Using WFS to build GIS support

Sandra Fällman
Department of Computing Science
Umeå University, Sweden
c99sfs@cs.umu.se

April 7, 2004

Abstract

As Internet today is our most important information resource, all kind of information need to be accessible from the web. Much of the information has reference to location. This type of information is called *spatial information* and has now entered the World Wide Web, providing web users with map information. In the attempt to *geo enable* the web, standard specifications has for a long time been a missing factor. This has led to interoperability problems among spatial Web Services, and instead of having a geo-spatial web community, today only stand alone systems are provided. The OpenGIS Consortium (OGC) is an association trying to solve the problem by offering open computing standards for geo-processing.

The main topic of this Master's Thesis project is to investigate the OGC standard called *Web Feature Services (WFS)*. The technique offers new possibilities to geo-processing by making it possible to interact with static map images through the exchange of XML documents. In the report, the background of the OGC standards is discussed, the design of the WFS is analyzed, and a framework investigation is performed. Also, a demonstration system is constructed, with servers implementing these OGC standards. The result is a flexible and distributed system with an interactive client, providing users with map and feature information of his/her needs.

Contents

List of Figures

List of Tables

1 Introduction

The society we live in today is a mobile and distributed society. Because of this, most of the information flowing around has reference to location. Nearly 80 percent of the government and business information has a geographic reference. The amount of spatial data existing, and the power that spatial data have, requires that our technologies and services are able to handle this kind of data.

For centuries geographic data (GIS data) has been collected. Although much of this data will remain on paper, much has also been digitalized. The GIS data, or spatial data, which is collected nowadays, is stored in databases, flat files or in Geographic Information Systems (GIS). The fact that there is no accepted standard for the representation of spatial data, and the fact that geographic resources are designed for many different purposes, makes it difficult to integrate services of GIS data. Now, with spatial data entering the scene of the World Wide Web, the number of people sharing this data has increased, and the key concept for the creation of distributed spatial Web Services, is interoperaility [OB02].

1.1 Master's Thesis project task

By using the technique Web Feature Service (WFS) to build GIS support, information from different sources can be merged together and be adapted to different roles and situations.

In this Master's Thesis project a demonstration system (demo system) is created, in which geographic data from different sources are merged together and delivered to an interactive client application. The client provides functionality for navigating in the information space and makes it possible to extend geographic information based on the needs of the user.

The main task is to develop a WFS interface on top of an Object Relational Database Management System (ORDMS) and evaluate how well the standard works with stored data.

The data to be served by the WFS server is stored tracks from existing command and control applications within SaabTech. Other data sources to use in the system, are map data served from already existing Web Map Services (WMS).

The literature part of the Master's Thesis project includes an investigation of the background of the techniques WMS and WFS and investigates what problems WMS and WFS are intended to solve. A critical analysis of the design of WFS is also performed. Interesting areas in this critical analysis are security, performance, scalability and reliability. For the implementation of the demo system, an investigation of possible and suitable frameworks for WMS and WFS is done. The central part of the literature study lies on WFS.

The implementation has one additional level (level2), which consists of the implementation of a smarter client application, to enhance the interactivity with the user. As a work for the future, one more level of the demo system could be realized, in which another external data source would be integrated in the demo system. This data source could be a database containing additional information about the features that can be seen on the map.

1.2 Goal and approach

The goal of the Master's Thesis project is to examine the technique of WFS, its possibilities and limitations, in order to determine its usefulness for SaabTech. To achieve this goal, both

a literature study and implementation part are required. The technique of WMS is also examined, although this technique is already known and used at SaabTech. The reason to this is that since the WFS technique is developed from the WMS technique, it is necessary to understand WMS to be able to understand and analyze WFS.

The literature study is divided into three different parts:

- Investigation of the background of WMS and WFS and what problems these techniques are intended to solve.
- Critical analysis of the design of WFS with focus on the areas security, performance, scalability and reliability.
- Investigation of possible and suitable frameworks for WMS and WFS.

1.3 Overview

In the following three sections, section 2, 3 and 4, the results from the literature study are presented: section 2 describes the background of WMS/WFS, section 3 the analysis of the design of WFS and section 4 presents the investigation to determine suitable frameworks for WMS and WFS in the demo system. Section 5 and 6 describe the implementation level1 and level2. In section 7, future work is discussed. Section 8 is a summary, and conclusions are presented here. The last section, section 9, contains acknowledgements.

1.4 SAAB and SaabTech

SAAB is one of the world leading high-technology companies and develops products and services mainly for the defense market, but also for the commercial market. SAAB's main focus lies in the area of defense, aviation and space [Tec04a]. SaabTech is a business unit of SAAB and is a merge of SAAB Avionics and SaabTech Systems and offers products and services that are central for the defense market, such as avionics, electronic warfare and decision superiority systems. Products are developed for land, air and naval use [Tec04b]. This Master's Thesis project is performed at the Air systems division at SaabTech.

2 Background of WMS and WFS

To call the world we live in today a mobile society is almost an understatement. Most activities in our daily life have to do with movements. All movements have spatial information to give. To present this type of information on the web is called *online mapping* or *web mapping*.

2.1 Lack of interoperability

Many organizations and companies have provided services for this kind of data for a time now, but these systems have all been implemented in different ways and therefore they cannot interoperate. Vendors perform isolated development and create unique systems. This reality leads to a situation where users are locked into single vendor solutions and the web client can only interact with its own vendors map server implementation, and no one else's. The situation can be visualized by thinking of each web mapping system as its own technology island; it is on its own without connection to other web mapping systems. Although the situation is slowly getting better, the lack of interoperability still exists [Con04c].

2.2 Addressing the problem

Since there exist a lot of servers, each providing valuable and different spatial information of interest for many different clients, the situation presented above is not desirable. The broad access to the information is strongly limited when each server has its own implementation, and does not follow a published interface specification.

A publicly documented interface specification would give the opportunity to vendors of web mapping services to follow the specification, and make their products interoperable with other web mapping services, implemented by other vendors.

But to have an interface publicly documented may not be sufficient. It needs to be a standard of an extent that it is accepted by many implementers [Con04c].

2.3 OGC

The problem of creating an open standard for web mapping was taken care of by the Open GIS Consortium (OGC). OGC offers open interfaces for the implementation of web mapping services. OGC provides two programs (OGC Specification Program and OGC Interoperability Program) for the planning, developing, reviewing, and last of all, officially adopting the specifications.

The Open GIS Consortium, Inc. (OGC) is an international industry consortium. Among the 254 members, there are government agencies, companies and universities, all working together in the development of these open geographical standards [Con04a].

The *OGC Implementation Specifications* are a set of interfaces that specify the request and response protocols for the interaction between open web-based servers and clients [Con04c]. Two of the specifications are the *Web Map Service Implementation Specification* (the WMS Implementation Specification) and the *Web Feature Service Implementation Specification* (the WFS Implementation Specification).

In a situation where multiple servers implement e.g. the OGC WMS Implementation Specification, a client application would have the possibility of accessing map data from each server, and merge them into one picture. For example, one server could be asked for a map layer showing all mountains in a specific area of the world, another server could be asked to provide a map layer of all lakes in the same area, and a third server could be asked for a map layer showing all cities in the area (assuming that the three servers serve this information; mountains, lakes and cities, respectively). These map layers can be requested by a client application and shown by the client as one map picture, with all three layers merged into one. All three servers are asked for the layer in the same way, since it is specified in the WMS Implementation Specification how a request for a map layer shall look like. The lack of interoperability is no longer present.

With the OGC Implementation Specifications, the isolated technology islands do not only develop bridges between each other, they are merged into one big continent where spatial data is easily exchanged between the different servers and clients.

2.4 OGC Web Services

When an application's process execute on another computer, located in a network, it is called distributing computing. The process uses the network (i.e. the interfaces, protocols, schemas etc.) for data transmission. These online computing processes are called Web Services. *OGC Web Services* is the definition of the standard interfaces and protocols, encodings and schemas, for Web Services that handle geo-spatial information [Wor03].

OGC Web Services support today only one distributed computing platform, Internet, and the Hypertext Transfer Protocol (HTTP) is used.

Since the area of web mapping is a large area, one interface specification is not sufficient to accomplish a freely exchanging of spatial information. Therefore, there exists several OGC Implementation Specifications.

2.5 WMS

One of the most famous specification of all OGC Specifications is the WMS Implementation Specification. It describes the functionality of an *OGC Web Map Service* (OGC WMS). An OGC WMS can be described as:

"An instance whose behavior follows an interface specification, describing the requests and responses to be handled by a standard map server" [GS04].

2.5.1 A Web Map Server

The functionality of a WMS server is to produce maps. A map is not defined as the geo-referenced data itself; it is a visual representation of the geographical data. When a client requests a map from a WMS server, a number of things can be specified in the request, e.g. what information the client wants to be presented on the map, what part of the earth that the client is interested in (an area), the size and format which is desirable for the retrieved map, the background color and transparency, and the coordinate reference system to be used. If the client asks for more than one map, and the requested maps are of the same part of the world (have the same *Bounding Box*), the same coordinate reference system and have the same output size, all maps can be merged together into one map which is retrieved

by the client. With a transparent background, lower map layers are visible through upper map layers. It does not matter if the different map layers come from the same WMS server or from different WMS servers, as long as the essential parameters are the same, they can be merged into one map picture. The specification enables web map clients to easily choose among different servers, each providing different map data, and build maps with the information that they need [Con04e]. Thus, the WMS Implementation Specification is a great advantage for all web map clients around the world, but it also serves a purpose for the suppliers of spatial data. Since WMS clients can freely choose and access WMS servers through a common interface, the same spatial data does not have to be served from more than one WMS server. This fact saves time and cost for holders of WMS servers and suppliers of spatial data.

2.5.2 OGC WMS operations

The WMS standard specifies three operations, the two first are required and the third is optional. These operations are shortly presented below.

1. Get Capabilities. With this request to a WMS server, the client gets a description of the map layers offered by the server.
2. GetMap. This is the request to retrieve a map layer from a WMS server.
3. GetFeatureInfo. With this operation the client can ask the WMS server for information about special features presented on the map. The operation is optional.

To send a request to a WMS server, the only thing needed is a web browser. By specifying the request in the form of a Uniform Resource Locator (URL), the server will respond. How the URL should be constructed depends on the chosen request. Common for all URLs is that they require a version number and a parameter telling the server which of the request types is chosen. With the GetMap request, the client is allowed to specify a number of additional parameters in the URL request, e.g. what information the client wants the map to show (i.e. map layers are specified) and what part of the world the map shall visualize. If the client wants to retrieve information about a certain feature shown on the map and the GetFeatureInfo request is specified, the client needs to specify in the URL, the map that is being queried and also the location on the map, which is of interest [Con04e].

2.5.3 Example

First a client may want to get information about what kind of map data the WMS server can serve. This is done by sending a GetCapabilities request to the server:

http://hostname/cgi-bin/mapserv?map=mapfile.map&VERSION=1.1.1&
REQUEST=GetCapabilities

The answer from the WMS server is a machine-readable description of its capabilities. With this information the client knows what map layers the server can deliver, and can then construct a valid query. In the following request, the client wants to have cities presented on the map, within a specified area:

http://hostname/cgi-bin/mapserv?map=mapfile.map&VERSION=1.1.1&
REQUEST=GetMap&LAYERS=cities&BBOX=17,58,19,60

In the next query, the client wants to have both cities and airports shown on the map (the area is the same as in the earlier question):

http://hostname/cgi-bin/mapserv?map=mapfile.map&REQUEST=GetMap&
VERSION=1.1.1&LAYERS=cities,airports&BBOX=17,58,19,60

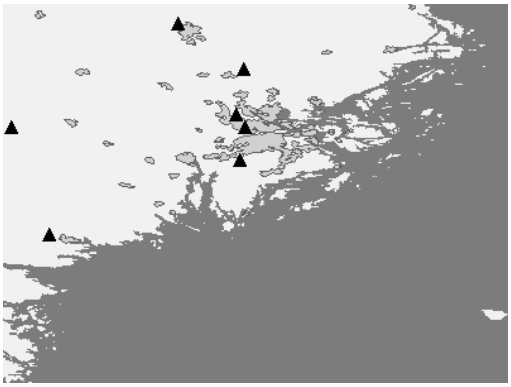

Figure 1: WMS query example

The layer *sea* is drawn by default in the color gray, and with a green background color, the layer *land* is created.

The URL is not formally correct according to the WMS Implementation Specification 1.1.1, some additional attributes have to be specified. But this report will not investigate WMS in detail, and these attributes are therefore not added to the URL, since this would require further explanation.

2.6 WFS

The OGC WFS Implementation Specification is a specification for how to handle *geographic features* (section 2.6.3) on the Internet. In the interaction between a client and a WMS server, static images are sent over the web. In the client interaction with a WFS server, the geo-spatial data exchanged is encoded in Geography Markup Language, GML (section 2.6.2).

The OGC WFS Implementation Specification describes the functionality of an OGC Web Feature Service (OGC WFS). An OGC WFS can be described as:

”An instance whose behavior follows an interface specification describing the requests and responses to be handled by a standard feature server” [GS04].

2.6.1 A Web Feature Server

The request to a WFS server is a query (or transformation operation) for one or more features that can be delivered by the server. The transport protocol used is HTTP. When the server receives the request, it executes the request and sends back the requested information to the client. Two of the requirements that exist for a WFS are that all interfaces must be defined in XML, and that the data store (in which the geographic features are stored) should be hidden from the client applications, i.e. the client should not be able to access the features by other means than through the WFS interface. One additional requirement for a WFS is that the features within the interface must be expressed with GML [Con04d].

2.6.2 Geography Markup Language

The Geography Markup Language is the schema used for modeling, transporting and storaging of spatial information. It is an XML grammar written in XML Schema. The language provides objects that make it possible to describe the geography surrounding us, like features, topology, units of measure, time, coordinate reference systems etc.[Con04b].

2.6.3 A WFS Feature

To give an explanation of what a geographic feature really is, can be done by describing it as an abstraction of a phenomenon existing in the real world. If it has a location relative to the earth, it is a geographic feature. Furthermore, a feature has a state. This state is defined by a set of properties. Each property has a name, type and value.

For example, a city can be a geographic feature and have the following three properties: name, population and location. Each of these properties has a name (e.g. city_name, city_pop, city_location), a type (e.g. a string, a number and a point) and a value (e.g. Stockholm, 1.000.000, and (17, 60)).

The type definition of a feature determines the number of properties, their names and types. Geographic features can have a property that might have a geometry value. These features are called geographic features with geometry [Con04b].

The geometries of geographic features are restricted to *simple geometries*, which means that their coordinates are two-dimensional. Points, line strings and polygons represent the geometries in a 2-dimensional reference system. Geometries may also be collections of other geometries, either collections of the same geometry type (homogeneous geometry collections), or collections of different geometry types (heterogeneous geometry collections) [Con04d].

2.6.4 OGC WFS operations

There are two types of WFS, the Basic WFS and the Transaction WFS. The basic WFS is a read-only WFS. Clients can therefore not modify any features served by a basic WFS. The basic WFS must support the following three operations:

1. GetCapabilities. When a WFS receives this request, it answers with a description of its capabilities, i.e. what feature types it serves and the operations supported on each feature type.
2. DescribeFeatureType. The WFS can be asked to describe the structure of any of the feature types it serves.
3. GetFeature. With this request, the WFS is asked to deliver features. The client should be able to constrain the query both spatially (i.e. ask for features which are placed in a specific area) and non-spatially. The client should also be able to specify what feature properties he/she is interested in receiving from the WFS.

