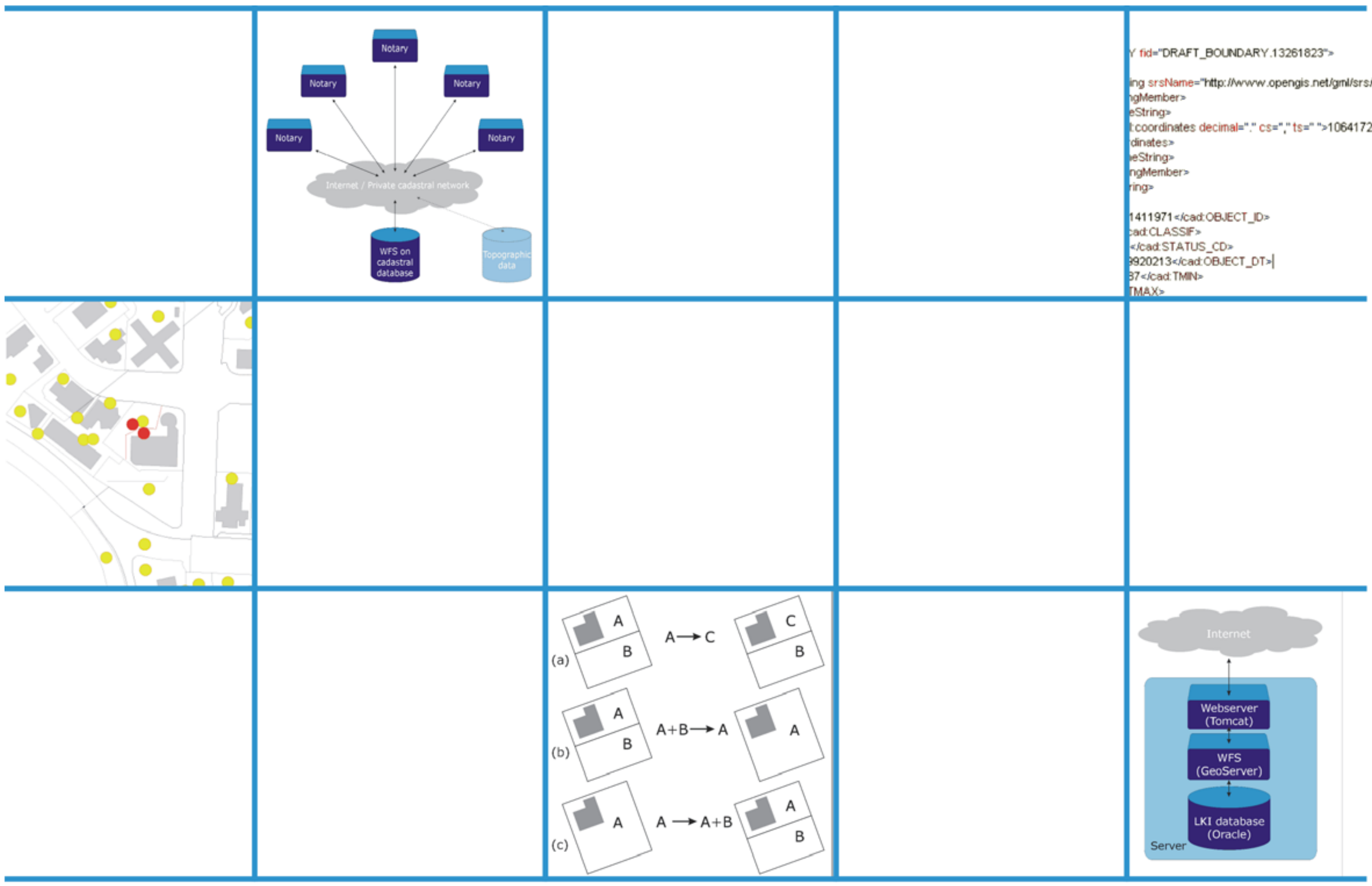
OpenGIS Web Feature Services for editing cadastral data

Analysis and practical experiences



```
Y fid="DRAFT_BOUNDARY.13261823">

ing srsName="http://www.opengis.net/gml/srs/epsg
ngMember>
eString>
l:coordinates decimal="." cs="," ts=" ">106417204,4
dinates>
eString>
ngMember>
ring>

1411971</cad:OBJECT_ID>
cad:CLASSIF>
</cad:STATUS_CD>
)920213</cad:OBJECT_DT>
37</cad:TMIN>
TMAX>
```

Master of Science thesis
T.J. Brentjens
Section GIS Technology
Geodetic Engineering
Faculty of Civil Engineering and Geosciences
Delft University of Technology

OpenGIS Web Feature Services for editing cadastral data

Analysis and practical experiences

Master of Science thesis Thijs Brentjens

Professor:	prof. dr. ir. P.J.M. van Oosterom (Delft University of Technology)
Supervisors:	drs. M.E. de Vries (Delft University of Technology)
	drs. C.W. Quak (Delft University of Technology)
	drs. C. Vijlbrief (Kadaster)

Delft, April 2004

Section GIS Technology
Geodetic Engineering
Faculty of Civil Engineering and Geosciences
Delft University of Technology
The Netherlands

Het Kadaster
Apeldoorn
The Netherlands

Preface

This thesis is the result of the efforts I have put in my graduation research project between March 2003 and April 2004. I have performed this research part-time at the section GIS Technology of TU Delft in cooperation with the Kadaster (the Dutch Cadastre), in order to get the Master of Science degree in Geodetic Engineering.

Typing the last words for this thesis, I have been realizing more than ever that this thesis marks the end of my time as a student at the TU Delft. However, I also realize that I have been working to this point with joy. Many people are "responsible" for this, but I'd like to mention the people who have contributed most.

First of all, there are of course people who were directly involved in the research project. Peter van Oosterom had many critical notes and - maybe even more important - the ideas born out of his enthusiasm improved the entire research. Many thanks go to Tom Vijlbrief, representing the Kadaster, for his helpful comments (on all topics) and for showing the way when it concerned Kadaster issues. Wilko Quak always was available to give technical support. But I'd like to thank especially Marian de Vries. As my direct supervisor, she helped solving technical problems, reviewing new pieces I wrote for this thesis and developing parts of the software. Especially their expertise and pleasant way of working, created the right atmosphere to perform the research in.

Besides the scientific interest of people, two people have contributed to the project without any direct interest (I think at least). Here, I have to mention Marco Baars, a good friend and fellow MSc student. He also reviewed my thesis, but more importantly he always has been available for a cup of coffee, lunch or a short walk outside. I'm grateful to my mother, Dorrit Brentjens, who also helped with reviewing my thesis and gave the necessary mental support to me, even during very difficult times for her.

For the upcoming years, I hope to work in the same friendly yet professional atmosphere as I've experienced during this MSc thesis project.

Delft, 28 April 2004

Thijs Brentjens

Contents

Summary

The need for sharing already available, but distributed geographic data and the need for facilitating the integration of Geographic Information Systems (GIS) with other information systems has been growing. In such a distributed environment, the ability to edit geographic data (i.e. besides querying, also deleting, updating and creating geographic features) is essential.

GIS web services are services using the Internet, for example to retrieve maps or query geodata. In this context, the OpenGIS Consortium (OGC) recently has developed several web service specifications that should support interoperability between services and clients of different vendors. One of these is the Web Feature Services specification. This specification is a unique open standard in the sense that it is the only specification that provides a standardized interface not only for querying (Basic WFS) but also for editing geographic data (Transactional WFS) over the Internet.

The OpenGIS Service Framework provides the framework for OpenGIS Web Services. These open, interoperable services use general web technologies like HTTP and XML and can be used to build applications dynamically. Web Feature Services are one of these services.

Web Feature Services are either Basic, which means that data can only be retrieved, or Transactional, which means that the server supports create, update and delete operations on some or all of its features. The WFS specification also defines operations for locking features, which can be used to maintain consistency.

The Geography Markup Language (GML) is an XML encoding for the modeling, transport and storage of geographic information including both the spatial and non-spatial properties of geographic features. In WFS, GML 2.1.2 is used as output format and to encode new geographic features.

The term *geometric transaction* is used for transactions consisting of operations to modify geographic features in a geo-database. Because of the geometric properties, geometric transactions are usually complicated to process. Not only common difficulties with geo-databases like encoding and computations with spatial operators play a role. In particular the relations of geographic features with features in the surroundings (either explicitly stored in some topological structure or spatial constraints, or implicitly from the geometric properties) make transactions complicated to process.

Since Web Feature Services can be used to edit geographic data over Internet, Web Feature Services should be able to process geometric transactions. Therefore for this MSc research project, the following research question has been answered:

How should geometric transactions be processed in a distributed GIS using OpenGIS Web Feature Services?

To answer this question literature has been studied and a case study has been done to develop a Web Feature Service. With this service, the abilities of WFS to process geometric transactions have been analyzed.
For editing the cadastral data, one could think of a notary who drafts a new boundary, because a parcel has to be split. This principle has been used in the case study. Both a web feature server and a client have been implemented.

The web feature server is composed of an Oracle Spatial 9i database, with GeoServer (an Open Source WFS) configured on top of it as web feature server. GeoServer has been installed in Tomcat (a Java servlet container) to provide Internet access, as illustrated in figure 1.

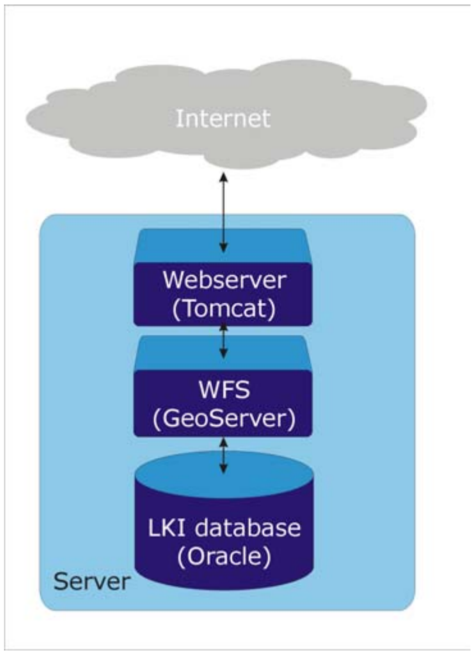


Figure 1 Developed web feature server

Because the tested existing WFS-clients were either not fully compliant with the specification, nor transactional or it would have cost too much time to adapt them for the case study service, a client needed to be developed in this research. The developed client uses Scalable Vector Graphics (SVG) in a standard web-browser for visualization and editing of features graphically. Java Server Pages (JSP) have been used to implement functionality to compose and send the transaction request. This way, functionality of the client is divided between a browser and Java web server, as illustrated in figure 2.

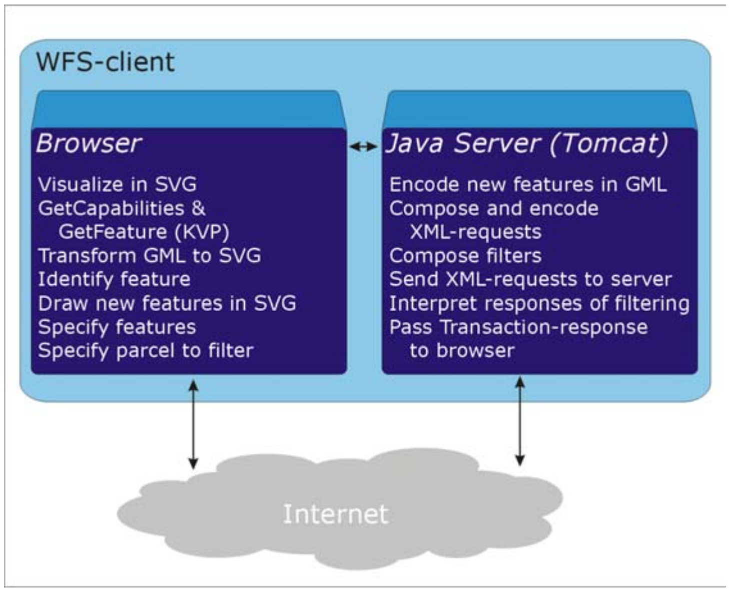

WFS-client
Browser
Visualize in SVG
GetCapabilities & GetFeature (KVP)
Transform GML to SVG
Identify feature
Draw new features in SVG
Specify features
Specify parcel to filter
Java Server (Tomcat)
Encode new features in GML
Compose and encode XML-requests
Compose filters
Send XML-requests to server
Interpret responses of filtering
Pass Transaction-response to browser
Internet

Figure 2 Functionality of the developed WFS client is divided between a browser and web server

Interoperability tests with other than the developed client showed that not all currently available clients that claim to be WFS-compliant, work correctly. However, the tests also showed the power of interoperable services, since a generic client - not having any knowledge of the data source – could retrieve features from the data source, because a web feature server was configured on top of it.

The case study showed that, in general, the WFS-specification provides sufficient operations to make sure that geometric transactions can be sent from client to server and can be processed successfully. Using general web technologies as HTTP, XML and GML, geographic features can be retrieved, created, modified and deleted. However, WFS misses operations to deal well with constraints and other application logic and misses some useful operations to deal with complicated data sets and transactions.

The main conclusion is that, although WFS is not advanced enough to strongly support transactions on complicated (e.g. with topology or other spatial constraints) data sets in a generic way, Web Feature Services are suitable and powerful interoperable GIS web services to edit geographic data over large networks as the Internet.

Samenvatting

De noodzaak om reeds beschikbare, maar verspreid opgeslagen geografische data te kunnen delen en hergebruiken en de noodzaak om Geografische Informatie Systemen te integreren met andere informatie systemen is sterk gegroeid. Het is essentieel om ook in zo'n gedistribueerde omgeving geografische data te kunnen bewerken. Dit houdt in dat de data niet alleen bevraagd moet kunnen worden, maar ook dat geografische objecten gewijzigd, verwijderd en gecreëerd moeten kunnen worden.

GIS Web Services zijn services die gebruik maken van internet, bijvoorbeeld om (digitale) kaarten op te vragen of geografische data te bevragen. Het OpenGIS Consortium (OGC) heeft in dit kader onlangs een aantal standaarden opgesteld die voor interoperabiliteit tussen services en clients van verschillende leveranciers moeten zorgen. Eén daarvan is de Web Feature Service (WFS) specificatie. Deze specificatie is een unieke, open standaard, omdat het de enige standaard is die een interface beschrijft die niet alleen het bevragen van geodata (een zogenaamde Basic WFS), maar ook het bewerken van geodata (Transactional WFS) via internet ondersteunt.

Het OpenGIS Service Framework verschaft het kader voor OpenGIS Web Services. Dergelijke open, interoperabele services gebruiken standaard internet technologiën, bijvoorbeeld HTTP en XML. Met deze services kunnen applicaties dynamisch worden samengesteld. Web Feature Services behoren tot OpenGIS Web Services.

Web Feature Services worden Basic WFS genoemd als data alleen kan worden opgevraagd. Men spreekt van Transactional WFS als de service ook operaties voor het creëeren, verwijderen en wijzigen van objecten ondersteunt op alle of op een deel van de objecten in de database. In de WFS specificatie worden ook zogenaamde locking operaties gedefinieerd, die gebruikt kunnen worden om de consistentie van een dataset te behouden bij transacties.

Geography Markup Language (GML) is een op XML gebaseerde codering om geografische informatie te modelleren, transporteren en op te slaan. Met GML kunnen zowel de ruimtelijke als de niet-ruimtelijke eigenschappen van geografische objecten beschreven worden. WFS maakt gebruik van GML versie 2.1.2 als output formaat en om nieuwe geografische objecten te beschrijven.

De term *geometrische transactie* wordt in deze scriptie gebruikt om transcaties aan te duiden die bestaan uit operaties om geografische objecten, opgeslagen in een geo-database, te bewerken. Juist vanwege de geometrische eigenschappen is het moeilijk om geometrische transacties te verwerken. Hierbij spelen niet alleen reguliere problemen voor geo-databases een rol, zoals het beschrijven van de objecten of berekeningen met ruimtelijke operatoren. Met name de relaties van geografische objecten met objecten in de directe omgeving maken het moeilijk transacties te verwerken. Het betreft hier relaties die ofwel expliciet opgeslagen zijn in een topologische structuur of andere ruimtelijke voorwaarden, ofwel impliciet volgen uit de geometrische eigenschappen zelf.

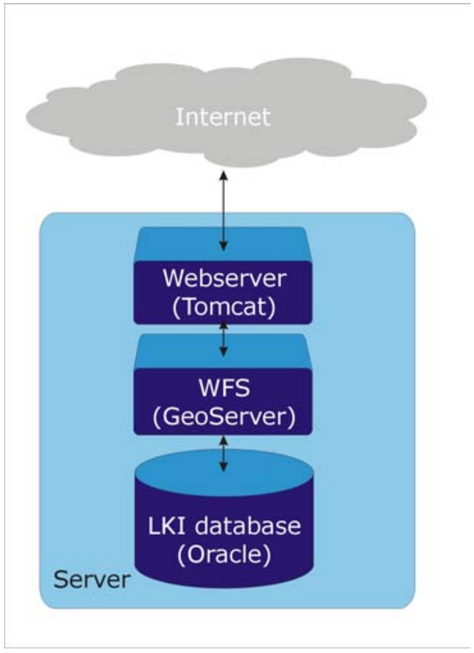
Aangezien Web Feature Services gebruikt kunnen worden om geografische data via internet te bewerken, moet het mogelijk zijn om met WFS geometrische transacties te verwerken. In de hoofdonderzoeksvraag van dit afstudeeronderzoek is dit als volgt geformuleerd:

Hoe moeten met OpenGIS Web Feature Services geometrische transacties verwerkt worden in een gedistribueerd GIS?

Om deze vraag te beantwoorden is literatuuronderzoek verricht en is een case studie gedaan waarbij een Web Feature Service is ontwikkeld. Met deze service zijn de mogelijkheden om met WFS geometrische transacties te verwerken geanalyseerd.

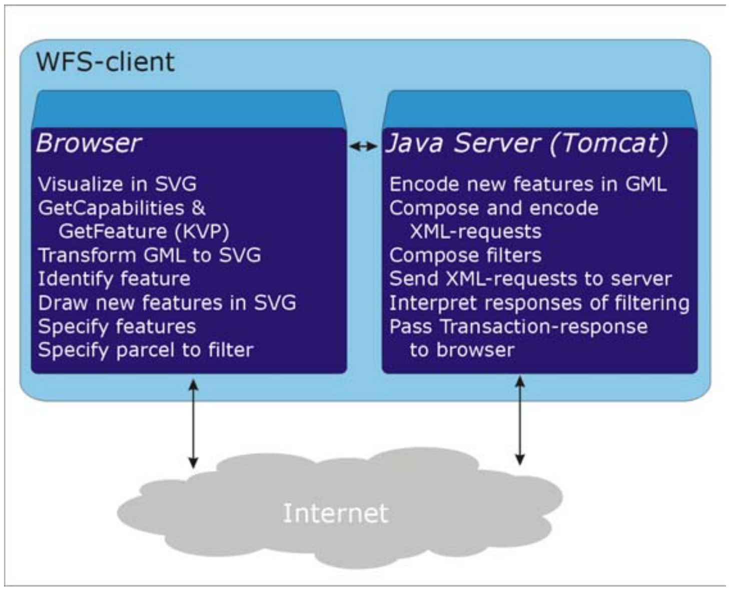
Het principe waarbij een notaris kadastrale geodata bewerkt is gebruikt in de case studie. Een notaris zal bijvoorbeeld nieuwe perceelsgrenzen schetsen bij de splitsing van een perceel. Voor deze case zijn zowel een web feature server als een cient ontwikkeld.

De server bestaat uit een Oracle Spatial 9i database. Hierop is GeoServer (een Open Source web feature server) geconfigureerd. GeoServer is geïnstalleerd in Tomcat, een Java servlet container. Via Tomcat wordt de server ontsloten naar internet, zie figuur 1.



Figuur 1 De ontwikkelde web feature server

Aangezien reeds bestaande WFS-clients niet volledig voldeden aan de specificatie, niet Transactional waren of omdat het te veel tijd zou kosten de clients aan te passen (indien mogelijk), is er besloten zelf een client te ontwikkelen tijdens het afstudeeronderzoek. Deze client gebruikt een standaard browser en Scalable Vector Graphics (SVG) om de data te visualiseren en objecten toe te voegen. Met Java Server Pages (JSP) is functionaliteit toegevoegd om transactie-requests op te stellen en te versturen. De functionaliteit van de client is dus verdeeld over een browser en een Java web server, zoals weergegeven in figuur 2.


WFS-client
Browser
Visualize in SVG
GetCapabilities & GetFeature (KVP)
Transform GML to SVG
Identify feature
Draw new features in SVG
Specify features
Specify parcel to filter
Java Server (Tomcat)
Encode new features in GML
Compose and encode XML-requests
Compose filters
Send XML-requests to server
Interpret responses of filtering
Pass Transaction-response to browser
Internet

Figuur 2 De functionaliteit van de client is verdeeld over een browser en een Java web server

Het testen op interoperabiliteit van de server met andere clients, heeft laten zien dat niet alle huidige clients, die wel claimen te voldoen aan de WFS-specificatie, correct werken. Echter, in de tests is ook de kracht van interoperabele services bewezen. Immers, een generieke client, die geen enkele kennis had van de dataset, kon wel verbinding maken met de database en ook data ophalen via WFS.

De case studie heeft ook laten zien dat in het algemeen de WFS-specificatie voldoende operaties bevat om een geometrische transactie in zijn geheel te versturen van de client naar de server en dat die transactie op de server verwerkt kan worden. Met reguliere internet technologiën als HTTP, XML en GML kunnen geografische objecten bevraagd, gewijzigd, gecreëerd en verwijderd worden. Echter, WFS mist operaties om goed om te gaan met voorwaarden en andere applicatie-specifieke zaken. Daarnaast missen er enkele operaties om op gecompliceerde data sets transacties te kunnen afhandelen.

De slotconclusie moet zijn dat, ondanks dat WFS niet geavanceerd genoeg is om transacties op gecompliceerde datasets (bijvoorbeeld met een topologische structuur of andere ruimtelijke voorwaarden) te ondersteunen, Web Feature Services toch krachtige, interoperabele GIS web services zijn om geografische data te bewerken via Internet.

1 Introduction

Over the last decade the need for sharing already available, but distributed geographic data and the need for facilitating the integration of Geographic Information Systems (GIS) with other information systems has been growing [2]. In such a distributed environment, the ability to edit geographic data (i.e. besides querying, also creating, updating and deleting geographic features) is essential.

The need for integration and interoperable geoprocessing between various (components of) GISs is also recognized by the OpenGIS Consortium (OGC) in its vision [8]. Interoperable geoprocessing refers to the ability of digital systems to 1) freely exchange all kinds of spatial information about the Earth and about objects and phenomena on, above, and below the Earth's surface; and 2) cooperatively, over networks, run software capable of *manipulating*[1] such information [8].

In this context, the OpenGIS Consortium recently has developed several GIS web services. GIS web services are interoperable services running over the Internet, for example to retrieve maps or query geodata. Web Feature Services (WFS) are developed by the OGC and can be used for operations such as [23]:

1) Getting or querying features based on spatial and non-spatial constraints
2) Creating a new feature instance
3) Deleting a feature instance
4) Updating a feature instance.

The WFS specification thus provides a standardized interface not only for querying (operation 1) but also for manipulating geographic data (operations 2, 3 and 4). This is a unique open standard, since no other open standard supports editing of geodata over the Internet.

Geography Markup Language (GML) is an eXtensible Markup Language (XML) grammar written in XML-Schema for the modeling, transport, and storage of geographic information [27] and is developed by the OGC. Geographic data can easily be exchanged between systems since GML must be used to express features within the WFS interface [23] and GML is also the preferred format in which data should be send to the client.

Research problem

Transactions can be used to make sure that modifications of several users do not interfere. Geometric transactions consist of one or more operations that edit geographic data. All operations in the transaction have to succeed to make the transaction succeed otherwise the transaction fails. With a Web Feature Service it should be possible to process geometric transactions. This research project focuses on 1) how geometric transactions should be processed in a distributed GIS and 2) how WFS deals with this kind of transactions. This is expressed in the main question:

How should geometric transactions be processed in a distributed GIS using OpenGIS Web Feature Services?

[1] Where 'manipulating' also refers to 'editing'

This problem is subdivided in 4 questions:

a) What are the main characteristics of a distributed GIS and GIS web services?
b) How should geometric transactions be processed in a distributed GIS environment?
c) What are OpenGIS Web Feature Services?
d) To what extent are Web Feature Services suitable for processing geometric transactions?

Questions a), b) and c) have been answered by studying literature. In a case study a Web Feature Service that is able to support simple cadastral transactions has been developed and tested to answer question d). With that service a notary should be able to sketch a provisional boundary and parcels, e.g. in order to support the process of splitting a parcel into tow or more parcels.

As far as known at this moment, no transactional web feature service has been realized and used to analyze WFS's transactional abilities. Therefore the developed service in the case study can be used to get some first practical experiences with transactional WFS.

Constraints

Developing an extensive graphical user-interface does not have a high priority in this research project. More specific, this means that, although a WFS client is needed and has been developed, the focus is on developing a working web feature service able of processing transactions in a distributed (and possibly heterogeneous) environment. Security issues are important, especially in cadastral transactions, but not much attention will be paid to them either. Existing methods for securing the data are assumed to be sufficient.

