaborative consortium processes.

The initiative was founded in 1994 after a series of preceding similar formations; namely GRASS (Geographic Resources Analysis Support System) which was developed by the U.S. Army, and OpenGIS which was the same concept as OGC under another name. The intricacies of geo-spatial tools had grown as technology advanced through the mid-80s, however most solutions were only applicable to certain systems or situations. Common standards for gathering, processing and representing geo-spatial data were missing. Furthermore, most tools were proprietary and impossible for all users to access. OGC (and its predecessors) were reactions to these shortcomings.

The heart of the OGC is the definition of standards [14]. A standard provides rules and guidelines for maximum interoperability. In essence, two different applications obeying the same standard should (with minor modification) readily be interoperable in a plug-and-play manner. The OGC standards define the encoding of technologies for use within Web, IT, enterprise and location-based services. This study will focus on the Web-based standards but it is important to note that the OGC domain encompasses much more.

The OGC is also concerned with compliance testing [15]. Software is deemed *compliant* if it implements OGC standards and has passed the official OGC Compliance Testing Program. The software is *implementing* if it uses OGC standards but has not gone through the Compliance Testing Program. The OGC plans to introduce a Interoperability Testing Program, evaluating if one compliant product is in fact inter-operable with others.

The Web OGC standards all strive to operate according to the *Web Services* paradigm in which modularized services are published, located and utilized by servers and clients [16]. GIS services can be categorized into three groups: data services offering access to geo-spatial data, processing services for modifying data and catalog services for publishing and locating data [17]. Data access standards are the most well-developed while the two remaining components require more attention. This statement is true for Web Services in general [18].

4.3.1. OGC Web Service Common

The OGC Web Service Common (OWS) [19] was established to unify all OGC web standards. The specification determines a set of operations, data structures and parameters that all OGC web standards should implement. This brings a number of advantages to the process: reducing the work required when writing and reading standards, shortening the OGC standard length, avoiding a 'reinvention of the wheel', increasing interoperability by discouraging non-essential differences and finally reducing the work required when programming.

One example defined by the OWS is *getCapabilities* operation in which a service advertises its supported operations. The mandatory parameters give a good indication of the purpose of the operation and how the service-orientation manifests itself. They are summarized in Table 1.

service	ServiceType	ExtendedCapabilities	meta data
ProviderName	Get	ServiceIdentification	Contents
request	ServiceTypeVersion	name	HTTP
ProviderSite	Post	ServiceProvider	AccessConstraints
AcceptVersions	Profile	DCP	AcceptLanguages
ServiceContact	URL	Operationsmetadata	Operation
Sections	Title	DatasetSummary	updateSequence
Constraint	Keywords	Abstract	Parameter
AcceptFormats	Fees	OtherSource	

Table 1: *The getCapabilities parameters*

The OWS standard has a number of future goals including increased collaboration with other standards organizations, improving service meta data document organization to better match WSDL and UDDI[1], improved meta data contents, more operations such as the Transaction operation, better integration with human language and handling of chained services.

The OWS specification leaves "chained services" undefined but the term is normally interpreted as many services pipelined together to form one GIS [17][6]. For example, a client requesting a raster image of certain data in a given projection will spark a sequence of service providers to contribute to the final result. An addressing service will locate resources, another will handle re-projection, a third instance will produce the raster data and so on. The sequence can be *client-coordinated*, *mediated* or *static*. Ideally it would be *dynamic* to include all of the mentioned approaches on-the-fly, but the OGC has yet to come close to such a solution. Some matters needing to be resolved are those of transparency, error-handling and and tracing.

4.3.2. Web Map Service

The OpenGIS Web Map Service (WMS) [20] defines methods to produce a visual map from a set of geo-spatial data, requested and returned over the HTTP protocol. The output formats can be pictorial (PNG, JPEG or GIF) or vector-based (graphical) (SVG, WebCGM). The service does not return data - the main output is a map. The map is associated with XML meta data which can contain additional information. A WMS can announce its services via responding to the getCapabilities request from clients.

1 Web Service meta data methods for discovering and contacting services. See W3C Recommendation *Web Services Description Language (WSDL) 1.1* and *UDDI v2 [OASIS 200302].*

The generation of the map requires at least 2 different CRSs - one for the bounding box of the underlying data and one for the outputted map. The service must take care to translate between these 2 systems. No particular Coordinate Reference System is imposed on the developer, however EPSG:4326 or CRS:84 are often suggested as supported projections due to their wide-spread use.

The standard defines layers as a basic unit of geographic information that may be requested as a map from a server. Layers can be stacked on each other and rendered in the same output map. The service can also produce limited information about features. Elevation and spatio-temporal data are available as meta data if the client so requests. The format of WMS feature data is not standardized since the main purpose of WMS is image, not feature, rendering.

The layer meta data - including name, coordinate reference system etc. - should be stored in accordance with ISO 19115 Geographic Information -- meta data or FGDC-STD-001-1998 Content Standard for Digital Geo-Spatial meta data.

4.3.3. Web Map Tile Service

A technique known as tile caching can speed up performance of WMS content. The corresponding OGC standard is called the Web Map Tile Service Implementation Standard (WMTS) [21]. WMTS is a complement to WMS and operates in the same way of dividing and delivering a raster map in smaller chunks. The provider holds the mandate to decide how data is partitioned, as opposed to WMS where the client can request data in any way and let the service provider perform *ad hoc* rendering each time. The WMTS server will pre-render images and use image caching mechanisms which makes it faster than a WMS service, but also less flexible. It can furthermore offer raster data in Web applications that may otherwise be restrictive with server-side rendering, perhaps only allowing static data content in a RESTful manner.

4.3.4. Simple Feature Access

The OpenGIS Simple Features Access (SFA) [22] specification schema supports storage, retrieval, query and update of feature collections via the SQL Call-Level Interface (SQL/CLI) [23]. Features are associated with both spatial and non-spatial data. The spatial data is represented as points, lines and polygons in 2 or 3 dimensions. A feature collection is stored in a table with columns dedicated to geometry where each row represents a feature. The geometry column links to a column in a 'geometry table' containing all the necessary data, such as vector type (point, line, polygon, curve etc.), projection and so on. Procedures and functions enable geo-spatial operations such as transformations, re-projections and geometric-based querying.

4.3.5. Web Feature Service

The OpenGIS Web Feature Service 2.0 Interface Standard (WFS) [24][25] is a set of methods for communicating information about features. A feature is defined as "an abstraction of real world phenomena" [26], which encompasses anything needing to be represented in a GIS. It is important to note that a feature is not synonymous to a vector entity. A vector is strictly limited to shapes (lines, points etc.) while feature entities can be classified as shapes, but also arbitrarily chosen categories such as "bridge", "wildlife habitat" and so on. Features are represented in the Geography Markup Language (GML) [27], an XML grammar developed by the OGC for representing geographic data. Alternatively the response comes in the format of key-value pairs. This makes the protocol non-RESTful and brings the possibility to use well-known Web architecture standards such as SOAP[2].

A basic WFS offers querying and retrieval of feature data at the property level. This lowers the amount of transferred data since only the specified and relevant information will be concerned. Furthermore it abstracts the underlying data storage method away from the client, requiring the client to understand only WFS. A transactional WFS (WFS-T) extends the functionality to offer modification of features directly against the data store.

Treating features separately brings the possibility to spatially analyze them in relation to each other, such as measuring, calculating and discovering patterns. This is not possible, or at least very difficult, with the WMS output consisting of a single image.

4.3.6. Well-Known Text and Well-Known Binary

Well-Known Text (WKT) [28] is a text-based markup language for representing geometric objects. Well-Known Binary (WBT) is the binary equivalent. Objects supported are points, lines, polygons and polyhedrons. For example, a point and a linestring are expressed as

```
POINT(80 22)
LINESTRING(80 22, 50 70, 100 30)
```

The standard also defines markup for transforming objects between different spatial reference systems. The format is often used and referenced by other OGC standards.

2 Protocol for communication and data exchange between Web Services and clients, see *W3C Recommendation:SOAP Version 1.2 Part 1, Messaging Framework*.

4.4. The DISMAR Project and Distributed Services

4.4.1. Distributed Service Architectures

An emerging trend in software engineering is the one of *service-oriented* and *distributed architectures*, a concept fully described in [3]. In brief, a distributed architecture consists of several geographically and logically separated service providers. The different service providers will perform operations for the client and send back data. How and where data is communicated depends on the nature of the service. The paradigm has been introduced as a response to the increasing size of systems, evolving in parallel with the decreasing size of client devices.

The paradigm is of relevance in this study, because the underlying data has been shown to come from many different sources. It is almost impossible to think in terms of a monolithic database or service provider - instead a final solution will have to think distributed services. The DISMAR project explained next is a similar real-world example of a distributed services Web mapping application.

4.4.2. DISMAR

The Data Integration System for Marine Pollution and Water Quality (DISMAR) project set out to accrue and integrate multi-sourced, heterogeneous data distributed among satellite sensors, *in situ* instruments and stored data. The overall aim of the project was to aid the governmental tracking of pollution [29]. The goals of the DISMAR project were 1. Harmonization of meta data and documentation of existing datasets, 2. Development of common methods to access data, 3. Development of common models to represent objects and 4. Integrating the data with other European similar efforts. The study resulted in a software model named the DISMAR PROtotype (DISPRO).

After careful evaluation of techniques it was determined that the rapid, extensive development and wide use of OGC standards made them the best option for the final DISPRO GIS software. WMS was chosen due to the property of most of the data (raster-like grids gathered by sensors). MapServer was chosen as the core server as the project desired an open-source solution well-adapted for WMS. MapServer was deemed to have enough functionality to fulfill the technical needs.

Data about pollution existed in-house with many different actors; federal commissions, research institutes and so on. They all had their internal systems suffering from a lack of interoperability with one another. DISPRO approached this limitation via *Product Generators, DISPRO Nodes,* the *DISPRO Portal* and *DISPRO Clients*. Illustration 7 summarizes the flow of data through the architecture.