With a Transaction WFS it is possible for the client to perform operations that modify features, i.e. the client can create, update and delete features that the WFS serves. The transactional WFS must support the three basic operations described above, and the additional Transaction operation. One more operation exists, the LockFeature operation. This operation is optional for a transactional WFS to implement. The LockFeature operation ensures that serializable transactions are supported by processing a lock request on a feature

while a transaction on that feature is being processed [Con04d].

2.6.5 Example

In this example scenario of a client interaction with a WFS server, the HTTP method post is used.

To be able to know what feature types the WFS serves, the client generates a GetCapabilities request to the server:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GetCapabilities version="1.0.0" service="WFS"/>
```

In the answer from the WFS server, the client receives information about what feature types the server handles and what operations the server supports on each feature type. To get a description from the server of the structure of a feature type (in this example the feature type called *cities* is chosen), the client generates a DescribeFeatureType request:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- get the XML schema for the feature type "cities" -->
<DescribeFeatureType outputFormat="XMLSCHEMA">
       <TypeName>cities</TypeName>
</DescribeFeatureType>
```

The answer from the server is a description of the feature type *cities* and the properties of that feature type. From the answer, the client can read what information is possible to receive about a feature of this feature type, and can then perform a GetFeature request to the server with feature type specified and possibly additional constraints specified (e.g. spatial constraints or non-spatial constraint).

A GetFeature request without additional constraints:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<wfs:GetFeature outputFormat="GML2" xmlns:gml="http://www.opengis.net/gml"
 xmlns:wfs="http://www.opengis.net/wfs" xmlns:ogc="http://www.opengis.net/ogc">
<!--
get all features of feature type cities and all their properties
-->
<wfs:Query typeName="cities">
</wfs:Query>
</wfs:GetFeature>
```

A GetFeature request with the constraint that the name of the received cities must contain the string ”Stockholm”:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<wfs:GetFeature outputFormat="GML2" xmlns:gml="http://www.opengis.net/gml"
 xmlns:wfs="http://www.opengis.net/wfs" xmlns:ogc="http://www.opengis.net/ogc">
<!--
get all features of feature type cities and all their properties,
where the names of the cities contain the string "Stockholm"
-->
<wfs:Query typeName="cities">
```

```
<ogc:Filter>
<ogc:PropertyIsLike wildCard="*" singleChar="#" escape="!">
<ogc:PropertyName>/cities/Name</ogc:PropertyName>
<ogc:Literal>*Stockholm*</ogc:Literal>
</ogc:PropertyIsLike>
</ogc:Filter>
</wfs:Query>
</wfs:GetFeature>
```

In this way, the client can communicate with the WFS server and ask the server detailed questions.

3 WFS design analysis

A Web Feature Service is an XML Web Service. This means that it is a web based application that communicates with other applications on the web by using the web technology standards XML, SOAP, WSDL and HTTP [ED02]. The analysis of the design of WFS will therefore partially be an analysis of general design issues regarding XML Web Services, but also an analysis of some WFS specific design issues.

3.1 Security in WFS - XML and SOAP security

XML Web Services are extremely successful, but have a mayor drawback: the security aspect. There are two big concerns about the security in XML Web Services:

- Authorization - restricting access to XML Web Services.
- Confidentiality - protecting the XML messages which are exchanged over the web.

To secure Web Services, the two standard techniques *Secure Sockets Layer* and *firewall based rules* are frequently used. To be able to use firewall based rules, the clients that want to access the XML Web Service must be known in advance since the IP and possibly mac addresses of the clients must be specified to the service. Because of this, the method is most used in private networks. On the Internet, the clients are unknown to the services, and in this situation, there are a number of Secure Socket Layer (SSL) implementations to enhance security. The SSL implementations encrypt and decrypt the messages exchanged over the web, and not only are the messages secured from being read while they are on their way, SSL also verifies that the messages come from the sender that they should come from, and not from someone else. This is security on transport level, but although the firewall and SSL implementations are effective, they are not considered to be enough for ensuring security in Web Services. Especially not regarding to the communication protocol used in XML Web Services, called Simple Object Access Protocol (SOAP) [ED02]. To secure XML Web Services, the messages have to be protected along their whole path, from client to service provider. To ensure a secure path for the XML messages, message-level security is required, and achieved by adding security mechanisms to the SOAP messages. With digital signatures and/or encryption, specific parts of an XML/SOAP message can be protected [SM04].

Since 1999, an encryption standard has been proposed for encrypting the information (data and tags) within an XML document. This allows senders of XML documents to encrypt specific parts of the document. Different parts of the XML document can be encrypted with different keys. This makes it possible to send the same XML document to multiple users, with the users only being able to decrypt the information relevant to them. The standard also makes it possible to create and verify digital signatures. The signature can be for an entire XML message, or for just a part of it. The encryption and digital signatures provides security on application level. Unfortunately, the techniques of digital signatures and encryption were not constructed specifically for XML-based protocols and the SOAP structure, but for generic well-formed XML documents. The use of the SOAP protocol has therefore further security aspects [ED02].

The SOAP standard defines the format of XML messages. Thus, a SOAP message is a well-formatted XML fragment, enclosed in SOAP elements. The SOAP specification also defines how a HTTP message, containing a SOAP message, looks like. Almost all XML Web Services use HTTP as the transport protocol, but since there are those that do not, this part of the SOAP specification is optional. SOAP has been implemented on many

different platforms and can therefore link different systems together. A reason why SOAP has been so successful is that it is a small protocol which is simple to implement. When SOAP came, it was assumed that the security aspect was handled on the transport layer, i.e. that it was a question for HTTP to provide security for the XML Web Service. SOAP was seen as a HTTP-based protocol, and the reasoning was that it was enough to have security in the level below. But SOAP moved from being a HTTP-based protocol to the general protocol that it is now, and it is not safe to rely on the security provided by HTTP any more. The *WS-Security Specification* [Mic02] adds security features to XML Web Services, by providing a complete encryption system [Mic04], and is intended to realize message level security for SOAP message exchange. Mechanisms for protecting SOAP messages are digital signature and encryption. The syntax for these mechanisms is based on the W3C Recommendations for XML signature and XML encryption. Further functions are also specified in the WS-Security Specification, such as format of message timestamps, how to send passwords securely etc. [SM04]. This report will not dig deeper into the bottomless area of securing XML Web Services.

3.2 Performance of WFS

When it comes to performance aspects of the WFS design, general performance aspects of XML Web Services have to be considered. This includes e.g. web caching and how locks are used. Web caching can improve performance if appropriate data is cached. How locks are used are also important for the performance of an XML Web Service. Locks should be used as efficient as possible, which means that they should only be used when it is needed, and only for as long as required to avoid data corruption during updates [CB03].

Beside these two aspects, one big performance aspect is the performance of XML/GML. Large XML/GML messages are being transferred between the service provider and the client. This is of critical concern to the performance. Two more design issues with performance aspects are discussed in this section: the lack of the two mechanisms Push and Multicast, and data replication.

3.2.1 Performance of XML/GML

GML is encoded in plain text XML and have three performance problems. These problems are being addressed in the OGC document *Binary-XML Encoding Specification* [Con04f]. This is not a specification nor a standard yet, but simply a discussion of the problems and possible solutions. Although it is neither a specification nor a standard yet, the ideas are presented in brief in this report, since they are relevant and because it is possible that the document will become an OGC Implementation Specification in the future.

First of all, an XML document is very bulky, the structure of the text is extremely redundant. As a result, the transfer of an XML document over the web is slow. By using general compression methods like GZIP the bulkiness can be reduced. Since XML is so redundant, with a big part of the document consisting of repeated blocks of text, it compresses a lot.

The second performance problem concerns the lexical scanning of XML. The lexical scanning means the process of transforming free text into tokens, so that the XML text can be processed internally. This process is very costly and is therefore a problem to performance.

The third performance problem is also a costly conversion, of text-encoded numerical coordinate and observation values.

Although the bulkiness of XML is reduced by using general compression methods, this does not solve the two last performance problems. But by using a binary encoding method for representing the XML data, and by making a small change to the method for specifying the values of coordinates, the scanning cost would be less and further space reductions would be achieved.

In brief, the *Binary encoding method* parses the XML stream into a sequence of tokens. Element openings and closings, empty element, special tags, comments etc. are all represented with a token. For making the processing effective, the head of every byte sequence contains the byte length of that structure. For space and time efficiency, element and attribute names are represented with integer values, and stored in a global table for strings. Furthermore, the way GML represents coordinate values can be made more efficient by replacing it by a numeric representation. This representation of course applies to all values representing numbers and arrays of numbers. Now GML uses special character-delimited sequences of textual numbers or hierarchical structures for the representation of coordinate values. This representation should be abandoned (or augmented) and replaced by the XML-Schema list representation of double values. By using the array of IEEE double floating-point numbers, this type of coordinate values can be processed very efficiently. Since the array can be read straight into an array in memory, it can be used directly by modern processors and requires at most a swapping of the byte order of the values, no parsing or conversion of coordinate values is required. To represent coordinate values this way would make GML as efficient as other formats used for feature encoding, e.g. the ESRI Shapefile format.

With the backward-compatibility mechanism provided, the binary blocks can be translated into the textual representation again. Although binary XML is very much smaller and less redundant than the textual encoding, it can be compressed with general compression methods and decrease in size even more. Binary encoding and textual encoding are equivalent, and there is no loss of information in the transformation them between. Further, binary encoding does not only apply to GML documents, but to all XML documents. Although, XML documents in general are not redundant and bulky to that grade that makes it worth to use binary encoding on them. The binary XML representation does not have to be much smaller than the textual representation for XML documents in general. But when dealing with GML documents, it is crucial to use binary XML [Con04f].

3.2.2 Push and Multicast

When a client wants to receive information from a WFS, it has to send a request to the WFS server. This means that the protocol in use is *Pull*. This is quite inefficient in some situations. Picture a scenario where a client is interested in following the movements of a hostile ship. The movements of the ship is not known, it can stop and start whenever, and change direction to wherever. To be sure not to loose track of the ship, the WFS client has to frequently ask the WFS for the position of the ship. Even if the ship stands still for a long time, the WFS has to be asked for the position, since it can start moving in a second. It is easy to see that having this situation would result in many unnecessary requests and responses between the WFS client and WFS server, when the Pull protocol is used. Therefore, this protocol can in some situations result in a heavy burden on the network. In the scenario described above, it would be better if WFS was designed to use the *Push* protocol. In that case, the server would recognize a change of position of the ship, and then deliver the new position to the client. This would eliminate unnecessary requests totally. The implementation of the Push protocol should be added to the WFS design. Another extension to the WFS Specification should be *Multicast*. In the scenario presented above, there could be multiple users wanting the same information from the WFS. If the

service could use Multicast to deliver information to all clients, instead of using *Singlecast* to each one of them, performance would be increased.

3.2.3 Data replication

Before the WFS Specification was introduced, the same feature information had to be served by multiple Web Feature Services since there was no open interface to follow. Now, instead of having the same feature information served by many servers, it is enough if the feature information is served by one OGC WFS since all client implementing the OGC WFS Specification can access this single WFS. But this situation could, theoretically, create a lot of pressure on a service, now that many more clients may want to access it. WFS servers providing feature data of interest to many clients could probably get a lot of problems with a highly loaded network, and thereby get performance problems. In this way of seeing it, performance will suffer from the interoperability, which the WFS Specification has created. The solution to the problem may be to use data replication. This would increase the performance of a highly loaded WFS. To use data replication would not only increase performance, but also create a backup service. In case something happens with one of the services, the second can take over and clients may only notice a decrease in performance for a while. Theoretically thinking, the scenario of a highly loaded WFS would not be an unusual situation, and careful reflections are therefore required from the system architects and system designers in order to create a well-functional distributed system of geographical data.

3.3 Scalability and reliability of WFS

A rising question is how the WFS works in practice, when it serves of a lot of information to many users at the same time.

By doing theoretical calculations, it can be shown that the WFS traffic do not have to cause performance problems. Although the XML/GML requests are bulky and redundant, they are often small. The two situations, where the WFS traffic could result in performance problem and a heavy loaded network, are when there are many users on the same network, performing frequent requests to the server.

This has to be investigated carefully in practice. In the implemented demo system, I found that requesting the WFS for a large number of features did not seem to have any affect on performance. But since proper tests with many clients performing simultaneous requests to the WFS for information were not done, nothing could really be said yet about performance of WFS in reality. What *can* be said, is that the scalability of WFS is strongly related to the performance of WFS, since expanding a WFS service may be difficult in a heavy loaded network.

3.3.1 The *MaxFeatures* attribute

The specification offers a feature which enhances the reliability of WFS by making it possible to specify maximum number of features wanted from a GetFeature request. The attribute is called *MaxFeatures*. It is an optional attribute. If it is set to a number, and the number is reached, the result is truncated right away [Con04d]. Often, a client does not know how big an answer from a GetFeature request might be, the request could result in the retrieval of ten feature instances or ten thousand feature instances. With this parameter set in the GetFeature request, the situation in which a GetFeature request results in a heavy

load on the network can be avoided.

3.3.2 Simultaneous transactions

Since the WFS Implementation Specification defines the optional *Transaction* operation, allowing clients to insert, delete and update feature instances, one thing that comes in mind is what happens when two or more clients want to access the same feature instance/ feature instances at the same time. A rising question is how the WFS Specification handles transactions occurring at the same time on the same features, i.e. how the WFS Specification handles simultaneous transactions.

The WFS Implementation Specification does not address the topic of how geographic features are persistently stored. The responsibility of making sure that changes to the spatial data is consistent lies upon the WFS and the data storage system. Concurrent transaction semantics are supported by many systems, e.g. relational database systems based on SQL, and therefore the WFS Specification defines operations allowing a WFS to support standard concurrent transactions.

The operation defined for supporting serializable transactions is the *LockFeature* operation. It allows the WFS to put a lock on one or more instances of a feature type, during the time a transaction operation is being processed on the feature instance/feature instances. The LockFeature operation is optional for a transactional WFS to implement. Because of possible network latency, the WFS may lock features for a longer time than commercial database locks would do. The LockFeature operation is therefore said to be a long term feature lock for the assurance of consistency . The LockFeature operation may have the attribute *expiry* set, which tells the WFS how long it should hold the lock on a feature instance, if a transaction that releases the lock never generates. When the time is up, the WFS releases the lock [Con04d].

This means that the WFS Specification provides a solution to the situation where two or more clients are interfering with the same feature instance at the same time, by allowing a transactional WFS, connected to data storage systems supporting concurrent transaction semantics, to implement the LockFeature operation and by this have the problem eliminated. One thing that one could (and should) reflect over, is how long such a lock on a feature instance lasts with various network latencies. Although a good protection is given for avoiding errors occurring in simultaneous transactions, it may not work in practice, depending on the latency of the network.

3.4 Additional aspects

Since WFS is a new technique, not many implementations using the technique exist today, and the existing ones are quite new and have not been running for a long time. Therefore, many negative aspects regarding the functionality of WFS have not turned up yet. The information that can be read today about WFS, is general information about the specification and scenarios presenting what can be achieved with the technique (i.e. what problems the technique solves). This information is very positive to the WFS technique, and perhaps does not give a balanced picture of the situation. This is important to be aware of when investigating a new technique such as WFS. In a few years, studies showing some negative aspect of WFS might turn up.

Another important design issue, is how *access control* is handled in the OGC WFS Implementation Specification. Being able to restrict users, or groups of users, to a appropriate

security level is desirable. Appropriate security level can be a feature, a set of features, or feature types. Unfortunately, nothing that indicates that this topic is included in the OGC WFS Implementation Specification, has been found.