Structure of the thesis

This thesis starts with a theoretical basis in chapters 2, 3 and 4. Chapter 2 is an introduction on distributed systems and more specifically distributed GIS. It also deals with interoperable geoprocessing. A key-issue in interoperable geoprocessing is editing data. As stated before, geometric transactions consist of operations to edit geographic data, which is the subject of Chapter 3. That chapter describes issues that should be accounted for when dealing with geometric transactions. In Chapter 4, technologies developed by the OGC that aim to support interoperable geoprocessing (in the context of web services) are discussed. Special attention is paid to Web Feature Services and the transactional characteristics.

In Chapter 5 the case study that has been performed is described. In addition to scenarios for a notary to split a parcel, it also provides a description of both client and server that have been developed in this research. In chapter 6 the results from the case study will be analyzed and will be compared to the theoretical basis from the first chapters. Chapter 7 summarizes the most important conclusions and gives some recommendations for future research and development.

2 Geographic data in distributed environments

In general, in distributed (database) environments data are shared over some kind of network. This chapter deals with issues of distributed environments and sharing geographic data sets in particular. It first defines geographic features. The need for sharing this geo-data is then illustrated in some examples. Basics on distributed systems and especially the client-server architecture are provided in section 2.2. To share the data over these networks, system components have to work together – interoperate - in some way. Open Systems (and GIS Web Services as a special case of Open Systems) use standards that support this interoperability. These topics, interoperability, Open Systems and GIS Web Services, are dealt with in section 2.3 on Interoperable geoprocessing. Section 2.4 gives some final remarks on these subjects.

2.1 Sharing geographic data sets

2.1.1 Geographic features

Data are at the heart of a Geographic Information System [36]. In order to provide spatial data for the system, the real world has to be modeled and captured. Two principal approaches for modeling exist: the field-approach and the object-approach [18, 31], which is represented in figure 2.1.

According to Molenaar [18], in the *field model* the earth's surface is considered as a spatio (-temporal) continuum. Terrain aspects are represented in the form of attributes, which are position dependant. In the data set, such a field is most often represented in a grid or raster.

The *object model* assumes that terrain features (i.e. the real world) can be defined as objects having geometric characteristics (a location or position and shape and/or topology) and several non-geometric characteristics. An object consists of thematic data and geometric data and can be identified by a unique identifier. Temporal data are considered an inherent aspect of geographic information as well [36].

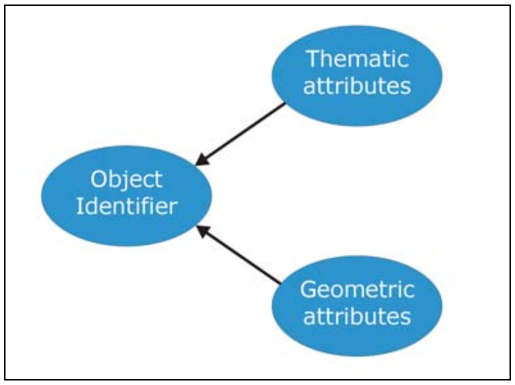


Figure 2.1 Geographic object (from [18])

This thesis almost exclusively deals with datasets containing objects. Here in most cases geographic objects are referred to as geographic features, which corresponds to the vocabulary of the OpenGIS Consortium (OGC). In the Geography Markup Language specification [27] the OGC defines a geographic feature as:

> *A geographic feature is an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth.*

This does not correspond with the object view very clear, but the link between geographic features and geographic objects is made in the explanation of this. In the GML 3.0 specification [27] is stated that a digital representation (e.g. in some geo-dataset) of the real world can be thought of as a set of features. The state of a feature is defined by a set of properties, where each property can be thought of as a {name, type, value} triple. The number of properties a feature may have, together with their names and types, are determined by its type definition.

2.1.2 Sharing geographic data

One can imagine that, in the course of time, a large set of heterogeneous, isolated geographic information systems and geographic datasets have grown [6]. On the other hand there is the necessity of sharing information from different databases [6, 15]. This is not only to avoid duplicating existing data (and therefore to avoid making extra costs [32]), but to benefit from the latest updates located in other databases as well [6, 15, 31]. The best quality can be found at the source.

This brings us to the issue of data integration. Based on [32][2] and [6], data integration can be defined as:

> *Data integration is bringing together spatial data that are stored in different forms and managed by different systems from a number of sources in order to allow users and applications to combine information from multiple sources correctly.*

There are however several problems concerning data integration. These difficulties can be divided in three categories [12, 26]:

1) Bringing together diverse information from a variety of sources.
2) Matching of supposedly similar entities in these sources (semantic integration).
3) Resolving inconsistencies across the source datasets.

This thesis deals with the first problem domain and especially in the context of distributed and possibly heterogeneous environments (Chapter 2) and GIS Web Services (Chapter 4). Uitermark [31] has developed a methodology to integrate spatial data semantically. That research thus focuses on the second problem.

[2] Vckovski [32] says on the various meanings of 'data integration': *"The term 'data integration' has been used with various meanings ranging from technical issues such as inter-conversion of raster and vector models, cartographic notions to general definitions such as bringing together of spatial data from a number of sources and disciplines, from different subsystems "each employing a different technique of data input from different media [3]"'.*

2.1.3 Some cases

As stated before, the need for dynamically combining data from several sources and editing (parts of) that data has grown. Sharing data in this context not only applies to retrieving the data and combining it, but sharing data can also include editing data. The following fictitious, but realistic examples illustrate this.

Update propagation

Top10NL contains topographic data on a scale of 1:10,000 for the Netherlands. GBKN is the Large Scale Base Map of the Netherlands (in Dutch: Grootschalige BasisKaart Nederland) and contains data on a scale of 1:500 or 1:1000 for urban areas and 1:2000 for rural areas. Now let's say that it is desired to maintain Top10NL partially based on GBKN-mutations[3]. If GBKN could propagate its mutations to Top10NL (i.e. send the mutations in some format to Top10NL or Top10NL could request all mutations in a certain time interval from GBKN), mutations to Top10NL could easily be made as well[4]. At least for Top10NL is signalled that objects in the real world have changed and steps could be taken to update the dataset.

When this simple network is extended with other datasets, let's say the national buildings database, an address database, etc., a large network of all kinds of geographic reference/base data could exist. See figure 2.2. In this network mutations in one dataset could (easily) be communicated to other datasets and thus create benefits, like lower maintenance costs and more up-to-date data, for all maintainers of the datasets.

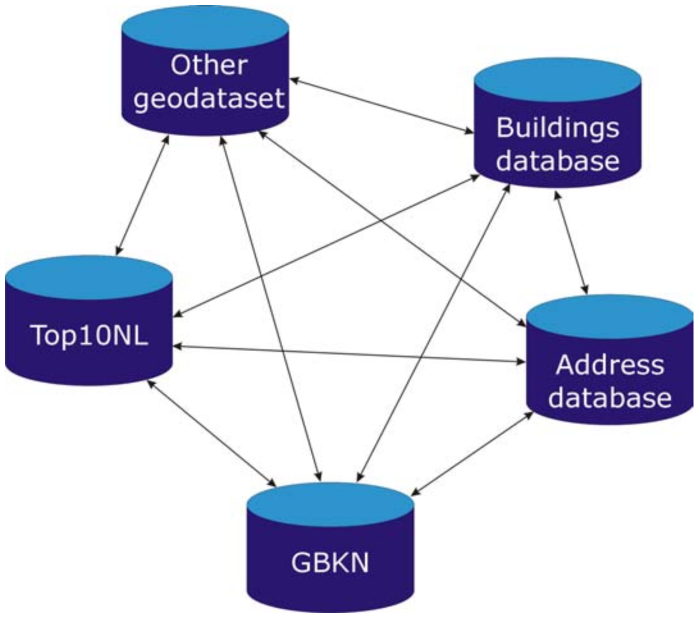


Figure 2.2 Network of geo-databases for mutation propagation

[3] The GBKN dataset is maintained by several institutions. In a way this dataset is distributed amongst its maintainers/producers as well.

[4] Note that semantics are not considered here, while these are critical for mutation propagation. Integrating the mutation-data thus will not be accomplished just by sending/receiving the mutations.

Reporting errors for car navigation

The use of car navigation has become more and more widespread in the last few years. But for data/map producers it is almost impossible to provide completely up-to-date data – i.e. geographic data of all roads in some area - without any errors on the media they use (e.g. CD-ROM's). However, keeping the data up-to-date and free of errors is essential for both producers and users. One could imagine that a producer wants to be notified of errors as soon as possible. If users could report where (the location of) errors exist in the database, data producers could quickly improve their dataset. In addition users wish to have the most recent data in their navigation systems. Users and data producers could form a huge network in which users report (the location of) errors to the central geo-database containing the road data and in return could download the most recent data to their navigation system. Figure 2.3 is a schematic representation of an example of such a network.

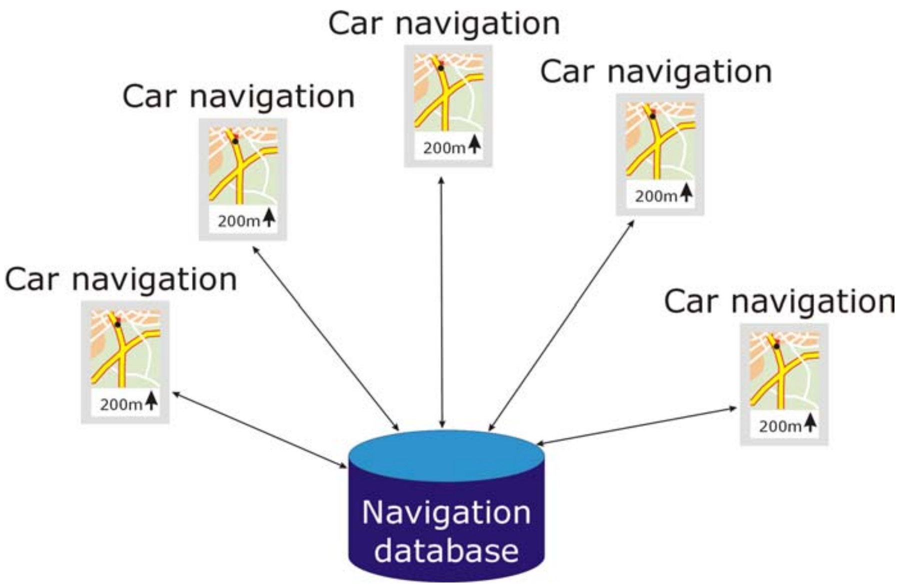


Figure 2.3 Network of car navigation users and geo-database for car navigation to report errors and get latest updates

GPS-tracks

Similar to car navigation, handheld GPS receivers are getting more and more popular as well. These handheld receivers are often used in outdoor sports and recreation, e.g. hiking, sailing or cycling tours. Lots of people like to exchange these tracks. As a result, in the last few years numerous websites have emerged that want to be a platform for the exchange of these GPS tracks. On such websites, people can upload and download their tracks. At the moment, this is most often a file-based system: text-files are uploaded and downloaded. But geo-databases could be useful to run underneath such a website, e.g. for searching or combining tracks.

Notary splits parcels

In the MSc thesis research project a case study has been performed. This case study investigated how the process of splitting a parcel could be supported by letting the notary sketch the new (preliminary) boundary and adding some information on new parcels. The notary could do this from his own office via an Internet connection. This is stored in a database from which for instance the surveyors can extract the areas where they need to measure new boundaries. Chapter 5 describes this case study in more detail.

2.2 Distributed systems

Out of the need to access and change information resources of other computer systems, distributed computing has arisen [36]. Section 2.2.1 gives an introduction on distributed (geo)database systems and deals with some definitions and other issues of distributed systems. The client-server architecture is often used in distributed systems and therefore discussed in section 2.2.2. Section 2.2.3 gives the main advantages and disadvantages of distributed computing.

Definitions given in this section are listed in the Glossary (Appendix A) as well. For databases in general, most of the definitions are taken from Sheth and Larson [30]. For issues related to geographical databases most definitions are based on the definitions of Buehler and McKee [8].

2.2.1 Basics of distributed (geo) database systems

In general, a distributed system allows its components (e.g. users, pc's, databases) to cooperate in some way. Sharing and managing data, applications and operations are examples of this. Several more formal definitions of a distributed system exist. The following definition describes best what is meant with distributed systems in this thesis:

> *A distributed system is a collection of autonomous computers linked in a network, together with software that will support integration [36].*

In such a distributed system the components can for instance share data or operations. For other definitions, especially on types of distributed databases, see [30] and [16].

An important characteristic of a distributed system is whether it is homogeneous or heterogeneous. A distributed (geo-)database system is called *homogeneous* if the component (geo-)databases are managed by the same type of software and *heterogeneous* otherwise [30]. When the database is heterogeneous it is more complex for the system as a whole to share data, because barriers to communicate and exchange information have to be overcome.

Data may be distributed among multiple databases in different ways [30]. Three types of distributed databases are discussed here in a little more detail: *replicated databases*, *federated databases* and *independent data servers*.

If multiple copies of some or all of the data are stored at multiple sites, the database is called a *replicated database* [5]. The main reason for using replicated data is to increase database availability. By storing critical data at multiple sites, the database can operate even though some sites have failed. Another goal is improved performance. But, since there are many copies of each data item, a transaction (manipulation of the data) is more likely to find the data it needs close by, as compared to a single copy database. This benefit is mitigated by the need to update all copies of each data item. In addition, keeping the copies consistent is also an issue. Thus, retrieving data is faster, editing data is slower [5].

Another type of distributed database is the *federated database system*. Sheth and Larson [30] treat federated databases extensively. Buehler and McKee [8] give the following description: federated databases are *separate* databases that are structured, perhaps with middleware or special database access software, in such a way that they can be queried as a single database. Federated databases are used when already existing databases, spread amongst different organisations, must co-operate or interoperate [16]. Therefore there must be agreement on the overall database schema and on which different locations manage which different parts of the data.

Influenced by the growth of the Internet, databases are more and more accessible over the Internet. These databases often are independent of each other, but still can serve a similar purpose or in the same domain. In other words, these databases act as *independent data servers* that can be accessed by end-users to combine data [4]. In such an infrastructure end-users should be able to search for the appropriate data by searching the meta-data data providers publish for their databases in catalogues. [4] describe such a model in the context of Internet GIS. An important advantage of this model is that data are managed at the source. There is no need anymore to manage copies of the data, which reduces the risk of inconsistencies in the database(s).

2.2.2 The client-server architecture

In the context of architectures for distributed database management systems (distributed DBMS), the client-server architecture is natural and widespread [36]. This holds also for Internet-based systems - as a special case of distributed systems, i.e. in a very large network. Because of this and because Web Feature Services are based on a client-server model, client-server computing is considered in this thesis.

The client-server architecture is a simple model of computing, in which system functionality is divided among the components that make requests (the clients) and the components that respond to them (the servers) [6]. Clients are elements that require services from the server. Mostly the server holds data and services available for transmission to the clients in the network [36]. Thus, servers can be defined as entities that own the resource [6]. The servers are usually powerful machines, capable of running complex system software. Clients typically may be PCs or workstations [36].

Buehler and McKee [8] give a less strict, but understandable description of the client-server principle:

> *"The network computing revolution (which includes the distributed geoprocessing revolution) is based on software entities (clients) that tell other software entities (servers) to do things for them. Software clients say, "Send me this specific data from your database!" or "Tell me what Internet address contains this information!" or "Take this data and do a correlation operation on it!" In a simple sense, your word processor is a client when you click on "Save" and the word processor instructs the operating system*

(acting as a server) to save your file to disk. Interoperability interfaces make it possible for diverse computers to request things of each other over networks and get predictable responses."

Note that Buehler and McKee link client-server computing and interoperability directly. Interoperability is discussed in section 2.3 in more detail.

An important concept in client-server computing is mediation. Wiederhold [35] discusses mediation elaborately, for this thesis it is sufficient to discuss some definitions and a few characteristics. Client-server applications mostly apply a 2-tier model [6]. The 2-tier model consists of an application front-end, which includes the graphic user interface and the application code, and a database at the back end, or a client and a server, respectively. A 3-tier model consists of an application, a layer called mediator, and a database [6, 30] as shown in Figure 2.4.

Wiederhold [35] defines a mediator as software that exploits encoded knowledge (application logic) about some sets or subsets of data to create information for another layer (e.g. to do some translation or for a higher layer) of application. Such a user application could be geographic data set integration [31].

Many mediators will access multiple databases to combine disjointed sets of data prior to analysis and reduction [35]. When accessing multiple databases the mediator needs to deal with semantic conflicts, because a mediator needs to 'understand' all these databases and these databases might have different (semantic) models [6]. This explains why, according to Wiederhold [35], the term mediation includes not only the interfaces between the user application and database servers but also the knowledge structures needed to transform data and any intermediate storage in order to deal with the abstraction and representational issues.

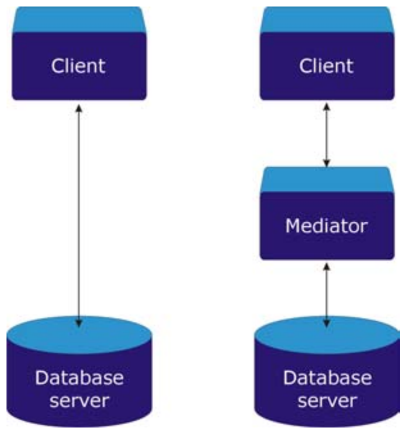


Figure 2.4 The concept of client-server computing in a 2-tier (left) architecture and 3-tier architecture using a mediator (right)

2.2.3 Advantages and disadvantages of distributed systems and client-server computing

For GIS applications, client-server computing holds many possibilities and GIS applications are often thought of naturally in a distributed context [36]. For example, every municipality maintains its own data, but that data could be used in nationwide systems as well. But the advantages must be weighed against the increased complexity of the distributed DBMS and client-server architecture [36]. Some important advantages and disadvantages of distributed systems and client-server computing are briefly discussed below.

A big advantage of distributed databases is that they allow the data to be shared globally while keeping control over locally kept data, for example on database modification, with the local sites. According to Worboys [36], distributed databases therefore are appropriate in cases where the data are themselves naturally distributed (regional environmental data, for example). He gives several other advantages of a distributed environment:

- In a distributed environment commonly occurring accesses to the local site from local users will be more efficient. This assumes that redundant copies of the data are used.
- Reliability of the system may be improved by distribution, because if a system goes down at one site, usage can still continue for the rest of the database. Here also is assumed that copies exist of the data.

Laurini [15] adds that, using a client-server architecture:

- There is the possibility to optimize each module and each of the modules can evolve separately.
- The volume of transferred data can be reduced.

If copies are avoided, a big advantage is that the data are managed at the source [4]. As stated before, this reduces the risk of inconsistencies in the data.

A disadvantage is that distributed databases have a more complicated structure to support, for example, handling queries where the data are fragmented across sites, or maintaining consistency of commonly accessed data items that are replicated across sites [36]. The distributed DBMS is correspondingly complex. Worboys [36] gives some examples of additional functionality that is required. The most important for this research is that remote access and transmission of queries and data using a communications network have to be supported. For this research these disadvantages are less relevant, since only one database is used, as in the model with independent data servers.

A difficult technical problem is how to connect *heterogeneous* databases to each other [15]. If there are *n* databases and if each of them is connected to all the others, *n(n-1)* connections have to be established. That implies that *n(n-1)* data converting or transforming systems need to be defined, otherwise the systems can not interoperate. It may be clear that this can be an enormous task, when the system consists of a large number of heterogeneous databases. This is where standards can play an important role. Standardization, interoperability and open systems will be discussed in the next section, 2.3.

2.3 Interoperable geoprocessing

2.3.1 Interoperability and interoperable geoprocessing

From the sections above it may be clear that there is a need to share and integrate (geographical) data from multiple, heterogeneous database systems. In order to do so, the systems need to interoperate in some way. Uitermark [31] distinguishes two different levels of interoperability, also see the first two points at page 4 of section 2.1.2. There is a technical level — or, the systems perspective — with an understanding of information processing issues, like network *protocols* and *standards* for data set files. And there is a semantics level — or, the data modeling perspective — with an understanding of the *semantics* of information processing.

In this thesis the technical level is considered. Interoperability refers to this definition of interoperability [8]:

> *Interoperability is the ability for a system or components of a system to provide information portability and interapplication, cooperative process control. Interoperability, in the context of the OpenGIS Specification, is software components operating reciprocally (working with each other) to overcome tedious batch conversion tasks, import/export obstacles, and distributed resource access barriers imposed by heterogeneous processing environments and heterogeneous data.*

Interoperability refers to the capability for applications running on dissimilar computers to exchange information and operate cooperatively using this information [6]. Bishr e.a. [6] also define application interoperability. With application interoperability they mean that two, or more, spatial information systems can exchange data seamlessly and users can query remote databases without any knowledge of their underlying data models. This makes chaining of services from different sources possible. This concept is discussed in section 2.3.3.

Portability refers to the ability to move from one system to another, without having to make (major) changes. Buehler and McKee [8] define, for example, user portability as “the ability of a user to move from one system to another without having to learn everything again”. Simultaneously, information portability could be described as the ability of a system to exchange information or data with another system. Bishr e.a. [6] relate portability to soft- and hardware only: according to them portability refers to the capability for software to run on different types of hardware.

As said before, the ability to edit geographic data (i.e. besides retrieving, also creating, updating and deleting geographic features) is essential. Buehler and McKee’s [8] definition of interoperable geoprocessing makes clear what interoperable systems in the context of GIS should be able to do:

> *“Interoperable geoprocessing” refers to the ability of digital systems to 1) freely exchange all kinds of spatial information about the Earth and about objects and phenomena on, above, and below the Earth's surface; and 2) cooperatively, over networks, run software capable of manipulating such information.*

Thus, interoperable geoprocessing is concerned with retrieving and editing geodata. Note that 'freely' here refers to 'free of troubles' and that fees may be asked to use a service.

Because of the *complexity* of current computing systems and because of systems being heterogeneous, standardization is required to realize interoperable systems [16, 32]. An important characteristic of a standard is that a standard provides - among other things - a break-down of the complexity, and more importantly, an agreed break-down [32]. Vckovski [32] argues that these standards are not just for external use. It can be necessary to provide "internal" standards when designing complex systems, creating syntactically and semantically well-defined interfaces between systems, problem domains, development groups and enterprises. These internal standards could for instance be useful in the cases mentioned in section 2.1.3: for example for mutation propagation between many large datasets or the network in which notaries sketch new boundaries in a cadastral database.

2.3.2 Problems with interoperability

The problems in sharing information in Geo-Information Infrastructures (GII) and interoperability [5] can be subsumed under two main categories [6]:

- *Political, Institutional and Economic problems*: The provision of information for the public requires legal considerations (e.g., copyright), proper institutional organization and data access rules, as well as pricing schemes.
- *Technical problems*: These include devising techniques for the provision of up-to-date inventory of the available data, mechanisms for seamlessly sharing information, update and consistency constraints and semantics.

The technical problems are the most relevant for this thesis. Bishr e.a. [6] divide the technical problems in two categories:

- Data modelling (which includes data migration and integration).
- System architecture.

According to Vckovski [32], the heterogeneities between the component datasets in a distributed system lead to problems that are common both in data migration and integration. These problems need to be tackled, to make the component systems interoperable. The heterogeneities that cause the problems exist on various levels, such as [32]:

- Different computer hardware and operating systems.
- Different DBMS software.
- Different concepts for data modelling, e.g. relational models, object-oriented models.
- Semantic heterogeneities.

The first three types are syntactical in nature. They occur at the software and hardware level, while the semantic heterogeneities usually occur at the application level [6]. For problems concerning data modelling, this subdivision in syntactical and semantical

[5] Because of the close relation between sharing information in a GII and interoperability [6], this categorisation can be applied to problems with interoperability as well.

problems is common (see [6, 32] and to a certain extent [15] as well). However, differences exist in classifying problems as syntactical or semantical.

Syntax concerns the format (the language, so to speak) in which information is represented. In the context of geographic information Bishr e.a. [6] discuss syntactic constructs. Syntactic constructs include spatial primitives to represent the geometry (e.g., nodes, arcs and surfaces) and topological relationships (e.g., interior, exterior or neighbourhood) of real world features. It also includes constructs to represent the thematic properties and relationships of the real world features (e.g., classes, subclasses, aggregation, associations, generalization or attributes).

Syntactic problems arise when differences exist in encoding the data [32], i.e. when the syntactic constructs of different systems are dissimilar. For example, the geometry may be limited to a few basic types such as points and line strings. Or alternatively, it may include much more sophisticated types such as grids, fields, curved primitives or polyhedrons - in case of 3D models. As with the geometry, spatial (and temporal) referencing may be accomplished in a variety of ways [6].

Differences in syntax are still a major problem area within the exchange of geodata. These issues however, are addressed by many format standardization efforts [32]. The XML-based data transfer standard developed by OGC, GML, should tackle this problem. GML will be discussed in section 4.2.3.

Semantic differences between geospatial databases occur when there is a disagreement about the meaning, interpretation, or intended use of the related objects [6]. For example, a building in system A can be defined differently in system B. When data from both systems need to be combined, the syntax might be the same, but the meaning of "building" (i.e. the semantics) is not. In addition, the enormous variety of *encodings* of geospatial semantics makes it particularly challenging to process requests for geospatial information [10].