Information Gathering The first step is to harvest raw data which can be done in many different ways. In the case of pollution, measurement is often performed via sensors, satellites, computations on aggregated data, ground stations etc. These tools produce information output streams and are maintained by the separate data providers involved in the collaboration.

Product Generator The raw data is processed and stored in a format native to the internal legacy system. Up until this point the internal system need not change anything in order to implement DISPRO. In other words DISPRO can be implemented on top of an already existing system, allowing the legacy storage methods to remain intact.

DISPRO Nodes Each data provider implements a WMS server (MapServer) capable of communicating and fetching data from the Product Generator to produce WMS maps according to the OGC specification. This requires each provider to implement some sort of interface between the WMS service and their product generator.

DISPRO Portal This is a central server also known as the *aggregator*. The most prominent feature is the meta data catalog for indexing all the separate DISPRO nodes maintained by the data providers. It will poll the Nodes at regular intervals via the *getCapabilites* request and update the meta data catalog accordingly. Nodes make themselves known to the Portal by placing their URL in the aggregator. The catalog itself is an XML database, doing away with the overhead involved in relational table querying and conversions. The XML entries are easily translated to and from WMS mapfiles.

DISPRO Client The user interface consists of many components functioning together. The Client queries the Portal for a desired service, receives an XML response with meta data from the catalog and initiates contact with the corresponding DISPRO Node. The output WMS map is presented in a thin interface built on *de facto* Web standards.

The Profiler allows the user to set and store preferences such as projection and layer groups (oil, algae etc.) which will automatically fetch the desired information upon each application start. The user profile mechanism alleviates the user from choosing the same settings over and over again, and also avoids unnecessary fetching of meta data since the information is persistently stored with the client. The *Web Gis* visually presents the information and has basic capabilities such as zooming, panning and clicking on the map, choosing and removing layers, searching for keywords and more.

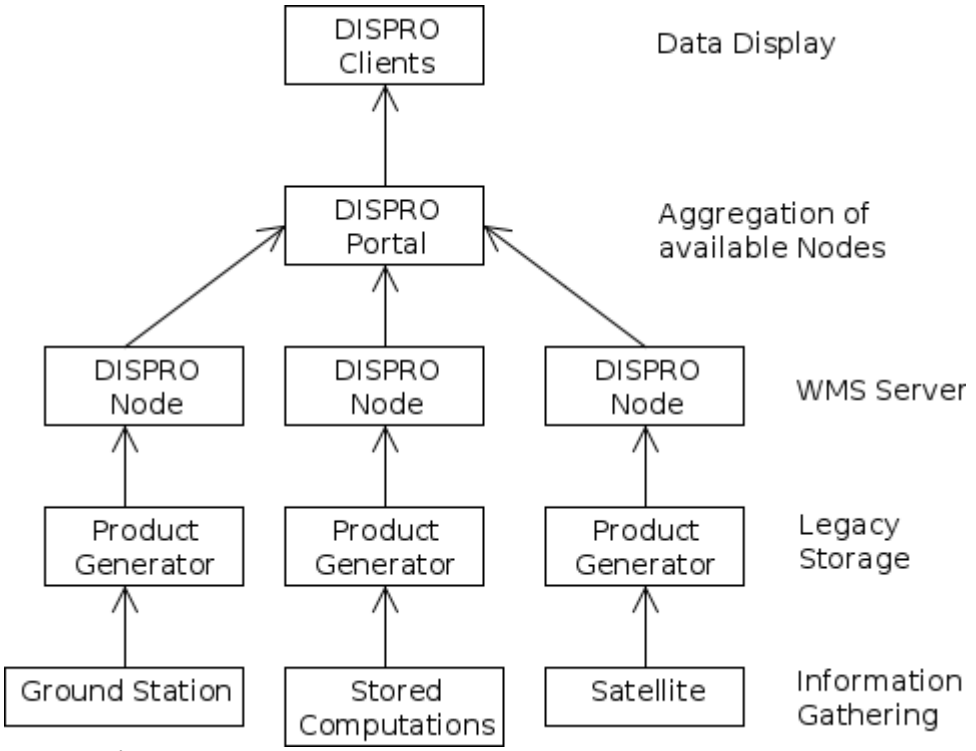

Illustration 7: *The DISPRO architecture*

The system was evaluated via two test scenarios which both yielded positive results. The design held up to standards and was highly graded from a user experience point of view. The chosen technologies were deemed useful and appropriate for the situation, with special value placed in the open-source, license-free attributes.

The heart of the final application is the meta data catalog. In spite of keen consideration going into its structure and accumulation, the writers see potential issues in the harmonization process. The system requires data providers to keep their own meta data and MapServer mapfiles up-to-date and valid which may not always be performed as desired. The writers admit the process is highly circumstantial and propose repeatedly reminding providers of the importance of meta data processing. They write: *"During the DISMAR project, the providers made the first step towards better meta data accuracy through agreeing on a standardized meta data structure and representation. "* [29 p 150]

5. Implementation

This chapter demonstrates the high-level implementation of the test model. The chapter is structured according to the layers; Data Source, Data Access and User Interface.

The concept of software layers has been mentioned as a key component. A software layer is a stand-alone part of an architecture that acts independently from others, allowing tools and technologies to be switched out with greater ease and increasing code maintainability. The test model is composed of three layers: Data Source (PostGIS for vector data and .tiff files for raster data), Data Access (MapServer and TinyOWS) and User Interface (OpenLayers). Each layer is made up of several different components, servers and clients scattered over a geographical and logical distance. Different providers of data provide their services unknowingly and independently of others. This structure aims to emulate a distributed services architecture.

The test model emulates a distributed services architecture by implementing two different server machines, unaware of each other. There is currently no central repository to look up or find services, and neither does a meta data catalog exist. Programmers of User Interfaces must have knowledge about addresses and methods to directly reach the Data Access servers, and Data Access servers must have knowledge about the binding to Data Sources. Table 2 summarizes tools and hosts used and Illustration 8 depicts the structure. A note - this is the architecture used for the test cases, however it would be possible to add or remove Data Access and Data Source nodes.

The test model can be classified as a minor GIS, mimicking the properties of a large-scale GIS.

Layer	Tools	Host
Data Source	PostGIS Shape files	Mid Sweden Uni. server Local machine
Data Access	MapServer TinyOWS Ubuntu Server	Mid Sweden Uni. server Local machine
User Interface	OpenLayers ExtJS GeoExt jQuery HTML, CSS, PHP	Mid Sweden Uni. server Local machine

Table 2: *Tools and hosts*

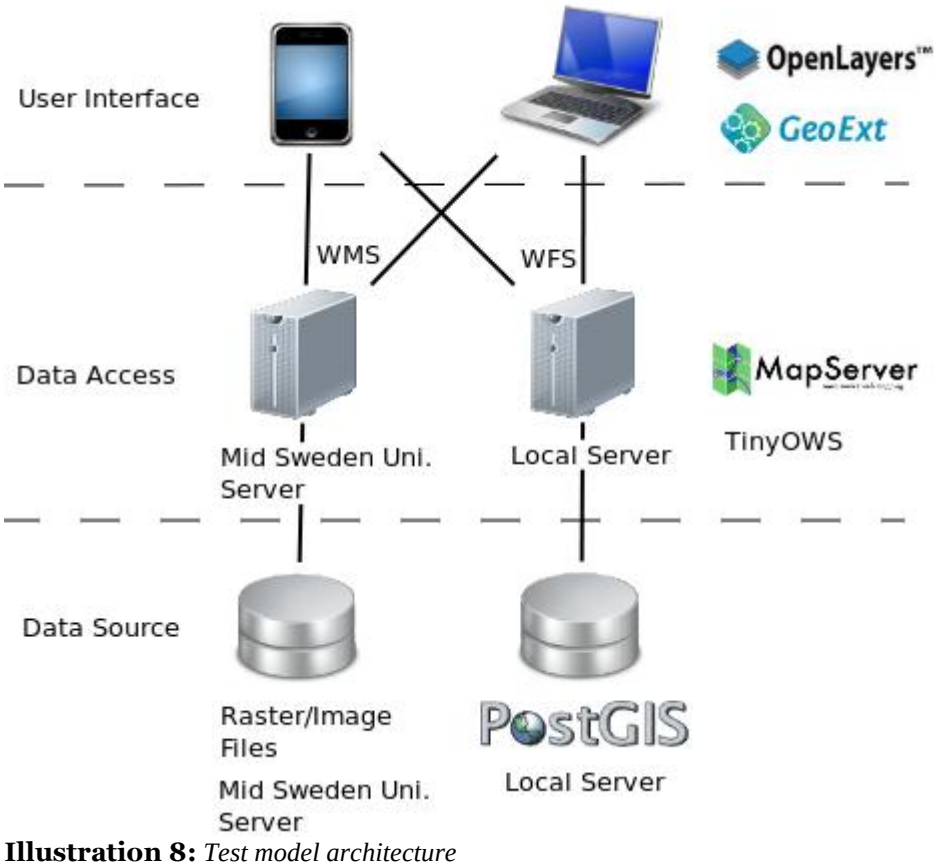

Illustration 8: *Test model architecture*

5.1. Data Source

This section will describe the storage of data, divided into storage of a base layer (raster data) and features (vector data). Test case database structure and will also be discussed.

5.1.1. Base Layer Storage

The base layer depicts the concerned geographic region and acts as a background canvas that places features in a human-understandable context. Data is delivered as a raster in the form of .tif files. The map is partitioned into equal-sized chunks, each containing a bit of the map at a particular zoom level. The smallest zoom level offers a view down to the building level, while the largest shows all of Jämtland and Nord-Sør Trøndelag. Each .tif file is associated with a set of files:

.dim - Contains XML-formatted meta data.

.htm - Presents meta data about the fraction on a human-readable format.