4 Framework investigation

4.1 OGC framework

Several organizations and companies have created software products implementing various OGC Specifications. These products can be found on the OGC web site [Con04g]. On this site, information about the products, e.g. which of the OGC Specifications they implement, version number etc., can also be found. The frameworks differ in many ways from each other, not only in which OGC Specifications they implement, but also in their behavior. Some products can only act as server, other clients, and some of them as both. There are both commercial products and products developed as open source projects. There are products developed for different platforms, e.g. Unix/Linux systems, Windows etc. Therefore, many factors have to be considered when creating a distributed system for the exchange of geo-spatial data, and the choice of framework/product has to be based on the structure, purpose and intentions of the particular system that is to be constructed.

4.2 The demo system

Since the demo system is described in detail later in the report (section 5 and 6), it will only be described perspicuous here, in order to give the reader enough information to be able to follow the discussion about the most suitable framework/frameworks for the servers in the demo system.

4.2.1 System architecture

The first level of the demo system, Demo System level1, consists of three servers. One server delivering feature data (server1), one server delivering map data (server2), and one server, that will merge information from the two previous servers and deliver maps to the client (server3). The client is, in this level of the demo system, an ordinary web browser. The second level of the demo system has no further requirements on the servers.

4.2.2 System requirements

The three servers in Demo System level1 have different requirements on the frameworks.

- Server1 is a WFS server. The framework chosen for this server must implement the OGC WFS Implementation Specification and must be able to act a WFS server.
- Server2 is a WMS server. This type of server already exists at SaabTech. Therefore, no framework has to be chosen for this server in the demo system. The framework to use implements the OGC WMS Implementation Specification and can act WMS server.
- Server3 has three roles: WFS client (to be able to receive features from server1), WMS client (to be able to receive maps from server2) and WMS server (to be able to merge the received information and deliver it as maps to the client). The framework chosen for this server must therefore implement both the OGC WFS Implementation Specification and the OGC WMS Implementation Specification and must be able to act WFS client, WMS client and WMS server.

The geographical data to be served by the WFS server is stored in a PostGIS database (section 5.3.1). An additional requirement for the framework chosen for the WFS server is

therefore that it is possible to connect to a PostGIS database.

To sum up, the following framework/frameworks are needed for the implementation of Demo System level1:

- For the WFS server (server1), a framework implementing the OGC WFS Implementation Specification. The framework has to be able to connect to a PostGIS database and has to be able to act WFS server.
- For the WMS server/WMS client/WFS client (server3), a framework implementing the OGC WMS Implementation Specification and OGC WFS Implementation Specification. The framework must be able to act WFS client, WMS client and WMS server.

4.3 Investigation criteria

The architecture of the demo system together with the system requirements make certain key-topics interesting to examine for the different frameworks. The analysis of suitable framework for the WFS server and the WMS server/WMS client/WFS client are done with these key-topics as a base. The key-topics are listed below.

1. Implemented OGC Specifications - what OGC Specification does the framework implement?
2. Possible roles in a distributed system (server/client/both) - what roles can the framework have in a distributed system?
3. Platforms - on what platforms can the framework run?
4. Data source connections - what data sources may be used with the framework?
5. Available documentation and instructions - how much information/instructions are available for the framework?
6. Price tag: is the framework a commercial product or open source project?
7. Development status - is the framework continuously being developed or not?

Answering these questions will give a good basis for the framework investigation.

4.4 Investigated frameworks

Since there exists many frameworks implementing the OGC Specifications, a first selection of which frameworks to investigate further has to be made. This selection is basically made from the impression the frameworks give when studying them in brief. The choice is made to investigate three open source products: MapServer (section 4.4.1), The GeoServer Project (section 4.4.3) and Deegree (section 4.4.2). MapServer is already used at SaabTech for WMS, and is the framework which will be used for the WMS server in the demo system. It is interesting to investigate MapServer further, to see if it is suitable for other servers in the demo system as well. The GeoServer Project and Deegree are completely new for SaabTech. In the following three sections, a short description of each of the framework is given, with the established criteria illuminated specially.

4.4.1 MapServer

The development of MapServer started at the University of Minnesota, with a project called ForNet. It was a project where NASA and the Minnesota Department of Natural Resources participated. MapServer is a development environment for creating Internet applications

used in situations where geographical data is to be handled. It is an open source project and has the advantage that it can run on Linux/Apache systems, which many commercial systems can not. MapServer can also run on Windows platforms like NT/Win98/Win95. MapServer is built upon other open source projects like Pearl etc. [oM04c].

4.4.2 Deegree

The development of Deegree started with a research initiative, founded at the Department of Geography, Bonn University. The main purpose of the research initiative was to work on real-world problems, geographical information systems and the Internet. Deegree is an open source product for creating GIS applications and is based on Java. The framework offers valuable parts for the builder of a distributed system of geographical data. The fact that all parts of Deegree implement the OGC Specifications makes the Deegree architecture extremely interoperable. Since it is a Java framework, Deegree is platform independent, which is also proven by the numbers of installations currently running on e.g. Windows (NT/2000), Linux, Mac OS X and Solaris. Furthermore, Deegree offers interfaces (read and write) to a wide selection of data sources [lat04b].

4.4.3 The GeoServer Project

Like Deegree, The GeoServer Project is a Java based framework. It implements the OGC WFS Implementation Specification. The framework is a fully transactional WFS, which means that the additional operations, allowing users to modify the features, are offered. Although both Oracle and Shapefile data formats are supported by the GeoServer Project, users are strongly recommended to use PostGIS for storage of the spatial data. To support several data storage formats is one of the primary goals for the GeoServer Project, but this goal is not achieved yet since PostGIS is the only data source on which good support exists [Pro04].

4.5 Comparison

The three different frameworks are compared from the investigation criteria, in the same order as the criteria are presented in section 4.3.

According to the OGC list of registered products, MapServer only implements the WMS Specification. Although it is said in the MapServer *UNIX Compilation and Installation HOWTO*, that ”New with version 3.7 of MapServer is support for the OpenGIS Consortium’s Web Feature Service standard.” [oM04d], I have not found any information saying that the OGC WFS Specification is actually implemented by MapServer. Also, there are several limitation with the MapServer WFS server. These limitations are listed in the MapServer *WFS Server HOWTO* documentation, available on the MapServer web site. One of the limitations is that the GetFeature request does not validate against the XML Schema [oM04e]. This limitation, together with the other limitations of the MapServer WFS server, makes me not consider it as a framework suitable for the implementation of a WFS. Deegree implements both the OGC WMS and WFS Specifications, along with a number of other OGC Specifications [Con04g]. According to The GeoServer project home page, GeoServer implements the OGC WFS Specification, although the project is not registered on the OGC list of products implementing OGC Specifications. For the WFS server, MapServer is not an alternative, and the choice stands between Deegree and The GeoServer Project.

MapServer is possible to configure as WMS server/client and WFS server/client according to the MapServer home page, although one should keep in mind that the WFS part of MapServer is not reliable. Deegree is possible to configure as WMS server/client and WFS server/client. For The GeoServer Project, there is a WFS server and a WFS client available.

MapServer can be compiled on many platforms and operating systems [oM04d], but since almost all documentation papers and installation instructions are for Unix users, it seems to be best suited for installation on Unix/Linux. Deegree is platform independent [lat04b] and should work fine on both Windows and Unix. For The GeoServer Project, it is assumed that it is built on a Unix system, documentation and examples are mainly for Unix users. But since The GeoServer Project is a Java framework, it should work on Windows too, and users are encouraged to test that alternative and send the documentation to The GeoServer Project [Geo04].

MapServer can be connected to, and receive data from, various data sources, e.g. PostGIS and Oracle Spatial [oM04d]. Deegree is also possible to connect to a number of data sources. The *Deegree-WFS-Configuration Description* describes the connection to Oracle Spatial, but interfaces for PostgreSQL/PostGIS are available [lat04b]. For The GeoServer Project, support exists for Oracle Spatial and Shapefile data formats, but it is recommended to use PostGIS, since this is the only data source for which good support exists [Geo04].

MapServer provides documentation for the construction of a WMS server/client and a WFS server/client. The documentation on WMS is detailed while the documentation on WFS is quite poor. For Deegree, detailed documentation exist for both WMS and WFS. The GeoServer Project has detailed documentation on installing the GeoServer framework, while the configuration instructions are a bit thin.

All three frameworks are open source projects.

Both in Deegree and in The GeoServer Project, there is a lot of activity going on, which is also shown on their web sites. Updates are also coming from MapServer. All three frameworks have new versions out from October 2003 or later, which shows the speed of the development in all three projects.

4.6 Result

Since nothing is found that proves that MapServer implements the OGC WFS Specification (and because the MapServer WFS has many limitations), MapServer is no alternative for the WFS server in the demo system. Since MapServer is already used for WMS at SaabTech, they are interested in knowing if the framework could also act WMS client to other WMS servers, and possibly even act WFS client to other WFS servers. Because of this, MapServer is chosen for the installation of the WMS server/WMS client/WFS client, although it is uncertain that it can act WFS client.

For the WFS server in the demo system, the choice stands between Deegree and The GeoServer Project. From the comparison, the conclusions are that both frameworks should work fine, since they both implement the OGC WFS Specification and can both be connected to a PostGIS database. Deegree is chosen over the GeoServer project to be used for the implementation of the WFS server. This choice is made since Deegree seems to be far developed, many OGC Specifications are implemented, good documentation and demos are available, updates come regularly etc. Deegree gives an impression of being a framework with a lot of flexibility and possibilities.

5 Implementation level1

The implementation consists of building a demo system in which a user can navigate and extend geographic data, based on his/her needs. This requires the implementation of several servers, each providing different data. The main task in the implementation is to develop a WFS interface on top of an Object Relational Database Management System (ORDMS). Since servers providing map data already exist at SaabTech, a special interest exists in investigate the technique of WFS and how SaabTech can benefit from integrating this technique in their future systems.

The demo system is divided into two levels, Demo System level1, and Demo System level2.

The first thing to accomplish in Demo System level1, is to create a WFS, which will serve feature data stored in a PostGIS database. When this task is accomplished, feature data from the WFS, and map data received from an already existing WMS server, will be merged into a third server, with mission to deliver the merged information, in map format, to a client.

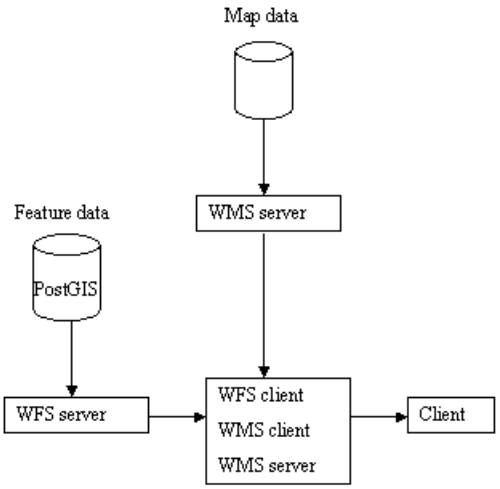

Figure 2: Overview Demo System level1

The data to be visualized through the WFS server are stored tracks from existing command and control applications within SaabTech.

5.1 Virtual experimental environment

Since the demo system required several servers, possibly running on different platforms, virtual machines were used for the implementation. By using a the program *vmware* [VMw04] for creating virtual computers, the computers can be configured to have the qualities they need, e.g. the amount of memory, number of hard disks, network cards, serial ports etc. can be configured. To use virtual computers for the implementation has several advantages. First, it is more practical to use only one physical computer, than several, because of security reasons. The process to get a computer connected to the Intranet may take time. It is therefore better to use a computer already connected to the appropriate net (i.e. the net of appropriate security level). I was assigned a computer in one of

the experimental computer halls. Another advantage of using virtual computers is that if something would go wrong and a re-installation is required, you only have to delete the virtual computer and create a new one. Furthermore, virtual computers are easy to backup by just making a clone of the computer when it is in a state worth saving. If the virtual machine would crash, just delete the broken machine and ”install” a new one.

5.2 Goal for SaabTech with Demo System level1

The main interest, which SaabTech has in achieving Demo System level1, is the implementation of the WFS server. For SaabTech it is also interesting to find out if it is possible to cascade Web Map Services and Web Feature Services. To have a cascading WMS would mean that clients could ask a WMS for map layers, which the WMS does not serve by itself, but is able to receive from other WMS. The client would thereby not need to keep track of several WMS servers; it only has to be aware of one. The client simply asks the WMS for map layers and the WMS delivers the map layers to the client. If the information comes from the WMS server itself, or from a remote WMS, is not important to the client. Having a cascading WMS would give an abstraction level towards the client and make it easy to change the system infrastructure behind the WMS server, through which the client has connection. Without the knowledge of the client, WMS and WFS servers can be added or removed. In the same way, WFS servers are possible to cascade.

5.3 Feature storage

The features to be served by the WFS in the demo system are stored in a PostGIS database. The database is installed on a virtual machine with the FreeBSD operating system. PostGIS is used since it is a free, open source database, which handles spatial constraints well.

5.3.1 PostGIS

PostGIS is developed as a spatial database technology research project. It is an extension to the PostgreSQL object-relational database system and allows its user to store GIS objects. PostGIS follows the OGC *Simple Features Specification for SQL* and has the intention of providing full OpenGIS support [Pos04]. The OGC Simple Features Specification For SQL defines a standard SQL schema for operations to be applied on simple features. The operations are storage, retrievals, queries and updates [Con04h]. A simple feature is a point, line, polygon, or a combination of these.

5.3.2 Feature types

The WFS handles two feature types. One feature type represents cities and is stored in a PostGIS table called *stad*. The other feature type represents targets, and is stored in a PostGIS table called *targets*. The tables are stored in the PostGIS database called *wfs*.

The target’s type (*category_id* in the database table) is represented by an integer value; 0, 1, 2 or 4, where 0 = air, 1 = land, 2 = sea and 4 = unknown.

The data stored in the database table *stad* is downloaded from Internet. The data stored in the database table *targets* is data from one of the command and control applications within SaabTech. The table holds data describing the state of the sea at a particular moment. Both data sets are converted to PostGIS format.

Table 1: *stad*

Column name	*Type*	*Description*
Gid	integer	unique id number
city_name	character varying	name of the city
gmi_admin	character varying	shortening of country and province name
admin_name	character varying	province
fips_cntry	character varying	shortening of the country name
cntry_name	character varying	country name
status	character varying	status of the city, e.g. capital
pop_rank	integer	population ranking
pop_class	character varying	population class
the_geom	geometry	location of the city

Table 2: *targets*

Column name	*Type*	*Description*
target_id	integer	unique id number used in the database table
targetnumber	character(255)	unique number for each target
timestamp	integer	the timestamp
cog	double precision	course over ground (clockwise from north, rad)
sog	double precision	speed over ground (m/s)
category_id	integer	an id describing the target type
geo_pos	geometry	location of the target

5.4 Web Feature Service

Possible frameworks to use for the WFS are analyzed in section 4, and the choice is to use Deegree for the implementation of the WFS. As mentioned in the framework analysis, the Deegree WFS is a Java framework for the creation of a Web Feature Service and implements the OGC WFS Specification 1.0.0.

Since WFS is the central point of the Master's Thesis project, the implementation of the WFS server will be documented in more detail than the WMS servers.

5.4.1 Installation of Deegree and Tomcat

The implementation of a WFS server in Deegree requires that a demo is installed and configured. This demo has connection to a MS Access database. For the implementation of the WFS server, a virtual computer is created with the operating system Windows XP.

The web interface of the Deegree WFS is a Java servlet. A servlet can be described as an applet that runs on the server side [Sun04b]. Since most web servers support the Java servlet technology, Deegree users are not limited to only one web server. The Apache Tomcat servlet engine is used in this demo system, because it is an open source project, and its integration with the Deegree framework is described in the *Deegree-WFS-Configuration Description* [lat04a]. The version of Apache Tomcat used in the demo system is Apache Tomcat 4.1.27.