The semantic diversity of spatial representations causes problems, which can become more complex than the syntactical problems [14, 32]. Solutions for these problems could for instance be found using ontologies, object classes and/or mediation. Uitermark [31] treats these issues in the context of data set integration. Because this thesis focuses on the technical level of interoperability, these issues will not be discussed further.

When information needs to be shared, several problems with respect to the system architecture may arise [6]:

- Each database management system has its own functionality and interfaces.
- The databases may be installed on different platforms, which support different network protocols.
- The application protocol, which defines the way two or more components (e.g. clients or servers) communicate, may present a problem.

2.3.3 Open systems and GIS Web Services

According to Bishr e.a. [6], software systems that are built on standards for portability and interoperability are called open systems. Buehler and McKee [8] demand more than portability and interoperability to call a system an open system:

> *An open system is a system that implements open interface specifications and standards that promote application portability, scalability, interoperability, diversity, manageability, extensibility, compatibility with legacy components, and user portability.*

An open system standard is an interface specification - a specification that describes services provided by a software product - to which any vendor can build products [6]. There are two important points for open system standards. First, the specification is available to any vendor and evolves through a consensus process that is open to the entire industry. Second, the specification defines only an interface; so different vendors can provide the standard interface on their proprietary systems. In addition, the specification must be detailed enough such that two independently developed components can work together based on the specified interface.

GIS Web services

GIS Web services can be considered a special type of open systems. Web services are self-contained, self-describing, modular applications that can be published, located, and dynamically invoked across the web [14]. Web services use the Internet as communications network.

Web services provide access to sets of operations accessible through one or more standardized interfaces. In the process, services may use other external services and operations. GIS Web Services can be grouped into three basic categories [1]:

- *Data services* (such as the OGC Web Mapping, Web Coverage and Web Feature Services) offer customized data to users [22]. These services are tightly coupled with specific data sets.
- *Processing services* are not associated with specific datasets. Instead they provide operations for processing or transforming data in a manner determined by user-specified parameters [2]. Such services can provide generic processing functions such as projection/coordinate conversion, rasterization / vectorization, map overlay, imagery manipulation, or feature detection and imagery classification.
- *Registry / catalogue services* are used to classify, register, describe, search, maintain and access information about Web services [22]. Types of registries are differentiated by their role such as registries for cataloguing data types, online data instances, service types and online service instances. The contents of registry / catalogue services are metadata describing other services.

In order for a sustainable and extensible GIS Web Services architecture to exist, the basic services should be accessed via standardized interfaces.

Alameh [1] argues that, once GIS Web Services are deployed, client applications can be built more flexibly by mixing and matching available services. Each client application is created by using multiple GIS and non-GIS Web Services like a service to pay for GIS services.

This service-based model is rapidly materializing as a result of the advancements in general web service technologies and the focused efforts of the Open GIS Consortium (OGC) in the areas of service categorization and interoperability of service interfaces [1, 8]. One of these technologies - Web Feature Services (WFS) – and other components - like GML to exchange the data and a spatial DBMS (in the case Oracle has been used) -

are subject of this thesis. WFS and other relevant standards developed by the OGC are discussed in chapter 4.

2.4 Final remarks

This chapter has dealt with geographic data in distributed environments. More specifically, it showed the need for sharing geographic data from heterogeneous databases and gave some basics on distributed systems. Interoperable geoprocessing and GIS Web Services are important subjects in this context. One of these types of services, Web Feature Services, should make editing of geographic data in a distributed, heterogeneous environment possible over the Internet. In order to edit data in a multi-user environment, transactions can be used to make sure that modifications of several users do not interfere. Transactional Web Feature Services are the subject of Chapter 4. But first in Chapter 3 geometric transactions will be discussed.

3 Geometric transactions

Chapter 2 has made clear that there is a need to share geographic data in a distributed environment, but there are a lot of problems to overcome. This chapter deals with geometric transactions, which is the subject of the research question 'How should geometric transactions be processed in a distributed GIS environment?'. Transactions are often used to deal with problems with edits on a dataset in a multi-user environment. In this chapter regular database transactions will be discussed first in section 3.1. Transactions on geographic features (geometric transactions) are more complicated. Why geometric transactions are more complicated than regular database transactions is explained in section 3.2. Section 3.3 summarizes the most important conclusions of this chapter.

3.1 Database transactions

This section (3.1) is a summary of [5]. Relevant parts of [5] for this thesis have been adapted.

3.1.1 Concurrency control and transactions

Commands that access the database are called database operations[6]. In SQL-terms (SQL stands for Structure Query Language, also see Appendix A: Glossary) these commands are constructed with SELECT, INSERT, UPDATE and DELETE-statements. In a multi-user environment, it is possible that these database operations apply to the same data in a database. The changes one user likes to make might interfere with changes that another user is making, which can result in inconsistencies in the data if no care is taken.

Concurrency control can avoid this kind of problems with interfering operations. Concurrency control is the activity of coordinating the actions of processes that operate in parallel, access shared data, and therefore potentially interfere with each other. The goal of concurrency control and recovery is to ensure that transactions execute atomically, meaning that

1) each transaction accesses shared data without interfering with other transactions, and
2) if a transaction terminates normally, then all of its effects are made permanent (the transaction is committed); otherwise it has no effect at all (the transaction is aborted and the database has to recover to the state it was in before the start of the transaction).

Transactions can consist of several database operations. It can be important that all operations in a transaction are processed together. This is why the database should be able to recover to the old situation in case a transaction fails. Otherwise, inconsistencies might occur in the database.

[6] These operations form a Data Manipulation Language (DML). Operations to define and change tables, databases, indexes, etc. are included in a Data Definition Language (DDL).

3.1.2 Processing transactions

For processing transactions, the database management system (DBMS) should support operations to start, commit and abort transactions. A program tells the database that it is about to start executing a new transaction. When all operations of the transaction - which is a series of update, delete and insert statements that belong together - have terminated normally, the transaction is committed to make all effects permanently. Until the transaction is committed, the changes are not visible for other users. Otherwise, the transaction is aborted and all effects of the transaction must be made undone.

When two or more transactions execute concurrently, their database operations execute in an interleaved fashion. That is, operations from one program may execute in between two operations from another program. This interleaving can cause programs to behave incorrectly, or interfere, thereby leading to an inconsistent database. The goal of concurrency control is to avoid interference and thereby avoid errors.

One way to avoid interference problems is not to allow transactions to be interleaved at all. An execution in which no two transactions are interleaved is called *serial*. From a user's perspective, in a serial execution it looks as though transactions are operations that the database processes atomically. Serial executions are correct because each transaction individually is correct and transactions that execute serially cannot interfere with each other.

For performance reasons, it might be necessary to allow for non-serial (that is concurrent) executions. Such executions are called *serializable*. More precisely, an execution is serializable if it produces the same output and has the same effect on the database as some serial execution of the same transactions. Serializable transactions are correct transactions in the sense that they do not interfere, e.g. updates are performed on different tables or on different records in one table.

In a database system, a scheduler controls the order in which database and transaction operations are executed. A scheduler is thus responsible for ensuring transaction serializability. To achieve this, *locking* can be used.

Locking is a mechanism commonly used to solve the problem of synchronizing access to shared data. The idea behind locking is simple. Each data item has a lock associated with it. Before a transaction T_1 may access a data item, the scheduler first examines the associated lock. If no transaction holds the lock, then the scheduler obtains the lock on behalf of T_1. If another transaction T_2 does hold the lock, then T_1 has to wait until T_2 gives up the lock. The scheduler thereby ensures that only one transaction can hold the lock at a time, so only one transaction can access the data item at a time. It goes beyond the scope of this thesis to treat locking more extensively.

3.2 Why geometric transactions are complicated

In this thesis the term *geometric transaction* is used for (database) transactions that are performed on geographic datasets. Geographic datasets contain geometry of objects and in particular operations on geometry make geometric transactions complicated. Some of the complicating issues are discussed in this section.

3.2.1 Geometry in transactions

The database should be able to deal with geometry. This includes encoding the geometry (for instance as some special data type, as a nested object or in some kind of table-structure) and being able to do computations with the geometric properties of geographic features. Especially when performing update or delete-operations in transactions, this is relevant, because one might use certain spatial operators to indicate which features are involved. One can imagine that selecting features that are intersected by a line needs more processing to find them than selecting features that have a certain value for one of its attributes. Spatial operators thus should be present in the database to support transactions on geographic features. Furthermore, the database could deal with spatial indexing and spatial clustering. These issues are not related to geometric transactions only, but are general issues that make geo-datasets complicated to deal with.

For geometric transactions that migrate geographic features from one dataset to another, geometric discrepancies (figure 3.1) might occur, for example due to surveying errors in both datasets. Laurini [15] proposes some sort of rubber-sheeting technique to solve geometric discrepancies. It could be necessary to deal with these discrepancies before transactions are committed to the database.

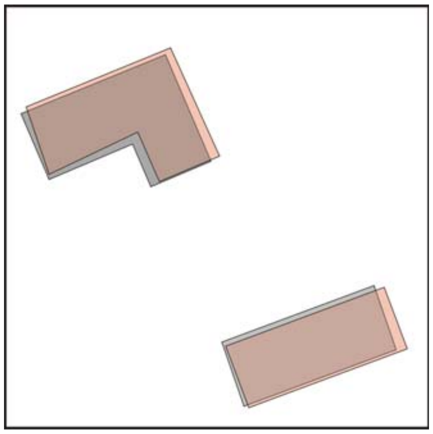


Figure 3.1 Geometric discrepancy between two datasets, based on Laurini 1994.The semi-transparent buildings are new versions of the underlying buildings, but the geometry of the objects is not exact the same.

Besides the problems mentioned above, geometric transactions are especially complicated because of the (spatial) relations of geographic features with their surroundings. To maintain consistency in a database, it is important that features can be identified that are somehow involved in a transaction, for instance because these features have to be locked. Sometimes the spatial relations between geographic features are explicitly stored in a topologic structure; sometimes relations follow from geometric/spatial constraints (for instance the coordinates of a point object lie within some polygon). Because of these relations, it is more difficult to identify all geographic features that are involved in a transaction. Sections 3.2.2 and 3.2.3 give some more explanation on this.

3.2.2 Topologically structured data

The GML 3.0 specification [27] describes topology as the branch of mathematics describing the properties of objects that are invariant under continuous deformation. For example, a circle is topologically equivalent to an ellipse because one can be transformed into the other by stretching. The constructs of topology allow characterization of the spatial relationships between objects using simple combinatorial or algebraic algorithms.

In geographic modeling, the foremost use of topology is in accelerating computational geometry and effective structuring for consistent modeling (e.g. partitions where no gaps or overlap may exist). Topology, realized by the appropriate geometry, also allows a compact and unambiguous mechanism for expressing shared geometry among geographic features to avoid redundant storage [27].

A GIS is different from many other database applications, because the topological edit operations can be complicated and related to many old and new objects [28]. This can be illustrated by following example. Assume that some dataset on soil types, with three objects, of figure 3.2 is topologically structured. For example, in the database the neighbours of all objects are stored explicitly.

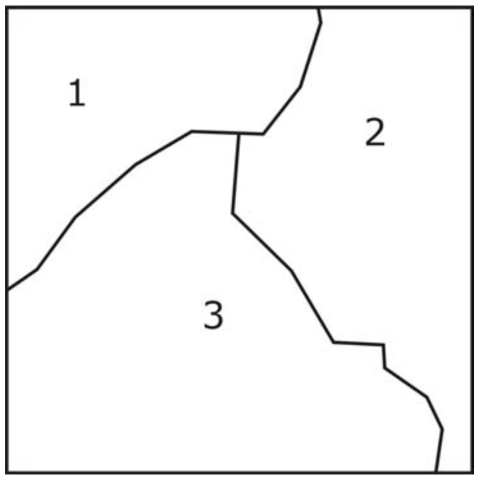


Figure 3.2 Example of topologically structured data.

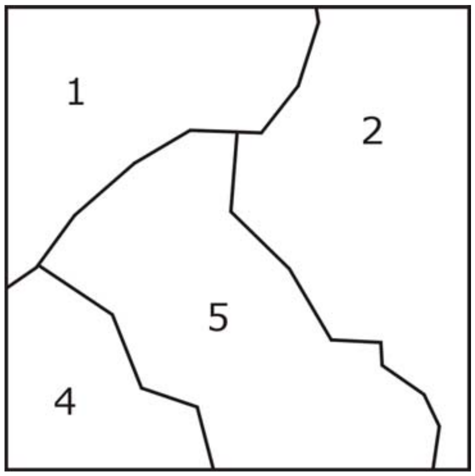


Figure 3.3 Inserting a new object by splitting object 3 into 4 and 5 involves other objects (object 1) as well.

Now assume that new objects are created in a transaction, by splitting Object 3 into object 4 and 5, as illustrated in figure 3.2. Now the neighbours of Object 1 and Object 2 have changed as well. To maintain consistency, the topological relationships of these objects therefore have to be updated.

The example shows that when data are topologically structured, processing the transaction holds more than just inserting the new object, because the dataset can get inconsistent. Time has not been taken into account, but time and maintaining history of features makes processing the transactions even more complicated. Oosterom [28] discusses how to maintain consistency in topologically structured, historical data.

3.2.3 Spatial constraints

In topologically structured data, features can get involved in a transaction, because the features' relationships with the subject(s) of a transaction are explicitly stored. But even when data are not topologically structured, it can be difficult to perform all operations of a transaction because of the relations of features with their surroundings or with other features.

To ensure consistency of a geographic dataset, integrity tests can be performed before a feature is modified (i.e. created, updated or deleted) [13]. These integrity tests might be necessary for spatial constraints like two geographic feature types that are not allowed to intersect or features that are not allowed to be in the vicinity of each other. Figure 3.4 gives an example of a small dataset where a new road has to be inserted. Assume that there are several spatial constraints for the dataset. For example, that roads have to be disjoint with buildings or that a new road has to connect to an existing road. In figure 3.5 the new road is tested on these conditions and is not accepted as being valid for the dataset, since the road intersects a building. The transaction thus will not be committed.

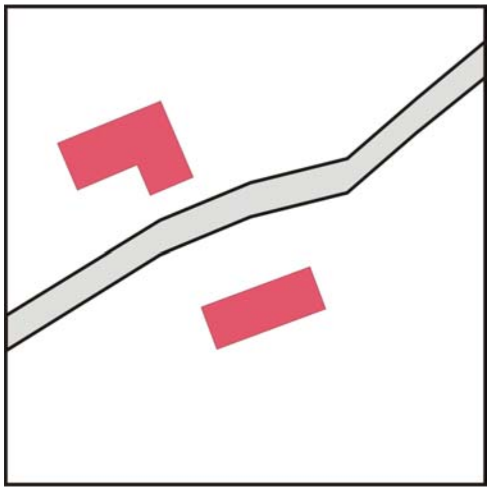


Figure 3.4 Small dataset of 2 buildings and a road segment, where another road is to be inserted.

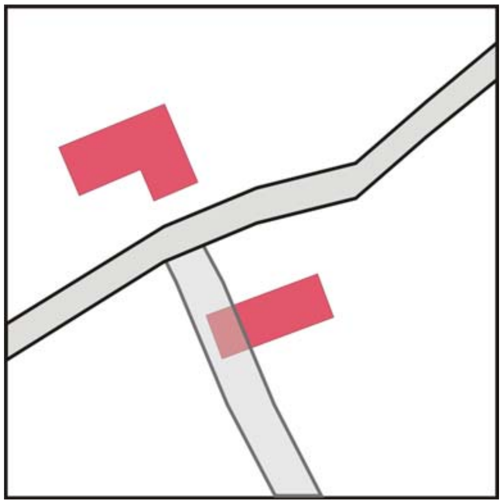


Figure 3.5 Integrity test before inserting the new road (slightly transparent) definitely. The road is not accepted, since it intersects a building.

All kinds of spatial constraints could be relevant for modifications on geo-datasets and thus for geometric transactions. These constraints can be specified in the Object Constraint Language. These constraints could be implemented in the database with SQL as assertions or could be handled at some other place, e.g. a mediating service or middleware that has to be used.

Before committing a geometric transaction, the operations have to be tested against these spatial constraints. This could be done in the database or before the database is accessed. In this context Garnett and Owens [13] describe Validating Web Feature Services, in which integrity and validation tests on the relevant geographic features are performed before committing the transaction.

3.3 Conclusion

From this chapter follows that transactions are executions of a program that edits the database by read or write operations. Transactions consist of a Start and an End-operation – that is, a transaction is either ended by Commit to make all effects of the operations permanent or Abort to undo all effects. Locking features ('freezing' the features for all other modifications than the modification of the transaction itself) can be used to avoid inconsistencies in the database, because transactions otherwise could interfere with other transactions. The term *geometric transaction* is used for (database) transactions that are performed on geographic datasets.

Because of the geometric properties, geometric transactions are usually complicated to process. In this context, not only common difficulties with geo-databases like encoding and computations with spatial operators play a role. In particular the relations of geographic features with features in the surroundings (either explicitly stored in some topological structure or spatial constraints, or implicitly from the geometric properties) make transactions complicated to process. If no care is taken, this will result in an inconsistent dataset, because the features in the surroundings could be affected by the transaction as well.

4 OpenGIS Web Feature Services

This chapter deals with research question c), what are OpenGIS Web Feature Services? First, in section 4.1, the OpenGIS Service Framework is described. Web Feature Services are developed with this framework in mind. Section 4.2 deals with relevant technologies for GIS Web services (HTTP and XML) and especially with technologies developed by the OpenGIS Consortium. It discusses Web Map Services (WMS), Web Coverage Services (WCS) and Web Feature Services (WFS). Since Web Feature Services are subject of this thesis, section 4.3 deals with them in more detail. That section summarizes relevant parts of the WFS specification on Basic WFS and Transactional WFS. Finally, in section 4.4 some conclusions are drawn.

4.1 OpenGIS Service Framework

The Open GIS Consortium, Inc. (OGC), a not-for-profit trade association dedicated to promoting new technical and commercial approaches to interoperable geoprocessing, was founded in 1994 in response to widespread recognition of the problem of non-interoperability and its many negative ramifications for industry, government, and academia [8]. OGC envisions the full integration of geospatial data and geoprocessing resources into mainstream computing and the widespread use of interoperable, commercial geoprocessing software throughout the global information infrastructure [8].

The OpenGIS Reference Model[7] (ORM) provides an architecture framework for the ongoing work of the OGC [25]. Further, the ORM provides a framework for the OGC Technical Baseline. The OGC Technical Baseline consists of the currently approved OpenGIS Specifications as well as for a number of candidate specifications that are currently in progress.

The OpenGIS Service Framework (OSF) identifies services, interfaces and exchange protocols that can be utilized by any application [25], as illustrated in figure 4.1. OpenGIS Services are implementations of services that conform to OpenGIS Implementation Specifications. Compliant applications, called OpenGIS Applications, can then "plug into" the framework to join the operational environment.

The OSF is designed to meet the following purposes [25]:

- Provide a framework for coordinated development of new and extended services
- Enable interoperable services through standard interfaces and encodings
- Support publishing, discovery and binding of services through service metadata
- Allow separation of data instances from service instances
- Enable use of a provider’s service on another provider’s data
- Define a framework that can be implemented in multiple ways

[7] Note that the OpenGIS Reference Model [25] is not a fixed standard, but a living document.

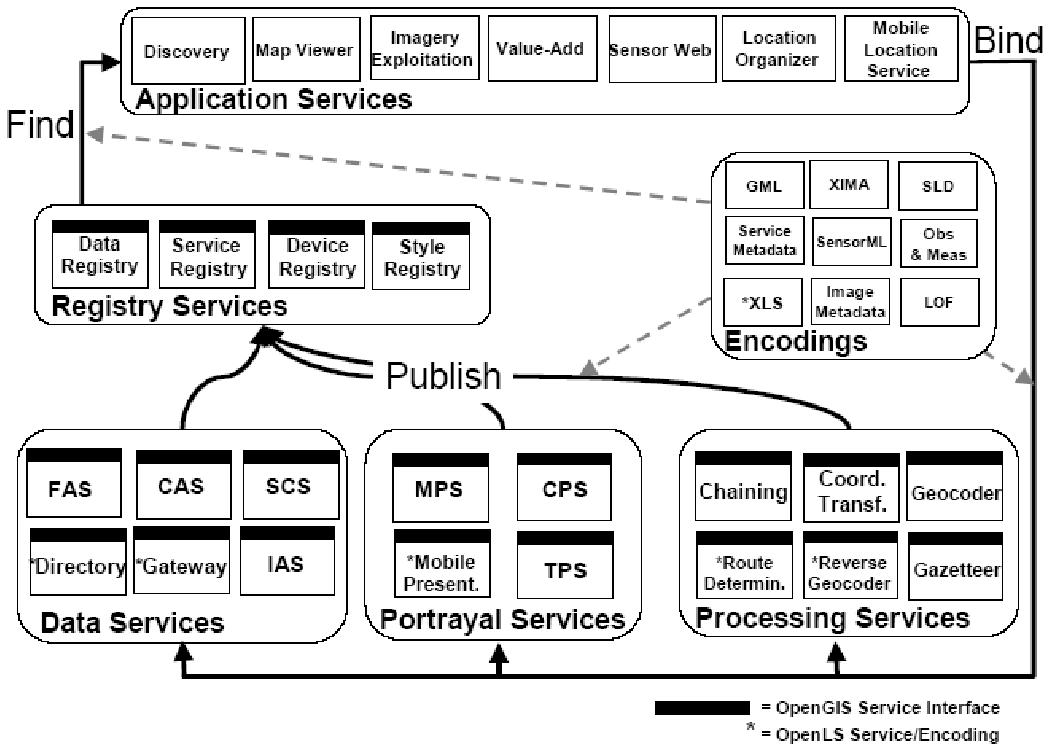


Figure 4.1 The OWS Service Framework (OSF) [25]

In the OpenGIS Reference Model some definitions are provided concerning the Service Framework. These key definitions for the Service Framework are:

- *A service is a distinct part of the functionality that is provided by an entity through interfaces.*
- *An interface is a named set of operations that characterize the behaviour of an entity.*
- *An operation is a specification of a transformation or query that an object may be called to execute. Each operation has a name and a list of parameters.*

An instance of a service may be associated with a specific instance of a dataset, or it may be a service that can be used to operate on multiple, unspecified datasets. The first case is referred to as a *tightly coupled service*, the second as a *loosely coupled service*. Service operations can be associated with data classes (data type) or with instances (data set).

The architecture is based on the publish/find/bind pattern in figure 4.2 and supports the dynamic binding between service providers and requestors since sites and applications are frequently changing in a distributed environment. Three essential roles are distinguished [25]:

- *Service provider*: publishes services to a broker (registry) and delivers services to service requestors
- *Service requestor*: performs service discovery operations on the service broker to find the service providers it needs and then accesses service providers for provision of the desired service.

- *Service broker*: helps service providers and service requestors to find each other by acting as a registry or clearinghouse of services.

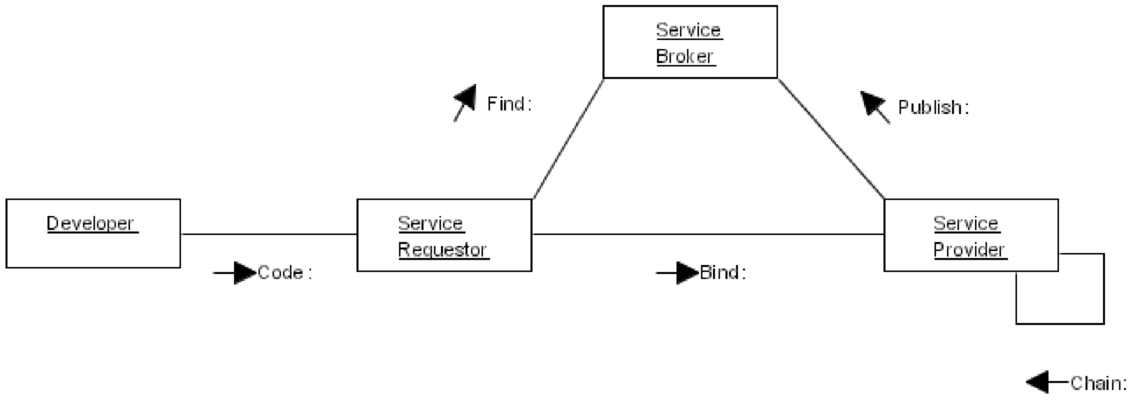


Figure 4.2 Publish-find-bind pattern from ORM [25].

By building applications to common interfaces, each application can be built without a-priori or run-time dependencies on other applications or services. Applications and services can be added, modified, or replaced without impacting other applications. In addition, operational workflows can be changed on-the-fly, allowing rapid response to time-critical situations. This loosely coupled, standards-based approach to development results in very agile systems - systems that can be flexibly adapted to changing requirements and technologies [25].

OSF services are accessible from *Application Services* operating on user terminals (e.g., desktop, notebook, handset, etc.) or servers that have network connectivity [25]. Users may use Application Services to access *Registry*, *Portrayal*, *Processing* and *Data Services* (see Section 2.3.3), depending upon the requirements and designed implementation of the application. Application Services commonly, but not necessarily, provide user-oriented displays of geospatial content and support user interaction at the user terminal. These services can be considered mediating services for users (clients) at one side and primary OSF services (like data services) on the other side.