.ini - Configuration file specifying the area covered in coordinates in the related projection.

.tab - Contains meta data.

.tfw - Used to georeference the .tif file in terms of X and Y orientation.

.tif - The raster image.

The total size of all files for all zoom levels (there are 16 zoom levels) is very large - somewhere around 70 GB. It would be impossible to send all images at once, so instead only specific portions of the data are sent upon request. Clients can cache delivered raster images in order to speed up performance.

The dataset is proprietary and provided by the project coordinators in collaboration with the federal geographic service departments. It is physically stored at the Mid Sweden University server. The map may not be used outside the realms of the project and any future use will involve a fee. Many providers of geographic data charge for their product via various license agreements. Unless deals can be settled between the Swedish and Norwegian emergency and geographic departments, the final solution will have to take this expense into the budgeting calculations.

5.1.2. Feature Storage

Features are kept in PostGIS, an open-source plug-in for the PostgreSQL database [5]. It is implemented on top of an already existing database as a set of tables, views and functions that add geo-spatial support.

PostGIS is compliant with the OGC SFA standard [22] and handles spatial data in the WKT/WKB format described in section 4.3. Tables are via functions transformed to associate with vector data in a given projection. The tuples thus become features, in accordance with SFA.

Since features are associated with a shape in a plane, queries such as 'Fetch all fire trucks within a range of 50 km from point X,Y' or 'Does area X cover area Y?' can be performed. PostGIS does not compromise original SQL functionality, so non-spatial tables, view and procedures can be used concurrently in the same database. The format is very well suited for delivering vector and associated non-spatial data to an application.

An SQL code snippet creating a spatially enabled table shows the use of the `AddGeometryColumn()` function included in PostGIS, and also sums up how spatiality is treated. The function will 1) create a new geometry column in the `pfa_facility` table, 2) enforce constraints that ensure only points are stored, 3) enforce constraints to ensure only 2-dimensional data is stored, 4) enforce constraints to ensure only SRID 3006 is used and 5) add an entry to the `geometry_columns` table generated by PostGIS for storing meta data about all geometry columns in the database. PostGIS uses the meta data table when performing operations and locating features.

```
CREATE TABLE pfa_facility(id serial not null primary key);
SELECT AddGeometryColumn('pfa_common', 'geom', 3006, 'POINT',
2);
```

PostGIS offers hundreds of functions for performing spatial operations on data. Most PostGIS functions start with *ST_* which stands for Spatiality Time. PostGIS was initially aimed at providing temporal as well as spatial capabilities, however the temporal aspect was not as successful and has received less attention from developers. A developer can define functions and procedures as with any SQL database. Some examples of functions:

`ST_Touches(geometry, geometry)`
Returns true if the two geometries spatially touch each other.

`ST_Disjoint(geometry, geometry)`
Returns true if the two geometries are disjoint, i.e. do not touch.

`ST_GeomFromText(text, [srid])`
Returns a geometry from a WKT string.

`ST_isvalid(geometry)`
Returns true if the geometry is valid. A geometry can be invalid if it does not conform to criteria specific to the geometry, for instance a polygon may not intersect itself.

5.1.3. Test Case Database Structure

To be deemed as sufficient in a test case, the database must 1) Be designed in a structure similar to the final design, 2) contain data and be

structured in accordance with the actual structure of the departments, 3) enable storage and retrieval of geo-spatial objects and 4) enable operations on geo-spatial data such as the ST_ functions exemplified in the previous section.

The test database was modelled in an agile process where the users of the final system - representatives from the respective departments - specified what data they find relevant. They produced documents of core entities such as 'fire stations', 'ambulances' and listed a few attributes important to note about each entity such as 'EKG', 'RAKEL number' etc. Each department worked separately from one another. The departments' data specifications are included in Appendix B. The information was compiled and a set of common attributes were identified.

It was determined that data could be generalized into three main categories: Equipment, Facilities and Vehicles. A set of *general* tables were created, containing information pertaining to all equipment, facilities and vehicles. Since the PostgreSQL database allows inheritance, children entities of these general entities were created to represent specific data. Illustration 9 shows the UML diagram produced for the project. Some notes about the diagram:

- Tables starting with *p_* pertain to the police, *a* to the ambulance and *f* to the fire department.
- Tables starting with *pfa_* pertain to all departments.
- Green tables pertain to all departments and contain data applicable to several objects in the database.
- Blue tables pertain to the police, yellow to the ambulance and red to the fire department.
- Not all relationships and attributes are shown in the diagram to increase readability.
- All facilities inherit from `pfa_facility` and all vehicles inherit from `pfa_vehicle`.
- Equipment is kept in one single table.
- Vehicles and facilities are the only objects associated with geometries, as specified by the `geom` attribute in `pfa_facility` and `pfa_vehicle`.

The tables included in the test model were:

- `pfa_nation`

- `pfa_facility`
- `f_station`

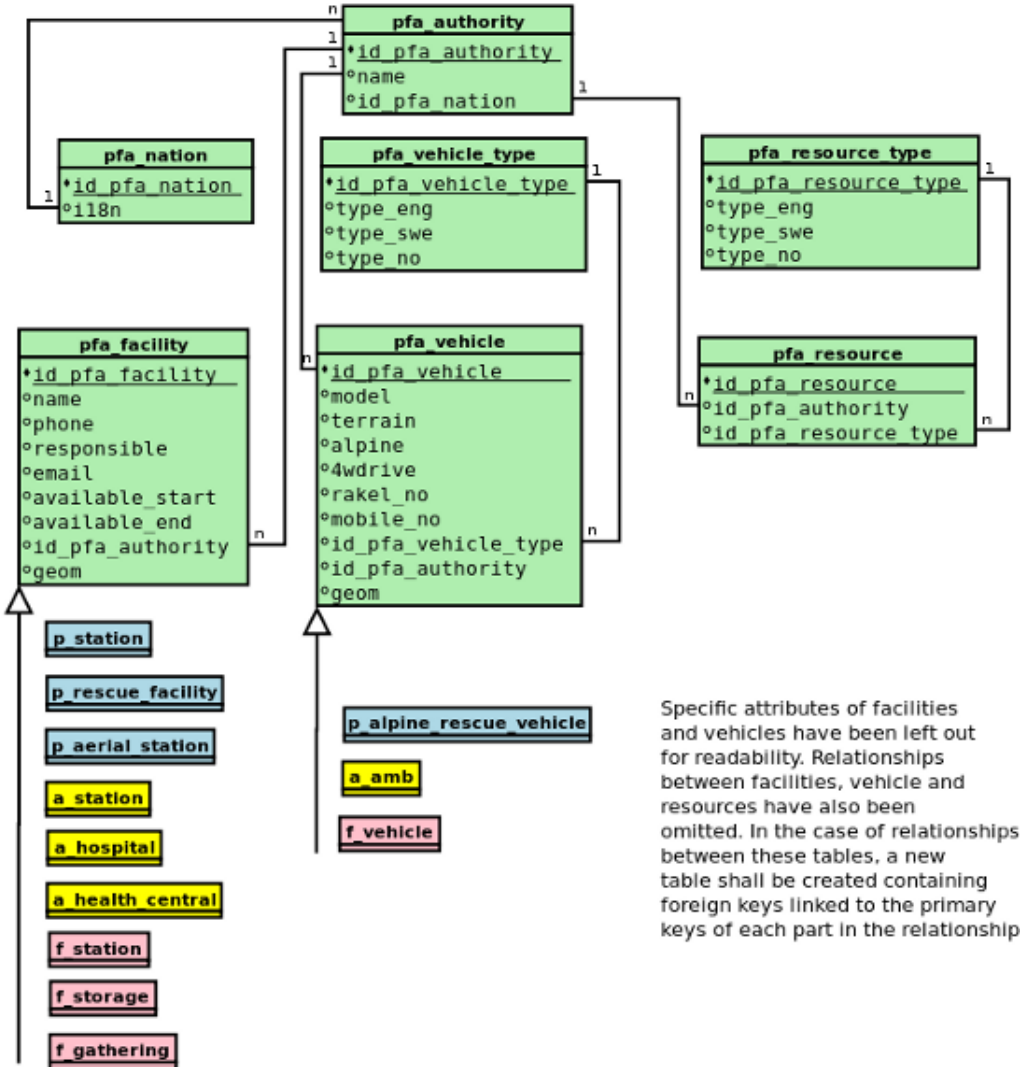

Illustration 9: *Database UML diagram*

The database design does not fully meet the requirements of the GGI2 project goals. Time constraints enforced a scaling down of the test database to only contained a limited set of core entities relating to vehicles, equipment and facilities. Information about estates, buildings and populations has been left out. The departments agreed that this order of priority was the best, reasoning that equipment, vehicles and facilities are more vital components in a rescue effort. From a test case point of view, the design can thus be deemed sufficient as it meets the requirements stated in the beginning of this section.

5.2. Data Access

The Data Access layer refers to the module used to access data contained in the Data Source layer. It shields the User Interface layer from the specific implementation details of the Data Source. Subsequently, client implementations need only know how to perform standard HTTP requests and accept a response on a given OGC standard format.

The Data Access layer is implemented as web servers accepting incoming requests, fetching data from the source and returning it to the client. Ubuntu Server 10.04 is used for tending the initial HTTP requests. A CGI script routes the request on to MapServer/TinyOWS.

Raster data is delivered as image files. Features are delivered as OGC GML or alternatively on the GeoJSON format [31]. A GeoJSON is a JSON object obeying standardized definitions that enable better handle geographic data. It requires no particular library since it is technically nothing more than a JSON object. Most geo-spatial tools can produce/consume GeoJSON objects.