In the configuration of Apache Tomcat, it has to be specified where Tomcat can find the root directory of the WFS installation. This can be done by using the Tomcat manager web

interface or by configuring the element <Context path> in the file *server.xml* in the Tomcat installation. The context path is set to */deegreewfs*.

5.4.2 Configuration of Deegree

Configuring the Deegree WFS requires that a number of XML files are configured.

First of all, the file *web.xml*, has to be configured. Tomcat uses this file to get information about what servlets there are for the application, the names of the servlets, parameters and access restrictions [lat04a].

Name web.xml
Location C:/deegreewfs/WEB_INF
Purpose give the Tomcat information about what servlets belong to the application.
Description This file contains the element <init-param>. This is used in the initialization of the servlet. The name of the parameter is *capabilities* and is compelling, but the name and position of the file that the parameter refers to, is up to the implementer to decide. This file is called *sample_wfs_capabilities.xml* in the WFS in the demo system.

```
...
<init-param>
   <param-name>capabilities</param-name>
   <param-value>file:///C:/deegreewfs/WEB-INF/xml/sample_wfs_capabilities.xml
   </param-value>
</init-param>
...
```

The point where the servlet is accessible also has to be specified. This is done with the element <servlet-mapping>. By doing the following mapping:

```
...
<servlet-mapping>
      <servlet-name>deegreewfs</servlet-name>
      <url-pattern>/wfs</url-pattern>
</servlet-mapping>
...
```

The Deegree WFS for the demo system is accessed with the URL:
http://hostname_XP:portnumber/deegreewfs/wfs

After this, the capabilities document has to be configured. In this document, the feature types that the WFS handles are defined.

Name sample_wfs_capabilities.xml
Location C:/deegreewfs/WEB-INF/xml
Purpose Defines the feature types served by the WFS.
Description The element <FeatureTypeList>, lists the feature types served by the WFS. The feature types, in their turn, are defined within the element <FeatureType>. Each feature type is further described in its own configuration document. For each feature type, the Java class responsible for accessing the feature type from its data source (e.g. database) has to be specified. Within the element <ResponsibleClass>, both the configuration document and the Java class are specified. The attribute *className* defines the responsible Java class and the attribute *configURL* defined the path to the configuration document for the feature type.

The WFS in the demo system serves two feature types, called *stad* and *targets*. Their configuration documents are called *config_postGIS_stad.xml* and *config_postGIS_targets.xml*, respectively. The path to these files are specified with the attribute *configURL*. Since the Deegree WFS demo has its feature types stored in an MS Access database, and the feature types to be served by the WFS in the demo system are stored in a PostGIS database, the attribute *className* is changed to *org.deegree_impl.services.wfs.postgis.PostgisDataStore*.

```
...
<WFS_Capabilities version="1.0.0">
...
<FeatureTypeList>
...
<FeatureType>
    <ResponsibleClass
    className="org.deegree_impl.services.wfs.postgis.PostgisDataStore"
    configURL="file:///C:/deegreewfs/WEB-INF/xml/config_postGIS_stad.xml" />
    <Name>stad</Name>
    ...
</FeatureType>
<FeatureType>
    <ResponsibleClass
    className="org.deegree_impl.services.wfs.postgis.PostgisDataStore"
    configURL="file:///C:/deegreewfs/WEB-INF/xml/config_postGIS_targets.xml" />
    <Name>targets</Name>
...
</FeatureType>
</FeatureTypeList>
</WFS_Capabilities>
```

Other information specified in the capabilities document is information about the service, like name and online resource. Further, the request methods for the different operations that the WFS server can perform have to be defined. The two possible methods are the HTTP methods GET and POST. For the Deegree WFS framework, the GetCapabilities request can be performed by using both the GET method and the POST method. Therefore both are defined. The same applies for the DescribeFeatureType request. It is not mandatory according to the OGC WFS Implementation Specification to implement the GET operation for the DescribeFeatureType request, but in Deegree it is implemented. The GetFeature request can only be performed by using the POST method.

```
...
<GetCapabilities>
    <DCPType>
    <HTTP>
    <Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    <Get onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    </HTTP>
    </DCPType>
</GetCapabilities>
<DescribeFeatureType>
...
    <DCPType>
    <HTTP>
    <Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    <Get onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    </HTTP>
    </DCPType>
</DescribeFeatureType>
<GetFeature>
...
    <DCPType>
```

```
    <HTTP>
    <Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    </HTTP>
    </DCPType>
</GetFeature>
...
```

After the capabilities document is configured, the configuration documents for the feature types have to be constructed. These documents can be seen as bridges between the data stores and the WFS. They specify the connections to the data stores and map each property of the feature types so that the properties can be access through the WFS. This means that each field in the data store is mapped with a name that is used when accessing the property through the WFS. Since the WFS server in the demo system serves two feature types, two configuration documents have to be constructed:

Name config_postGIS_stad.xml
Location C:/deegreewfs/WEB-INF/xml
Purpose Link the WFS to the data store that contains the feature type *stad*.
Description The feature type *stad* is represented by the PostGIS table with the same name. The part of the configuration document describing the connection to the database table specifies name of the database table, user-name, password etc.

```
...
<DatastoreConfiguration type="POSTGIS">
<Connection
    name="stad">
    <driver>org.postgresql.Driver</driver>
    <logon>jdbc:postgresql://hostname_db:portnumber/wfs</logon>
    <user>user name</user>
    <password>password</password>
</Connection>
...
```

After the data store is specified, each property of the feature type has to be mapped in order to be accessible through the WFS. This mapping is done with the element <MappingField>. Within this element, the element <Property> exists, which has an attribute *name* and *type*. This specifies the name through which the property of the feature type is accessible through the WFS, and its type in Deegree. The element <DatastoreField> specifies the name and type of the property that is used in the data store, i.e. the name and type of a field in the PostGIS database table.

For example, two of the properties of the feature type *stad* are *id* and *name*. In the PostGIS database table these properties are called *gid* and *city_name*, and are of the types integer and varchar, respectively. With the following configuration, these two properties are accessible through the WFS with the names */stad/ID* and */stad/Name*.

```
...
<FeatureType name="stad">
...
<MappingField>
    <Property name="/stad/ID" type="INTEGER" />
    <DatastoreField name="stad.GID" type="INTEGER" />
</MappingField>
<MappingField>
    <Property name="/stad/Name" type="VARCHAR" />
    <DatastoreField name="stad.CITY_NAME" type="VARCHAR" />
```

```
</MappingField>
...
```

The feature type *stad* also has a property which is of the geometry type POINT in the PostGIS database and has the name *the_geom* in the PostGIS table. The types used in Deegree are the ones defined in the class java.sql.Types [Sun04a], with the additional type named GEOMETRY. The property is mapped with the GEOMETRY type in Deegree, and is accessible through the name */stad/coordinates*.

```
...
<MappingField>
    <Property name="/stad/coordinates" type="GEOMETRY" />
    <DatastoreField name="stad.THE_GEOM" type="GEOMETRY" />
</MappingField>
...
```

According to the deegree-WFS-Configuration Description it should be possible to perform mapping between different data types in future versions of Deegree.

The other feature type served by the WFS is called *targets*:

Name config_postGIS_targets.xml
Location C:/deegreewfs/WEB-INF/xml
Purpose Link the WFS to the data store that contains the feature type *targets*.

Since the configuration of the *targets* feature type is similar to the configuration of the *stad* feature type, it will not be described in this section. It can be found in appendix D.

The last part of the configuration is the construction of two .xsd files, one for each feature type. In the demo system, these files are called *stad.xsd* and *targets.xsd*. The locations of the documents are specified in the configuration document for each feature type. The .xsd file is a schema file, which defines the elements of a feature type specified in the configuration document for that feature type. The .xsd files are used for validating the elements of the feature types.

The .xsd schema files can be found, together with the other Deegree configuration files, in appendix D.

5.4.3 A simple Java client

In order to test the WFS server and get familiar with how the server works, a simple client is written. The client is written in Java and uses the POST method for sending requests to the WFS server.

```
\>java WFSclient request_file.xml
```

If no XML request file is given to the client, the *GetCapabilities* request is sent to the WFS server.

Name WFSclient.java

Purpose	Send a request to the WFS server and print the answer from the WFS server on standard output.
Argument	An XML file with the request to the server.

Test runs with this client can be found in appendix A.

The result is a WFS server with PostGIS connection, shown in figure 3.

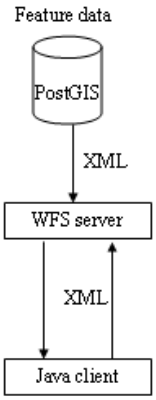

Figure 3: A Web Feature Service with PostGIS connection.

5.5 Web Map Services

The framework for the two WMS servers needed in the demo system is MapServer. The WMS with mission to serve map data is referred to as the *WMS1*, and the WMS server with mission to merge feature data from the WFS server and map data from the WMS1 into maps with features for the clients, is referred to as the *WMS2*. During the compilation of MapServer, it can be chosen what kind of service MapServer is supposed to be, e.g. WMS server/client and/or WFS server/client. The WMS1 is installed on the same machine as the PostGIS database (the FreeBSD machine) and compiled to be a WMS server. Since MapServer requires one configuration file, a copy of this file is taken from an already running MapServer WMS server. The map data that the WMS1 uses is stored in the PostGIS database. For the WMS2, a new virtual machine is created, with Linux (Redhat) operating system. MapServer is installed and compiled to be able to act WMS server, WMS client and WFS client. A copy of the same configuration file taken for the WMS1 server, is also taken for the WMS2 server. This results in two identical MapServer WMS servers, but one of them with the additional possibilities to act WFS and WMS client.

The WMS1 only needs a few configurations to be able to serve maps in Demo System level1, but the WMS2 needs some additional configurations in order to be able to receive maps from the WMS1 and feature data from the WFS, and to merge this information into maps for the clients.

5.5.1 Configuration of MapServer

The MapServer runs as a CGI application on the web server [oM04a]. The configuration file for the MapServer framework is called *Mapfile*. The information in this file defines the data that the MapServer can serve. The name of the Mapfile can be selected freely, but normally they have the extension *.map*.

The Mapfile for the WMS1 MapServer installation is called *wms1_sfa.map*.

Name	wms1_sfa.map
Location	/usr/local/apache/cgi-bin/
Purpose	configuration file for the WMS1.

Description
The Mapfile is configured by editing several objects. The *WEB* object defines the web interface, and has a keyword called *METADATA*, which makes it possible to define arbitrary data such as title and online resource.

```
...
WEB
  METADATA
    WMS_TITLE "wms1_sfa"
    ...
    WMS_ONLINERESOURCE "http://hostname_FreeBSD/cgi-bin/mapserv?"
    ...
  END
END
...
```

One of the most important object is the *Layer* object. This object defines the layers that the WMS can serve. In each *Layer* object, there are several other objects describing the Layer, e.g. the name of the layer and the connection to the data store where the data describing the layer is stored. All layers that a WMS server can serve have to be defined in the Mapfile. The order in which they are defined in the map file decides the order in which they are drawn on the map, i.e. the first layer defined in the map file is the first to be drawn and the others are drawn on top. Since the map data to be served by the WMS1 is stored in a PostGIS database, the *CONNECTIONTYPE* is set to *postgis* and the parameters host, port, database name and user name, for the connection to the PostGIS database, are defined with the *CONNECTION* keyword. One of the layers served by the WMS1 is a layer representing lakes.

```
...
LAYER
  METADATA
      WMS_TITLE "Lakes"
      ...
  END
  NAME "Lakes"
  CONNECTIONTYPE postgis
  CONNECTION "user=username dbname=name_of_db host=hostname_db port=portnumber"
  TYPE POLYGON
  CLASS
```

```
    COLOR 225 225 227
    OUTLINECOLOR 100 100 100
  END # end of class object
END # end of layer object
```

The layer is accessed by its name (*lakes*).

The client can receive a map with this layer by giving the following URL to a web browser:
http://hostname_FreeBSD/cgi-bin/mapserv?map=wms1_sfa.map&VERSION=1.1.1&
REQUEST=GetMap&LAYERS=lakes&BBOX=10,57,19,60

Another layer served by the WMS1 is *cities*. This layer is a remote layer for the WMS2 to access, i.e. a client can ask the WMS2 for this layer, although the WMS2 server does not serve the layer by itself. Instead the WMS2 asks the WMS1 server for the layer.

```
...
# This layer is a layer, which can be accessed from another WMS server
LAYER
  METADATA
      WMS_TITLE "Cities"
      ...
  END
  NAME "cities"
  CONNECTIONTYPE postgis
  CONNECTION "user=username dbname=name_of_db host=hostname_db port=portnumber"
  TYPE POLYGON
  CLASS
    COLOR 255 204 102
    OUTLINECOLOR 100 100 100
  END # end of class object
END # end of layer object
```

As can be seen above in the Layer definition, nothing has to be made to the configuration on the local side, i.e. in the Mapfile of the WMS1. The configuration for the cascading is only done on the remote side, i.e. in the Mapfile of the remote WMS (the WMS2).

5.6 Server cascading

5.6.1 WMS server cascading

The WMS2 server is configured to be able to receive a layer from the WMS1 server. This makes it possible to ask the WMS2 for a layer which it does not store itself. Instead, the layer is served by the WMS1 server. The client is unaware of the cascading. By configuring the layer definition in the WMS2 Mapfile, the layer can be made remote.

The Mapfile for the WMS2 MapServer installation is called *wms2_sfa.map*, and contains one remote layer.

Name	wms2_sfa.map
Location	/var/www/cgi-bin/
Purpose	configuration file for the WMS2.

Description
The WMS2 has a layer called *remote_cities*, which is a remote layer. When the client asks the WMS2 for this layer, the WMS2, in its turn, asks the WMS1 for the layer named *cities*. The *CONNECTION* is set to the URL of the WMS1, together with the name of the WMS1 Mapfile. The *CONNECTIONTYPE* is changed from *PostGIS* to *WMS*.

```
...
LAYER
  METADATA
      "wms_title" "Remote_cities"
      "wms_name" "cities"
      "wms_server_version" "1.1.1"
      ...
  END
  NAME remote_cities
  TYPE POLYGON
  CONNECTIONTYPE WMS
  CONNECTION "http://hostname_FreeBSD/cgi-bin/mapserv?map=wms1_sfa.map"
  ...
END # end of layer object
...
```

Furthermore, the web server must be able to access the temporary directory where temporary images and files are put. The path to this directory is specified with the keyword *IMAGEPATH* in the *WEB* object (the object which defines how the web interface will act). The *IMAGEURL* keyword specifies the base URL and takes the web browser to the temporary images and files [oM04b]. It is important that the temporary directory is writable by the user which the web server runs as, and that the path is correctly specified.

```
...
WEB
  IMAGEPATH "/tmp/"
  IMAGEURL '/tmp/'
END
...
```

With these configurations, the client (e.g. a web browser) can receive a map from the WMS2 (which runs on the linux machine), containing this remote layer, by giving the following URL to the browser:

http://hostname_Linux/cgi-bin/mapserv?map=wms2_sfa.map&VERSION=1.1.1&
REQUEST=GetMap&LAYERS=remote_cities&BBOX=10,57,19,60

The same map with the same layer can also be received by accessing it directly from the WMS1 (which runs on the FreeBSD machine):

http://hostname_FreeBSD/cgi-bin/mapserv?map=wms1_sfa.map&VERSION=1.1.1&
REQUEST=GetMap&LAYERS=cities&BBOX=10,57,19,60

With a working WMS server cascading, the only thing remaining for a complete Demo System level1 is the configuration of the WMS2 to receive feature data from the WFS server.