4.2 Relevant technologies for GIS web services

In this section, two general web technologies (HTTP and XML) that are defined by the World Wide Web Consortium (W3C) are discussed first. GML (Geography Markup Language) and the OGC Filter Encoding make use of XML and are described next. The last part of this section deals with technologies, which can be used to retrieve and/or edit geographic data through the Internet. These technologies - Web Map Services (WMS), Web Coverage Services (WCS) and Web Feature Services (WFS) – will be compared.

4.2.1 HTTP[8]

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. The HTTP protocol is a

[8] Summary of [11].

request/response protocol. Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. The HTTP protocol uses URL to identify resources. The following scheme is used to locate network resources via the HTTP protocol:

```
http_URL = "http:" "//" host [ ":" port ]  [ abs_path [ "?" query ]]
```

The question mark ("?") is used to delimit the boundary between the URL of a queryable object, and a set of words used to express a query on that object. For example: `http://www.someserver.com/someapp.cgi?theparameter=somevalue` locates the server and passes a value to a resource called "someapp.cgi" on that server.

For this thesis two HTTP methods (GET and POST) for communication between client and server are relevant. The GET method means: retrieve whatever information (in the form of an entity, for example an HTML-document or an XML-document) is identified by the Request-URL. The Request-URL is the URL that is invoked by the client. If the Request-URL refers to a data-producing process, the produced data shall be returned as the entity in the response and not the source text of the process (unless that text happens to be the output of the process).

POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URL.

4.2.2 XML

Bos [7] describes XML as:

> *"XML is a set of rules for designing text formats that let you structure your data. XML is not a programming language, and you don't have to be a programmer to use it or learn it. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. XML avoids common pitfalls in language design: it is extensible, platform-independent, and it supports internationalization and localization."*

XML is a meta language that allows for creation and formatting of specific document markups [9]. XML can be considered a family of technologies [7]:

> *"XML 1.0 is the specification that defines what "tags" and "attributes" are. Beyond XML 1.0, "the XML family" is a growing set of modules that offer useful services to accomplish important and frequently demanded tasks. XLink describes a standard way to add hyperlinks to an XML file. XPointer is a syntax for pointing to parts of an XML document. An XPointer is a bit like a URL, but instead of pointing to documents on the*

Web, it points to pieces of data inside an XML file. CSS, the style sheet language, is applicable to XML as it is to HTML. XSL is the advanced language for expressing style sheets. It is based on XSLT, a transformation language used for rearranging, adding and deleting tags and attributes. XML Schemas help developers to precisely define the structures of their own XML-based formats.“

Figure 4.3 gives an example of XML-encoded data. Some XML-technologies (XML Schema, XSL/XSLT and XSQL) are discussed below.

```
<university name=”Delft University of Technology”>
      <country>Netherlands</country>
      <city>Delft</city>
      <faculty name=”Civil Engineering and Geosciences“>
            <bachelorstudents>608</bachelorstudents>
            <masterstudents>367</masterstudents>
      </faculty>
      <faculty name=”Information Technology and Systems”>
            <bachelorstudents>298</bachelorstudents>
            <masterstudents>179</masterstudents>
      </faculty>
      ...
</university>
```

Figure 4.3 Example of XML-encoded (fictitious) data, using tags and attributes.

```
<xs:complexType name="DRAFT_BOUNDARY_Type">
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
        <xs:element    name="FID"    nillable="FALSE"    minOccurs="1"
maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element    name="SHAPE"    type="gml:LineStringPropertyType"
minOccurs="1" maxOccurs="1"/>
        <xs:element   name="OBJECT_ID"   nillable="FALSE"   minOccurs="1"
maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        …
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 4.4 Part of an XML-schema

An *XML-schema* is a document that describes the valid format of an XML dataset. The specification of XML Schema can be found at [53]. Definitions in XML Schema include what elements are (and are not) allowed at any point; what the attributes for any element may be; the number of occurrences of elements; etc. [54]. Figure 4.4 gives a piece of an XML-Schema to define a geographic feature. An XML-document can be validated against its XML Schema. If the document adheres to all rules in the XML-schema, the document is called *valid*.

XSL allows developers to describe transformations using XSL Transformations (*XSLT*). XSL Transformations can convert XML documents into HTML, another XML-document or other textual output [9].

For this thesis, *XSQL* is a relevant technology as well. XSQL Pages are templates that allow anyone familiar with SQL to [43]:

- Assemble dynamic XML "datapages" based on one or more parameterized SQL queries, and
- Transform the "datapage" to produce a final result in any desired XML, HTML, or Text-based format using an associated XSLT Transformation.

Using XSQL, one can retrieve XML-encoded data from a relational database by using SQL-statements. The XSQL page in figure 4.5 retrieves one feature from an Oracle database (table CL_BOUNDARY), encapsulates it in XML-tags and transforms it to another text-based format using an XSLT-stylesheet (`boundaryOracle2GML.xsl`), in this case to an XML-document.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="xslt/boundaryOracle2GML.xsl"?>
<xsql:query    connection="conn"    xmlns:xsql="urn:oracle-xsql"    max-
rows="1">

SELECT  FID,  OBJECT_ID,  SHAPE,  BBOX,  OBJECT_DT,  TMIN,  TMAX  FROM
CL_BOUNDARY

</xsql:query>
```

Figure 4.5 XSQL to retrieve data from an Oracle database and transform that using an XSLT-stylesheet

4.2.3 Geography Markup Language[9]

The Geography Markup Language (GML) is an XML encoding for the modelling, transport and storage of geographic information including both the spatial and non-spatial properties of geographic features. Figure 4.6 gives a simple example of a GML-encoded feature, in this case a road. GML uses XML-related technologies as well, like XLink or XPointer.

The GML specification [27] defines the XML Schema syntax, mechanisms, and conventions that:

- Provide an open, vendor-neutral framework for the definition of geospatial application schemas and objects;
- Allow profiles that support proper subsets of GML framework descriptive capabilities;

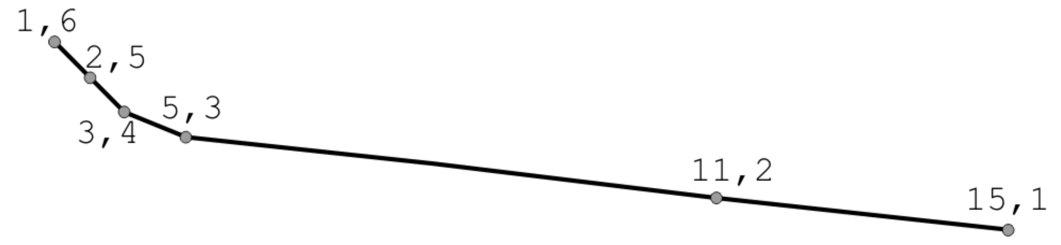
[9] This section is a summary of the GML specification [27].

- Support the description of geospatial application schemas for specialized domains and information communities;
- Enable the creation and maintenance of linked geographic application schemas and datasets;
- Support the storage and transport of application schemas and data sets;
- Increase the ability of organizations to share geographic application schemas and the information they describe.

Implementers may decide to store geographic information in GML, or they may decide to convert from some other storage format on demand and use GML only for schema and data transport.

A GML application schema is an XML Schema written according to the GML rules for application schemas (see [27]) and which defines a vocabulary of geographic objects for a particular domain. For example topographic data, cadastral data or road transport data. In such a definition, objects and their application specific properties are defined and references are given to other XML Schemas that define standardized types and elements of GML (see Appendix D for an example of an application schema for GML 2.1.2). For instance *features.xsd* is referred to for the definition of geographic features, *topology.xsd* if topologic relations are modeled or *geomteryBasic0d1d.xsd* and *geomteryBasic2d.xsd* for basic geometric primitives[10]. An example of a GML application schema for the topographic domain is the recent GML prototype for the Dutch Topographic Service [34].

At the time of writing, the most recent version of GML is GML 3.0. According to the OGC, GML 3.0 provides a variety of kinds of objects for describing geography including features, coordinate reference systems, geometry, topology, time, units of measure and generalized values. Some of these will be discussed shortly. For more details on GML 3.0 see the specification [27].



```
<featureMember>
   <ex:Road gml:id="r2">
      <curveProperty>
         <LineString srsName="somelistofcrs.xml#1234">
            <coordinates>1,6 2,5 3,4 5,3 11,2 15,1</coordinates>
         </LineString>
      </curveProperty>
   </ex:Road>
</featureMember>
```

[10] GML 3.0 has several schemas that define the geometry model. GML 2 has one (*geometry.xsd*).

Figure 4.6 Example of a GML-encoded geographic feature, adapted from [27]. De geometry of the road is encoded in GML. The attribute srsName specifies the spatial reference system.

A geographic feature is an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth [27]. This follows the general definition of a feature given in ISO 19109 [48] and the OGC Abstract Specification Topic 5 [19]. The feature XML schema (schemas can be found at http://schemas.opengis.net/gml/) provides a framework for the creation of GML features and feature collections.

Associations between features can be defined in GML. One of these associations is 'features being a member of feature collections'. A *GML Feature Collection* is a collection of GML feature instances that can behave as a GML feature. GML Feature Collections are themselves valid GML features and can have `gml:location` and other properties as defined in their GML Application Schema.

With GML 3.0 *topology* can be encoded. The conceptual model underlying the representation of topology in GML is that of Topic 1 of the OGC Abstract Specification (ISO DIS 19107). The model describes the correspondence of topological and geometric relationships up to three dimensions.

One of the requirements in developing of GML 3.0 (and previous versions) is strict separation of data and presentation. Therefore none of the GML data description constructs have built-in capability to describe the styling information. In GML 3.0 there is a possibility to describe default styles. The *default styling* mechanism was created as a separate model that can be "plugged-in" to a GML data set. The term "default" signifies very relaxed relation to other parts of the GML model. The style information that is assigned to a data set may be used for styling but may also be completely ignored.

At the moment of writing, the specification of Web Feature Services uses GML 2.1.2 [21] instead of GML 3.0. GML 3.0 is far more comprehensive in describing geography than GML 2.1.2. In a nutshell, GML 2.1.2 only provides encodings for geographic features, feature collections, geometry and coordinate reference systems. Topology, temporal aspects and default styling thus are not part of the GML 2.1.2 specification.

4.2.4 Filter encoding[11]

A *filter expression* is a construct used to constraint the property values of an object type for the purpose of identifying a subset of object instances to be operated upon in some manner. In the Filter Encoding Implementation Specification an XML encoding for filters is described. Using the numerous XML tools available today, such an XML representation can be easily validated, parsed and then transformed into whatever target language is required to retrieve or modify object instances stored in some object store. For example, an XML encoded filter could be transformed into a WHERE clause for a SQL SELECT statement to fetch data stored in an SQL-based relational database. Similarly, an XML encoded filter expression could be transformed into an XPath or XPointer expression for fetching data from XML documents.

[11] This section is a summary of the Filter Encoding Implementation Specification [20], of which relevant parts have been adapted.

The OGC filter encoding is a common component that can be used by a number of OGC web services. Any service that requires the ability to query objects from a web-accessible repository can make use of the XML filter encoding. For example, a Web Feature Service may use the XML filter encoding in a *GetFeature* operation to define query constraints. Other services based on the Web Feature Service could also make use of the OGC filter encoding.

The OGC filter encoding defines several operators and other predicates that can be used to construct a filter. For example spatial operators (e.g. a boundingbox), comparison operators (e.g. a property is between value A and B), feature identifiers and literals. A literal is any part that is to be used exactly as it is specified. Figure 4.7 gives two examples of OGC filter encodings.

```
(a)
<Filter>
   <PropertyIsLessThan>
      <PropertyName>DEPTH</PropertyName>
      <Literal>30</Literal>
   </PropertyIsLessThan>
</Filter>

(b)
<Filter>
   <And>
      <Within>
         <PropertyName>WKB_GEOM</PropertyName>
         <Polygon name="1" srsName="EPSG:4326">
            <outerBoundaryIs>
               <LinearRing>
                  <coordinates>
                     -98.5485,24.2633...
                  </coordinates>
               </LinearRing>
            </outerBoundaryIs>
         </Polygon>
      </Within>
     <PropertyIsBetween>
        <PropertyName>DEPTH</PropertyName>
        <LowerBoundary>400</LowerBoundary>
        <UpperBoundary>800</UpperBoundary>
     </PropertyIsBetween>
   </And>
</Filter>
```

Figure 4.7 Two examples of OGC filter encodings, from [20]. Filter (a) checks whether the DEPTH is lower than 30, filter (b) combines spatial and non-spatial predicates to check for the geometric property WKB_GEOM to lie within a region defined by a polygon and to check for the DEPTH to be between 400 and 800.

4.2.5 OpenGIS Web Services: the WMS, WCS and WFS interfaces

This section discusses three GIS web services the OpenGIS Consortium has defined. First, Web Mapping Services (WMS) - to retrieve digital maps over the Internet - are briefly discussed. This is followed by services to retrieve coverages, Web Coverage

Services (WCS). Web Feature Services (WFS), which are services to retrieve and edit geographic data, are discussed after WCS. This section concludes with a short comparison of these services.

WMS[12]

A Web Map Service (WMS) produces maps of georeferenced data. The WMS specification [24] defines a "map" as a visual representation [13] of geodata; a map is not the data itself. The maps are generally rendered in a pictorial format such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or in Web Computer Graphics Metafile (WebCGM) formats. Although GML is raw data and not a visual representation, GML could be used as output format as well. The WMS-specification standardizes the way in which maps are requested by clients and the way that servers describe their data holdings.

The WMS specification defines three WMS operations: *GetCapabilities* returns service-level metadata, which is a description of the service's information content and acceptable request parameters in XML; *GetMap* returns a map image whose geospatial and dimensional parameters are well-defined; *GetFeatureInfo* returns information about particular features shown on a map, by adding to the map URL additional parameters specifying a location (as an X, Y offset from the upper left corner), the format in which the information should be returned (e.g. as GML, but without the schema) and the number of nearby features about which to return information. This way extra information on geographic features can be requested, but the data cannot be edited.

A standard web browser can ask a Web Map Service to perform these operations simply by submitting requests in the form of Uniform Resource Locators (URLs). WCS and WFS use this technique as well. Figure 4.8 gives an example of a WMS-request to get a map from a server.

```
http://mapserver.somewhere.nl/servlet/wms?
?WMTVER=1.0
& REQUEST=map
& SRS=EPSG:28992
& LAYERS=BUILDINGS,ROADS,RIVERS
& STYLES=default,default,default
& BBOX=160000.0,440000.0,170000.0,450000.0
& WIDTH=600
& HEIGHT=600
& FORMAT=GIF
& TRANSPARENT=TRUE
& SERVICENAME=someservice
```

Figure 4.8 Example WMS-request for retrieval of map image

A basic WMS classifies its georeferenced information holdings into "Layers" and offers a finite number of predefined "Styles" in which to display those layers. The behaviour of a Web Map Service can be extended to allow user-defined symbolization of feature data instead of named Layers and Styles. Furthermore, individual map Layers can be

[12] Summary of [24]

[13] Digital representations are meant here.

requested from different Servers. The WMS specification thus enables the creation of a network of distributed Map Servers from which Clients can build customized maps.

A "Cascading Map Server" is a WMS that behaves like a client of other WMSes and behaves like a WMS to other clients. For example, a Cascading Map Server can aggregate the contents of several distinct map servers into one service. Furthermore, a Cascading Map Server can perform additional functions such as output format conversion or coordinate transformation on behalf of other servers. In terms of the client-server architecture, a Cascading Map Server is a mediator for Web Map Services.

WCS[14]

The Web Coverage Service (WCS) supports electronic interchange of geospatial data as "coverages" - that is, digital geospatial information representing space-varying phenomena. This corresponds with the field model, as described in section 2.1.1 on page 3. The Web Coverage Service provides available data together with their detailed descriptions; allows complex queries against these data; and returns data with its original semantics (instead of pictures), which can be interpreted, extrapolated, etc. - and not just portrayed. Unlike WFS, which returns discrete geospatial features, the Web Coverage Service returns representations of space-varying phenomena that relate a spatio-temporal domain to a (possibly multidimensional) range of properties.

The Web Coverage Service consists of three operations: *GetCapabilities*, *GetCoverage,* and *DescribeCoverageType*. The *GetCapabilities* operation returns an XML document describing the service and the data collections from which clients may request coverages. The *GetCoverage* operation of a Web Coverage Service is normally run after *GetCapabilities* has determined what queries are allowed and what data are available. The *GetCoverage* operation returns (for example as in figure 4.9) values or properties of geographic locations, bundled in a well-known coverage format. In fact, any format is allowed, but the specification provides five formats (GeoTIFF [38], HDF-EOS [39], DTED [37], NITF [42] and GML [27]) of which the server should at least support one. The *GetCoverage* syntax and semantics bear some resemblance to the WMS *GetMap* and WFS *GetFeature* requests, but several extensions support the retrieval of coverages rather than static maps or discrete features.

```
http://wcs.somewhere.nl/servlet/wcs?
?SERVICE=WCS
& VERSION=1.0.0
& REQUEST=GetCoverage
& COVERAGE=coverageName
& CRS=EPSG:28992
& BBOX=160000.0,440000.0,170000.0,450000.0
& WIDTH=10000
& HEIGHT=10000
& FORMAT=image format
```

Figure 4.9 Example WCS-request for retrieval of coverage data

[14] This is a summary of [26].

WFS[15]

The OGC Web Map Service allows a client to overlay map images for display served from multiple Web Map Services on the Internet. In a similar fashion, the OGC Web Feature Service allows a client to retrieve geospatial (vector-)data encoded in Geography Markup Language (GML) from multiple Web Feature Services. A drawback of this is that the client has to do the rendering itself.

But besides operations to retrieve data, WFS describes operations to edit (insert, update, delete) data on the level of a feature. With WFS (requests for) transactions on geographic features can directly be sent to a single database. This makes WFS the only open, interoperable standard with which data can be edited over the Internet.

Because of the operations defined in the WFS specification, (i.e. 'select', insert, delete, update, lock) a WFS is close to a Data Manipulation Language (DML). This raises the question whether WFS should not also provide a Data Definition Language (DDL), similar to SQL. Such a DDL could for instance be used to define new feature types. Discussing WFS as a DDL is out of the scope of this thesis, but could be a very interesting future development.

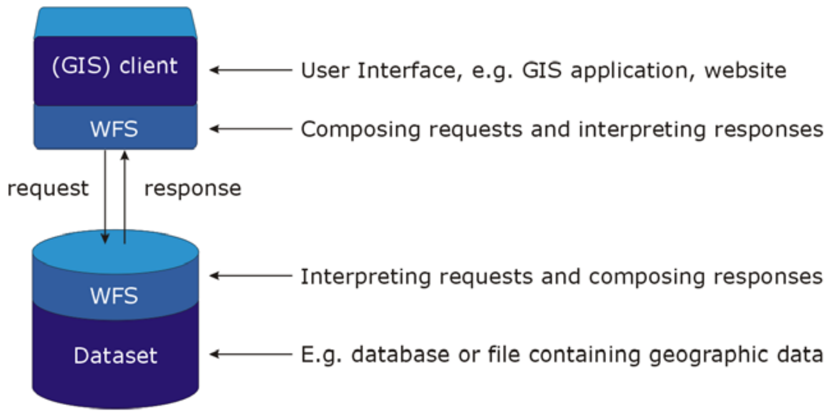

(GIS) client
WFS
request
response
WFS
Dataset
User Interface, e.g. GIS application, website
Composing requests and interpreting responses
Interpreting requests and composing responses
E.g. database or file containing geographic data

Figure 4.10 Web Feature Service

In the specification [23] the Web Feature Service interface is described, as illustrated in figure 4.10. The specification describes the WFS operations. WFS supports INSERT, UPDATE, DELETE (in a *Transaction*-request) and QUERY (*GetFeature*) operations on geographic features using HTTP as the distributed computing platform. It also provides operations to describe the service and the contents of the service, such as *GetCapabilities* and *DescribeFeatureType*.

The specification defines a transaction as a logical unit of work that is composed of one or more data manipulation operations [23]. Transactions can be encoded in XML. It is the function of a Web Feature Service, in its interaction with the data storage system used to persistently store features, to ensure that changes to data are consistent. However, the document also acknowledges the fact that many systems do support standard concurrent transaction semantics and so proposes optional operations that will allow a Web Feature Service to take advantage of such systems (e.g. relational database

[15] This is a summary of [23].

systems based on SQL). For example, Web Feature Services could support a WFS-operation for locking (LockFeature).

Why WFS could also be useful for internal work processes

For all kinds of organisations that use GIS or geographic data, a big advantage of Web Feature Services (and the service-based model in general) is that using open standards allows for the separation of different components of the working environment. That is, organisations don’t have to change the entire working environment when one of the components (e.g. client or server) needs to change.

Take for example a client-server application for editing geographic data. In a closed system, the server and client can only be used in cooperation with each other, i.e. they are dependant. But if the server is WFS compliant, any client that is WFS compliant can easily be used since it communicates with the server via an open standard. If the client needs to change (for what reason), an organisation can choose in fact any (generic) client that is WFS compliant. Note that generic WFS-clients probably need to be adapted to perform (application) specific tasks, but these changes are most often very small compared to changing the entire environment.

In addition, changing the software demands much less new (technical) knowledge and skills, since the protocols and technologies that are used are still the same. Using systems and services based on open standards therefore could allow for more flexibility in the choice of software compared to closed systems, where client and server cannot be separated since they use specific product protocols and technologies.

Comparison

To a certain extent Web Map Services, Web Coverage Services and Web Feature Services work similarly. All are interfaces for handling geographical data in web services. However, each of them serves a different purpose. Table 1 compares the most important operations/requests.

Operation/request	WMS	WCS	WFS
Describe capabilities	✓	✓	✓
Retrieval of graphics	✓	✓	-
Retrieval of unrendered / raw geodata	✓	✓	✓
Information on a specific feature	✓	-	✓
Selection of data is possible by:	Bounding Box, Time, Elevation	Bounding Box, Time, values of attributes	Spatial and non spatial filters, as in the Filter Encoding
Editing geodata	-	-	✓

Table 1. Comparison of the most important operations of WMS, WCS and WFS.

Because WFS is the only OGC technology that supports modifications on geographic features, WFS has been chosen for this research. Therefore WFS will be described in more detail in the next section.

4.3 WFS in more detail[16]

4.3.1 Basics on WFS

With WFS data can be handled on the feature level. In the WFS specification [23] operations for both retrieval and manipulation of features are defined, as well as the supporting operations.

Processing requests would proceed as follows (figure 4.11):

1) A client application would request a capabilities document from the web feature server. Such a document contains a description of all the operations that the server supports and a list of all feature types that it can service. The operation that enables this is *GetCapabilities*.
2) A client application (optionally) makes a request to a Web Feature Service for the definition of one or more of the feature types that the WFS can service. Clients can do this by sending a *DescribeFeatureType* request.
3) Based on the definition of the feature type(s), the client application generates a request as specified in the specification. This can either be a *GetFeature* request to retrieve features or a *Transaction* request to modify features.
4) The request is sent to a web server using either HTTP GET or HTTP POST.
5) The WFS is invoked to read and service the request.
6) When the WFS has completed processing the request, it will generate a status report and hand it back to the client. In the event that an error has occurred, the status report will indicate that fact.

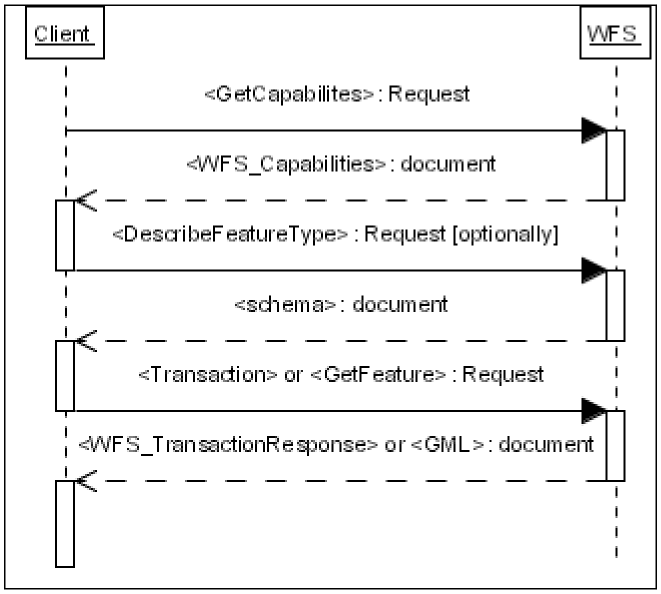

Client
WFS
<GetCapabilites> : Request
<WFS_Capabilities> : document
<DescribeFeatureType> : Request [optionally]
<schema> : document
<Transaction> or <GetFeature> : Request
<WFS_TransactionResponse> or <GML> : document

Figure 4.11 UML sequence diagram for processing requests in general.

[16] Summary of [23]. For the technical details see [23].

Two classes of Web Feature Services are defined:

- *Basic WFS.* A basic WFS would implement the *GetCapabilities*, *DescribeFeatureType* and *GetFeature* operations. This would be considered a READ-ONLY Web Feature Service.
- *Transaction WFS*. A transaction Web Feature Service would support all the operations of a basic Web Feature Service and in addition it would implement the *Transaction* operation. This kind of WFS can be considered a READ-WRITE Web Feature Service. Optionally, a Transaction WFS could implement the *LockFeature* operation.