The test model implements the open-source MapServer [32] and TinyOWS [33] web servers. TinyOWS was initially a stand-alone project but has lately been merged into MapServer, enabling both tools to be used in unison and tuned via the same configuration files. MapServer does not support the WFS-T (WFS Transactional) format and thus only delivers static data. TinyOWS delivers as well as writes data to the data store. The application uses TinyOWS for transactional requests, in other cases MapServer. Both servers run on the Mid Sweden University Server but could theoretically operate on different machines. The use of two different servers, as opposed to one, emulates the possibility of using several Data Access nodes in a distributed architecture.

A MapServer/TinyOWS configuration file specifies layers. Each layer can come from different data sources, produce vector or raster output and have different namespaces, styles and projections. In the case of WMS, the server is responsible for rendering a map based on the client's request. The layer mechanism allows heterogeneous data sources to be compiled in one configuration file. The client is not aware of the multitude of various sources since the server acts as a single point of entry. If compared to the DISPRO model discussed in section 4.4, the Data Access layer would correspond to the DISPRO Node.

The feature/vector layers used for the test case correspond to the tables `f_station` and `pfa_facility` in the database. Performing geo-spatial calculations through functions is possible by specifying SQL statements directly in the server configuration files. In other words, if the client requests all fire trucks within a given area, there is a corresponding layer

dedicated to that exact request. The result is a regular set of features transferred via WFS.

5.3. User Interface

The User Interface layer is a web application designed to run in a regular web browser. It is implemented via standard web techniques: HTML, CSS, Javascript and PHP. The raster base map is a WMS layer and the features are all partitioned into vector layers fetched via WFS. Editable features are transferred via WFS-T. The appearance is illustrated in Illustration 10. The interface makes heavy use of a set of open-source Javascript frameworks:

OpenLayers This has become the *de facto* technology for web mapping applications. OpenLayers offers functionality for handling maps, layers, features, rasters and geo-spatial operations in the browser. OpenLayers is used for handling the map of the application, retrieving the background image, setting up connections with WFS servers, formatting the servers' response and adding the features to the map. OpenLayers can handle data on many OGC standard formats, including GeoJSON, GML, KML, vector, raster, WMS, WFS, TMS and WFS-T. OpenLayers integrates well with mobile devices.

ExtJS A browser layout framework offering a set of tools performing common, otherwise tedious, functions such as aligning elements in the window, adding buttons, toolbars, handling events, creating forms and much more. ExtJS is meant to alleviate the programmer in creating visually harmonizing interfaces with properly handled response mechanisms.

GeoExt ExtJS and OpenLayers are not seemlessly integrated with one another. For instance, an OpenLayers map action such as drawing a polygon may need to be included in an ExtJS toolbar. GeoExt is the mediator between the two frameworks.

jQuery A well-used Javascript library for performing a plethora of common operations. The library is used in the application for tasks such as producing AJAX requests, minimizing the code and organizing the application.

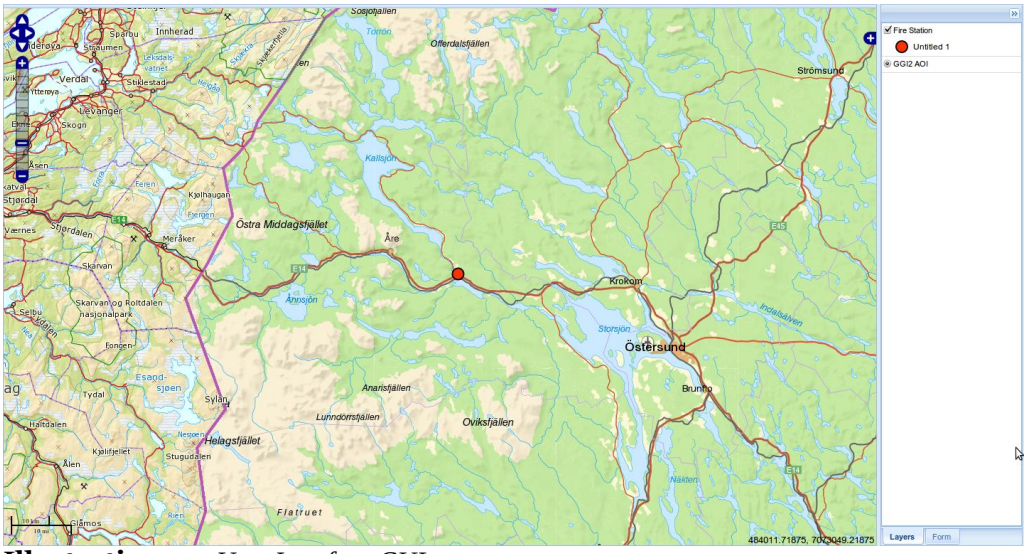

Illustration 10: *User Interface GUI*

5.3.1. Web Application Structure

The Web application structure (OpenLayers, ExtJS, GeoExt and jQuery libraries not included):

```
`-- ggi2
  |-- config
  |   `-- ggi2.json
  |-- config.php
  |-- css
  |   `-- app.css
  |-- index.html
  `-- js
     |-- App.js
     |-- init.js
     |-- Map.js
     `-- Model.js
```

The Javascript files in the js folder are responsible for most of the content. Capitalized files contain singleton objects, ie. each session has one App, Map and Model object respectively. The App object is responsible for rendering layout and appearance and providing containers for data display, such as the eastern panel with a map legend and form. The Map object takes care of any map-related operations such as creating a map panel, fetching layers, displaying feature popups and so on. The Model object provides the application with configuration data fetched from a configuration file. The purpose of the Model is to shield user interface logic from data retrieval logic. The Map object is not aware of the origin of configuration data, leaving it subject to change without any major impact on other objects than the Model. This approach loosely mimics the Model-View-Controller pattern in which an

underlying data model provides the view (interface objects/classes) with data. Future expansion of the application will most likely involve a large degree of configuration parameters and further communication with different servers - this can all be handled by the Model object. The Javascript source code is included in Appendix C.

Illustration 10 depicts how the Model acts as an intermediary between the Data Access layer and the User Interface. The App is not directly in touch with the Model - its only responsibility is content and layout

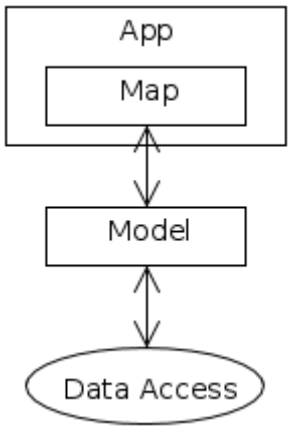

Illustration 11:
Interface structure

rendering. The Map exists in a map panel within the App.

5.3.2. Configuration file

All information required in the application is maintained in the configuration file *ggi2.json*. It contains data about the different available hosts (Data Access nodes), what layers are available and what hosts offer what layers. The client calls a PHP file through a regular AJAX request and the PHP file echoes the configuration information back to the application. The information is then used to create a map and fetch data. It also dictates what projection, units and other geo-spatial parameters to use application-wide. The contents of ggi2.json are included in Appendix C.

The ggi2.json file can be considered a meta data repository kept within the application. Ideally, this would be implemented as a meta data service node - a centralized server or repository producing information about hosts dynamically on request. There is a redundancy danger in keeping all such information in a configuration file within the application, since the data is subject to constant change. Time constraints forced this setup. However, since the information is retrieved through a PHP file, the application Javascript never needs to change even if the fetching of meta data does. If the future offers more

time to implement a meta data repository, the same PHP file can be modified but still produce the same information.

5.3.3. Testing TMS

As a test, the base WMS layer was swapped out with a Tile Mapping Service (TMS) layer hosted at the Mid Sweden University server. The swap went smoothly and only required re-programming of the base layer calling function to better suit the implementation of tiles. In a future implementation, this could be performed via a selection model where different base layer maps might be offered. The user could switch as desired between base layers and different calling functions would be called.

The rendering of tiles is initially much slower than WMS which momentarily gives the impression that the application is broken. However, after the rendering of a tile is performed, it will be cached and all subsequent calls will be much faster.

6. Results

The introductory chapter presented a number of goals. This chapter describes the objective results of the study, how the goals were met and possible contamination of test results. The first goal strived to create an environment for an empirical study to be conducted. The second goal performs the empirical study.

Goal *Enable testing of a few of the most common geo-spatial Web standards by using technologies for accessing and performing simple operations such as fetching data about a fire station, modifying it and writing it back to the store. This should be done through a user interface.*

The test model successfully manages to fetch features from a data source for display in a Web interface. A form enables editing of the feature's associated attributes and changes are written back to the data source. The application consists of many different technologies, nodes and components, all stitched together in a web of OGC standards. The standards used are WFS, WFS-T, WMS, SFA and TMS. All data does not exist in the same projection and re-projection has sometimes been necessary. Most of the frameworks used are developed enough to take care of re-projection automatically.

The interface offers a view of a background map and options to show vector layers in the map, displaying icons corresponding to resources. Icons can be clicked whereupon a dialog box appears with further information about the feature. The user can edit features and save the changes which will reflect in the Data Source. Illustration 12 shows the selection of a feature - a popup will hover over the feature and a form will appear in the right-side menu. The form allows users to modify the feature by filling out the form and hitting 'Save'.