5.6.2 WFS server cascading

To cascade the WFS and WMS servers in Demo System level1 is not possible. The reason for this is that an OGC Web Service can choose which of the two HTTP request methods GET and POST to implement [Con04d]. Both request methods do not have to be implemented. Deegree implements both GET and POST for the GetCapabilities request and the DescribeFeatureType request. But for the GetFeature request, only POST is implemented. MapServer only implements the GET method [oM04e]. By following email-lists, it can be read that MapServer has plans on implementing the POST method, but nothing has happened yet (i.e. in the moment of writing).

Based on these facts, the conclusion that MapServer cannot act WFS client to a Deegree WFS server, is made.

5.7 System architecture

Demo System level1 consists of the following three virtual computers:

- WMS1
  - OS: FreeBSD 4.8
  - Installations: MapServer (compiled to act WMS server), PostgreSQL/PostGIS (database containing both map data and feature data).
- WMS2
  - OS: Linux 2.4.18
  - Installations: MapServer (compiled to act WMS server/client and WFS client), Apache web server (integrated in the Redhat distribution).
- WFS
  - OS: Windows XP
  - Installations: Deegree-demo WFS 1.2.1 (OGC WFS 1.0.0), Apache Tomcat 4.1 (servlet engine).

The design of the implemented system is illustrated in figure 4.

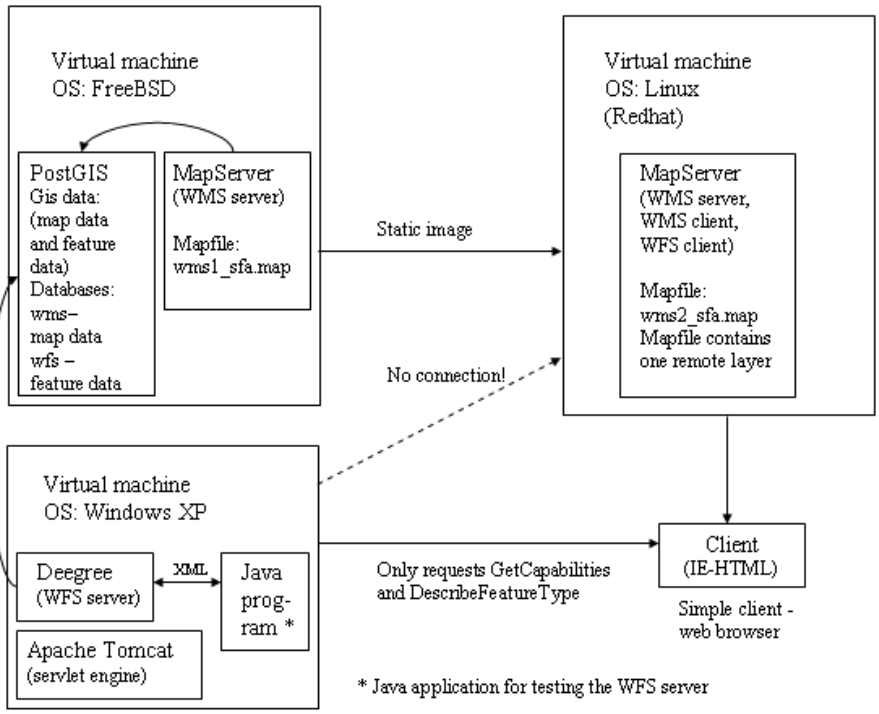

Figure 4: System design level1

5.8 Result

My opinion about Deegree is that the fact that it uses a number of XML files for configuring the WFS, makes it very easy to adjust to the implementers special needs. Once you have understood the arrangement of the different XML configuration files, you can easily connect to chosen data source/data sources, and there is no problem in having different features stored in different data stores (e.g. in different databases).

The MapServer WMS cascading was possible to perform and works fine, but the WFS cascading between MapServer and Deegree does not work. The reason for this is that MapServer and Deegree implement different HTTP request methods for the WFS *GetFeature* operation. To make the WFS-WMS cascading work, one possible approach could be to change framework on the WMS2 server to Deegree. Since I have not worked with the Deegree WMS server/client, I can not say for sure that it would work, but with my experience of Deegree, I would be surprised if it did not. But because MapServer is the framework used for WMS servers at SaabTech, there was no point in constructing a WMS server in Deegree to make the WFS-WMS cascading work. The alternative approach would have been to implement the WFS server with MapServer. The cascading would then probably work when both parts are MapServer installations, but there are many limitations with the MapServer WFS. Furthermore, MapServer does not implement the OGC WFS Implementation Specification, which is a big limitation. My suggestion is therefore not to use MapServer as a WFS. The MapServer developers have done a good job in building a WMS framework, but they have not got that far in their development of WFS. Since the big interest at SaabTech was to achieve level2 of the demo system, I chose, together with my tutors at SaabTech, not to set up a MapServer WFS server to make the WFS server cascading work, but to instead

move on to the implementation of Demo System level2.

The accomplished Demo System level1 is a system of cascading WMS servers, illustrated in figure 5. This system shows that it is possible to have a MapServer WMS cascading, but also that it is not possible to cascade a Deegree WFS server to a MapServer WFS client.

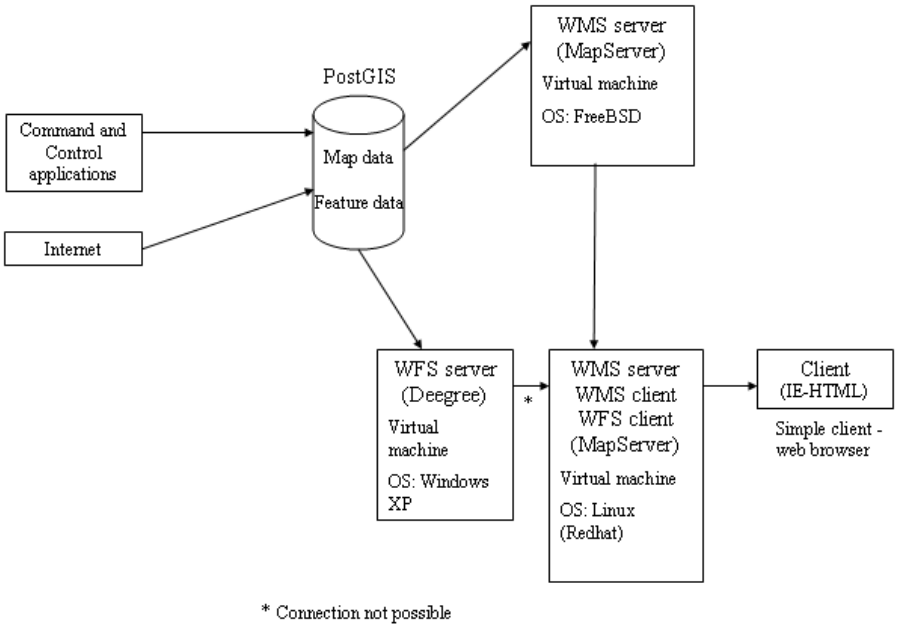

Figure 5: Demo System level1

6 Implementation level2

Although SaabTech is interested in investigate the possibilities of cascading WMS servers and WFS servers (Demo System level1), the goal of the implementation is to achieve a functional Demo System level2, since this is a system that SaabTech might have use for. Products offering approximately the same functionality as provided by Demo System level2 exist, but SaabTech wants to investigate if the technique of WFS can be integrated in these systems and thereby take advantage of all the possibilities that an open standard offers.

The implementation of Demo System level2 consists of building a smarter client application to enhance the interactivity with the user, and to connect the client directly to the WMS server and WFS server.

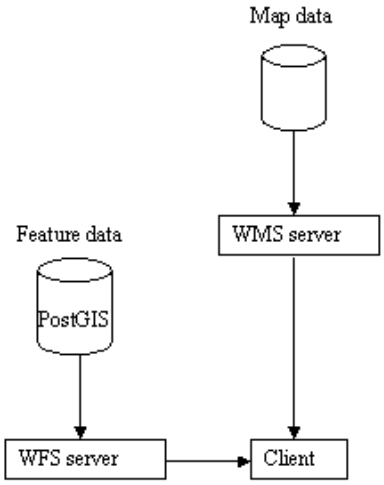

Figure 6: Overview Demo System level2

6.1 Goal for SaabTech with Demo System level2

To have a client that is connected directly to different WMS and WFS servers has a lot of advantages. Not only does it provide a great interoperability and freedom for the client, but it also has an important performance aspect. Imagine a scenario in which it is interesting to follow the movements of ships. The positions of the moving ships changes all the time and their positions on the map need to be frequently updated, which requires a lot of requests to the WFS server containing the ship feature type. On other hand, the sea, i.e. the area in which the ships are moving, may be the same for a long period of time, which means that while the WFS server needs to be continuously asked for new positions of the ships, the WMS server does not have to be asked over and over again for the same map. With this scenario, having the system illustrated with Demo System level1, the WMS sever would be asked to deliver a map each time the WFS server is asked for the new position of the ships, even if the map has not changed. This would mean that a lot of unnecessary requests to the WMS server are sent, which means a lot of unnecessary map data transfers. The result is a heavy loaded network. In Demo System level2, the WMS server is only requested when the client needs a new map. In this way, the data flow is minimized since no unnecessary requests and responses are sent.

Furthermore, to have an interactive client opens a world of possibilities in geo processing.

In the scenario described above, the users may want to receive information about a particular ship, e.g. information about the status of a ship (friendly or hostile), what country the ship belongs to etc. With an interactive client, the user could simply mark the ship of interest to receive information about it.

6.2 The client application

The implementation of the client application is an extension of a web map client application constructed at SaabTech. With the original web map client application, users can choose MapServer WMS server to connect to, and then choose among the available MapServer layers to have presented on the map. The user can also navigate, and zoom in and out the map. This client application is extended in Demo System level2 with additional WFS functionality. In Demo System level2, the users must be able to perform the following (besides the already existing functionalities):

- Select and connect to a WFS server.
- Get information about what feature types the WFS server can serve.
- Select a feature type and present the features instances of this feature type on a map.
- Select a feature instance and receive information about this feature instance.

6.2.1 Use case view

Figure 7 illustrates the possibilities a user has with the extended application.

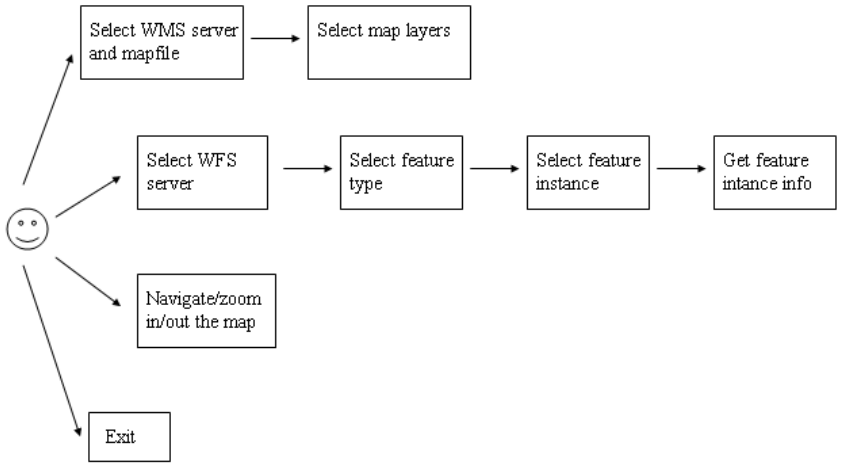

Figure 7: Use case view

In appendix B, screen shots of the application can be found.

6.2.2 Class view

The client application is written in Java. Figure 8 illustrates the classes' relations to each other. The most important classes, along with their most important methods, are described in Appendix C.

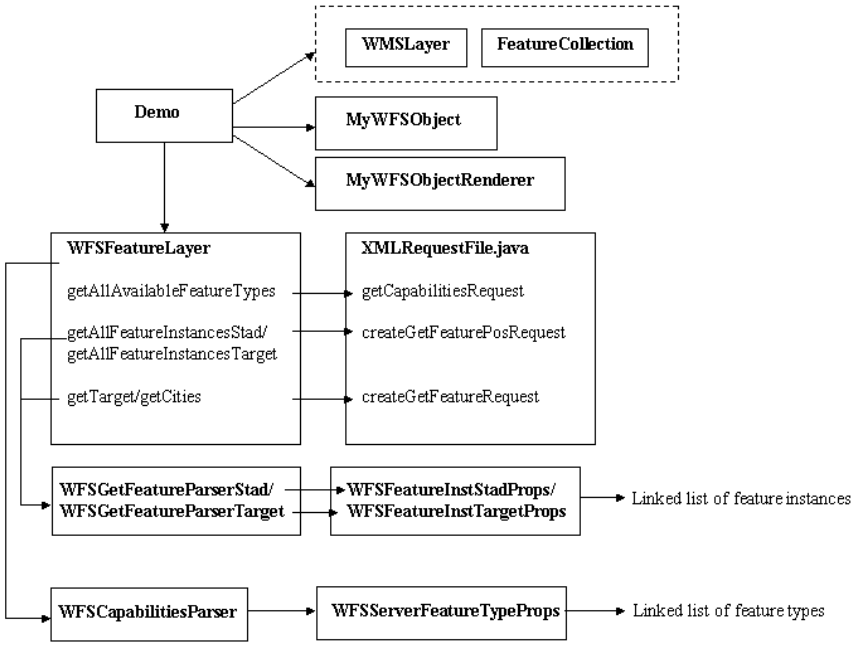

Figure 8: Class relations

6.3 System architecture

The WMS2 from Demo System level1 is now removed and the client has direct connection to the WMS1 and the WFS. The Java client application runs on the Win XP computer. The system design is illustrated in figure 9.

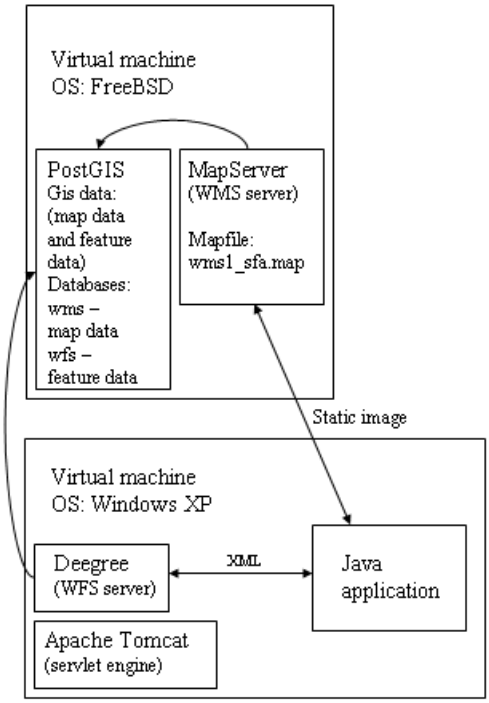

Figure 9: System design level2

6.4 Result

Demo System level2 contains a smarter client application, which is connected to two of the servers constructed in Demo System level1 (the WFS server and one of the WMS servers). In the system, maps are requested from the WMS and features are requested from the WFS. With the client application, users can choose freely what servers it wants to connect to, i.e. the client is not limited to use specific servers as in Demo System level1. Any OGC servers available can be connected to, and it is up to the client application to interpret the answers.

The implemented system works fine, no problems were encountered during implementation. With this system, no unnecessary requests are sent to the server. The questions that are sent, when users are working with the client application, are questions constructed to receive the necessary information, but nothing more. In this way, performance is maximized and the network-load is decreased.

The accomplished Demo System level2 is illustrated in figure 10.