Section 4.3.2 discusses Basic WFS, section 4.3.3 discusses Transaction WFS. This section deals with issues that concern all WFS. This includes the way requests are encoded and sent and some practical issues like identifying features and error reporting.

Versions of WFS

Servers and clients can support more than one version of a WFS specification. In the first step, the version of the web feature server and client can be determined. To get to a mutually understood version, a procedure for version negotiation is described in the specification.

HTTP as distributed computing platform

WFS requests are sent between client and server using the HTTP protocol. HTTP supports two request methods: GET and POST, both have been discussed in section 4.2.1. Depending on the type of request, GET or POST is used.

Encoding of requests

There are two methods of encoding WFS requests. The first uses XML as the encoding language, the second uses keyword-value pairs to encode the various parameters of a request. An example of a keyword value pair (KVP) is "`REQUEST=GetCapabilities`", where "`REQUEST`" is the keyword and "`GetCapabilites`" is the value. Figure 4.12 gives examples of both a KVP and an XML encoding of a GetFeature request.

In general KVP requests are shorter, but using KVP for transaction requests is limited. KVP-requests are sent using HTTP GET, while XML-requests have to be sent with HTTP POST. In both cases (XML and KVP), the response to a request or the exception report must be identical.

Within the interface, features should be encoded in GML, version 2.1.2. This means that when features are sent to a web feature server, they must be encoded in GML 2.1.2. However, web feature servers are allowed to support other *output* formats as well. Filters, as described in the Filter Encoding Implementation Specification [20], are to be used for defining a set of feature instances to operate upon. Filters can be used in both KVP and XML encoded requests. Figure 4.12 illustrates the use of a filter in an XML-request as well.

```
(a)
http://www.someserver.com/servlet/wfs
?request=GetFeature
```

```
&FEATUREID=TEST_BOUNDARY.1000


(b)
<?xml version="1.0"?>
<GetFeature
 version="1.0.0"
 service="WFS"
 xmlns="http://www.opengis.net/wfs"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:cad="http://www.someserver.com/cad"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
 xsi:schemaLocation="http://www.opengis.net/wfs
  ../wfs/1.0.0/WFS-basic.xsd">
   <Query typeName="TEST_BOUNDARY">
      <ogc:Filter>
         <ogc:FeatureId fid="TEST_BOUNDARY.1000"/>
      </ogc:Filter>
   </Query>
</GetFeature>
```

Figure 4.12 KVP (a) and XML (b) request to retrieve the feature of type TEST_BOUNDARY with FeatureId 1000. (a) is sent using HTTP GET, (b) using HTTP POST.

Identifying features

The WFS-specification [23] assumes that "every feature instance that a particular WFS implementation can operate upon is uniquely identifiable. That is to say, when a WFS implementation reports a feature identifier for a feature instance, that feature identifier is unique to the server and can be used to repeatedly reference the same feature instance".

It seems to be that the OpenGIS specifications only allow one attribute of a feature to be the unique identifier. This seems to be the case in both the WFS-specification and the OpenGIS Filter encoding specification [3] (that also deals with feature identifiers), where nothing is said on having a key composed of multiple attributes and all examples use one (numeric) value as the identifier.

Exception reporting

In the event that a Web Feature Service encounters an error while processing a request or receives an unrecognized request, it shall generate an XML document indicating that an error has occurred. The format of the XML error response is specified in the WFS-specification.

A *ServiceExceptionReport* element can contain one or more WFS processing exceptions. Individual exception messages are contained within the *ServiceException* element. The contents of this *ServiceException* element can be in any format the server "wishes". The optional *code* attribute may be used to associate an exception code with the accompanying message. A client can use these codes to provide the correct (error) message, if the client knows what code means what. However, these codes are not specified, but should be defined by the service provider (at the server side).

4.3.2 Read-only service: Basic WFS

A Basic WFS implements the *GetCapabilities*, *DescribeFeatureType* and *GetFeature* operations. These will be described here shortly. Appendix D contains examples of the requests and responses for *GetCapabilities* and *DescribeFeatureType* operations.

A Web Feature Service must be able to describe its capabilities. Specifically, it must indicate which feature types it can service and what operations are supported on each feature type. A client can request for such a capabilities document by sending the *GetCapabilities* request to a web feature server. The server should respond to this with an XML-document, describing following issues:

- General information on the service itself (*Service section*): the name of the service, the title, an abstract on the server, keywords for catalogue searching, the top-level HTTP URL (OnlineResource) and information on fees and access constraints.
- The *capabilities section* specifies the list of requests that the WFS can handle and what resources (URL's) should be called for the requests. E.g. whether the WFS supports transaction requests or not.
- The *featuretypelist section* provides a list of feature types with their spatial reference system and bounding box. The list could also specify which operations are allowed per feature.
- The *filter capabilities section* is optional. It lists the filters (of the Filter Encoding Implementation Specification) that the web feature server supports.

The standard capabilities of a Web Feature Service can be extended if a service provider wishes so. The specification defines *VendorSpecificCapabilities*, which can be added to the regular capabilities. In this element extensions of a vendor can be included.

The function of the *DescribeFeatureType* operation is to generate an XML-schema description of feature types serviced by a WFS implementation. The schema descriptions define how a WFS implementation expects feature instances to be encoded on input and how feature instances will be generated on output. The feature geometry must be expressed using GML geometry description.

A Web Feature Service must be able to service a request to retrieve feature instances. In addition, the client should be able to specify which feature attributes to fetch and should be able to constrain the query spatially and non-spatially, similar to SQL where clauses. The *GetFeature* operation allows for the retrieval and uses filters from the Filter Encoding [20] to constrain the data to retrieve.

The server processes the *GetFeature* request and returns an XML document, containing the result set, to the client. With a *GetFeature* request one or more feature types can be queried. A client can constrain the feature set by using filters from the filter encoding specification. For instance, one feature (identified by a `<FeatureId>`) needs to be fetched from a Web Feature Service. Figure 4.12 (b) in section 4.3.1 gives the XML-encoded request.

In the *GetFeature* request the output format can be specified. The default value is GML 2, but if a web feature server offers other output formats, a client can request for one of these in the request. A web feature server may offer any other output format, as long as it is advertised in the capabilities document. Requesting certain versions of features (if a server supports feature versioning of course) is accommodated for as well in the WFS-specification.

4.3.3 Modifying features: Transactional WFS

In addition to a Basic WFS, a Transactional WFS supports operations to modify geographic features as well; that is create, update, and delete operations on geographic features. In order to do so, a Transactional WFS implements the *Transaction* operation and could optionally implement the *LockFeature* operation.

The only operation that can be encoded as a KVP-request, is the *Delete* operation. If other operations are desired, then the request has to be encoded in XML. A *Transaction* element may contain zero or more *Insert*, *Update*, or *Delete* elements that describe operations to create, modify or destroy feature instances:

- For *Insert* statements, the state of a feature must be expressed in GML and must validate to the schema the corresponding *DescribeFeatureType* operation generates.
- The *Update* element describes one update operation that is to be applied to a feature or a set of features. Filters (as specified in the filter encoding – described in 4.2.3) can be used to define such a set. The operation contains one or more properties and their associated new values.
- The *Delete* element is used to indicate that one or more feature instances should be deleted. The scope of a delete operation is constrained using the *Filter* element as described in the filter encoding.

When the transaction has been completed, a Web Feature Service will generate an XML response document indicating the completion status of the transaction. This status report states whether the transaction has been completed successful, partial successful or failed. When a web feature server handles a transaction atomically (as one entity), it is not possible that a transaction is partial successful. The status of partial successful has been defined, to allow for web feature servers that don't support atomic transactions to give reports which operations succeeded and which operations failed. Note that the web feature server cannot advertise whether transactions are dealt with as atomic transaction (thus as one entity) or not.

In response to an *Insert* operation, a Web Feature Service shall also generate a list of newly generated feature identifiers assigned to the new feature instances in the same order as in the *Transaction* request. In Appendix D examples are given for *Transaction* requests and responses.

Besides the three modifying operations, a *Transaction* may contain a *Native* element. This element is intended to allow access to vendor specific capabilities of any particular Web Feature Service. It contains the vendor specific command or operation.

Figure 4.13 is a simplified sequence diagram[17] illustrating the messages that might be passed back and forth between a client application and a Web Feature Service in order to process a typical transaction request.

[17] Sequence diagrams are a type of UML (Unified Modelling Language) diagram.

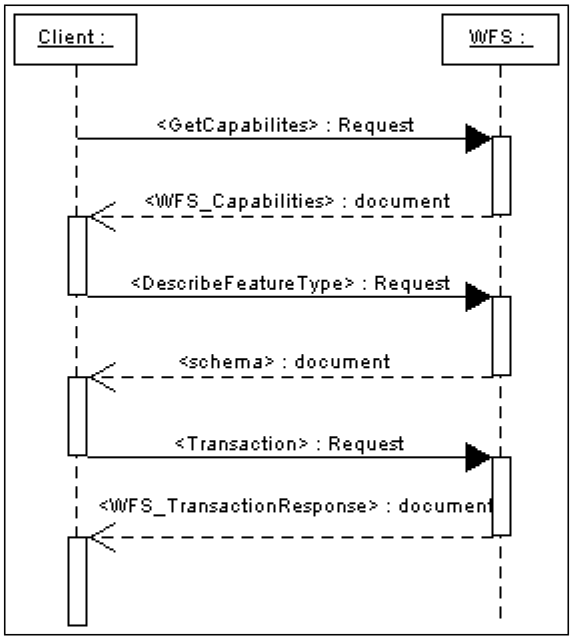

Client :
WFS :
<GetCapabilites> : Request
<WFS_Capabilities> : document
<DescribeFeatureType> : Request
<schema> : document
<Transaction> : Request
<WFS_TransactionResponse> : document

Figure 4.13 Sequence diagram for processing a transaction request.

The purpose of the *LockFeature* operation is to expose a long-term feature locking mechanism to ensure consistency. The lock is considered long term in order to lock features longer than the duration of a single transaction. The *LockFeature* operation is optional and does not need to be implemented for a WFS implementation to conform to the WFS specification. If a WFS implements the *LockFeature* operation, this must be advertised in the capabilities document.

The *LockFeature* element contains one or more *Lock* elements that define a locking operation on feature instances of one feature type. The *expiry* attribute is used to set a limit on how long a Web Feature Service should hold a lock on feature instances in the event that a transaction is never issued that will release the lock. Once the specified number of minutes has elapsed, a Web Feature Service may release the lock if it exists and abort any related transaction. Any further transactions issued against that lock using a lock identifier generated by the service will fail. A *Lock* element uses a filter to specify what feature instances should be locked.

In response to a *LockFeature* request, a Web Feature Service generates an XML document. This document will contain a lock identifier that a client application can use in subsequent WFS operations (like a *Transaction*) to operate upon the set of locked feature instances.

A WFS can also support the *GetFeatureWithLock* operation. By using this operation instead of the GetFeature operation, a client requests for features to be retrieved and lock them at the same time.

Assuming that a WFS implementation supports the optional *LockFeature* and/or *GetFeatureWithLock* operations, the *releaseAction* attribute is used to control how locked features are treated when a transaction request is completed. With this attribute locks can be ended.

4.4 Conclusion

The OpenGIS Service Framework provides the framework for OpenGIS Web Services. These open, interoperable services use general web technologies like HTTP and XML and can be used to build applications dynamically. The only services that allow for retrieval of geographic data on the feature level and allow modification of features are services that comply with the Web Feature Services specification.

Web Feature Services are either Basic, which means that data can only be retrieved, or Transactional, which means that the server supports create, update and delete operations on some or all of its features. Transactions may be composed of several of these operations of any type. Transactions can thus be composed very flexible, but still can be handled as atomic transactions by the server. The WFS specification also defines operations for locking features, which can be used to maintain consistency.

WFS requests can be encoded in the URL, using keyword-value-pairs, or in XML. XML requests are more expressive and transactional requests using *Insert* and *Update* operations, XML requests are the only requests allowed. Web Feature Services make use of the Filter Encoding to specify sets of features to operate upon. Within the interface, GML 2.1.2 is used to describe the features. This means that the server must support GML 2.1.2 as output format. Servers are allowed to provide any other output format as well. However, in transactions features have to be expressed in GML 2.1.2.

However, GML 2.1.2 is not the most recent version of GML. GML 3 is the latest, which provides encodings for topologically structured data, temporal aspects and default styling as well.

5 Case study: Notary sketches new parcel

The case study to sketch new parcels and their boundaries in a Web Feature Service plays an important role in the research. The case study has been used to test the abilities of Web Feature Services to process geometric transactions. Therefore both a web feature server and a client have been implemented. In addition, other WFS clients have been tested on the server to test interoperability. This chapter describes the case study itself and the service that has been developed. This comprises the role of the notary in cadastral transactions of parcels (section 5.1.1), a scenario to split parcels (section 5.1.2), the data model (section 5.1.3), the architecture of the service (section 5.2) and the implementations of the web feature server (section 5.3) and client (section 5.4). The last section summarizes the main conclusions and remarks on this case study.

5.1 Case study: the starting point

This section gives some background information on the case study. First, the role of the notary in cadastral transactions is briefly discussed in section 5.1.1. Section 5.1.2 gives scenarios how splitting a parcel could be done using a Web Feature Service. The data model that is needed for implementation of the scenario is discussed in section 5.1.3.

5.1.1 The role of a notary in cadastral transactions

In the Dutch cadastral system, when two or more parties have made an agreement on changing ownership of a cadastral parcel, they have to contact a notary to make a deed in which this agreement is formalized [29]. The notary sends this deed to the Kadaster (the Dutch cadastre). In order to change the ownership juridically, this deed has to be registered by the Kadaster. Changing ownership's rights includes regular transition of the ownership of a single parcel, merging two parcels, or splitting a parcel into two or more parcels (figure 5.1).

In case of a regular transition of rights (e.g. somebody buys a parcel from somebody else), no changes are made in the geometric properties of a parcel. In this case the notary can give a map of the parcel as confirmation which parcel is involved.

When two parcels are merged to one new parcel (in most cases someone buys a neighbouring parcel), the notary could indicate this geometrically as well. However, it would be sufficient if in the administrative system ownership were changed.

Splitting a parcel is the most interesting action (geometrically speaking), since this always involves changes in the geometry of parcels. New boundaries of parcels have to be determined. This involves surveying the new boundaries as well. The process of splitting up a parcel (for example two parties decide that part of one existing parcel will change from the current owner to the other party) is used in the case study and will be described in more detail.

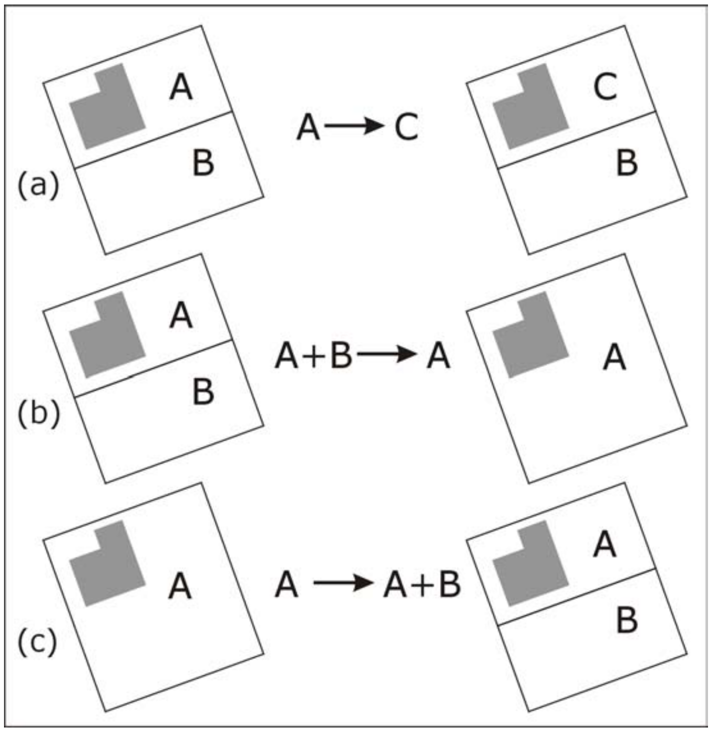


Figure 5.1 Three basic types of cadastral transaction: (a) regular – where C buys the parcel of A-, (b) merging two parcels – where A buys the parcel of B and creates one new parcel - and (c) splitting one parcel – where B buys a part of the parcel of A.

5.1.2 Scenario for splitting parcels using a Web Feature Service

In the current Dutch cadastral system, the process of changing a parcel geometrically - e.g. splitting up a parcel - could be described as follows [29]:

1) When a parcel needs to be changed in a cadastral transaction, a notary initiates the administrative process by sending a deed describing the transfer to the Kadaster. The Kadaster then changes the owner's rights in the cadastral registration (AKR). An annotation is made in the system that surveying has to be done.
2) A cadastral surveyor contacts the parties involved to point out the changes of the parcel's boundaries. Because of efficiency reasons the changes need not be surveyed directly, but this could be done at another time.
3) As soon as the surveying has been done, the surveyor makes the necessary changes (for instance, creating the new parcels and deleting the old one) in LKI, the database with the geometric properties of the parcels. After these changes have been checked and approved, the transaction is completed

At this moment only a sketch of the notary on the cadastral map is provided (in the first step) to indicate where the new boundary should be. This sketch is added as an illustration to the deed the notary sends to the Kadaster. Since the actual surveying of the new boundary could be done a few weeks after sending in the deed [29], there is a (short) period in which the fact that the parcel is involved in a cadastral transaction is known in the cadastral system, but for others this future change is not visible in the LKI-database.

WFS and other new technologies that have been developed in the field of ICT (e.g. XML and XSQL) and interoperable GIS (e.g. GML) might offer new possibilities to the Dutch Cadastre and notaries. When creating the deed the notary could, for instance, also sketch a provisional parcel boundary via WFS at the same time. This way, future changes to the LKI-database are known faster and others (notaries, individuals, municipalities, etc.) could benefit from this as well. In addition, the sketch can be enhanced with extra topographical data or orthophotos. The concept of editing cadastral data in a Web Feature Service will be tested in this case study.

For editing the cadastral data, one could think of a notary who drafts a new boundary, because a parcel has to be split up. This way, changes in the geometry of parcels caused by a cadastral transaction, are geometrically represented in the cadastral database. These drafts are made before the actual surveying is done.

This scenario for splitting a parcel could be used for testing WFS ability to edit geographic data:

1) The notary logs on to the system
2) The notary gives the parcel number of the parcel involved.
3) To draw a map, the server returns[18]:
    a. Lines/boundaries from LKI
    b. Labels/parcels from LKI
    c. Lines/boundaries from DRAFT_BOUNDARY if available
    d. Labels/parcels from DRAFT_PARCEL if any
4) The notary drafts the boundaries to be added and annotates with points and labels (e.g. A, B, C, etc.) as in the deed which part of the old parcel should belong to whom.
5) The request is being sent to the server where it is processed as one transaction. The data are checked and validated, for instance whether the labels are still unique and the features are in the correct area[19]. If this kind of validating tests is passed successfully, the features are inserted.
6) The server returns the result of the transaction.

The data from LKI serve as the background for drawing. With the bounding box other data (like orthophotos or topographic data from other WFS or WMS services) can be added in the client as additional background information.

5.1.3 Data model

For the case study, a selection has been made from the original LKI-dataset. An area of 1 square kilometre near the city of Gouda has been selected for developing the WFS. This area is chosen, because besides cadastral data, other topographic data (Top10NL) is available for this area in another database. This topographic data could be used as additional background data by accessing another web service, e.g. a WMS or WFS, whatever is installed on the database.

[18] On the current cadastral maps street names, numbers of houses and (cadastral) buildings are shown as well. These have been left away for the case study, to make the service less complex.

[19] To perform these operations, it is desired to lock an area. However, WFS does not have an operation to lock an area. See section 6.2.

The cadastral data are stored in millimetres in the database. The data are stored in the RD-coordinate system, which officially uses coordinates in meters. As soon as other data should be combined with this cadastral data, the coordinates of the cadastral data would be 1000 times too large. Therefore the coordinates should be either converted from millimetres to meters in the database or in a view. Since GeoServer 1.1.0 (see section 5.3.2) is not able to retrieve features from views, it is decided to convert the data to meters in the database.

In the case study no attention is paid to topology, because this would be too complex for the proof of concept. Also the table in LKI containing references to "enclaves" in parcels – `parcelover` – is left aside. In addition, it is not necessary to store ownerships data, because these data could be accessed through a reference to the AKR-database[20].

All this leaves two tables that will be used from the current database of LKI: `parcel` and `boundary`. For both of them a selection for the city of Gouda has been made, resulting in the tables `cl_boundary` and `cl_parcel2`. Because `cl_boundary` contained some unnecessary columns for the case study, the table `test_boundary` has been created from `cl_boundary`.

Because the official boundary still has to be surveyed by a cadastral surveyor to be legally binding, the boundary and/or parcel drafted by the notary should not be stored in the original tables with the legally binding boundaries and/or parcels directly. Consequently, the tables `draft_boundary` and `draft_parcel` have been created to store the drafts of boundaries and parcels respectively. Figure 5.2 shows the four relevant tables, Appendix B gives more details on the tables. The notary thus performs transactions on respectively `draft_boundary` and `draft_parcel`.

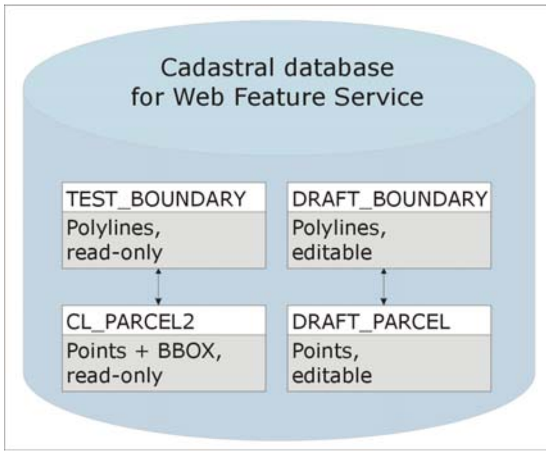


Figure 5.2 Cadastral database containing the 4 relevant tables for the case study

5.2 The service to draft parcels

5.2.1 Service architecture

The service should make it possible for notaries to access the LKI-database over the network, as illustrated in figure 5.3. In addition, topographic data from another Web

[20] The relation between LKI and AKR is made by X_AKR_OBJECTNUMMER in the LKI-database.

Feature Service could be retrieved via internet. The server on LKI and the client should support transactions on `draft_parcel` and `draft_boundary`. Since the notaries do not need advanced GIS-tools for complex analyses of data, a simple front-end (e.g. a browser to view and edit geodata) that can "talk" WFS would be sufficient. This makes the client as light as possible.

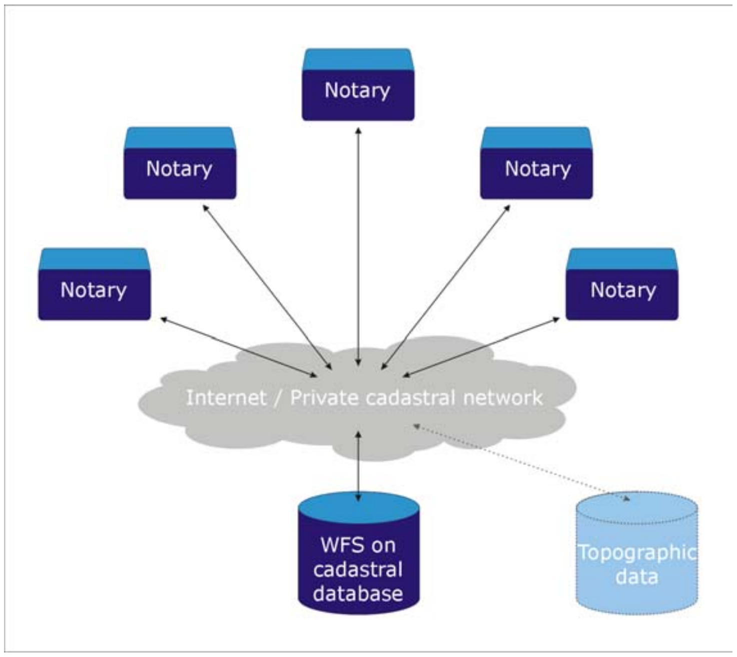

Notary
Notary
Notary
Notary
Notary
Internet / Private cadastral network
WFS on cadastral database
Topographic data

Figure 5.3 Web Feature Service for notaries to draft new parcels and boundaries in cadastral database, optionally using topographic data as background.

The client should consist of a viewer/editor and some part that "talks" WFS, i.e. translates events in the viewer/editor into valid WFS-requests and handles communication of requests and responses with the web feature server. The client should also translate GML data into some graphic representation to display the cadastral data.

The server has a data layer (consisting of an Oracle database with LKI-data), which is accessed by a WFS-layer to "talk" WFS. It will be sufficient if the server provides the data in GML only. Other output formats are not necessary.