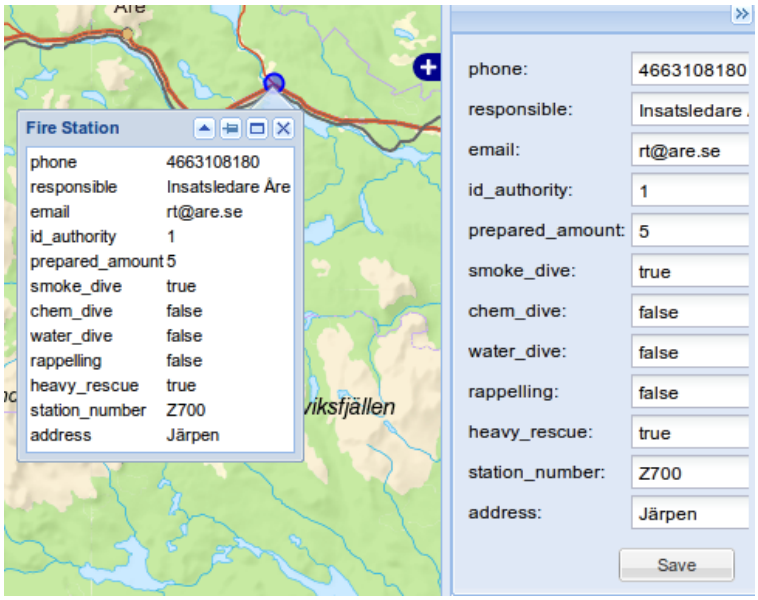

Illustration 12: *A feature has been clicked*

Goal *The test model must emulate a potential solution by implementing software layers. Each layer must be implemented through at least two different technologies to portray the scenario of a heterogeneous architecture. The modularity of layers and usability of Web standards shall be tested by swapping between technologies and estimating the degree of effort required to perform a swap. Each of the following software components (layers) must at least be involved: Data Source, Data Access, User Interface.*

The test GIS is constructed according to the thin thread model and contains all three stipulated layers, as depicted in Illustration 8 on page 30. A software layer can not be considered a layer until separation of concerns and true modularity has been proven - thus the demand for each layer to be implemented through at least two different technologies. This means that swapping out a tool in favor of another should not result in issues and re-structuring of the whole architecture, as long as the new tool communicates via the same OGC standard. Such functionality is desirable in a distributed services situation where nodes may change sporadically.

Modularity was tested by swapping out tools and frameworks to other alternatives. A small or non-existent effort required to swap a tool indicates a high level of separation, modularity and decoupling, where 'effort' shall be interpreted as 'human involvement needed to re-program parts of the application to fit the new tool'. Table 3 contains a summary of each layer and the alternative tool used. The architecture was tested in all possible constellations, shown in Table 4.

Layer	Original component	Alternative	OGC standard
User Interface	OpenLayers	QuantumGIS	WMS, WFS, GeoJSON, GML
Data Access	MapServer/TinyOWS	GeoServer	WMS, WFS
Data Source, vector data	PostGIS	Plain GML file	GML, SFA

Table 3: *Alternative tools tested*

OpenLayers	MapServer/TinyOWS	PostGIS
QuantumGIS	MapServer/TinyOWS	PostGIS
OpenLayers	GeoServer	PostGIS
QuantumGIS	GeoServer	PostGIS
OpenLayers	MapServer/TinyOWS	GML file
QuantumGIS	MapServer/TinyOWS	GML file
OpenLayers	GeoServer	GML file
QuantumGIS	GeoServer	GML file

Table 4: *Constellations tested*

The Web User Interface was alternated with QuantumGIS, an open-source desktop GIS for creating maps, importing layers from external resources, testing different projections and more [36]. Illustration 2 on page 9 was produced in QuantumGIS (all fire stations and district offices in Norway). Importing both raster and vector layers from the Data Access nodes into QuantumGIS required no re-programming in any part of the chain. This was the smoothest swap case performed, meaning it required the least amount of human involvement.

The Data Access node offering vector data through MapServer/TinyOWS was alternated with GeoServer. It required no interception on the Data Source level. The User Interface level required a minor modification in the configuration file due to the new DNS path. Using a configuration file to define all layer meta data proved useful, since it leaves the Javascript/HTML untouched.

Only vector data was swapped out on the Data Source level. The background raster map remained the same apart from the TMS swap, but such an alteration is inherently tightly coupled with the OpenLayers

code and produces no fair assumptions. The alternative source used was a plain GML file containing the same core data as the PostGIS source (fire stations). The exact same attributes and objects were used. It required alteration on the Data Source level since new (vector) layers had to be defined. Both GeoServer and MapServer need re-configuration to serve the new source since both tools are tightly coupled with their sources. An intermediary proxy placed between the Data Source and Data Access layer is a potential solution, but in reality it just pushes the problem in another direction. There is no getting around manual configuration of Data Access servers in the current solution.

Overall, the estimation has been made that if the involved geo-spatial Web standards had not been used, the whole project would have taken more time. It would have required the invention of an own protocol or an arbitrary way to represent data which would have added programming hours. To purchase software with proprietary protocols involved, is outside of the project budget and is thus not a feasible option.

7. Discussion

The discussion is divided into 5 different parts. *7.1* discusses the results of the empirical study. *7.2* evaluates the usability of OGC standards within the given context. *7.3* presents a proposed architecture for the final solution. *7.4* discusses ethical aspects and implications of this study. The final section *7.5* gives the GGI2 project a roadmap of where to go from here.

7.1. The Results

The empirical results show that some layers of the architecture required more reconfiguration and human involvement when modules were swapped, while others required less. However, the substance of the manual labor was more connected to implementations of different frameworks, not data representation.

The results have not been achieved through empirical measurements, but are rather estimations made arbitrarily in relation to each other. In this case – where the final report is aimed at producing an estimation for a project – such an outcome is acceptable. However, more firm results might have been produced if from the beginning, there had been a clear, measurable, empirical low-level goal. Such a goal was hard to establish initially, and if anything it shows that measuring the quality of a system is not a simple task.

Those modules that seemed unaffected should not be taken for granted. For instance, the User Interface configuration file seemed to be the only part reconfigured. In a larger implementation, nodes may be added, removed and renamed quite often. Who should be responsible for keeping the configuration file up to date? The problem remains even if a proper meta data catalog is implemented. Unless meta data gathering becomes automated, there will always be a degree of forced human involvement. This also raises the concern of governance and responsibility.

The framework stack has been developed by a user community over many years. Each framework has been programmed to function well with the others. The OGC standards for Web communication is the glue that binds them together, but there may be unforeseen consequences in switching out any of the tools. One example is how Data Access nodes are addressed. The User Interface must have knowledge of the address to reach a service. What if the address changes without notice, such as in the switch between technologies?

The data used has been produced *ad hoc* for the test case. Thus, the study cannot make any hard conclusions regarding information already

present in a data source somewhere. The result is highly bound to an isolated case. The study makes no claim to function with or fit the needs of those departments with existing data storage implementations. This may be regarded as a skewing of results, since reality looks very different and includes many more sources. Furthermore, traffic and data volumes in the test case are not the same as reality.

Only one or two map layers have been available in the interface at once. This is not an accurate portrait of reality - a reality that may call for 100 layers. Performance may suffer severely from a larger set of layers, especially if they come from a vast amount of differing data sources with differing bandwidth. Furthermore, the traffic against the test case servers is very low which enables exclusive and instant response. It may not be the case in other setups.

7.2. Using OGC Standards

Technical standards are developed to promote interoperability between modules in a software architecture. Standards are fine-tuned over the course of years by developer communities, ensuring constant optimization and growth. This report has shown that building a GIS based on OGC standards is beneficial and does indeed enable swapping between OGC-compliant/implementing tools. The swapping was not 100% effortless but did to a degree cut down human involvement.

Many useful frameworks rely on the standards; OpenLayers, GeoExt, MapServer, GeoServer, TinyOWS and PostGIS. This is another reason to strive for compliance when gathering and washing data - the information can easily be integrated into the mix. Using well-known and documented frameworks is not only convenient, but also comes with a large community of programmers ready to expand the product and help their peers. The author experienced this directly when occasionally getting stuck in the process, eventually finding help and solutions from others who had suffered the same problems. Hadn't it been for the help of other programmers, the whole test model may have been delayed to the point of failure.

Using OGC standards requires an understanding of the very fundamentals of geodetics, thus the report's theoretical section has largely focused on these aspects. For example, the WFS standard requires a grasp of the concept of features, which in turn calls for a grasp of vector data. The concept of projection, used directly in coding in OpenLayers, requires an understanding of the very purpose of projections.

The GGI2 project is advised to let any future technical solutions comply with the OGC standards, especially when considering formats for data storage. The standards should act as a pivoting point and guidance, to

unify participants and give the project a trajectory. *Homogeneity* should be the lead word - and what better way to achieve homogeneity than through compliance with well-developed standards? Following standards will unify the process, but still requires central governance. Someone must be responsible for the whole architecture.

One major weakness of the test model is its small scale. An optimal solution will be a large architecture involving many various nodes and more complex data. Problems not foreseen in this study will most likely arise. Will the tools tested in the model be able to handle larger volumes of data and a more intense traffic flow?

7.3. Proposed Structure

Two additional modules are proposed as an addition to the model. The proposed modules are the *Proxy Node* and the *Meta Data Catalog*. These will ensure a higher degree of decoupling and more efficient development.

Several results of the study arrive at the same conclusion - the User Interface and Data Access layers need to be decoupled. Data regarding the *where*, *how* and *what* of service-providing nodes should not be stored locally within the interface application. If the time comes to build clients for other purposes using the same services, the system instantly becomes very prone to redundancy errors.

A centralized *Proxy Node* could act as a single point of entry for clients (as opposed to clients communicating with many various Data Access nodes). Clients would send all queries to the proxy server. This would give better control of how data is delivered. The task of the Data Access nodes should be only to provide data, while the Proxy Node would perform Meta Data Catalog lookups and structure and unify data representation. Clients would only need to know one address.

A potential issue with the Proxy Node implementation is load balancing. In other words - one single point of entry runs the risk of becoming a bottle-neck. The server would need to be fast and capable enough, or alternatively server clustering technology could be used.

Another issue with the Proxy Node is the question of what actual software/framework to use. The software would have to be able to accept, process and deliver geo-spatial data, preferably with caching mechanisms to speed up performance. Also, it must be sophisticated enough to perform lookups in a meta data catalog (the second proposed module). And finally, it must accept all service types, i.e. WMS, WFS, TMS and so on. The author has not managed to find any server software capable of all these tasks.