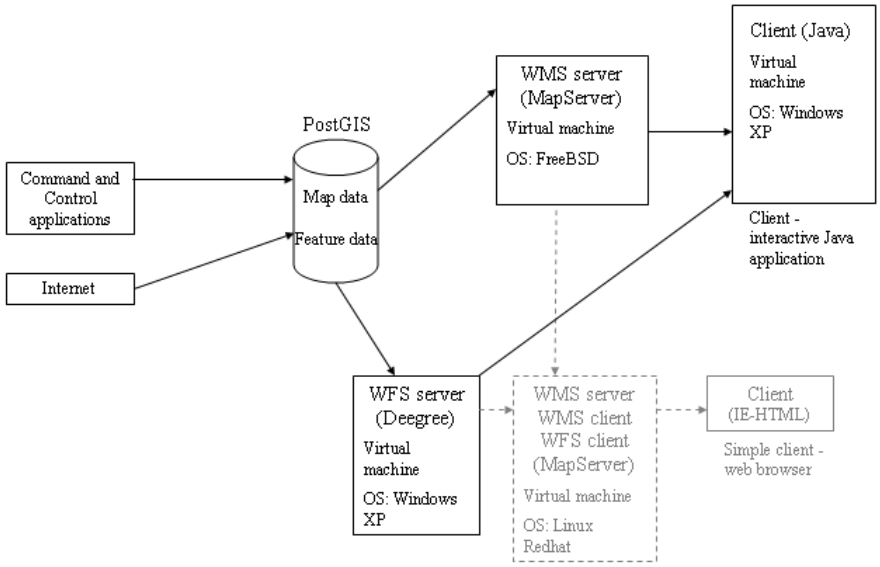

Figure 10: Demo System level2

7 Future work

Although the possibilities of the WFS technique are starting to show with the implementation of Demo System level2, there are a number of things interesting to examine further.

First of all, the client should be written as a general application. I had only time to do this partially. With a fully general implementation of the client application, the user can connect to any OGC WFS server available, receive information about what feature types that are available, having the feature instances presented on a map, and receive more information about chosen feature instances.

A possible scenario in which the WFS technique could be used, and in which the performance would be tested, is a situation where feature instances continuously are moving. To be able to examine this scenario, a function for continuously updates of features would be an interesting implementation topic.

A third level of the demo system is also a work for the future. In this level, the client application would be connected to servers containing additional information about specific properties of a feature type. One server could for example contain information served by a ship manufacturer, which would give the user the possibility to receive even more information about a feature instance, e.g. type of ship, year of construction, size, construction material etc. This would result in a system, in which additional information about features (that the WFS does not provide) could be served to the user. Figure 11 illustrates how Demo System level3 would look like.

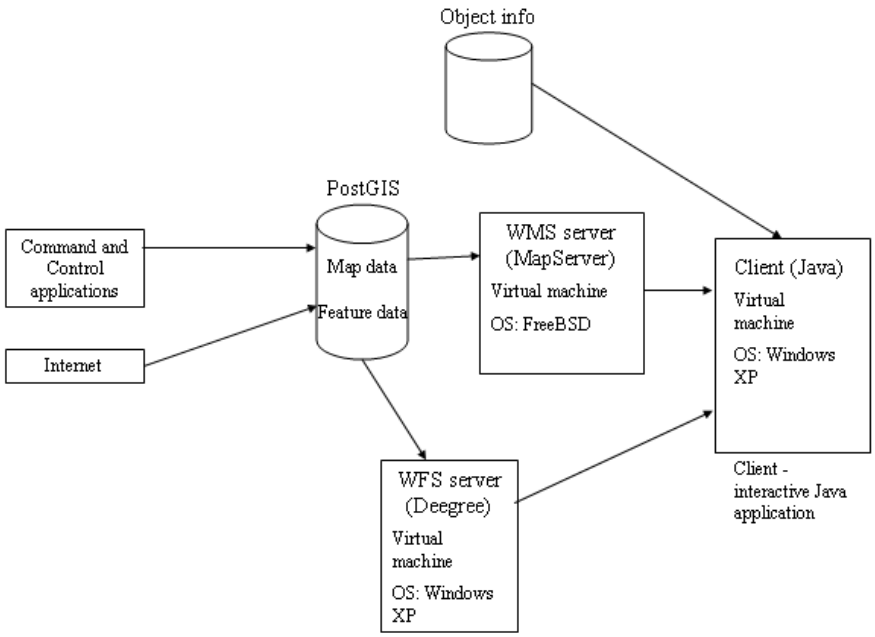

Figure 11: Demo System level3

8 Summary and conclusion

The techniques of WMS and WFS are intended to be the solution to the lack of interoperability, which has existed for a long time in geo-spatial Web Services. With these two OGC Standards we have moved from the situation in which one server requires its own special client application, to the situation where one application can be used with all geo-spatial Web Services implementing the appropriate OGC Specification. OGC is the consortium providing these open specifications that helps the geo-spatial world on the Internet to become interoperable. WMS and WFS make it possible for users of geo-spatial Web Services to only use one application, and still be able to connect to different Web Services providing geo-spatial information.

Since a WFS is an XML Web Service, the security, performance, scalability and reliability of XML Web Services in general have to be considered when analyzing the design of WFS. Beside these general aspects, there are additional topics regarding to the WFS design specifically. The absence of the Push protocol and Multicast functionality is one lack in the WFS design. The functionality of WFS would increase with the implementation of these. Another limitation in the design is the lack of access control. This security topic seems to be left out from the WFS Implementation Specification, which is very unfortunate since being able to control user access on application level is desirable.

Analyzing a new technique as WFS can be quite difficult since documents and technical papers and reports discussing negative aspects of the technique, can be hard to find. Because the technique is new, the negative sides of the technique may not have started to show yet. It is easy to buy the concept of WFS without reflecting over how well designed the technique really is, when all information is either informative or positive (often both). This has to be kept in mind during the whole process of analyzing and implementing the technique. When working with new techniques, critical thinking is important.

The conclusions from the framework investigation are that MapServer can not be recommended to use for the implementation of a distributed WFS system, since it does not implement the OGC WFS Implementation Specification. It seems that MapServer has focused on WMS, and left the goal of offering a good WFS support for later. In contrast to MapServer, The GeoServer Project has put all strength in developing a WFS server and left the WMS part totally. Deegree is the framework that gives the impression of being the most flexible framework for implementing geographical Web Services. While MapServer focus on the WMS part, and The GeoServer Project focus on the WFS part, Deegree focus on both, is easy to configure and can be connected to a wide variety of data-sources.

The conclusion from the implementation of Demo System level1, is that Deegree is a well functional framework for implementing a Web Feature Service. MapServer can be used in WMS server cascading, but since MapServer has big limitations when it comes to the technique of WFS, it cannot be used in the cascading with a WFS server implemented in Deegree. The reason for this is not only because MapServer does not implement the OGC WFS Implementation Specification, but also because of a design limitation in the OGC Specifications. The fact that the OGC Specifications accept both the HTTP request methods GET and POST is a big limitation to the interoperability, which the specifications are constructed to provide. With the freedom of choosing request method to implement, the frameworks may implement different request methods, and may therefore not be able to communicate, *although* they implement the OGC Specifications, which indicates that they should be interoperable. This means that if you have a WMS/WFS server/client, and want it to be able to communicate with other WMS/WFS servers/clients, implemented in other frameworks, it is not enough to check that the frameworks implement the OGC WMS/WFS Implementation Specification, you also have to check which of the two HTTP

request methods the frameworks implement. In order for the different server/clients to be able to communicate, the frameworks that they are built in have to implement the same HTTP request method. MapServer uses GET and Deegree POST for the WFS operation *GetFeature*, and therefore these two frameworks cannot be connected in WFS cascading, and would not be able to connect even if MapServer would implement the OGC WFS Implementation Specification.

Taking the implementation of the demo system to level2 is necessary in order to see the true advantages of the WFS technique. With this system, the possibilities of the WFS technique is easy to understand, and there are practically no limits for how far the Demo System level2 can be developed. The Demo System level2 is a totally heterogenous distributed system of geographic information, implemented in open source software. It is platform independent and implements the OGC WMS and WFS Implementation Specifications. With this system, interoperability in geo-processing has taken another step forward.

9 Acknowledgment

A big thanks is sent to following persons, who all have contributed in the work with this Master's Thesis project.

My tutor at the university, Mikael Rännar. Thank you for helping me with the report writing!

I thank my three tutors at SaabTech for all help and assistance!
Lars Schylberg, my main tutor and a true GIS expert.
Ulf Carlström, a talented (and busy) developer who has written the application I developed further.
Mats Jönsson, the person who has helped me with everything. Thank you for all time you have spent on me Mats.

I would also like to thank all nice people I have met at SaabTech.
♡

(The picture on the report's front-page is the PostGIS symbol [Pos04]).

References

[CB03] Microsoft Cris Boar. *XML Web Services in the Organisation*. 2003.

[Con04a] Open GIS Consortium. Online resources: About us, January 2004. `http://www.opengis.org/about`.

[Con04b] Open GIS Consortium. Online resources: Opengis geography markup language (gml) implementation specification, January 2004. `http://www.opengis.org/docs/02-023r4.pdf`.

[Con04c] Open GIS Consortium. Online resources: Opengis web map server cookbook, January 2004. `http://www.opengis.org/docs/2003/20031113_wmscookbook.pdf`.

[Con04d] Open GIS Consortium. Online resources: Web feature service implementation specification, January 2004. `http://www.opengis.org/docs/02-058.pdf`.

[Con04e] Open GIS Consortium. Online resources: Web map service implementation specification, January 2004. `http://www.opengis.org/docs/01-068r2.pdf`.

[Con04f] OpenGIS Consortium. Online resources: Binary xml-encoding specification, January 2004. `http://www.opengis.org/docs/03-002r8.pdf`.

[Con04g] OpenGIS Consortium. Online resources: Ogc registered products, March 2004. `http://www.opengis.org/resources/?page=products`.

[Con04h] OpenGIS Consortium. Online resources: Simple features specification for sql, February 2004. `http://www.opengis.org/docs/99-049.pdf`.

[ED02] P. Samarati E. Damiani, S. De Capitani di Vimercati. Towards securing xml web services. *The acm Digital Library*, November 2002.

[Geo04] The GeoServerProject. Online resources: Build documentation, January 2004. `http://geoserver.sourceforge.net/documentation/developer/build.htm`.

[GS04] ESRI GIS and Mapping Software. Online resources: Coming to terms with interoperability and standards a brief glossary, January 2004. `http://www.esri.com/news/arcuser/0403/glossary.html`.

[lat04a] lat/lon. *deegree-WFS-Configuration Description*. at/lon, 2004.

[lat04b] lat/lon. Online resources: Deegree home page, January 2004. `http://deegree.sourceforge.net/index.html`.

[Mic02] Microsoft. Online resources: Web services security, April 2002. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglob%spec/html/ws-security.asp`.

[Mic04] Microsoft. Online resources: Xml web services basics, March 2004. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebs%rv/html/infoset_whitepaper.asp`.

[OB02] Z. Lacroix O. Boucelma, M. Essid. A wfs-based mediation system for gis interoperability. *ACM Press, NY, USA*, 2002. Pages: 23-28. ISBN:1-58113-591-2.

[oM04a] University of Minnesota. Online resources: Howto for getting started with mapserver, February 2004. `http://mapserver.gis.umn.edu/doc/getstarted-howto.html`.

[oM04b] University of Minnesota. Online resources: Mapfile reference - mapserver 4.0, February 2004. `http://mapserver.gis.umn.edu/doc40/mapfile-reference.html#web`.

[oM04c] University of Minnesota. Online resources: Mapserver, January 2004. http://mapserver.gis.umn.edu.

[oM04d] University of Minnesota. Online resources: Mapserver unix compilation and installation howto, January 2004. http://mapserver.gis.umn.edu/doc40/unix-install-howto.html.

[oM04e] University of Minnesota. Online resources: Mapserver wfs server howto, January 2004. http://mapserver.gis.umn.edu/doc40/wfs-server-howto.html.

[Pos04] PostGIS. Online resources: Postgis, February 2004. http://www.postgis.org.

[Pro04] The GeoServer Project. Online resources: The geoserver project, January 2004. http://geoserver.sourceforge.net/html/index.php.

[SM04] T. Imamura Y. Nakamura S. Makino, K. Tamura. Implementation and performance of ws-security. *International Journal of Web Services Research*, Jan-Mar 2004. 1(1), pages: 58-72.

[Sun04a] Sun. Online resources: Java api, February 2004. http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Types.html.

[Sun04b] Sun. Online resources: Java servlet technology, February 2004. http://java.sun.com/products/servlet.

[Tec04a] Saab Technologies. Online resources: Saab, March 2004. http://www.saab.se.

[Tec04b] Saab Technologies. Online resources: Saabtech, March 2004. http://www.saabtech.se.

[VMw04] VMware. Online resources: Vmware, February 2004. http://www.vmware.com.

[Wor03] GEO World. Canada goes online continued: Ogc web services explode geoprocessing. *GEO World*, August 2003.

A WFS Tests

WFS test runs done with the simple Java client described in section 5.4.3.

GetCapabilities request

```
C:\java_applikationer>C:\j2sdk1.4.2\bin\java WFSclient GetCapabilities.xml


requested file: c:/deegreewfs/WEB-INF/xml/requests/GetCapabilities.xml
-----------------------------------------------
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- get  the Capabilities of the WFS server -->
<GetCapabilities version="1.0.0" service="WFS"/>


Response from wfs-server
-----------------------------------------------
<WFS_Capabilities version="1.0.0" updateSequence="null">
<Service>
    <Name>WebFeatureServer</Name>
    <Title>deegree WFS</Title>
    <Abstract>Web Feature Server maintained by the lat/lon GmbH</Abstract>
    <OnlineResource>http://hostname_XP:portnumber/deegreewfs</OnlineResource>
</Service>
<Capability>
<Request>
<GetCapabilities>
    <DCPType>
    <HTTP>
    <Get onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    <Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    </HTTP>
    </DCPType>
</GetCapabilities>
<DescribeFeatureType>
    <SchemaDescriptionLanguage>
    <XMLSCHEMA/>
    </SchemaDescriptionLanguage>
    <DCPType>
    <HTTP>
    <Get onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    <Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    </HTTP>
    </DCPType>
</DescribeFeatureType>
<GetFeature>
    <ResultFormat>
    <XML/>
    <GML2/>
    <FEATURECOLLECTION/>
    </ResultFormat>
    <DCPType>
    <HTTP>
    <Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs" />
    </HTTP>
    </DCPType>
</GetFeature>
</Request>
</Capability>
<FeatureTypeList>
    <Operations>
    <Query/>
    </Operations>
    <FeatureType>
```

```
        <Name>targets</Name>
        <Title>targets</Title>
        <Keywords></Keywords>
        <SRS>EPSG:4326</SRS>
        <LatLonBoundingBox minx="-2.5E7" miny="-2.5E7" maxx="2.5E7" maxy="2.5E7" >
        </LatLonBoundingBox>
        <Operations>
        <Query/>
        </Operations>
    </FeatureType>
    <FeatureType>
        <Name>stad</Name>
        <Title>realcities</Title>
        <Keywords></Keywords>
        <SRS>EPSG:4326</SRS>
        <LatLonBoundingBox minx="-2.5E7" miny="-2.5E7" maxx="2.5E7" maxy="2.5E7" >
        </LatLonBoundingBox>
        <Operations>
        <Query/>
        </Operations>
    </FeatureType>
</FeatureTypeList>
</WFS_Capabilities>
Done
```