5.2.2 Application logic

Shortly, application logic consists of some intelligence (some rules) that concerns the application. For this case study the following issues, of very different kind, can be considered application logic:

- **Authorization and rights of users**. One can imagine that not every user is allowed to perform every (type of) transaction at a server. It is not said that WFS must deal with the authorization, but authorization should be dealt with. For services as developed in the case study, it could be very critical to have some degree of control on users and their rights on the data too. For example, the notary should only be able to do manipulations on features of type `draft_boundary` and `draft_parcel`. For notaries, the other feature types (from

LKI) in the same service are only accessible for retrieving data. Now assume that a surveyor/editor is using the same Web Feature Service to maintain the boundaries, the surveyor/editor then needs rights to edit the LKI-database directly as well. Thus, different users need different rights on different feature types.

Authorization issues and the rights of users should ideally be dealt with at the server side, since the data at the server needs to be protected. Some mediating service could perform this task as well. This could be dangerous when a service is directly accessible for any client (as will be the case in most WFS), since the mediating service could be ignored then.

- **Transaction constraints**. Splitting a parcel in this case study means that at least one boundary and two locations for parcels should be inserted in the draft tables. This means that a transaction should always be composed of an insert-operation for one feature of type `draft_boundary` and two features of type `draft_parcel`, as in figure 5.4. This constrains the transactions' structure.

```
<Transaction>
      <Insert>
            <DRAFT_BOUNDARY>
                  …
            </DRAFT_BOUNDARY>
      </Insert>
      <Insert>
            <DRAFT_PARCEL>
                  …
            </DRAFT_PARCEL>
            <DRAFT_PARCEL>
                  …
            </DRAFT_PARCEL>
      </Insert>
</Transaction>
```

Figure 5.4 Structure of transaction for splitting a parcel, i.e. inserting one new boundary and two new locations for parcels.

Transaction constraints could be dealt with at the server side, client side or by a mediating service, if present. Since transactions affect the data at the server, it seems most naturally to check on these transaction constraints at the server side. The ideal situation would be that at all levels these constraints could be tested, to avoid unnecessary errors and sending transactions that will not be processed anyway.

- **Validation of features**. The features should be validated against certain constraints, both syntactical and semantical by nature. Questions like 'are new features encoded in valid GML?', 'is the date and time of creation of this new feature conform system time?', 'are the drafted parcels (points) on either side of the drafted boundary?' are examples of this. Similar to transaction constraints, validation of features should at least be done at the server. If clients would allow for validation before a transaction is send, clients can be held back to send invalid features, thereby saving bandwidth and processing time at the server. This means however that clients need to know what features are valid and should know what constraints hold.

The WFS specification provides operations for clients to get (part of) this application logic, which is discussed in sections 6.3 and 6.4.

Although important, only some of the issues mentioned above are implemented in the presented case study. The reason for this is that the highest priority has been on implementing both a WFS-client and web feature server to test the suitability of processing transactions.

For Web Feature Services holds what holds for most client/server architectures: the server holds the data and the client is responsible for the user interface. But application logic like above has not been dealt with yet. Where to put these issues always is a design problem. WFS and application specific issues are discussed in Chapter 6.

5.2.3 Requirements for the server

Since new features have to be created in the cadastral database, the web feature server should be transactional. The server has been built on top of the Oracle 9i database, because the data are stored in an Oracle 9i database. In terms of WFS, the server should be able to:

- Interpret *GetFeature* requests to make corresponding SQL-statements and encode the result of the SQL in GML for sending it back to the client.
- Support *filters* to select features on the feature identifier and a spatial filter to select features in a bounding box.
- Interpret *Transaction* requests to make corresponding SQL-statements (i.e. for example, GML-features should be mapped to the table-structure to make SQL-statement for *Inserts*) and encode the new id's (in case of *Inserts*) in a WFS-response to the client.
- Make valid WFS responses for *error reporting*.
- Provide a Capabilities document in response to a *GetCapabilities* request; this could be static XML if the available feature types do not change (that is: new feature types are defined or deleted). Otherwise, this document should be built dynamically.
- Provide XML-schema per feature type in response to a *DescribeFeatureType* request.

Especially the GetFeature and Transaction operations are critical to have. The other operations are important, but not necessary to test the abilities of WFS for transactions. Since GeoServer 1.1.0 on top of Oracle does not allow locking per feature and developing such a locking mechanism is too complex for this thesis, locking is ignored.

5.2.4 Requirements for the client

Functionality necessary or critical for the client to test the concept of editing geodata is discussed here. Other functionality, for instance concerning a good user interface, is left aside in the case study, because this is not critical to test the main goal of a transactional

Web Feature Service: supporting the operations to retrieve geographic features and to support transactions on geographic features.

The goal of the client is to demonstrate and test WFS for the case study. Only functionality necessary for the case study has been implemented. The case study data contains lines and points. Thus, the client will not deal with creating polygons.

The client for this case study, should be able to:

- Connect to a web feature server (preferably more than one) and optionally other services, like WMS.
- support the retrieval of GML for all data (*GetFeature*);
- support a filter for selecting features in the bounding box of a parcel (complex *Filter* operations);
- visualize GML;
- identify (GML-)features, i.e. give the thematic attributes of a feature;
- draw points for features of type `draft_parcel` ;
- draw lines for features of type `draft_boundary`;
- make WFS-requests in XML for inserting new features, deleting features and updating features that can be send with a WFS *Transaction*;
- send the WFS-requests to the server and interpret the responses.

Section 5.4 discusses the implemented client.

5.3 Web Feature Servers

At first, the plan in this research was to develop a transactional web feature server using XSQL, which is the subject of section 5.3.1. While developing a server with XSQL several other alternatives became available for the web feature server. Open Source projects as Deegree [46] and GeoServer [47] released implementations of a web feature server. Since developing a web feature server with XSQL would have consumed lots of time there was decided to investigate the use of web feature servers as Deegree or GeoServer.

The Deegree web feature server was not complete for transactions, in the sense that not all operations were supported for all geometry types (e.g. inserts on points were supported, but not on polylines). Therefore, the Deegree server was not suitable to use in this research.

GeoServer claimed to be fully compliant as transactional web feature server, i.e. for all feature types all transactional operations were said to be supported. Therefore GeoServer was investigated further and used to develop the case study service. How the service has been developed with GeoServer, is discussed in section 5.3.2.

5.3.1 XSQL

Using XSQL, one can retrieve XML from an Oracle database. GML can be generated in an XSQL-page by using XSLT[21]. This principle has been used to generate GML from the cadastral database in the MSc research.

[21] Note that XSLT is not a powerful programming language, but nothing more and nothing less than a tool to transform XML-encoded data in another format (e.g. XML, HTML or text). It has some functions/methods to make computations on the data, but those functions/methods are not very powerful.

As mentioned earlier, with XSQL one can include SQL-statements in a webpage. This includes INSERT, UPDATE and DELETE-statements as well. In this research, editing the data in the database with XSQL has not been tested. Because GeoServer (section 5.3.2) was released while still making valid GML and there were several other steps to be taken before a WFS could be developed with XSQL, there was decided to proceed with GeoServer.

Although XSQL only has been used for retrieving GML from the database, it can be said that XSQL should provide a very flexible way to retrieve and edit data. The reason for this is that, using XSQL, the data from the database is directly encoded in XML. The structure of the data thus remains untouched. This direct encoding in XML, which can be transformed with XSLT to GML, enables the representation of features containing multiple geometric properties. Therefore it should be able to develop a web feature server with XSQL to interpret the requests and the geographic features in requests.

A possible advantage of XSQL is that less components are necessary, since the XSQL pages directly work at the DBMS. So for the server only a DBMS and a webserver with XSQL-pages could be required.

5.3.2 GeoServer: an open source web feature server

On September 22 2003 the GeoServer Project [47] announced its first full release of GeoServer, an open source web feature server. GeoServer is a *full implementation* of the WFS Specification of the OpenGIS consortium. It is developed in Java and supports versions 0.0.14, 0.0.15 and 1.0.0 (the current version) of the Web Feature Service specification. GeoServer version 1.1.0 has been extensively tested against the official OGC test suite [47].

GeoServer can be configured as a transactional Web feature server and is originally developed for PostGIS[22]. But GeoServer is developed to be able to deal with other datasources as well. At this moment, GeoServer can act as a Basic WFS on ESRI's shapefiles and GeoServer could be used on an Oracle Spatial database and ESRI's ArcSDE as a Transactional WFS. The fact that GeoServer supports Oracle is relevant, as Oracle is also the platform of the Kadaster.

[22] PostGIS is an open source spatial database technology that adds support for "OpenGIS style" geographic objects to the PostgreSQL object-relational database [51].

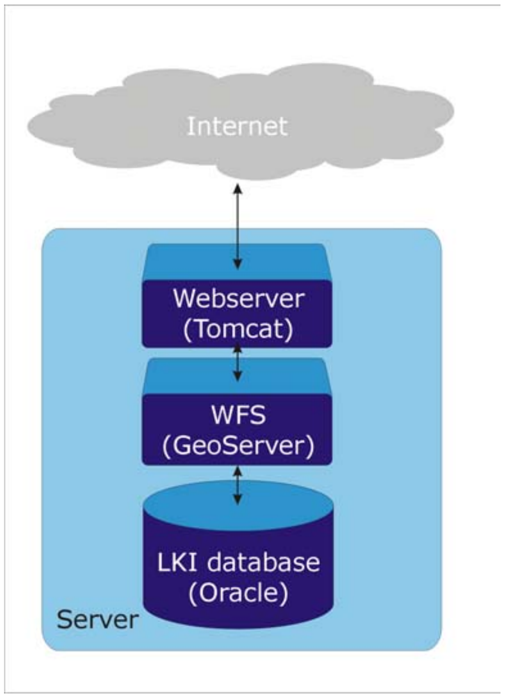


Figure 5.5 The server consisting of the dataset, GeoServer to access the data and communicate according to the WFS-specification and Tomcat as the webserver and Java platform.

Because of its transactional capabilities and because GeoServer already had implemented all spatial filters (as it is a full implementation of the WFS-specification), GeoServer 1.1.0 was chosen for further development of the Web Feature Service. GeoServer 1.1.0 has been configured to serve as a Transactional WFS on the cadastral database LKI in Oracle 9i (figure 5.5). Appendix F gives some details on configuring GeoServer on top of an Oracle database.

GeoServer assumes that each feature is uniquely identifiable by one property. Compound keys therefore are not supported in GeoServer. Since in the original LKI-database parcels and boundaries are identified by a compound key (OBJECT_ID, OGROUP and TMAX), GeoServer could not deal with this data. Therefore an attribute (the name 'FID' has been chosen) has been added to all four tables to act as the primary key.

Furthermore, GeoServer 1.1.0 contained some bugs that troubled especially processing transactions. In the MSc-project time has been spent on debugging and testing, in cooperation with the developers of the GeoServer-project (see Appendix E for an example of a problem). Most problems have been solved so that transactions can be performed. Only *Inserts* on features containing more than one geometric property fail, but it is expected that this will be solved in one of the next versions of GeoServer. Besides that, all transaction operations are supported correctly at the moment. The only thing that is not convenient is that views are not supported yet in GeoServer.

5.4 WFS clients

5.4.1 Existing transactional clients

At first, existing clients that claim to be WFS compliant have been tested. The investigated clients are ESRI's ArcExplorer, Intergraph's GeoMedia Viewer, Moximedia's client (http://www.moximedia.com) and the Open Source clients GeoClient (http://www.mycgiserver.com/~amri/geoclient.cocoon.xml) and OpenMap (http://openmap.bbn.com).

According to ESRI, ArcExplorer (free available for download at http://www.esri.com) should be able to act as a basic WFS-client after installation of a plug-in. However, ArcExplorer – version 4.0.1, Java has been tested - can't connect to any other Web Feature Service than services based on ESRI's Arc-products. ArcExplorer thus is not suitable for visualizing the GML output of the web feature server.

Intergraph (http://www.intergraph.com) offers GeoMedia Viewer, which can be ordered for free. This viewer can be extended to have a Web Feature Service as data source. Data can only be retrieved, so the viewer is not transactional. In this research, the viewer has been used for testing interoperability of the server (Chapter 6).

Moximedia offers a (commercial) transactional WFS client. This client is part of the framework Moximedia offers for OpenGIS web services. However, the client does not visualize GML, but uses a Web Map Server for visualization. This makes querying the data more difficult and thus less suitable for the case.

The open source initiative called GeoClient currently can only connect to a sort of stripped down version of a web feature server and therefore is not fully WFS-compliant. Since editing is not supported, this client is not suitable for the case study.

BBN Technologies' OpenMap package is an Open Source JavaBeans based programmer's toolkit [50]. Using OpenMap, one can quickly build applications and applets that access data from legacy databases and applications. OpenMap provides the means to allow users to see and manipulate geospatial information. OpenMap does not support WFS as a data source, but can be extended. However, this demands quite an effort.

The conclusion of the above is that, although many clients (websites) are available for web mapping, for Web Feature Services and in particular transactional Web Feature Services, not much clients are available yet[23]. Adapting these existing clients is either not possible, since the source code is not available, or too much time would be consumed with it. Therefore it is decided to develop a transactional WFS client for this research.

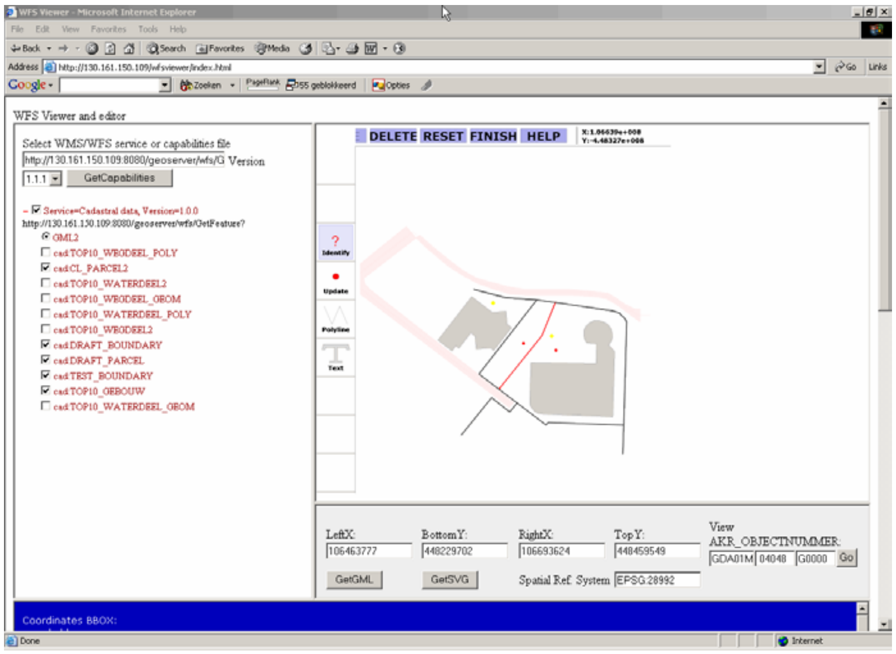
5.4.2 Developed client in the case study

The developed client is based on a client made by the Section GIS Technology (TU Delft) that has been used to visualize GML-encoded data from Top10NL as SVG (Scalable Vector Graphics) in a web browser [33, 34]. This SVG-client is adapted to use a Web Feature Service as data source. The client did not support transactions yet. Therefore the

[23] At the website of OpenGIS (http://www.opengis.org/resources/?page=products) conforming products are listed.

client has been extended with functionality to draw and encode features and send these in a WFS-transaction to the server.

Both the *GetCapabilities* and the *GetFeature* request (for retrieval of the features in GML) are send as Keyword-Value-Pair-requests and are issued using Javascript. If SVG is chosen to visualize the data, XSLT is used in the browser to do the transformation from GML to SVG. A screenshot of the client in figure 5.6 shows some features in SVG. The transformation to SVG is done at the client side, after receiving the GML.


WFS Viewer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Go Links
Zoeken PageRank 55 geblokkeerd Opties
GetCapabilities
DELETE RESET FINISH HELP
Identify
Update
Polyline
Text
GetGML
GetSVG
Go
Coordinates BBOX:
Done
Internet

Address: http://130.161.150.109/wfsviewer/index.html

WFS Viewer and editor

Select WMS/WFS service or capabilities file: http://130.161.150.109:8080/geoserver/wfs/G Version 1.1.1

- [✓] Service=Cadastral data, Version=1.0.0
http://130.161.150.109:8080/geoserver/wfs/GetFeature?
[✓ GML2]
- [ ] cad:TOP10_WEGDEEL_POLY
- [✓] cad:CL_PARCEL2
- [ ] cad:TOP10_WATERDEEL2
- [ ] cad:TOP10_WEGDEEL_GEOM
- [ ] cad:TOP10_WATERDEEL_POLY
- [ ] cad:TOP10_WEGDEEL2
- [✓] cad:DRAFT_BOUNDARY
- [✓] cad:DRAFT_PARCEL
- [✓] cad:TEST_BOUNDARY
- [✓] cad:TOP10_GEBOUW
- [ ] cad:TOP10_WATERDEEL_GEOM

LeftX: 106463777 BottomY: 448229702 RightX: 106693624 TopY: 448459549
View AKR_OBJECTNUMMER: GDA01M 04048 G0000
Spatial Ref. System EPSG:28992

Figure 5.6 Viewing features as SVG from a web feature service

For navigation (zooming and panning) the client uses standard SVG-functionality. Support for SVG has to be available in the browser; a plug-in (e.g. from Adobe) can be used for this. Note that not for all web browsers such a plug-in is available, for instance for Mozilla. The thematic attributes of features can be queried by clicking on a feature. A table with the attributes is generated from the GML in the client.

The part of the client that is responsible for the retrieval and visualization of GML is generic in the sense that all Web Feature Services could be used as a data source. For the case study, support for transactions had to be implemented however. The functionality to draw and encode new features is developed for the case study and is not generic (yet).

Transaction support at the client side has been implemented with Java Server Pages (JSP, [41]). JSP has good support for XML and with JSP it is easy to add (complex) functionality to a web-based application like the WFS-client. This support for XML is necessary to compose and send the requests to the web feature server.

In order to use JSP-technology, a Java servlet container like for instance Tomcat [45] or Resin [52] needs to be installed at a server. Processing is partially done at the server, the results are returned to the browser. This way, functionality of the WFS-client is distributed over the browser and a web-server (figure 5.7). Note that the machine that has been used for the client and the web feature server are in fact not the same server.

Using Java Server Pages makes the user-side of the client light, because most processing (e.g. constructing the transaction requests) is done at the Java web server. The browser does not need to download plug-ins, applets or what so ever. However, there is more traffic on the network, because the browser has to communicate with the server where the Java Server Pages are.

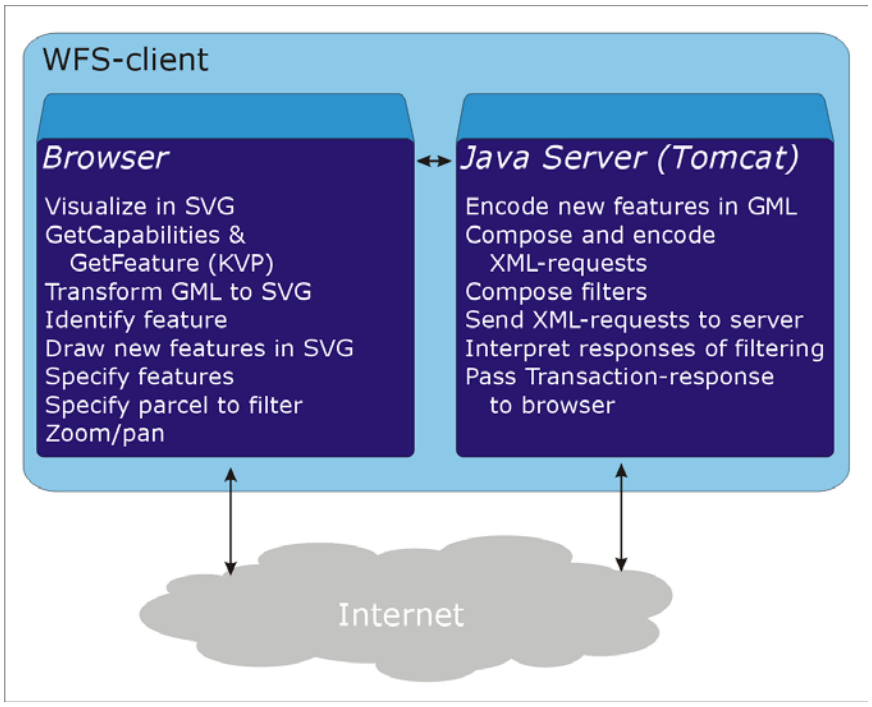


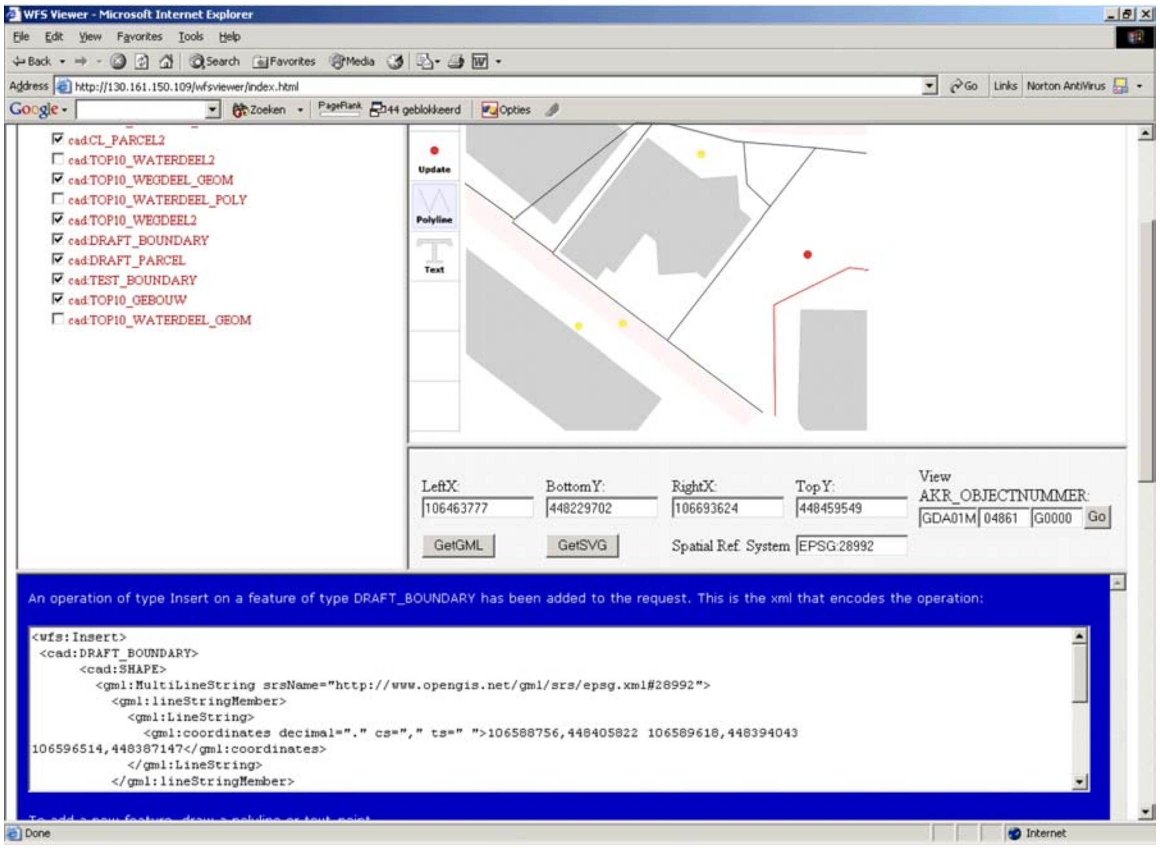
Figure 5.7 Functionality of the WFS-client divided between browser and server

Functionality to compose and send Transaction requests is divided between browser and web server as follows:

- *Inserting new features*. The shape of a feature (i.e. its coordinates) is drawn in SVG. The coordinates are passed to a Java Server Page that creates a form to fill in the other thematic attributes of the new feature. After submitting the form, the feature is encoded with JSP (at the server) as GML. Then, an XML-string containing the GML-feature and WFS-transaction-elements is sent to the browser, see figure 5.8. This string is added to the Transaction request.
- *Deleting and updating features*. The feature is selected in SVG and its feature identifier is used by the Java server to create an HTML-form from the feature in GML. Then the user can choose to specify values and update the feature or to delete the feature. Both are dealt with in JSPs to construct the corresponding part of the WFS-request. This part is again passed to the browser and added to the Transaction request.
- The user can *send the Transaction request* from the browser to a JSP. This will send the (raw) XML-request to the web feature server and passes the response (successful or failed) of the server to the browser.

The filters in XML are encoded at the server side of the client, i.e. in the JSP on Tomcat. It concerns a filter to select a feature on its feature identifier and two filters to select features of different types that lie in the bounding box of a parcel.

For more details on the client, see Appendix C. That appendix provides more detailed descriptions on what functionality is performed where and gives some schemas on the different Java Server Pages that have been developed.