The Proxy Node would perform lookups in the *Meta Data Catalog*. In other words, the only node in the whole architecture cognisant of all Data Access nodes would be this catalog. It could potentially be implemented using the open-source GeoNetwork OpenSource [37], the only OSGeo-approved meta data framework. Similarly to OGC but smaller in size, OSGeo is a foundation approving and recommending open-source geo-spatial frameworks [37]. OSGeo will only recommend frameworks meeting certain quality standards, meaning GeoNetwork is probably the best open-source bet. The author has not tested GeoNetwork and does not exclude the possibility of using alternative frameworks. The proposed architecture and framework stack is depicted in Illustration 13.

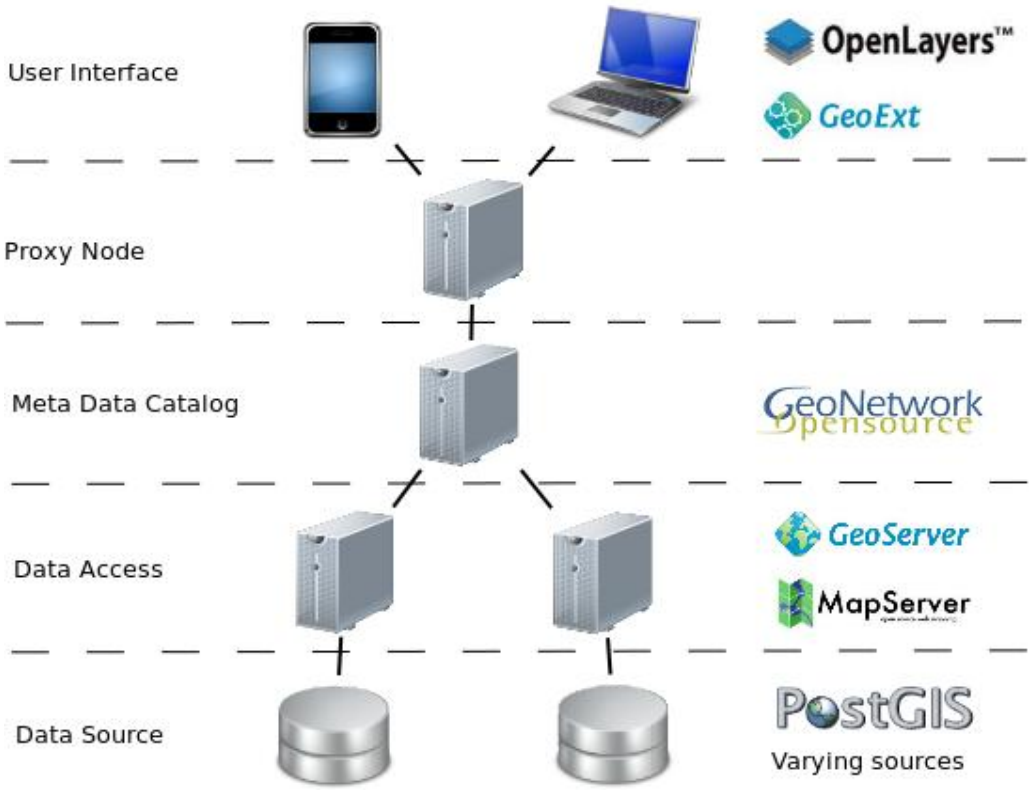

Illustration 13: *Proposed architecture*

A problem with the Meta Data Catalog is that human involvement and continuous maintenance is still required, albeit in a more proper and organized way. Just as the DISPRO project participants experienced - someone has to be responsible for keeping the catalog up to date.

Current technology is not advanced enough to gather meta data automatically. However, the author is hopeful that one day standards will be developed to 'crawl' the Web and find and index Web Services. It is already possible to search content of Web pages through search

engines such as Google and Bing. Seeing the emerging trend of distributed services, there is great reason to believe we will one day be able to crawl Web Services as well. Perhaps a simple search string, eg. 'Norwegian towns', will lead to a set of results, including the service available at https://ws.geonorge.no/SKWS52/. This would be beneficial to society as a whole, putting valuable tools and information in the hands of regular users.

The fact that OSGeo only recommends one meta data tool, compared to ten web mapping tools and eight geo-spatial libraries, indicates that the field of meta data is less developed than others. Emerging organizations such as the Dublin Core Metadata Initiative spark hope for the future.

7.4. Ethical Aspects

This study finds itself touching the topic of government and responsibility. The emergency services save citizens' lives on behalf of the government and the people.

On one hand, the emergency services should make sure their technical tools are reliable in order for rescue operations to succeed. This might mean purchasing software from companies using proprietary protocols for communicating geo-spatial data. But on the other hand, the government has a larger purpose to serve all aspects of society. The OGC is an international collaboration that works non-profit to unify and harmonize – to solve the problems that the GGI2 project is facing. Participation and use of the OGC standards is the fuel behind the standards' very existence. If the government doesn't promote international, open collaboration – what does that imply to the citizens?

Promoting for-profit companies by purchasing and using their products will increase the companies' standing on the market. The vendor with the most followers will have the power to decide *how* we view geo-spatial data. The consequences of such a scenario are only subject to wild speculation. We must ask ourselves – do we want commercial powers to drive the way we view our world?

Promoting non-profit collaboration by using and contributing to the standards researched in this report will increase openness in the geo-spatial Web community. Anyone will have the possibility to veto any decisions or introduce a new idea. Certainly, large non-profit organizations are not immune to corruption, malicious individuals and internal power struggles. Good intentions and rights ambitions can aid these problems. The GGI2 project has the power to make a choice and bring their good ambitions to the open geo-spatial community.

7.5. Where to Go from Here

The most pressing issue for future progress is to investigate how existing data sources can be integrated into the architecture. A large, monolithic database is not an option since it would require massive efforts, space, bandwidth and maintenance. It would involve data subject to a large amount of regulations, and would furthermore be very prone to updating errors, should data exist both in the monolithic database as well as locally at some department. To achieve a functioning Web portal, the GGI2 project must thoroughly investigate and solve each of the following steps:

1) From a technical point of view, research exactly where and how all desired data is currently stored.

2) Investigate ways to communicate data. Confidentiality, authorization issues and license costs could become obstacles.

3) Agree on common protocols for communication. The OGC Web standards have been proven a good option and the author suggests using the standards as a central focus.

4) If ready for use, serve the data via MapServer and integrate it into the application. If not, wash the data to an appropriate format. This step is far from trivial and could involve a lot of work.

5) Owners of data must expose their resources via APIs agreeing with the protocol. If owners of data do not collaborate and engage in this step, the project will fail. Some participants may need technical help - something the project coordinators must provide.

6) Further develop the application to take care of primarily security issues, but also performance optimization and user interaction.

7) Find resources to host the Data Access nodes and other crucial servers involved. The servers used in the test cases will not be an option. This can give rise to questions regarding responsibility - who is going to be responsible for maintenance and hosting when so many different departments are involved?

These steps call for technical expertise. Including a team of computer technicians/Web programmers is an absolute must to achieve the desired Web portal - at least one person for each layer, working full-time to fulfill their part of the architecture.

The governing organs heavily weigh the financial factor into their decision. If the Web portal is deemed an unrealistic goal due to budget or time constraints, an option could be to eliminate the geo-spatial element. An alternative is a simple tree or catalog Web interface listing

resources in some structured manner. The catalog would be searchable and intuitively organized. Although simpler, this alternative would still require steps 1), 2) and 3) and the help of programmers. An exemplifying excerpt:

```
`-- Fire Stations
   `-- Järpen Fire Station
      |-- Height Vehicles
      |   `-- SDZ 805
      `-- Fire Vehicles
         |-- SDZ 705
         |-- SDZ 706
         |-- SDZ 707
         `-- SDZ 708
```

Outlooks are bright for the fire, police and ambulance departments of Jämtland and Nord-Sør Trøndelag. Collaboration is key, and the GGI2 initiative in itself proves the definitive existence of a positive attitude towards cooperation.

References

[1] U.S. Department of Commerce Technology Administration, National Institute of Standards and Technology. Gallaher et al. *Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry*. NIST GCR 04-86

[2] "RAKEL" on 9 Mar 2012, https://www.msb.se/sv/Produkter--tjanster/RAKEL/

[3] Krafzig et al., 2005, *Enterprise SOA: Service-Oriented Architecture Best Practices*, ISBN: 0131465759

[4] Hürsch, Walter L. & Lopes, Cristina Videira, 1995, *Separation of Concerns*. College of Computer Science, Northeastern University, Boston, MA.

[5] Obe, Regina O. & Hsu, Leo S, 2011, *PostGIS in Action*. Manning Publications, ISBN: 9781935182269 .

[6] Alameh, Nadine, 2001, *Scalable and Extensible Infrastructures for Distributing Interoperable Geographic Information Services on the Internet*. PhD publication from Massachusetts Institute of Technology. Dept. of Civil and Environmental Engineering.