DescribeFeatureType request

```
C:\java_applikationer>C:\j2sdk1.4.2\bin\java WFSclient DescribeFeatureType.xml


requested file: c:/deegreewfs/WEB-INF/xml/requests/DescribeFeatureType.xml
-----------------------------------------------
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- get the XML schema for the feature type "stad" -->
<DescribeFeatureType outputFormat="XMLSCHEMA">
        <TypeName>stad</TypeName>
</DescribeFeatureType>


Response from wfs-server
-------------------------------------------
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema attributeFormDefault="unqualified"
            elementFormDefault="qualified"
            xmlns:gml="http://www.opengis.net/gml"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import schemaLocation="http://www.deegree.org/xml/schemas/wcts/feature.xsd">
</xsd:import>
<xsd:import schemaLocation="http://www.deegree.org/xml/schemas/wcts/geometry.xsd">
</xsd:import>
<xsd:element name="ResultCollection" substitutionGroup="gml:_FeatureCollection">
</xsd:element>
<xsd:element name="stad" substitutionGroup="gml:_Feature">
<xsd:complexType>
<xsd:sequence>
<xsd:element minOccurs="0" name="THE_GEOM" type="xsd:geometry">
</xsd:element>
<xsd:element minOccurs="0" name="GID" type="xsd:integer">
</xsd:element>
<xsd:element minOccurs="0" name="CITY_NAME">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:maxLength value="255"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
```

```
</xsd:element>
<xsd:element minOccurs="0" name="GMI_ADMIN">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:maxLength value="255"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element minOccurs="0" name="ADMIN_NAME">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:maxLength value="255"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element minOccurs="0" name="FIPS_CNTRY">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:maxLength value="255"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element minOccurs="0" name="CNTRY_NAME">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:maxLength value="255"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element minOccurs="0" name="STATUS">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:maxLength value="255"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element minOccurs="0" name="POP_RANK" type="xsd:integer">
</xsd:element>
<xsd:element minOccurs="0" name="POP_CLASS">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:maxLength value="255"></xsd:maxLength>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element minOccurs="0" name="PORT_ID" type="xsd:integer">
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
Done
```

GetFeature request

```
C:\java_applikationer>C:\j2sdk1.4.2\bin\java WFSclient GetFeatureStadFilter2.xml


requested file: c:/deegreewfs/WEB-INF/xml/requests/GetFeatureStadFilter2.xml
-------------------------------------------
<?xml version="1.0" encoding="iso-8859-1"?>
<wfs:GetFeature outputFormat="GML2" xmlns:gml="http://www.opengis.net/gml" xmlns
:wfs="http://www.opengis.net/wfs" xmlns:ogc="http://www.opengis.net/ogc">
    <!--
       get all cities and all their properties, where the name of the city
```

```
       contains the string "erlin"
    -->
    <wfs:Query typeName="stad">
        <ogc:Filter>
            <ogc:PropertyIsLike wildCard="*" singleChar="#" escape="!">
            <ogc:PropertyName>/stad/Name</ogc:PropertyName>
            <ogc:Literal>*erlin*</ogc:Literal>
            </ogc:PropertyIsLike>
        </ogc:Filter>
    </wfs:Query>
</wfs:GetFeature>


Response from wfs-server
---------------------------------------------
<?xml version="1.0" encoding="UTF-8" ?>
<ResultCollection xmlns:gml="http://www.opengis.net/gml"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://www.deegree.org http://hostname_XP:portnumber/
                  deegreewfs/wfs?request=DescribeFeatureType&amp;typename=stad">
<gml:boundedBy>
    <gml:Box>
    <gml:coord>
        <gml:X>13.3275728225708</gml:X>
        <gml:Y>52.5162734985352</gml:Y>
    </gml:coord>
    <gml:coord>
        <gml:X>13.3275728225708</gml:X>
        <gml:Y>52.5162734985352</gml:Y>
    </gml:coord>
    </gml:Box>
</gml:boundedBy>
<gml:featureMember>
    <stad fid="ID273">
        <stad.Port_id>0</stad.Port_id>
        <stad.Pop_class>1,000,000 to 5,000,000</stad.Pop_class>
        <stad.coordinates>
            <gml:Point srsName="EPSG:4326">
            <gml:coordinates cs="," decimal="." ts=" ">13.3275728225708,52.5162734985352
            </gml:coordinates></gml:Point>
        </stad.coordinates>
        <stad.Status>National and provincial capital</stad.Status>
        <stad.Fips_cntry>GM</stad.Fips_cntry>
        <stad.Name>Berlin</stad.Name>
        <stad.ID>273</stad.ID>
        <stad.Cntry_name>Germany</stad.Cntry_name>
        <stad.Admin_name>Berlin</stad.Admin_name>
        <stad.Gmi_admin>DEU-BER</stad.Gmi_admin>
        <stad.Pop_rank>2</stad.Pop_rank>
    </stad>
</gml:featureMember>
</ResultCollection>
Done
```

B Java application

Appendix B shows some screen shots of the Java client application. The left side in the application window holds the maps, and the right side displays the client and server communication.

In figure 12, the users specifies the WFS server to connect to. In the same way, the user can specify what WMS server (*Mapserver* in the menu) to receive map layers from.

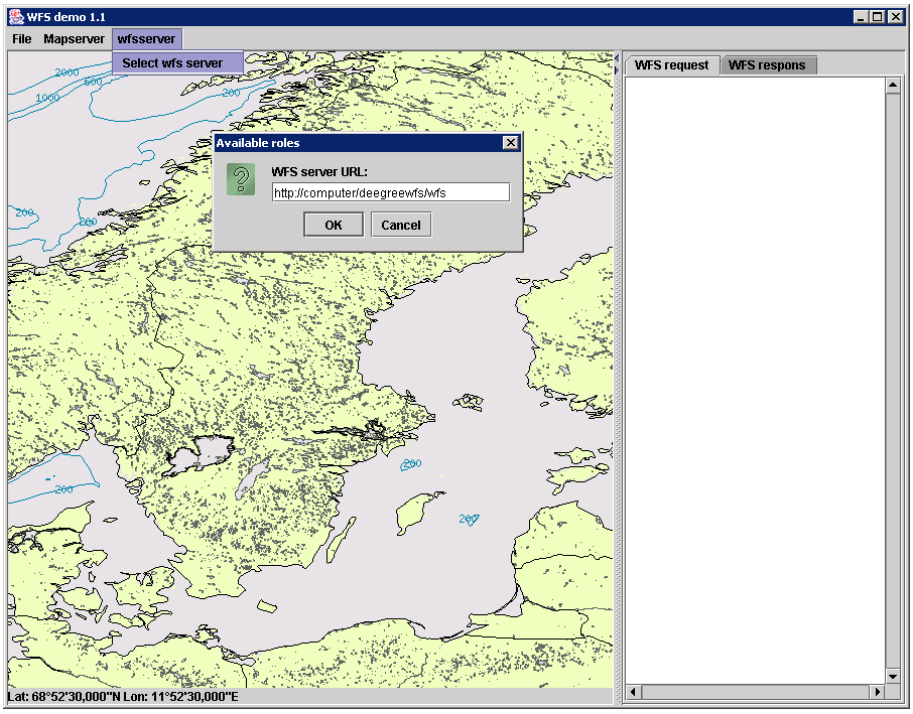

Figure 12: Select WFS server

In Figure 13, the user has pressed right mouse button to open the menu from which map layers and feature types can be chosen. Some map layers have already been added in the figure, but no features have been added yet.

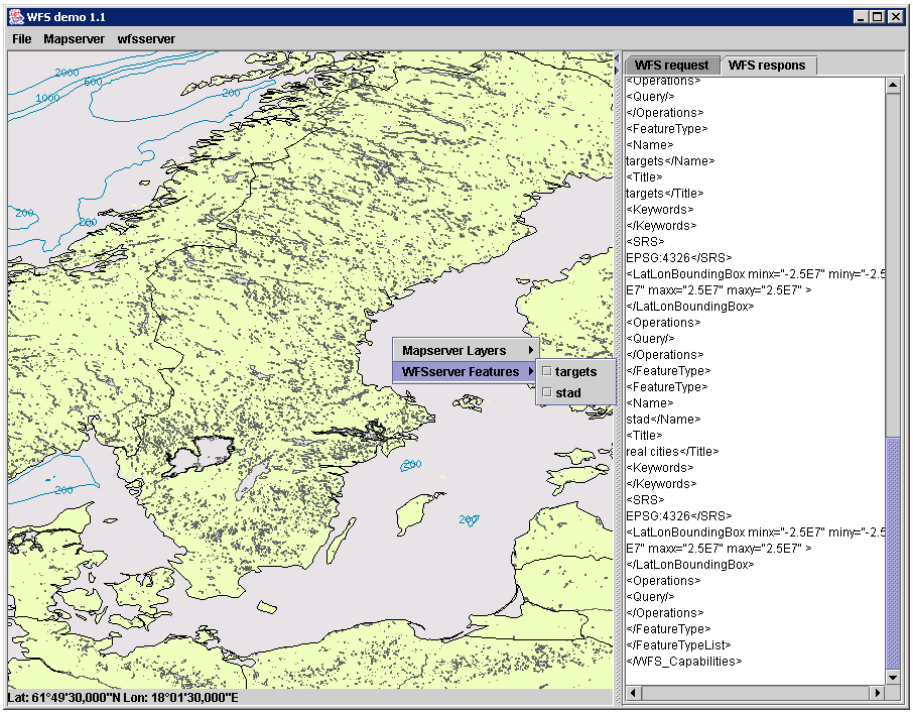

Figure 13: Available feature types

In figure 14, the feature type *Targets* has been selected. All instances of the *targets* feature type are drawn on the map. The targets are represented with triangles, drawn in different colors depending on the category of the target.

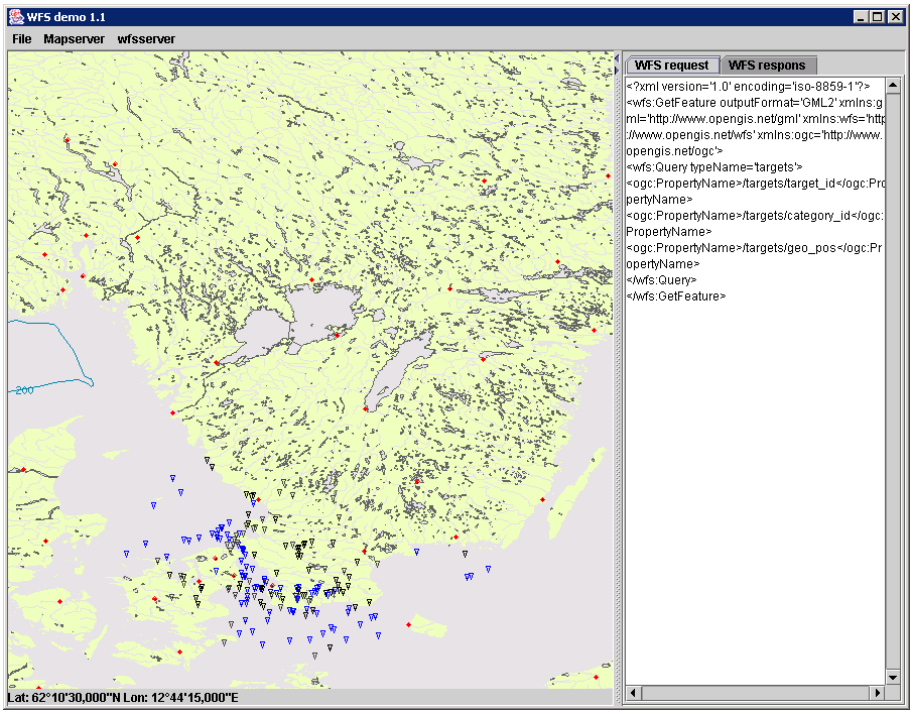

Figure 14: *Targets* feature instances

In figure 15, one target is marked and the user wants to receive more information about the marked target. The information available about the target is shown in, and can be selected from, the check-box. The marked target has been observed on the sea.

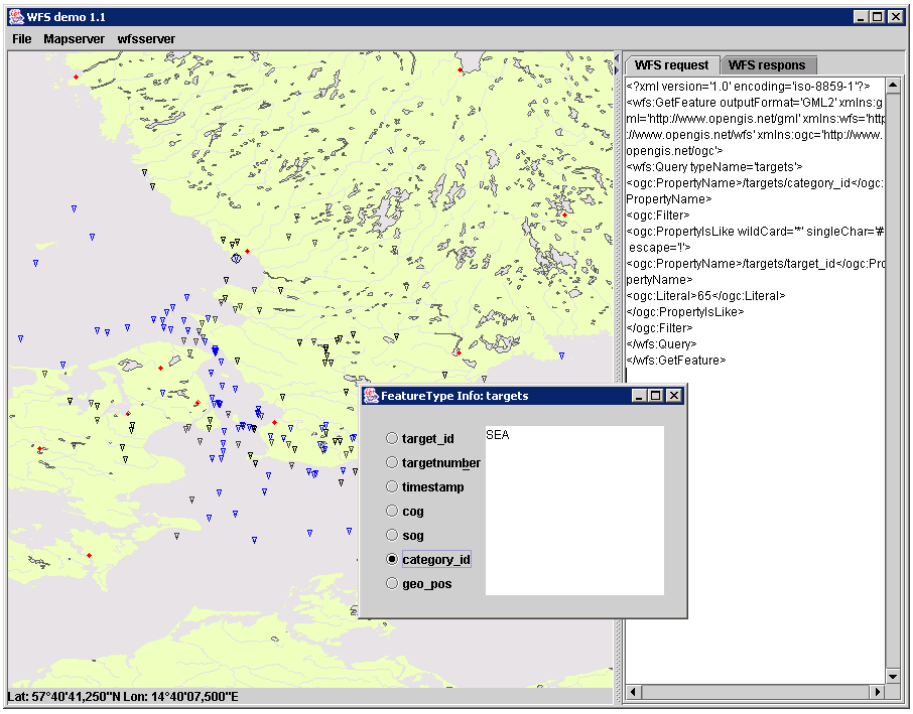

Figure 15: Select feature instance

In figure 16, the map has been zoomed out, and different map layers have been selected (e.g. country borders). Both of the feature types (*targets* and *stad*) are selected from the menu, and all features of those feature types are drawn on the map. The features of feature type *stad* are represented with red dots.

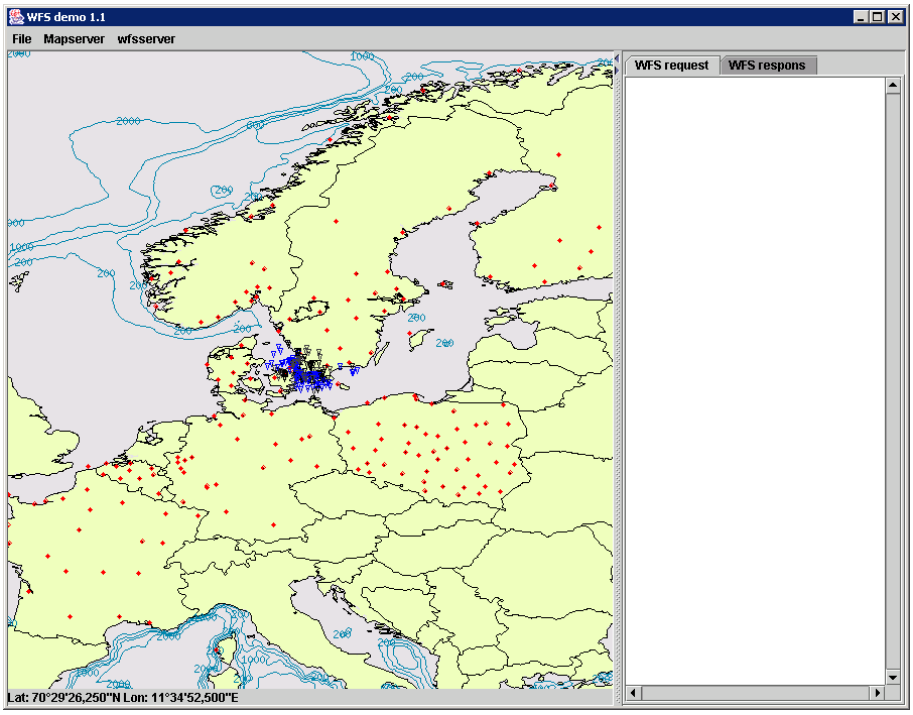

Figure 16: Different map layers and both available feature types are selected

C Class description

Short presentation of the most important classes in the Java application written for Demo System level2.