WFS Viewer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://130.161.150.109/wfsviewer/index.html
Go Links Norton AntiVirus
Google Zoeken PageRank 44 geblokkeerd Opties
Update
Polyline
Text
Done
Internet

- [✓] cad:CL_PARCEL2
- [ ] cad:TOP10_WATERDEEL2
- [✓] cad:TOP10_WEGDEEL_GEOM
- [ ] cad:TOP10_WATERDEEL_POLY
- [✓] cad:TOP10_WEGDEEL2
- [✓] cad:DRAFT_BOUNDARY
- [✓] cad:DRAFT_PARCEL
- [✓] cad:TEST_BOUNDARY
- [✓] cad:TOP10_GEBOUW
- [ ] cad:TOP10_WATERDEEL_GEOM

LeftX: 106463777 BottomY: 448229702 RightX: 106693624 TopY: 448459549

View AKR_OBJECTNUMMER: GDA01M 04861 G0000 Go

GetGML GetSVG Spatial Ref. System EPSG:28992

An operation of type Insert on a feature of type DRAFT_BOUNDARY has been added to the request. This is the xml that encodes the operation:

```
<wfs:Insert>
 <cad:DRAFT_BOUNDARY>
      <cad:SHAPE>
        <gml:MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
          <gml:lineStringMember>
            <gml:LineString>
              <gml:coordinates decimal="." cs="," ts=" ">106588756,448405822 106589618,448394043
106596514,448387147</gml:coordinates>
            </gml:LineString>
          </gml:lineStringMember>
```

Figure 5.8 From SVG to GML/WFS. The white box below – in the blue frame - contains a part of the WFS-request to add to the Transaction request.

5.5 Conclusion

These conclusions can be drawn from this chapter:

- Boundaries drawn or changed by a notary are not stored in the table with the current boundaries directly, because the new or changed boundaries still have to be surveyed by a cadastral surveyor. Consequently, tables containing the draft boundaries and parcels have to be created in the same DBMS.
- With XSQL, one can retrieve and edit XML-encoded data from or in a database by using SQL-statements. In this research, data in the database has not been edited with XSQL nor a complete web feature server has been built with XSQL. But GML can be generated in an XSQL-page by using XSLT. With XSQL and XSLT the features from the cadastral data containing multiple geometric attributes can be presented in GML without losing any data and more specific losing any geometric attribute(s). Because SQL-statements to edit data are part of

XSQL as well, it should be possible to develop a transactional web feature server with XSQL.

- By adding a new column as identifier to each table in Oracle, it is possible to have a Transactional WFS running on the cadastral database (in Oracle) using GeoServer 1.1.0. GeoServer is a full implementation of the WFS-specification. All operations from the WFS-specification and all filters from the Filter Encoding are supported. This makes GeoServer very suitable for testing WFS.
- Since the cadastral data are stored in millimetres and the coordinate system (the Dutch RD-system) that is used for the data officially is in meters, the cadastral data has been converted to meters. This way, data from other sources (e.g. WFS, WMS) can be added in a client.
- Because existing WFS-clients are either not fully compliant with the specification, nor transactional or it costs too much time to adapt them for the case study service, a client needed to be developed in this research. The developed client uses SVG for visualization. SVG can be generated by transforming the GML stream with an XSLT stylesheet. The client uses a standard web-browser and Java Server Pages (JSP). The JSPs are used for implementing functionality to compose and send the transaction request. This way, the client remains extremely light at the browser.

6 Suitability of WFS for transactions: analysis

For this research, the main research question is on the suitability for transactions of WFS as an interoperable service. This chapter analyses some important topics. First, tests on interoperability of the implemented software (both client and server) are described in section 6.1. In section 6.2 the way WFS handles transactions is discussed. When performing transactions, features are modified. These features could be validated against the constraints datasets have. How clients and servers could deal with this validation, is the topic of section 6.3. Section 6.4 discusses other application logic that is relevant for transactions. Conclusions on WFS' suitability for transactions are drawn in section 6.5.

6.1 Interoperability testing

A basic principle for interoperable services, in this case WFS, is that any WFS client should be able to communicate with any web feature server that is based on open standards. This means that both client and server have to comply with the specification, i.e. open standard. If the specification does not contain errors or omissions, any generic WFS-client and WFS-server should be able to interoperate.

6.1.1 Server

Several interoperability tests have been performed 1) to test whether the server and client of the implemented service are compliant with the WFS specification or not and 2) to test whether other WFS products can interoperate with the server or client. Validation of output (responses, including the GML output stream) of GeoServer and using other WFS clients were used to test interoperability.

For interoperability of the developed server, it is important that the web feature server can provide valid GML. Therefore GeoServer has been tested on the GML it sends to clients. This test showed that the GML was not valid at first. This was not caused by GeoServer, but by errors in the hand made XML Schema documents for two feature types. After correcting these errors using standard XML tools (see Appendix F for more details), the GML tested valid.

The above shows that validation of XML is essential to make sure that the XML output (in this case GML) is compliant with the specification. If the input or output of the XML is not compliant with the service, the service is not truly interoperable. This results in generic clients (as GeoMedia Viewer, see section 6.1.2) that can't use the service correctly, for example.

GeoServer has been tested for transactions using the client of the case study from Chapter 5. All kinds of *Transaction* requests have been constructed based on the WFS-specification.

Transactions consisting of insert, update and delete operations, in random order and different quantities have been sent to the server with the case study client. These transactions all were processed successfully and the responses are valid WFS responses.

In addition, other requests (*GetFeature, GetCapabilities, DescribeFeatureType*) have been sent to the server, which also are processed successfully. All requests have been constructed (and have been validated) as in the specification. Table 6.1 lists the requests of which the responses of the web feature server have been tested and validated using XMLSpy (Appendix F).

Response to request	Additional tested for	Valid response
GetCapabilities		✓
GetFeature (output GML (2.1.2))	DRAFT_BOUNDARY	✓
	DRAFT_PARCEL	✓
	TEST_BOUNDARY	✓
	CL_PARCEL2	✓
	TOP10_GEBOUW	✓
Transaction	Insert operations mixed with delete and delete operations	✓
Incomplete transaction	Error report is validated	✓

Table 6.1 Requests of which responses of GeoServer have validated against the corresponding schemas.

GeoServer claims to be compliant to the WFS-specification[24]. GeoServer 1.1.0 has passed the CITE tests of the OpenGIS Consortium. These tests are developed by the OGC and are used to test whether a product is compliant to the specification. Based on this, one may conclude that GeoServer is fully compliant to the WFS specification.

To test interoperability of the server in practice other generic clients have been used. One client managed to connect to the web feature server and retrieve features. This is described below.

6.1.2 Clients

Besides the case study client, GeoMedia Viewer with an extension for WFS has been tested as a basic client[25]. GeoMedia Viewer does not support editing data, but can retrieve data from a Web Feature Service. With this viewer, the service has been accessed. The results of this test are:

- Some changes to the schemas are necessary for GeoMedia Viewer to understand the schemas. GeoMedia Viewer does not accept restrictions (see Appendix D for a full schema of `draft_boundary` and `draft_parcel`) that are defined in the XML-schemas of feature types. If these parts of the schema are left away (see figure 6.1), GeoMedia Viewer accepts the features.

[24] In fact GeoServer is the reference implementation of a web feature server, which means that GeoServer can be used to evaluate other implementations (http://cite.occamlab.com/reference/).

[25] ESRI's ArcExplorer also has been tested, but never managed to connect to the web feature server, see section 5.4.1.

- All features are completely retrieved and visualized.
- GeoMedia Viewer orientates all features differently (the x and y coordinates are swapped and the features have been mirrored in the y axis) than the case study client. GeoMedia Viewer gets the correct coordinate system, but somehow features are not displayed correctly, see figure 6.2. However, this is an internal problem to deal with the features from the GML encoding in GeoMedia Viewer.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema
  targetNamespace="http://130.161.150.109:8080/geoserver"
  xmlns:cad="http://130.161.150.109:8080/geoserver"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="1.0">
<xs:import namespace="http://www.opengis.net/gml"
schemaLocation="http://130.161.150.109:8080/geoserver/data/capabilities/gml/2.1.2/feature
.xsd"/>
<xs:complexType xmlns:xs="http://www.w3.org/2001/XMLSchema" name="DRAFT_BOUNDARY_Type">
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
        <xs:element          name="SHAPE"          minOccurs="0"          nillable="true"
type="gml:MultiLineStringPropertyType"/>
        <xs:element name="OBJECT_ID" minOccurs="0" nillable="true" type="xs:decimal"/>
        <xs:element name="CLASSIF" minOccurs="0" nillable="true" type="xs:decimal"/>
        <xs:element name="STATUS_CD" minOccurs="0" nillable="true" type="xs:decimal"/>
        <xs:element name="OBJECT_DT" minOccurs="0" nillable="true" type="xs:decimal"/>
        <xs:element name="TMIN" minOccurs="0" nillable="true" type="xs:decimal"/>
        <xs:element name="TMAX" minOccurs="0" nillable="true" type="xs:decimal"/>
        <xs:element name="SOURCE" minOccurs="0" nillable="true" type="xs:string"/>
        <xs:element name="QUALITY" minOccurs="0" nillable="true" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element              name='DRAFT_BOUNDARY'              type='cad:DRAFT_BOUNDARY_Type'
substitutionGroup='gml:_Feature'/>

</xs:schema>
```

Figure 6.1 Stripped XML Schema document that is accepted by GeoMedia Viewer, because all restrictions have been left away.

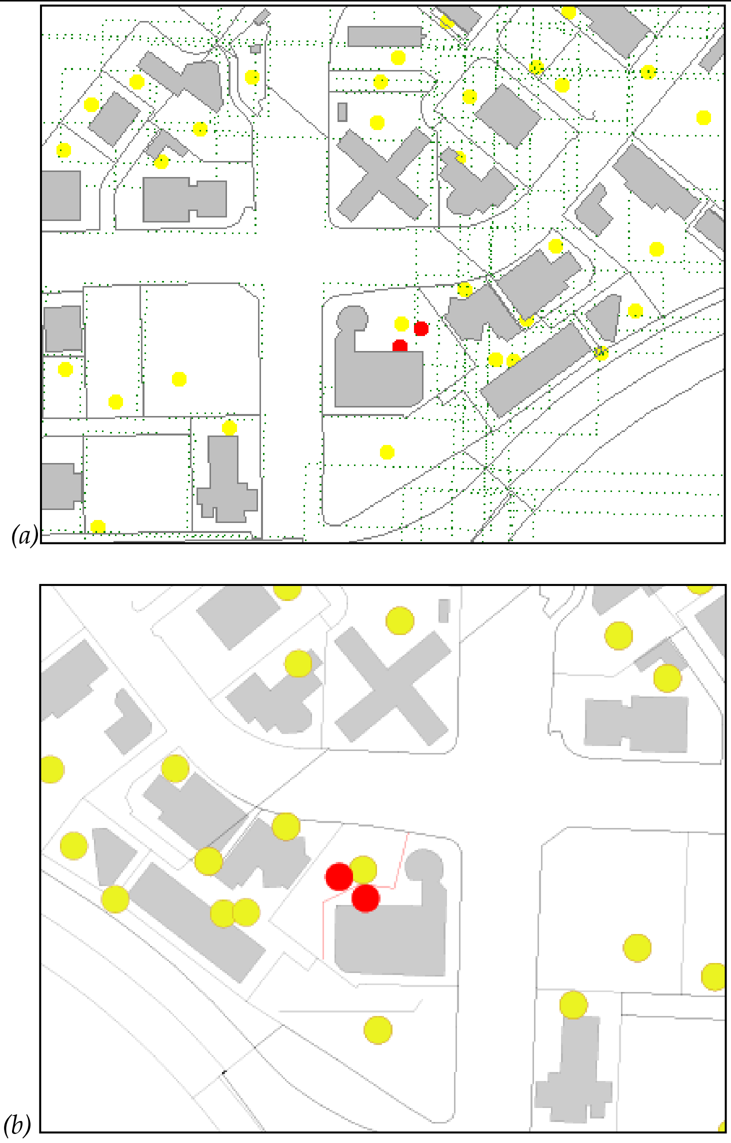


Figure 6.2 (a) is a screenshot of GeoMedia Viewer, (b) of the case study client of approximately the same area. It shows that the orientation of features in GeoMedia Viewer is quite different from the orientation in the case study client.

However, GeoMedia Viewer managed to connect to the server and retrieve all features. This shows that WFS actually can work as an interoperable service. That is, some client, not knowing anything about the internal setup of the server, can retrieve data from that server because both WFS client and the web feature server are compliant with the WFS-specification. If existing applications (like GIS-software for analysis or viewers) are able to connect to a web feature server, data could also easily be used in thick clients like GeoMedia Pro or ArcGIS.

The fact that WFS uses standard web technologies as HTTP and XML (GML and the WFS-requests/responses self) makes integration of geographic data in web-applications relatively easy. Common web technologies like Java Server Pages and SVG can be used for that, since these are very suitable to deal with XML. In addition, for these common standards (technologies) all kinds of tools – both commercial and freeware - are available, which makes development of open, interoperable services less difficult.

6.2 Handling transactions

For processing geometric transactions, WFS only provides an interface. This means that WFS describes the operations and the requests/responses for sending geometric transactions. Processing a transaction (this includes how the transaction is dealt with in the database, e.g. atomic) is up to the server. Below, some issues are discussed that are important to make sure that the transaction is processed correctly.

- Transactions have to start and end with an operation like START and COMMIT/ABORT. In WFS, the transaction is handled as an XML-element. That is, it has to start with <Transaction> and end with </Transaction>, everything in between is the content of the transaction. The server can assure that the entire transaction is received, by checking that the XML-request is well formed and valid. This means that any incomplete request, for example because the network connection was lost while sending the transaction, can easily be rejected.
- Transactions may consist of several operations that modify features. The fact that transactions in WFS can consist of Insert, Update and Delete-elements is very convenient for sending large, complex transactions to one server. In the case study, composing and sending such transactions has been accomplished. WFS transactions can be dealt as 'true' database transactions (in case a web feature server is based on a DBMS), in the sense that if one of the operations (that is part of the transaction) fails, the entire transaction fails.

  However, web feature servers are allowed to process transactions partially, e.g. if the server is not based on a DBMS. How a transaction is dealt (e.g. atomic or not), can't be advertised by the server. This means that clients can't know how transactions are dealt at the server, even if they have asked for the capabilities.
- Important to note is that transactions, as defined in the WFS specification, can only be performed on one web feature server. Transactions running over multiple servers can only be performed in another way. For example, this can be done by using the locking operations and some mediator that manages the locks. That mediator manages the "parent" transaction (i.e. the transaction that accesses several servers) by splitting that up in a transaction for every server that is involved. Every individual transaction must succeed to make the parent transaction succeed.
- Processing the transactions includes ensuring serializability. Using and managing locks to ensure serializability is the responsibility of the server. However, the WFS specification describes operations to ask for locks and release locks at the feature level. This should allow clients to request servers to acquire and release locks, because they want to modify certain features. The locking-operations of WFS appear provide was to ask for locks and release these again on the request of a client. These could be used to implement long transactions, but this has not been tested.

- For certain operations to edit geographic databases, it is necessary to lock an area. For example, if some person is editing features in an area, it could be forbidden for others to add new features to that area, since this can cause inconsistencies. The WFS specification only defines operations to lock individual features, not areas.
- Compound keys are not allowed in WFS. In addition, it is not possible to advertise in the XML Schema definition of a feature type which attribute is the identifier, since GML always uses the fid-attribute to indicate the value of the id.

6.3 Validation of features

Datasets often have restrictions that features should meet with. For instance, a road cannot intersect a house, a house should at least have an area of 10 square meters, addresses must have a postal code or a boundary must have a reference to the parcel left and right of it. Of course, these restrictions should also hold for all new features or changed features. Validation of (changes in) features should prevent that a dataset will contain invalid features.

The server would be the best place to perform the validation checks. However, if the client is able to construct features that already are consistent with the constraints in the data model of the server, unnecessary transactions can be avoided.

Since a transactional web feature service could have numerous clients that modify features, the chances increase that invalid features are submitted to the dataset. This makes validation a very important issue. The WFS-specification defines some operations and mechanisms that can be used for validation. To what extent these mechanisms are sufficient for validation of features, is the topic of this section.

6.3.1 Validation of single features

In any Web Feature Service, every feature type must be defined in an XML Schema document at the server side. This XML Schema document may contain all kinds of constraints on attributes/elements of a feature, see figure 6.3 for an example. Such a schema can be used to validate features at the server side, not between features or objects.

In addition, clients can ask for the schema description by sending the *DescribeFeatureType*-request. Clients could validate newly created or changed features against the corresponding schema, before sending them to the server. Validation checks thus can be also done at the client side. However, for some clients these schemas are difficult to understand, as described in section 6.1 for GeoMedia Viewer.

```
<xs:complexType name="DRAFT_BOUNDARY_Type">
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
        <xs:element name="FID" nillable="false" minOccurs="1" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
```

```
       <xs:element     name="SHAPE"     type="gml:LineStringPropertyType"     minOccurs="1"
maxOccurs="1"/>
         <xs:element name="OBJECT_ID" nillable="true" minOccurs="1" maxOccurs="1">
           <xs:simpleType>
             <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
             </xs:restriction>
           </xs:simpleType>
         </xs:element>
          <xs:element name="CLASSIF" nillable="false" minOccurs="1" maxOccurs="1">
           <xs:simpleType>
              <xs:restriction base="xs:integer">
               <xs:maxLength value="11"/>
             </xs:restriction>
           </xs:simpleType>
          </xs:element>
…
</xs:complexType>
```

Figure 6.3 Example of a part of an XML-schema. Restrictions are defined by the <xs:restriction>-element.

If a client does not validate features for some reason (a client is not obligated to validate features against the schema), the server could still do the validation. The server can do this by validating all new features and changes against the schema description in the web feature server, but this is not obligatory in the WFS-specification.

Validation of data is ideally done as close as possible to the database self. The reason for this is that inconsistencies may arise in the database because the validation process was missed[26].

In the database constraints can be defined. For instance, the length of an attribute can be constrained in the definition of a table in a relational database. If a feature is passed from the web feature server to the database and the database does not accept the feature because an attribute is not valid (e.g. too long), the database throws an exception. This exception is passed to the web feature server. The web feature server then constructs an error report and sends that to the client, as in figure 6.4. The WFS specification does not specify the contents of the error report. So the error message could literally be the error message of the DBMS, but it may also be some error of the web feature server that is installed on the DBMS. In any case, the result is that the server has not accepted the feature, so the dataset is still correct in the sense that it only contains valid features. The client has also been notified that the transaction failed.

```
<wfs:WFS_TransactionResponse
   version="1.0.0"
   xmlns:wfs="http://www.opengis.net/wfs"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wfs
http://130.161.150.109:8080/geoserver/data/capabilities/wfs/1.0.0/WFS-transaction.xsd">
   <wfs:TransactionResult handle="">
      <wfs:Status>
         <wfs:FAILED/>
      </wfs:Status>
      <wfs:Locator></wfs:Locator>
      <wfs:Message>org.vfny.geoserver.responses.wfs.WfsTransactionException:   Row   adding
failed.
        at
org.vfny.geoserver.responses.wfs.TransactionResponse.execute(TransactionResponse.java:289
)
```

[26] This does not mean that validation should only be done at the database. It can be very beneficial if validation also is done in the client to give feedback to the user quickly.