[7] Image and information taken with permission on 3 Mar 2012 from http://earthobservatory.nasa.gov/Features/GRACE/page3.php

[8] Image and information taken with permission on on 3 Mar 2012 from http://en.wikipedia.org/wiki/Ellipsoid

[9] "EPSG Geodetic Dataset" on 3 Mar 2012, http://www.epsg.org/geodetic.html

[10] ISO 19111:2007 *Geographic Information -- Spatial Referencing by Coordinates*, definition 4.5

[11] W3C Recommendation *Scalable Vector Graphics 1.1 (Second Edition)*

[12] Image taken with permission from author Jeffrey Dunn on 2 Mar 2012 at http://outsidetheneatline.blogspot.com/2009/08/did-you-know-6-raster-vs-vector.html

[13] "About OGC" on 2 Mar 2012, http://www.opengeospatial.org/ogc

[14] "OGC Standards" on 2 Mar 2012, http://www.opengeospatial.org/standards

[15] "OGC Compliance" on 2 Mar 2012, http://www.opengeospatial.org/compliance

[16] ISO/IEC 40270:2011, *Information technology -- W3C Web Services Policy 1.5 -- Framework*

[17] Alameh, Nadine, *Chaining geographic information Web services*, *Internet Computing, IEEE* , vol.7, no.5, pp. 22- 29, Sept.-Oct. 2003

[18] W3C Working Group Note 11 February 2004, *Web Services Architecture, 3.5.3 The Role of meta data*

[19] OGC 06-121r9 *Web Services Common Standard*

[20] OGC 06-042 *Web Map Service*

[21] OGC 07-057r7 *Web Map Tile Service*

[22] ISO 19125 *Geographic information -- Simple feature access*

[23] ISO/IEC 9075-3:2003 *Information technology -- Database languages -- SQL -- Part 3: Call-Level Interface (SQL/CLI)*

[24] OGC 09-025r1*Web Feature Service*

[25] ISO/DIS 19142 *Geographic information -- Web Feature Service*

[26] ISO 19101:2002 *Geographic information -- Reference model*, definition 4.11

[27] ISO 19136:2007 *Geographic information -- Geography Markup Language (GML)*

[28] ISO/IEC 13249-3:2011 *Information technology -- Database languages -- SQL multimedia and application packages -- Part 3: Spatial*

[29] Tuama, Éamonn Ó & Hamre, Torill, 2007, *Design and Implementation of a Distributed GIS Portal for Oil Spill and Harmful Algal Bloom Monitoring in the Marine Environment*, Marine Geodesy, 30:1-2, p 145-168

[30] "Vfront" on 2 Apr 2012, http://www.vfront.org

[31] "GeoJSON" on 2 Apr 2012, http://www.geojson.org

[32] "MapServer" on 2 Apr 2012, http://www.mapserver.org

[33] "TinyOWS" on 2 Apr 2012, http://www.tinyows.org

[34] Leff, A, 2001, *Web-application development using the Model/View/Controller design pattern*. Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International p 118-127

[35] "SOSI" on 13 Apr 2012, http://www.statkart.no/nor/sosi/

[36] "QuantumGIS" on 14 Apr 2012, http://www.qgis.org/

[37] "GeoNetwork" on 15 Apr 2012, http://geonetwork-opensource.org/

[38] "OSGeo" on 15 Apr 2012, http://www.osgeo.org/

[39] "Dublin Core" on 15 Apr 2012, http://dublincore.org/

Appendix A: GGI2 Survey with the purpose to extract information about data

Translated from Swedish

Answer these questions from your own point of view.

1. What is your job, and what are your tasks at work?

2. What is your role in the project?

3. What do you believe are the general goals of the project?

4. Who do you believe are the primary parties? Are secondary parties involved?

5. Who do you believe are the end users of the project?

7. What potential problems do you think could prevent the project from Achieving its goals?

8. What are the priorities made in the project goals?

10. Do you as a project participant have any wishes for the project?

11. What information within your organization could be used in the project? For example: Background maps, resource overview, map overview, name database, address database, buildings etc.

12. In what form is the information stored (digitally/paper etc.) This information shall be available (on this page or brought to the session).

13. Where is the information stored?

14. Who is responsible for it?

15. What information not in your possession would be useful to you?

15. Do you see any issues of confidentiality that could potentially effect the results of the project?

Appendix B: The departments' own categorization of data

All material in this appendix has been provided directly by the end users and was used by the author to model a database

Health Departments

Ambulanestasjoner

- Koordinater
- Kontaktpunkt (Utalarmeringssentral)
- Antall amb
- Terrengfordon
- Døgnberedskap
- administrativ nummer til stasjon
- Ikke fast/innmontert utstyr som kan taes med ut i fra oppdrag.ressurses tilgjengelig på stasjon
- utrustning
- 4 wdrive

Ambulanse

- Hvilken Ambulansestasjon
- Terrängkapacitet ja/nej
- Radionummer /Rakel/
- Personell
    - utbildning
    - Delegasjon
- Mobilnummer til ambulans
    - antall faste båre
    - Båretyper
        - Våkummadras

- Fixasjonsutstyr
- Varmematriell
- intubasjon/ventilasjon
- Medisinsk uystyr
    - defifrillator
    - EKG utstyr
- egemiddel/vædsker

Helsebuss/ambbuss

- Båreplasser
- Sitteplasser
- Personell ombord kvalifisering/utbildning.

Ambulansehelikopter baser

- Koordinater
- Kontaktpunkt (sentral)
- Type helikpter
- Båreplasserutrusning
- Mulighet til underhengende operasjoner
- Utbildning personell
- Ikke fast/innmontert utstyr som kan taes med ut i fra oppdrag.

Sykehus

- Koordinater
- Hvilken sentral tilhører sykehuset.
- Kontaktnummer
- Type sykehus
- Kapasitet
- Hvilke typer pasienter kan de ta
    1. Traume
    2. Hodeskade
    3. Brannskader

Hälsocentraler

- Akutnr
- Kordinater
- sjukvårdsgrupper
- förmåga

Fire Departments

Brandstationer
(Grundinformation)

- Koordinater
- Kontaktuppgifter: Vem är ansvarig för stationen, telefonnummer, e-mail, m.m. Vilken kommun tillhör stationen.
- Samlingslokaler
- Beredskap: hur många har beredskap vid stationen och anspänningstid. Vilken larmcentral larmar ut enheten.
- Fordon
- Förmågor
- Symbol för vilken typ av fordon som finns på stationen. Vi behöver klargöra vilka benämningar vi har i respektive land.
- Symbol för förråd

Samlingslokaler

antal personer
sagesman (polis, räddtj, sjukvård)
Kontakt (person + telefonnr)
Ägare (fysisk ägare)
Kokmöjligheter ja/nej
Dusch ja/nej
Telefon ja/nej
Värme ja/nej
Vatten ja/nej

Förmågor
Typ: Rökdykning, Kemdykning,. Tung räddning (buss), . Klargör benämnigar. Vattendykare. Rappelering.

Fordon

- Anropsnummer (RAKEL)
- Mobilnummer
- fordonstyp

Fordonstyp

- Släckbil: Utrustning grovt beskrivet för varje fordonstyp
- Tankbil: vattenmängd
- Höjdfordon:
    .1 antall meter
- xx
- xx

Förråd

- Utrustning

Utrustning
Typ: Defribrilator. miljöräddning (läns, pump...). Vattendammsugare (restvärde). Tält, bårar. Slangförråd. Utrsutning för att skapa värme. Vacuummadrass, helikopterbår. Motorsprutor/pumpar . Hydrauliska klippverktyg. Skum. PPV-fläkt. Bårmateriell

Police Departments

Polisstation

Telefonnummer till LKC (Ledningscentralen för denna stationen)
nummer

Hund		x antal Hundar
Ledningsfordon		antal
Insatsstyrka SWAT		befintlighet, direkttelenummer
Andre polititjenestemenn		specifik förmåga
Förhandlare	antal	
Eftersök MSO		antal
Öppettider		klockslag
Polisskotrar	antal	

Hund

Förare
Telefon
Typ (Lavin, krimsök, nark, bomb, likhund)

Helikopterbasering

Kontaktnummer		nummer
Beredskap	Tid	
Nattkapacitet		ja eller nej

Polisflygplan (mindre)

Kontaktnummer		nummer
Beredskap	Tid	
Nattkapacitet		ja eller nej

Stationeringsplats för räddningsstyrka till fjälls

Kontaktnummer	nummer
Fordon	x antal Fjällräddningsfordon
Alpin förmåga	ja eller nej

Lavinhund x antal Hundar

Fjällräddning fordon

Typ
Antal bårplatser

Appendix C: User Interface Javascript code including configuration file

App.js

```
/*
 * App
 *
 * Represents the application structure
 *
 * Singleton object
 *
 * Lena Sundin March 2012
 */

function App(mapPanel) {

   //Internal variables not accessible by returned instance
   var viewPort = null;
   var mapPanel = mapPanel;
   var formPanel = new Ext.FormPanel({
      frame : true,
      bodyStyle : 'padding:5px 5px 0',
      defaultType : 'textfield',
      buttons : [{
         text : 'Create Event',
         handler : function() {
            createEvent();
         }
      }]
   });

   /*
    * Create tree to display layers
    */
   var createTree = function() {
      var LayerNodeUI = Ext.extend(GeoExt.tree.LayerNodeUI,
         new GeoExt.tree.TreeNodeUIEventMixin());
      return new Ext.tree.TreePanel({
         autoScroll : true,
         plugins : [{
            ptype : "gx_treenodecomponent"
         }],
         loader : {
            applyLoader : false,
            uiProviders : {
               "custom_ui" : LayerNodeUI
            }
         },
         root : {
            nodeType : "gx_layercontainer",
```

```
          loader : {
            baseAttrs : {
                      uiProvider : "custom_ui"
            },
            createNode : function(attr) {
            if(!mapPanel.layers.getByLayer(attr.layer).data.layer.features)
            return GeoExt.tree.LayerLoader.prototype.createNode.call(this, attr);
                      attr.component = {
                                xtype : "gx_vectorlegend",
                                layerRecord :
mapPanel.layers.getByLayer(attr.layer),
                                showTitle : false,
                                cls : "legend"
                      }
            return GeoExt.tree.LayerLoader.prototype.createNode.call(this, attr);
            }
          }
        },
        rootVisible : false,
        lines : false
      });
    }
    /*
     * Return singleton instance of this class
     */
    var getInstance = function() {
      if(!App.singletonInstance) {
        App.singletonInstance = createInstance();
      }
      return App.singletonInstance;
    }
    /*
     * Create singleton instance
     */
    var createInstance = function() {

      //Application viewport
      var tree = createTree();
      viewPort = new Ext.Viewport({
        layout : "border",
        id : "viewport",
        items : [mapPanel, {
          region : 'east',
          collapsible : true,
          split : true,
          margins : '0 5 0 0',
          width : 200,
          layout : 'fit',
          items : new Ext.TabPanel({
            id : "tabPanel",
            border : false,
            activeTab : 0,
            tabPosition : 'bottom',
            items : [{
                      //Tab 1
                      title : "Layers",
                      layout : 'fit',
                      items : tree
            }, {
```

```
                    //Tab 2
                    title : "Form",
                    id : "formPanel",
                    items : formPanel,
                    autoScroll : true
            }]
          })
        }]
      });

      return {
        formPanel : formPanel,
        getFormPanel : function() {
          return this.viewPort.find("id", "formPanel")[0].items.items[0];
        },
        setFormPanel : function(items, handlerFunction) {
          this.formPanel = new Ext.FormPanel({
            frame : true,
            bodyStyle : 'padding:5px 5px 0',
            defaultType : 'textfield',
            items : items,
            buttons : [{
                      text : "Save",
                      handler : function(el) {
                                handlerFunction(el);
                      }
            }]
          });
          this.getFormPanel().destroy();
          this.viewPort.find("id", "formPanel")[0].add(this.formPanel);
          this.viewPort.find("id", "tabPanel")[0].setActiveTab("formPanel");
          this.viewPort.doLayout();
        },
        viewPort : viewPort
      }
    }
    //Return the singleton instance of this object
    return getInstance();
}
```