Demo.java
Class contains the main method and is the main class in the application.

mouseClicked(MouseEvent e) - A feature instance on the map is marked with a left mouse button click. If a feature is marked, and left mouse button is clicked and either Control, Shift or Alt button is pressed, different actions take place. With Control pressed, a dialog box appears from which the user can select and receive information about the marked feature instance. With Shift pressed the mapped is zoomed out, and with Alt pressed the map is zoomed in.

run() - method sets up the application window with all its functionality, e.g. the layout, menus etc. It sets the default WMS to be the MapServer running on the FreeBSD machine, and the default WFS server to be the WFS server running on the XP machine.

CreateTargetObjects/createStadObjects - creates target/stad objects and adds them to a collection containing all chosen feature instances.

DeleteFeatureInst(String featureTypeName) - deletes the features of specified feature type from the collection of feature instances.

createWFSDialog(String feature_type, String feature_name) - creates a radio button dialog window, in which the user can choose what information to receive about a selected feature instance.

exit() - closes the window upon user request.

main() - creates an object of the class *Demo.java* and performs the run method on it, which starts the application.

WFSFeatureLayer.java
Class represents a WFS layer, which means that it contains all methods/operations on the WFS features. This class extends the class *ObjectFeatureLayer.java*.

makeWFSRequest(String XMLrequest) - Performs a request (received as parameter) to the WFS server. The answer from the WFS server is returned.

GetAllAvailableFeatureTypes() - Performs a GetCapabilities request to the server and parses the result generated from the server.

GetAllFeatureInstancesTarget()/getAllFeatureInstancesStad() - Performs a GetFeature request to the WFS server and parses the result generated from the server.

getTarget(String featurename, String chosen_info) - Performs a GetFeature request to the server with the feature instance and feature attribute specified, and parses the result generated by the server.

XMLRequestFile.java
Class represents an XML file. It contains methods for creating different XML files, which are to be sent as requests to the WFS server.

getCapabilitiesRequest() - Creates a GetCapabilities XML request file.

createGetFeaturePosRequest(String featuretype) - Creates a GetFeature request for the feature type sent as argument. The request is restricted so that not all attributes of the feature instances are returned. Only the parameters necessary for creating feature representations on the map are asked for.

createGetFeatureRequest(String featurename, String featuretype, String chosen_info) - Creates a GetFeature request with restrictions: the feature instance of interest and parameter of interest.

WFSServerFeatureTypeProps.java
Class represents a feature type and contains the attributes needed to be stored about a feature type.

WFSFeatureInstStadProps.java and WFSFeatureInstTargetProps.java
These classes represent a feature instance of the feature type ”stad” and ”targets” respectively. They contain all attributes of an instance of respective feature type and get/set methods for all these attributes. The attributes of a feature instance are stored as a linked list.

WFSCapabilitiesParser.java
Class represents a parser for the XML file received after a GetCapabilities request is sent to the WFS server. The parser creates a new object of the class WFSServerFeatureTypeProps for each feature type it finds in the received XML file. The result of the parsing is a linked list of feature types with the attributes set for each feature type.

startElement - is called when the parser encounters a new feature type tag in the received XML file. A new object of the class WFSServerFeatureTypeProps is then created and added to the linked list of feature types.

endElement - is called when an attribute of a feature type is encountered by the parser. The attribute for the feature type is then set.

WFSGetFeatureParserStad.java
Class represents a parser for the XML file received after a GetFeature (feature type = stad) request is sent to the WFS server. For each feature instance, a new object of the class WFSFeatureInstStadProps is created. The result of the parsing is a linked list of feature instances with the attributes set for each feature instance.

startElement - is called when the parser encounters a new feature member (feature instance) tag in the document it parsers. A new object of the class WFSFeatureInstStadProps is then created and added to the linked list of feature instances.

endElement - is called when an attribute of a feature instance is encountered by the parser. The attribute for the feature instance is then set.

WFSGetFeatureParserTarget.java
Class represents a parser for the XML file received after a GetFeature (feature type = targets) request is sent to the WFS server. It has the same functionality and methods as the class *WFSGetFeatureParserStad.java*.

MyWFSObject.java
Class represents a graphic WFS feature, i.e. the graphical representation of a WFS feature instance. When the user selects a feature type from the menu, all feature instances of this

feature type is requested from the WFS server, and for each feature instance, an object of this class is created and put into the collection of feature objects.

MyWFSObjectRenderer.java
Defines and renders the WFS objects on the map.

Two of the classes, which already existed in the application before the WFS extension was made, are the following:

WMSLayer.java
Class represents a WMS layer.

FeatureCollection.java
Class represents a collection of features. This collection holds the graphical representation of the WFS feature instances.

D Deegree configuration files

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>

<servlet>
<servlet-name>deegreewfs</servlet-name>
<servlet-class>org.deegree_impl.enterprise.WFSServlet</servlet-class>
<init-param>
<param-name>capabilities</param-name>
<param-value>file:///C:/deegreewfs/WEB-INF/xml/sample_wfs_capabilities.xml
            </param-value>
</init-param>
<init-param>
<param-name>debug</param-name>
<param-value>ERRORS_AND_COMMENTS</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>deegreewfs</servlet-name>
<url-pattern>/wfs</url-pattern>
</servlet-mapping>
</web-app>
```

sample_wfs_capabilities.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<WFS_Capabilities version="1.0.0">
<Service>
<Name>WebFeatureServer</Name>
<Title>deegree WFS</Title>
<Abstract>Web Feature Server maintained by the lat/lon GmbH</Abstract>
<OnlineResource>http://hostname_XP:portnumber/deegreewfs?</OnlineResource>
</Service>
<Capability>
<Request>
<GetCapabilities>
<DCPType>
<HTTP>
<Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs"/>
<Get onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs"/>
</HTTP>
</DCPType>
</GetCapabilities>
<DescribeFeatureType>
<SchemaDescriptionLanguage>
<XMLSCHEMA/>
</SchemaDescriptionLanguage>
<DCPType>
<HTTP>
<Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs"/>
<Get onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs"/>
</HTTP>
</DCPType>
</DescribeFeatureType>
<GetFeature>
<ResultFormat>
<XML className="org.deegree_impl.services.wfs.XMLResponseHandler"/>
<GML2 className="org.deegree_impl.services.wfs.GMLResponseHandler"/>
```

```
<FEATURECOLLECTION className="org.deegree_impl.services.wfs.
                    FCResponseHandler"/>
</ResultFormat>
<DCPType>
<HTTP>
<Post onlineResource="http://hostname_XP:portnumber/deegreewfs/wfs"/>
</HTTP>
</DCPType>
</GetFeature>
</Request>
</Capability>
<FeatureTypeList>
<Operations>
<Query/>
</Operations>
<FeatureType>
<ResponsibleClass className="org.deegree_impl.services.wfs.postgis.
            PostgisDataStore" configURL="file:///C:/deegreewfs/WEB-INF/xml/
            config_postGIS_stad.xml"/>
<Name>stad</Name>
<Title>real cities</Title>
<SRS>EPSG:4326</SRS>
<LatLonBoundingBox minx="-25000000" miny="-25000000" maxx="25000000"
            maxy="25000000"/>
<BoundingBox SRS="EPSG:4326"
                    minx="-25000000" miny="-25000000" maxx="25000000"
                    maxy="25000000" />
<ScaleHint min="0" max="0.0124726" />
</FeatureType>
<FeatureType>
<ResponsibleClass className="org.deegree_impl.services.wfs.
            postgis.PostgisDataStore"
            configURL="file:///C:/deegreewfs/WEB-INF/xml/config_postGIS_targets.xml"/>
<Name>targets</Name>
<Title>targets</Title>
<SRS>EPSG:4326</SRS>
<LatLonBoundingBox minx="-25000000" miny="-25000000"
            maxx="25000000" maxy="25000000"/>
<BoundingBox SRS="EPSG:4326"
                    minx="-25000000" miny="-25000000" maxx="25000000"
            maxy="25000000" />
<ScaleHint min="0" max="0.0124726" />
</FeatureType>
</FeatureTypeList>
</WFS_Capabilities>
```

config_postGIS_stad.xml

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<DatastoreConfiguration name="cite" type="POSTGIS">
<Connection name="connection_name">
<driver>org.postgresql.Driver</driver>
<logon>jdbc:postgresql://hostname_db:portnumber/wfs</logon>
<user>username</user>
<password>password</password>
</Connection>
<FeatureType name="stad">
<OutputFormat>
<GML2 responsibleClass="org.deegree_impl.services.wfs.postgis.
            PostgisDataStoreOutputGML">
<!-- URL where an application can find the schema of the
                feature type schema -->
<SchemaLocation>file:///C:/deegreewfs/WEB-INF/xml/stad.xsd
                </SchemaLocation>
</GML2>
```

```
<XML responsibleClass="org.deegree_impl.services.wfs.postgis.
            PostgisDataStoreOutputXML">
<!-- URL where an application can find the schema of the
                feature type schema -->
<SchemaLocation>file:///C:/deegreewfs/WEB-INF/xml/stad.xsd
                </SchemaLocation>
</XML>
<FEATURECOLLECTION responsibleClass="org.deegree_impl.services.
            wfs.postgis.PostgisDataStoreOutputFC"/>
</OutputFormat>
<MappingField>
<Property name="/stad/ID" type="INTEGER"/>
<DatastoreField name="stad.GID" type="INTEGER"/>
</MappingField>
<MappingField>
<Property name="/stad/Name" type="VARCHAR"/>
<DatastoreField name="stad.CITY_NAME" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/stad/Gmi_admin" type="VARCHAR"/>
<DatastoreField name="stad.GMI_ADMIN" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/stad/Admin_name" type="VARCHAR"/>
<DatastoreField name="stad.ADMIN_NAME" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/stad/Fips_cntry" type="VARCHAR"/>
<DatastoreField name="stad.FIPS_CNTRY" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/stad/Cntry_name" type="VARCHAR"/>
<DatastoreField name="stad.CNTRY_NAME" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/stad/Status" type="VARCHAR"/>
<DatastoreField name="stad.STATUS" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/stad/Pop_rank" type="INTEGER"/>
<DatastoreField name="stad.POP_RANK" type="INTEGER"/>
</MappingField>
<MappingField>
<Property name="/stad/Pop_class" type="VARCHAR"/>
<DatastoreField name="stad.POP_CLASS" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/stad/Port_id" type="INTEGER"/>
<DatastoreField name="stad.PORT_ID" type="INTEGER"/>
</MappingField>
<MappingField>
<Property name="/stad/coordinates" type="GEOMETRY"/>
<DatastoreField name="stad.THE_GEOM" type="GEOMETRY"/>
</MappingField>
<MasterTable name="stad" targetName="stad">
<!-- name of the table column that stores the id of a feature -->
<IdField number="true" auto="false">GID</IdField>
</MasterTable>
<CRS>EPSG:4326</CRS>
</FeatureType>
</DatastoreConfiguration>
```

config_postGIS_targets.xml

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
```

```
<DatastoreConfiguration name="cite" type="POSTGIS">
<Connection name="connection_name">
<driver>org.postgresql.Driver</driver>
<logon>jdbc:postgresql://hostname_db:portnumber/wfs</logon>
<user>username</user>
<password>password</password>
</Connection>
<FeatureType name="targets">
<OutputFormat>
<GML2 responsibleClass="org.deegree_impl.services.wfs.postgis.
                                    PostgisDataStoreOutputGML">
<!-- URL where an application can find the schema
                        of the feature type schema -->
<SchemaLocation>file:///C:/deegreewfs/WEB-INF/xml/targets.xsd
                </SchemaLocation>
</GML2>
<XML responsibleClass="org.deegree_impl.services.wfs.postgis.
                                    PostgisDataStoreOutputXML">
<!-- URL where an application can find the schema of the feature
                type schema -->
<SchemaLocation>file:///C:/deegreewfs/WEB-INF/xml/targets.xsd
                </SchemaLocation>
</XML>
<FEATURECOLLECTION responsibleClass="org.deegree_impl.services.wfs.
            postgis.PostgisDataStoreOutputFC"/>
</OutputFormat>
<MappingField>
<Property name="/targets/target_id" type="INTEGER"/>
<DatastoreField name="targets.target_id" type="INTEGER"/>
</MappingField>
<MappingField>
<Property name="/targets/targetnumber" type="VARCHAR"/>
<DatastoreField name="targets.targetnumber" type="VARCHAR"/>
</MappingField>
<MappingField>
<Property name="/targets/timestamp" type="INTEGER"/>
<DatastoreField name="targets.timestamp" type="INTEGER"/>
</MappingField>
<MappingField>
<Property name="/targets/cog" type="DOUBLE"/>
<DatastoreField name="targets.cog" type="DOUBLE"/>
</MappingField>
<MappingField>
<Property name="/targets/sog" type="DOUBLE"/>
<DatastoreField name="targets.sog" type="DOUBLE"/>
</MappingField>
<MappingField>
<Property name="/targets/category_id" type="INTEGER"/>
<DatastoreField name="targets.category_id" type="INTEGER"/>
</MappingField>
<MappingField>
<Property name="/targets/geo_pos" type="GEOMETRY"/>
<DatastoreField name="targets.geo_pos" type="GEOMETRY"/>
</MappingField>
<MasterTable name="targets" targetName="targets">
<!-- name of the table column that stores the id of a feature -->
<IdField number="true" auto="false">target_id</IdField>
</MasterTable>
<CRS>EPSG:4326</CRS>
</FeatureType>
</DatastoreConfiguration>
```

stad.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema xmlns:gml="http://www.opengis.net/gml"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import schemaLocation="http://www.deegree.org/xml/schemas/
    wcts/geometry.xsd"/>
<xsd:import schemaLocation="http://www.deegree.org/xml/schemas/
    wcts/feature.xsd"/>
<xsd:element name="stad" substitutionGroup="gml:_Feature">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="THE_GEOM" type="xsd:geometry"
                minOccurs="0"/>
<xsd:element name="GID" type="xsd:integer"
                minOccurs="0"/>
<xsd:element name="CITY_NAME" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="GMI_ADMIN" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="ADMIN_NAME" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="FIPS_CNTRY" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="CNTRY_NAME" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="STATUS" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="POP_RANK" type="xsd:integer"
                minOccurs="0"/>
<xsd:element name="POP_CLASS" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="PORT_ID" type="xsd:integer"
                minOccurs="0"/>
</xsd:sequence>
```

```
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

targets.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:gml="http://www.opengis.net/gml"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import schemaLocation="http://www.deegree.org/xml/schemas/
    wcts/geometry.xsd"/>
<xsd:import schemaLocation="http://www.deegree.org/xml/schemas/
    wcts/feature.xsd"/>
<xsd:element name="stad" substitutionGroup="gml:_Feature">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="geo_pos" type="xsd:geometry"
                minOccurs="0"/>
<xsd:element name="target_id" type="xsd:integer"
                minOccurs="0"/>
<!--
<xsd:element name="" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
-->
<xsd:element name="targetnumber" minOccurs="0">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="255"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="timestamp" type="xsd:integer"
                minOccurs="0"/>
<xsd:element name="cog" type="xsd:double" minOccurs="0"/>
<xsd:element name="sog" type="xsd:double" minOccurs="0"/>
<xsd:element name="category_id" type="xsd:integer"
                minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```