Map.js

```
/*
 * Map
 *
 * Contains all code related to the map, layers, features etc.
 *
 * Singleton object
 *
 * Lena Sundin March 2012
 */

function Map(MODEL) {

    var layers = [];
    var projection = MODEL.projection;
    var extent = new OpenLayers.Bounds(MODEL.extent[0], MODEL.extent[1],
MODEL.extent[2], MODEL.extent[3]);
```

```
   var layerStore = new GeoExt.data.LayerStore();

   //Some initial configuration, creating map and base layer
   //Map
   var map = new OpenLayers.Map('map', {
     projection : new OpenLayers.Projection(projection),
     maxResolution : MODEL.maxResolution,
     resolutions : MODEL.resolutions,
     units : MODEL.units,
     numZoomLevels : MODEL.numZoomLevels,
     maxExtent : extent,
     panDuration : 100,
     controls : [new OpenLayers.Control.MousePosition(), new
OpenLayers.Control.Navigation(), new OpenLayers.Control.LayerSwitcher(), new
OpenLayers.Control.PanZoomBar(), new OpenLayers.Control.KeyboardDefaults(), new
OpenLayers.Control.ScaleLine()]
   });

   //Base layer
   var baseHost = MODEL.hosts[MODEL.layers["BASE"].host];
   var base = MODEL.layers["BASE"];
   var baseLayer = new OpenLayers.Layer.TMS(base.friendlyName, baseHost.url, {
     layername : base.layername,
     type : base.type,
     tileSize : new OpenLayers.Size(base.tileSize[0], base.tileSize[1]),
     serverResolutions : MODEL.resolutions,
     transitionEffect : 'resize'
   });
   map.addLayer(baseLayer);
   map.zoomToMaxExtent();

   /*
    * Fetch all WFS layers
    * @param arrWFS Array of WFS layers
    * @param hosts Array of hosts
    */
   this.initWFS = function(arrWFS, hosts) {
     for(var v = 0; v < arrWFS.length; v++) {
       //For each layer in the config
       var entry = arrWFS[v];
       //Create new layer
       var saveStrategy = new OpenLayers.Strategy.Save();
       var layer = new OpenLayers.Layer.Vector(entry.friendlyName, {
         strategies : [new OpenLayers.Strategy.BBOX(), saveStrategy],
         projection : projection,
         styleMap : new OpenLayers.StyleMap({
           "default" : new OpenLayers.Style(entry.style)
         }),
         protocol : new OpenLayers.Protocol.WFS({
           version : "1.0.0",
           srsName : projection,
           url : hosts[entry.host].url,
           featureType : entry.layername,
           geometryName : entry.geometryName
         })
       });
     }

     //initWFS
```

```
      //Write back to data source
      var saveAttributes = function(option) {
        if(option == "cancel" || option == "no")
          return;
        var newAttr = APP.formPanel.getForm().getValues();
        map.selectedFeature.attributes = newAttr;
        map.selectedFeature.state = OpenLayers.State.UPDATE;
        var response =
map.selectedFeature.layer.protocol.commit([map.selectedFeature]);
        Ext.Msg.alert("Done", "Changes saved.");
      }
      //initWFS
      //Save feature after user hits 'Save' in the east form panel
      var saveAction = function(el) {
        Ext.Msg.show({
          title : "Save?",
          msg : "Do you want to save the changes?",
          buttons : Ext.Msg.OKCANCEL,
          fn : saveAttributes
        });
      }
      //initWFS
      //Feature popup
      var selectCtrl = new OpenLayers.Control.SelectFeature(layer);
      function createPopup(feature) {
        var html = "<table border=0 class='selectedFeatureClass'><tr><td>";
        for(var v in feature.attributes) {
          html += v + "</td><td>" + feature.attributes[v] + "</td></tr><tr><td>";
        }
        html += "</td></tr></table>";

        var popup = new GeoExt.Popup({
          title : layer.name,
          location : feature,
          html : html,
          maximizable : true,
          collapsible : true
        });
        popup.on({
          close : function() {
            if(OpenLayers.Util.indexOf(layer.selectedFeatures, this.feature) > -1)
{
                    selectCtrl.unselect(this.feature);
            }
          }
        });
        map.selectedFeature.popup = popup;
        popup.show();
      }

      //retriveWFS
      //Feature form
      function featureSelect(feature) {
        if(map.selectedFeature != undefined && map.selectedFeature != null)
          map.selectedFeature.popup.destroy();
        map.selectedFeature = feature;
        createPopup(feature);
        var items = [];
        for(var v in feature.attributes) {
```

```
          items.push({
             fieldLabel : v,
             name : v,
             allowBlank : true,
             value : feature.attributes[v]
          });
        };
        APP.setFormPanel(items, saveAction);
      };

      //
      layer.events.on({
        featureselected : function(e) {
          featureSelect(e.feature);
        }
      });
      map.addControl(selectCtrl);
      selectCtrl.activate();
      saveStrategy.activate();
      map.addLayer(layer);
    }
    //Begin population of map and related data
    //Fetch all WFS layers
    this.initWFS(MODEL.layers.WFS, MODEL.hosts);

    //This is a singleton class
    var getInstance = function() {
      if(!Map.singletonInstance) {
        Map.singletonInstance = createInstance();
      }
      return Map.singletonInstance;
    }
    //Returned object managed by the interface
    var createInstance = function() {

      return {
        map : map,
        layers : layers,
        layerStore : layerStore,
        selectedFeature : null
      }
    }
    //Return the singleton instance of this object
    return getInstance();
}
```

Model.js

```
/*
 * Model
 *
 * Separates rest of application from data operational
 * work such as fetching layers and initiating the
 * application with the correct configuration parameters.
 *
 * Singleton object
 *
 * Lena Sundin March 2012
```

```
 */


function Model() {

   //Internal variables not accessible by returned instance
   var conf = null;

   /*
    * Get configuration parameters.
    */
   var initModel = function() {
      $.getJSON("config.php", function(data) {
         MODEL.hosts = data.hosts;
         MODEL.layers = data.layers;
         for(var v in data.map) {
            MODEL[v] = data.map[v];
         }
         init(); //Global function in init.js
      });
   }

   //This is a singleton class
   var getInstance = function() {
      if (!Model.singletonInstance) {
         Model.singletonInstance = createInstance();
      }
      return Model.singletonInstance;
   }

   //Returned object managed by the interface
   var createInstance = function() {
      return {
         initModel : initModel
      }
   }

   //Return the singleton instance of this object
   return getInstance();
}
```

init.js

```
/*
 * init.js
 *
 * Initiate the whole application.
 *
 * Lena Sundin March 2012
 */

var APP;
var MODEL = new Model();
var EVENT = new Event();
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";

$(document).ready(function() {
   MODEL.initModel();
```

```
});
function init() {
   var MAP = new Map(MODEL);
   var mapPanel;

   /***********Map panel**************/
   mapPanel = new GeoExt.MapPanel({
      region : "center",
      id : "mappanel",
      xtype : "gx_mappanel",
      map : MAP.map,
      zoom : 3,
      split : true,
      tbar : new Ext.Toolbar({})
   });

   /***********Initiate application layout*************/
   APP = new App(mapPanel);
}
```

config/ggi2.json

```
{
   "hosts" : {
      "GISLAB1" : {
         "url" : "http://gislab.miun.se/cgi-bin/ggi2wms"
      },
      "GISLAB2" : {
         "url" : "http://gislab.miun.se/mapproxy/service"
      },
      "GISLAB3" : {
         "url" : "http://gislab.miun.se/cgi-bin/tilecache.cgi/"
      },
      "GEOSERVER1" : {
         "url" : "http://localhost:8080/geoserver/ggi2/ows"
      }
   },
   "map" : {
      "maxResolution" : 1562.5,
      "extent" : [234500, 6825100, 601500, 7225100],
      "projection" : "EPSG:3006",
      "units" : "m",
      "numZoomLevels" : 13,
      "panDuration" : 100,
      "resolutions" :
[1562.5,781.25,390.625,195.3125,97.65625,48.828125,24.4140625,12.20703125,6.1035156
25,3.0517578125,1.52587890625,0.762939453125,0.3814697265625]
   },
   "layers" : {
      "BASE" : {
         "host" : "GISLAB3",
         "layername" : "ggi2",
         "type" : "png",
         "tileSize" : [256, 256],
         "friendlyName" : "GGI2 AOI"
      },
      "WFS" : [{
         "host" : "GEOSERVER1",
```

```
        "layername" : "f.station",
        "friendlyName" : "Fire Station",
        "geometryName" : "the_geom",
        "style" : {
          "fillColor" : "#ff3300",
          "pointRadius" : 8
        }
      }]
   }
}
```