```
        at
org.vfny.geoserver.responses.wfs.TransactionResponse.execute(TransactionResponse.java:66)
        at org.vfny.geoserver.servlets.AbstractService.doService(AbstractService.java:280)
        at org.vfny.geoserver.servlets.AbstractService.doPost(AbstractService.java:225)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:760)
        …
        </wfs:Message>
        </wfs:TransactionResult>
</wfs:WFS_TransactionResponse>
```

Figure 6.4 Example of an error message, caused by an attribute that violated a constraint in the database. The message itself (between <wfs:Message> and </wfs:Message>) is constructed by the web feature server.

However, besides some elements to indicate that an error occurred, the web feature server is completely free to specify the contents of the error message. This can result in ugly, hard to understand messages, like in figure 6.4. For a generic client these messages cannot be interpreted to give useful feedback to the user. A user thus may not understand what went wrong when sending the transaction. In practice, to give useful feedback to a user, a generic client therefore should validate features itself.

If error messages were standardized (e.g. using error codes), it would be easier to give useful feedback to clients. Of course, it is very difficult to standardize all errors of different (heterogeneous) systems. However, common errors like features that don't validate against the schema or features that violate a constraint should be standardized. This way, generic clients can give useful feedback for at least part of the errors ('known' or standardized errors, like in HTTP) that might occur.

6.3.2 Validation of features in their environments

As described in Chapter 3, changes in features could affect other (already existing) features in the dataset as well. In a topologically structured dataset features cannot just be inserted, changed geometrically or deleted, since relations between features are stored and modifications affect other features than the features in the transaction alone. Spatial constraints (e.g. a road is not allowed to intersect a house) on non-topologically structured data could also be relevant. Therefore some validation and processing is necessary to maintain the topologic structure or to check other spatial constraints.

The current specification of WFS (1.0.0) uses GML 2.1.2 to describe new features in insert operations. Since topology is not supported in GML 2.1.2, topology is hard to describe in the transaction itself. Thus, the server should do some checking and maybe some extra processing to insert new features. But deletion of features and changes in geometry affect the topologic structure as well. Therefore, the server should perform validation checks and compute the topologic structure of the affected area again for every transaction that is performed.

Take for example a topological model where it is necessary that objects refer to other objects. If new features are created, these new features must refer to other (possibly new) objects as well, even before the transaction is sent. Otherwise the transaction contains invalid features. This means that these new features have to know the id of features that don't have an id yet.

This problem can be solved by either reserving identifiers at the server before new features are created (so the client requests new ids from the server, creates new features and then sends the transaction), or by providing temporary references which are changed into the actual ids as soon as the transaction is processed at the server. The latter leaves the database in an inconsistent state until the temporary ids have been replaced, so preferably new ids are acquired before the database is affected. But in WFS there is no such mechanism or operation.

A good generic client might understand topological structures. For such a generic client it is difficult to provide features that are valid in the topologically structured dataset, because clients don't know what that structure exactly is. If a server would also have an output format that supports topology (e.g. GML 3.0), clients could retrieve topology from the server. However, transactions are still difficult to process, since a client must send its features in GML 2.1.2 format to the server at this moment. Topology then can hardly be taken into account if the current WFS-specification is implemented.

The future version of WFS will be based on GML 3. Because GML 3 is more comprehensive and complicated than GML 2.1.2, GML profiles could be used to make the use and implementation of GML 3 easier. Discussion to define such GML 3 profiles is going on at the moment.

For other spatial constraints holds actually the same. Some additional checking is necessary, before the actual transaction is performed. If features do meet with the spatial constraints, then the features can be committed to the dataset. Otherwise, the transaction will fail entirely on a DBMS based WFS (or partially on non-DBMS based servers, since these do not support true transactions) and an error message must be sent to the client.

Validation and processing features at the server side can be implemented, without violating the WFS-specification. The reason for this is that it is an issue for the server to deal with internally[27]. However, generic clients will not know anything about restrictions (topological and other spatial constraints) on the relations between other features unless these are encoded somewhere, e.g. in the GML application schema. If features are not accepted, the web feature server will respond with an error report. But again, since the contents of the error message are not standardized, it is difficult for a generic client to interpret these messages and give feedback to the user.

6.4 Application logic

This section deals with application logic in systems that use WFS for exchanging and manipulating geodata. Examples of application logic are mentioned in Chapter 5, like authorization, users rights, billing/costs and transaction constraints. Spatial constraints on features can be considered application logic as well. To what WFS is capable of dealing these constraints, has already been discussed in section 6.3.

Regarding application logic, choices have to be made where to implement this. WFS limits most of these choices because the specification defines the operations and the requests and responses both client and server should deal with. For transactional WFS, clients must have functionality to compose the requests for manipulation of features and functionality to communicate these requests to the server. A client is not allowed to do that in another way. For example, just sending coordinates and the type of a new feature to the server and construct new features at the web feature server itself is not allowed.

[27] For example, GeoServer 1.2 is going to support some validation checks internally.

Some choices are thereby already made; some issues concerning application logic still have to be dealt with in the service.

Application logic like *user rights and authorization* are not dealt with in the web feature service itself. This means that anyone with access to the network where the server resides on (e.g. the Internet) could have access to the service. For authorization sufficient technologies are available (e.g. SHTTP or certificates). Authorization itself therefore is not a big problem.

Once a user is logged on, a user can use *GetCapabilities* to know what operations are allowed at what feature type. However, in a Web Feature Service it is not possible to constrain the rights of users per user type. So if it is necessary for example to allow that user type A may read and write feature type I, then it is not possible in WFS to allow user type B to only read feature type I. Therefore some application service should manage user rights or the server should do that some other way, since it is not dealt with in the WFS specification.

There is an element in the Capabilities document (`<AccessConstraints>`) that may contain a text block to describe any access constraints of the service. The service provider should make sure that a clear description is given in the `<AccessConstraints>` how a client can get access. This element could be used to refer to an application service or an application that manages authorization, but how a client should interpret this - for instance to access that application service -, is not clear since the server is completely free (there are no fixed codes) to fill in the element. The other way around, how web feature servers can be protected against clients that access the service and ignore the application service that manages authorization (if possible), is not discussed in the WFS specification either.

As mentioned in Chapter 5, constraints may exist on *the structure of a transaction*, that is constraints[28] on how many and which operations may be used in a transaction. The server should check these constraints or some mediating application service could do that for the web feature service. However, if the constraints are checked somewhere else than at the server, web feature servers should make sure that clients use those application services and that application services have the "knowledge" they need to check the constraints. This seems to be sensitive for errors, since changes in application logic should also be known in the application service.

If WFS deals with such transaction constraints directly, checking transactions - or validating transactions – would be easier and more robust. For instance, similar to feature type schemas, WFS could also provide XML-schemas for transactions that clients can request, to make sure that the transactions they send are of the correct structure. It also seems to be more natural to do these checks (if necessary) at the server, because the type of modifications that is allowed on a dataset is tightly coupled to the data layer and consequently to the web feature server or the DBMS.

[28] In fact, dealing with all kinds of constraints in a data model is a generic problem. Using a Model Driven Approach (MDA) constraints have to be modeled first, for instance in UML (Unified Modeling Language) and OCL (Object Constraint Language). The actual implementation is a derivative of the model. For more information on MDA, UML and OCL, visit the website of the Object Management Group (http://www.omg.org/, [49]).

From this section becomes clear that transactional web feature services could demand for some mediating service to manage user rights and other application logic, e.g. billing. This service is an application service for the web feature service, which can be accessed by clients.

But if it is compulsory to use some application service, there still needs to be a way to protect the web feature service against clients that try to send requests to the web feature service and ignore the application service. The WFS specification does not define how to refer to application services that should be used with the web feature server. The problem is thus how to make sure how requests from (possible) other clients that do not use the appropriate application service, are rejected.

Dealing with application logic is a generic problem for all kinds of web services. Dealing with this application logic and issues concerning application services are not treated more extensively, because that is out of the scope of this thesis.

6.5 Conclusion

The following conclusions can be drawn from this chapter:

- As should be the case with interoperable services, the developed server can be accessed with another (WFS-)client than the developed client. With another client it is possible to retrieve data from the server, although there still are some problems with retrieving all data correctly. This shows that realizing robust, interoperable services is quite a difficult task, because implementation of the specification is a very exact matter. Common tools as XML validators are helpful when building interoperable services. The fact that features from an unknown data source can be retrieved, visualized and queried shows the power of (interoperable) Web Feature Services.
- Transactions can be composed of Insert, Update and Delete operations very flexible. The number of these operations and the types can be mixed freely. This is very convenient for large or complex transactions, but it can cause problems if transactions are constrained somehow.
- For transactional web feature servers, validation of features is important. WFS provide operations to request the schema description of feature types. With this schema description WFS can validate single features, that is, the attributes of features can be checked. This is necessary to create valid new features.

  Validation against other constraints (e.g. topological or spatial constraints) is not dealt with by WFS. For clients it is not possible to get descriptions of such constraints. In addition, if features do not pass the constraints, it is not possible to inform clients clearly about the constraints that have been violated. The reason for this is that the error messages are not standardized but depend on the web feature server. This way, (generic) clients cannot interpret these messages to give useful feedback to a user and thus it is difficult for a user to correct the features if necessary.
- Since WFS uses GML 2.1.2 in the interface and GML 2.1.2 does not support topology, transactions on topologically structured data are difficult to realize. In addition, WFS doesn’t have an operation to acquire new ids. Such an operation

could be necessary in case new objects need to refer to each other in a topologically structured dataset.

- WFS misses an operation to lock an area, in addition to locking individual features.
- Application logic concerning transactions could partially be dealt with in an application service (e.g. billing) and partially at the server (e.g. transaction constraints, validation of features). However, referring to these services from the web feature server is not dealt with by the WFS specification.

Although there are some problems to overcome especially with constraints on datasets, the conclusion can be drawn that WFS is suitable to retrieve and modify geographic features over the Internet or other large networks.

7 Conclusions and recommendations

7.1 Conclusions

GIS Web Services focus on the technical part of problems concerning interoperable geoprocessing. The OpenGIS Consortium has developed several GIS Web Services, based on ICT standards of the World Wide Web Consortium (W3C), like XML and HTTP. In this context, Web Feature Services are a unique type of services, since they are the only open standards-based services that can be used for editing geographic data.

In order to modify data in a multi-user environment, transactions can be used to make sure that modifications of several users do not interfere. Geometric transactions are usually complicated to process because of the geometric properties. In this context, not only common difficulties with geo-databases like encoding, indexing and computations with spatial operators play a role. But in particular the relations of geographic features with features in the surroundings (either explicitly stored in some topological structure or in other spatial constraints, or implicitly from the geometric properties) make geometric transactions complicated to process.

Because Web Feature Services are based on common web technologies, WFS can benefit from other developments in these technologies and web services in general. Also, for these technologies lots of tools - commercial and free - are available that can be used to develop interoperable WFS clients and servers. As is shown, transactional Web Feature Services can be developed with XML, Java and Java Server Pages and SVG. These technologies can be used to develop light, yet powerful, WFS-clients.

Interoperability tests showed that not all currently available clients that claim to be WFS-compliant, work correctly. The clients either can't connect to a web feature service, or can't retrieve all data correctly. However, the tests also showed the power of interoperable services, since a generic client - not having any knowledge of the data source – could retrieve features from the data source, because a web feature server was configured on top of it.

The WFS-specification provides sufficient operations to make sure that geometric transactions can be sent from client to server (processing these transaction requests is up to the server). Using general web technologies as HTTP, XML and GML, with WFS geographic features can be retrieved, created, modified and deleted. The fact that the *Insert*, *Update* and *Delete* operations can be used in one transaction allows for very flexible geometric transactions, thereby making it possible to process complex, geometric transactions in a web environment.

The defined operations for locking existing features can also be used to define transactions over more than one web feature server or long transactions. However, WFS lacks an operation to lock an area, which is necessary for certain applications.

GML 2.1.2 is used in WFS to tackle problems with the encoding of geographic features. However, GML 2.1.2 does not support topology and temporal aspects and lacks default styling, which could be very practical. With GML 3.0 topology, temporal aspects and default styling can be encoded.

The filters from the Filter Encoding Specification are sufficient to select the features on which (spatial) operations should be performed. Another issue with the use of GML in WFS is that it is not possible to have compound keys.

The WFS specification offers a mechanism (using the *GetCapabilities* and *DescribeFeatureType operations*) that makes validation of (single) features possible, both at the client and server side. This way, transactions should only contain valid features, at least valid for the corresponding feature definition.

At this moment, WFS does not provide sufficient operations to deal with transactions where topology or spatial constraints are involved. It is not the responsibility of WFS to actually execute the validation and processing of this kind of transactions, but there is no mechanism in WFS for clients to retrieve the information they need to construct features that do not violate the topological structure or the spatial constraints of the dataset. In addition, new objects in (a transaction) cannot refer to each other using the correct identifiers, since WFS does not have an operation to acquire feature identifiers before the actual features have been created. Using generic clients for transactions on topologic data or data with spatial constraints is therefore difficult.

Error messages are not standardized, which makes it very difficult for (generic) clients to give useful feedback to a user in the case errors have occurred during a transaction.

From the case study can also be concluded that application logic is only partially accounted for in WFS. Per application, application logic should be divided between other (mediating) services like application services, supporting services like billing services, the server itself and the client.

The main conclusion is that Web Feature Services are suitable and powerful, interoperable GIS web services to edit geographic data over large networks as the Internet. But WFS (version 1.0.0) is not advanced enough to strongly support transactions on complicated (e.g. with topology or other spatial constraints) data sets in a generic way.

7.2 Recommendations

Based on the results and conclusions of this thesis, these recommendations are given:

- Incorporating GML 3.0 in WFS would allow WFS to deal with topology, temporal aspects and default styling much easier. However, if WFS will use GML 3.0, web feature services will become more complicated, since GML 3.0 is more complex than GML 2.1.2. How to use GML 3.0 and maybe even what parts of GML 3.0 should be used in WFS therefore needs to be investigated.
- Error reporting should be enhanced in WFS. Common errors should be standardized, to make it possible for a client to give useful feedback to the user.
- To support transactions better, additional operations and extensions of existing operations should be defined in the next WFS specification. It concerns an operation to lock areas and an operation for clients to acquire identifiers from the server to create new features. Also, the capabilities document should be extended

with an element to state how (e.g. atomic) transactions are dealt with at the server.

- Where to deal with application logic will remain a difficult design issue, especially in interoperable services. However, some issues like different rights per user or user group, constraints on the structure of transactions and validation against topologic and spatial constraints, tend to be tightly coupled with a dataset and could thus (nearly always) be dealt with at the server side. For instance, constraints on the structure of transactions could easily be handled by defining XML Schemas for transactions for each application. But also UML and OCL could be used to define constraints. Similar to the *DescribeFeatureType*- and *GetCapabilities*-operation one could imagine that an operation to get such a schema or constraints is then needed to inform a client of these restrictions and make validation of transactions possible (for example *GetConstraints*). The same could hold for spatial constraints, which might be modelled in XML as well, so that clients and servers can exchange these constraints.

  Research to determine which issues could and should be dealt with in the WFS-specification as well as how these issues could be part of the WFS-specification, has to be done to allow (generic) interoperable clients and servers to deal better with application logic (including how constraints should be passed).
- Research to the need and possibilities to extend WFS with a Data Definition Language should be done.
- As soon as more transactional clients and servers are available that claim to be WFS-compliant, extensive testing could be done to test whether WFS can deal with transactions sufficiently in an interoperable manner.

References

[1] Alameh, N., 2002, *Service Chaining of Interoperable Geographic Information Web Services*. Greenbelt, USA, accessed at http://web.mit.edu/nadinesa/www/paper2.pdf.

[2] Alameh, N., 2001, *Scalable and Extensible Infrastructures for Distributing Interoperable Geographic Information Services on the Internet*. PhD Dissertation Submitted to MIT Libraries. In [1].

[3] Aybet, J. 1990, Integrated mapping systems – data conversion and integration. *Mapping awareness*, 4(6), 18-23.

[4] Berg, C. van den, Tuijnman, F., Vijlbrief, T., Meijer, C., Oosterom, P. van, Uitermark, H., Multi-server internet GIS: Standardization and practical experiences. In Goodchild, M.F., M.J. Egenhofer, R. Fegeas, and C.A. Kottman, editors (1999), *Interoperating Geographic Information Systems*, Boston, USA, (International Conference and Workshop on Interoperating Geographic Information Systems, Santa Barbara, California, USA, December 3-4 and 5-6, 1997) pages 365-378.

[5] Bernstein, P.A., Hadzilacos, V., Goodman, N., *Concurrency control and recovery in database systems*, Addison-Wesley Publishing Company, 1987, USA.

[6] Bishr, Y., Radwan, M., Molenaar, M., 1998, *Aspects of interoperability: a GII perspective*, International Institute for Aerospace Survey and Earth Sciences (ITC), The Netherlands.

[7] Bos, B., 1999, *XML in 10 points*. http://www.w3.org/XML/1999/XML-in-10-points.

[8] Buehler, K. and McKee, L., 1998, *The OpenGIS Guide: Introduction to Interoperable Geoprocessing and the OpenGIS Specification.* OpenGIS Consortium Inc., Waltham (USA).

[9] Eckstein, R., Casabianca, M., 2001, *XML Pocket Reference*, O'Reilly & Associates, 2nd edition.

[10] Egenhofer, M.J., 2002, Toward the Semantic Geospatial Web, *ACM-GIS'02*, November 8-9, 2002, USA.

[11] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., 1999, *Hypertext Transfer Protocol -- HTTP/1.1*, W3C.

[12] Flowerdew, R., 1991, *Spatial Data Integration*. In [17]

[13] Garnett, J., Owens, B., 2003, *Validating Web feature server Design Document*, Refractions Research Inc., Victoria, Canada, accessed at http://vwfs.refractions.net/docs/Validating_Web_Feature_Server.pdf.

[14] ISO, 2001, ISO/TC 211 Geographic Information / Geomatics, *http://www.statkart.no/isotc211/,* in [1].

[15] Laurini, R., 1994, Sharing Geographic information in distributed databases. *Proceedings URISA*, pp. 441-454.

[16] Laurini, R., 1998, Spatial Multidatabase Topological Continuity and Indexing: a Step towards Seamless GIS Data Interoperability. *International Journal of Geographical Information Sciences*, 12(4):373–402, june 1998.

[17] Maguire, D.J., Goodchild, M.G., Rhind, D.W. (eds), 1991, *Geographical Information Systems: Principles and Applications*, London, United Kingdom, Longman.

[18] Molenaar, M., 1998, *An Introduction to the Theory of Spatial Object Modelling for GIS*, London, UK.

[19] OGC, 1999, *The OpenGIS Abstract Specification Topic 5: Features*, version 4, Reference number: 99-105r2, OpenGIS Consortium Inc, USA.

[20] OGC, 2001, Vretanos, P. (editor), *Filter Encoding Implementation Specification, version 1.0.0*, Reference number: OGC 02-059, OpenGIS Consortium Inc., USA.

[21] OGC, 2002 a, Cox, S., Cuthbert, A., P., Lake, R., Martell, R. (editors), *OpenGIS Geography Markup Language (GML) Implementation Specification, version 2.1.2*. Reference number: OGC 02-069, OpenGIS Consortium Inc., USA.

[22] OGC, 2002 b, *OGC Web Services Initiative 1 (OWS1) Baseline Documents Page*, http://ip.opengis.org/ows1/docIndex.html, in [1].

[23] OGC, 2002 c, Vretanos, P. (editor), *Web Feature Service Implementation Specification.* Reference number: OGC 02-058, OpenGIS Consortium Inc., USA.

[24] OGC, 2002 d, Beaujardière, de La, J. (editor), *Web Map Service Implementation Specification, version 1.1.1*, Reference number: OGC 01-068r3, OpenGIS Consortium Inc., USA.

[25] OGC, 2003 a, Buehler, K. (editor), *OpenGIS Reference model*, Reference number: OGC 03-040, OpenGIS Consortium Inc., USA.

[26] OGC, 2003 b, Evans, J. (editor), *Web Coverage Service (WCS) version 1.0*, Reference number: OGC 03-065r6, OpenGIS Consortium Inc., USA.

[27] OGC, 2003 c, Cox, S., Daisey, P., Lake, R., Portele, C., Whiteside, A. (editors), *OpenGIS Geography Markup Language (GML) Implementation Specification, version 3*. Reference number: OGC 02-023r4, OpenGIS Consortium Inc., USA.

[28] Oosterom, P. van, 1997, Maintaining consistent topology including historical data in a large spatial database. In Chrisman, N. (Ed.), *Proceedings of Auto-Carto 13*, Bethesda: ACSM & ASPRS, pp. 327-336.

[29] Rodriques Lopes, D.L., Zevenbergen, J.A., Boekhold, R., *Kadasters (dictaat)*, in Dutch, TU Delft, September 1998.

[30] Sheth, A.P., Larson, J.A., 1990, Federated Database Systems for Managing Distributed, heterogeneous and autonomous Databases. *ACM Computing Surveys*, Vol. 22, No. 3, September 1990.

[31] Uitermark, H., 2001, *Ontology-based geographic data set integration*, PhD-thesis, Deventer, the Netherlands.

[32] Vckovski, A., 1998, *Interoperable and distributed processing in GIS*, Taylor & Francis Ltd., London.

[33] Vries, M.E. de, *GML and SVG: from content to presentation*. SVG Open Developers Conference, Zürich, Switzerland, July 15-17, 2002

[34] Vries, M. de, Tijssen, T.P.M., Stoter. J.E., W. Quak, and Oosterom, P.J.M. van. *The GML prototype of the new TOP10vector object model*. Report for the Topographic Service, Delft University of Technology, GISt No. 9, Delft 2001

[35] Wiederhold, G., 1992, Mediators in the Architecture of Future Information Systems, *Computer*, March 1992, pp. 38-49.

[36] Worboys, M.F., 1995, *GIS: a computing perspective*, Taylor & Francis, London.

Websites

[37] *DTED,* http://assist.daps.dla.mil/docimages/0002/45/22/89020.PD8, last visited 23 April 2004

[38] *GeoTIFF*, http://www.remotesensing.org/geotiff/geotiff.html, last visited 23 April 2004

[39] *HDF-EOS tools and information center*, http://hdfeos.gsfc.nasa.gov/hdfeos/index.cfm, last visited 26 April 2004.

[40] *Introduction to JavaServer Pages*, http://developer.netscape.com/viewsource/kuslich_jsp/kuslich_jsp.html, visited 13 April 2004

[41] *Java Server Pages Technology,* http://java.sun.com/products/jsp/index.jsp, last visited 26 April 2004.

[42] *NITFS Technical Board Public Page,* http://www.ismc.nima.mil/ntb/baseline/1999.html , last visited 23 April 2004

[43] *Oracle Technology Network*, http://otn.oracle.com, visited 21 April 2004

[44] *Scalable Vector Graphics at the W3C website,* http://www.w3.org/TR/SVG/intro.html, visited 13 April 2004

[45] *Website of the Apache Jakarta Project*, http://jakarta.apache.org/tomcat/, last visited 26 April 2004.

[46] *Website of Deegree project*, http://deegree.sourceforge.net/, last visited 26 April 2004.

[47] *Website of GeoServer Project*, http://geoserver.sourceforge.net, last visited 30 March 2004.

[48] *Website of ISO TC 211 Geographic Information/Geomatics*, http://www.isotc211.org, last visited 26 April 2004.

[49] *Website of the Object Management Group*, http://www.omg.org/, last visited 24 April 2004.

[50] *Website of OpenMap*, http://openmap.bbn.com, last visited 23 April 2004

[51] *Website of PostGIS*, http://postgis.refractions.net, visited 5 April 2004.

[52] *Website of Resin*, http://www.caucho.com/resin/, last visited 26 April 2004.

[53] *XML Schema*, http://www.w3.org/XML/Schema, last visited 21 April 2004

[54] *XML Schema, a brief introduction*, http://lucas.ucs.ed.ac.uk/xml-schema/, last visited 19 March 2004.

[55] *XSQL - Combining XML and SQL*, http://xsql.sourceforge.net/, visited 13 April 2004.

Appendix A: Glossary

In this Glossary definitions are taken from the thesis. Only new references are explicitly mentioned.

Term/Abbreviation *Entire term*	***Explanation***
Client/server	The client-server architecture is a simple model of computing, in which system functionality is divided among the components that make requests (the clients) and the components that respond to them (the servers).
Federated database	Federated databases are separate databases that are structured, perhaps with middleware or special database access software, in such a way that they can be queried as a single database.
Geometric transaction	A geometric transaction is a (database) transaction that is performed on geographic datasets.
GML Geography Markup Language	The Geography Markup Language (GML) is an XML encoding for the modelling, transport and storage of geographic information including both the spatial and nonspatial properties of geographic features.
HTTP Hypertext Transfer Protocol	The HTTP protocol is a request/response protocol. The HTTP protocol uses URI's (Uniform Resource Identifiers, for instance http://www.someserver.com/index.jsp) to identify resources. HTTP GET and POST are used respectively to retrieve something (document, result of a process, etc.) from the server or send something (a message, XML-request, etc.) to the server.
Interoperability	Interoperability is the ability for a system or components of a system to provide information portability and interapplication, cooperative process control. Interoperability, in the context of the OpenGIS Specification, is software components operating reciprocally (working with each other) to overcome tedious batch conversion tasks, import/export obstacles, and distributed resource access barriers imposed by heterogeneous processing environments and heterogeneous data.

Java	An object-oriented programming language developed by Sun Microsystems.
Javascript	With Javascript, Java-like code can be embedded in HTML-pages. Javascript is not as strong as Java, but can easily be used to add some functionality to HTML-pages.
JSP Java Server Pages	JSP is a presentation layer technology that sits on top of a Java servlets model and makes working with HTML easier. It allows for mixing static HTML content with server-side scripting to produce dynamic output. By default, JSP uses Java as its scripting language [40].
Mediator	A mediator is software that exploits encoded knowledge (application logic) about some sets or subsets of data to create information for a higher layer of application.
OGC OpenGIS Consortium	The Open GIS Consortium Inc., a not-for-profit trade association dedicated to promoting new technical and commercial approaches to interoperable geoprocessing, was founded in 1994 in response to widespread recognition of the problem of interoperability and its many negative ramifications for industry, government, and academia.
Replicated database	A database is called a replicated database if multiple copies of some or all of the data are stored at multiple sites.
Service, Web Service	Web services are self-contained, self-describing, modular applications that can be published, located, and dynamically invoked across the web. Web services provide access to sets of operations accessible through one or more standardized interfaces.
SQL Structured Query Language	SQL is a standard language to ask (complex) questions to and edit data in databases.
SVG Scalable Vector Graphics	SVG is a language for describing two-dimensional graphics in XML [44].

Transaction, Database transaction	A transaction is a particular execution of a program that edits the database by means of read and write operations. In terms of SQL, these write operations are INSERT, UPDATE and DELETE operations.
WCS Web Coverage Service	The Web Coverage Service (WCS) supports electronic interchange of geospatial data as "coverages" – that is, digital geospatial information representing space-varying phenomena.
WFS Web Feature Service	Web Feature Services allow clients to retrieve and edit geographic (vector-)data encoded in Geography Markup Language (GML).
WMS Web Map Service	A Web Map Service (WMS) produces maps of georeferenced data in a digital format, e.g. JPEG, GIF or PNG. The WMS specification standardizes the way in which maps are requested by clients and the way that servers describe their data holdings.
XML Extensible Markup Language	XML is a meta language that allows for creation and formatting of specific document markups. With XML, sets of tags (like in HTML) can be defined for a specific application domain.
XML Schema	An XML-schema is a document that describes the valid format of an XML dataset. This definition includes what elements are (and are not) allowed at any point; what the attributes for any element may be; the number of occurrences of elements; etc.
XSL / XSLT Extensible Stylesheet Language / XSL Transformation	XSL allows developers to describe transformations using XSL Transformations (XSLT). XSL Transformations can convert XML documents into HTML, another XML-document or other textual output.
XSQL	XSQL is the combination of XML and SQL to provide a language and database independent means for storing SQL queries, clauses and query results [55].

Appendix B: Table definitions case study

Because in the original cadastral database, features in `test_boundary` are uniquely identified by its `OBJECTID` and `TMAX`, an extra column (`FID`) had to be added to the tables as a unique identifier. The same holds for the table `cl_parcel2`. Below, the table structures for all four tables in the case study are given with some extra remarks. For the `draft_parcel` and `draft_boundary` tables the column FID has been added to be primary key.

Test_boundary	
Fid	Int
shape	SDO_geometry
object_id	Int
classif	Int
status_cd	Int
object_dt	Int
tmin	Int
tmax	Int
source	String
quality	String

Cl_parcel2	
Fid	Int
ogroup	Int
object_id	Int
slc	Int
classif	Int
location	SDO_geometry
z	Int
rotangle	Int
accu_cd	Int
oarea	Float
object_dt	Int
tmin	Int
tmax	Int
sel_cd	String
source	String
qulity	String
vis_cd	String
akr_area	Float
bbox	SDO_geometry
municip	String
osection	String
sheet	String

parcel	String

Draft_boundary	
Fid	Int
object_id	Int
classif	Int - Classification, e.g. 'new boundary' or 'striked out'
shape	SDO_geometry
status_cd	Int - e.g. 'processed by notary'
object_dt	Int - Date of creation of object
tmin	Int – System time of 'birth' object
tmax	Int - System time of 'death' object
source	String – Always 'Notary'
quality	String – Always 'Draft'

Draft_parcel	
Fid	Int
akte_datum	Int – user date
tmin	Int – System time of 'birth' object
tmax	Int - System time of 'death' object
source	String – Always 'Notary'
quality	String – Always 'Draft'
municip	String – cadastral municipality
osection	String
parcel	String
pp_i_ltr	String – 'D' (for 'deelperceel', or 'part of parcel')
pp_i_nr	String – Provided by AKR, therefore this could be ignored
x_akr_objectnummer	String: Combination of municip, osection, parcel, pp_i_ltr, pp_i_nr; Probably the reference used by the notary.
line_id1	Int – Referring to the first part of the parcel's boundary
location	SDO_geometry
d_location	String
label	String - Label used by notary to annotate in the deed which part of the draft is the parcel ("het perceel aangeduid met A ..").
status	String – For instance 'needs further processing' or 'processed'.
restperceel	String – Indicates whether the rest of the parcel still has the same owner.
aktenummer	String: The deed number of the transaction.

Appendix C: Client in detail

This appendix documents the Java Server Pages that have been developed for the case study client. These Java Server Pages (JSPs) deal with the construction and sending of XML-requests for WFS transactions. This appendix also provides a general description of the client that extends section 5.4.2.

General description

The client consists of an SVG-layer that is capable of retrieving data from a web feature server in GML and it can visualize that GML as SVG. The Adobe SVG-plug-in is needed. The Viewer/editor uses standard SVG-functionality for zooming and panning. In this layer new feature instances (points and polylines) can be drawn in SVG as well. Points are used for adding features of type `draft_parcel`, polylines for adding features of type `draft_boundary`. This layer uses HTML and JavaScript for retrieving the GML stream and an XSLT transformation is done to make SVG out of the GML-stream.

The SVG-layer uses keyword-value-pairs for retrieving data. Filtering is not implemented in this layer. Therefore, when requesting for raw GML or GML to transform to SVG, the entire data set is returned. For real world applications, this is too much, so some selection of the data has to be made.

There is one exception to this, because functionality has been implemented to retrieve data that is in the bounding box of a parcel (a feature instance of type CL_PARCEL2). The X_AKR_OBJECTNUMMER needs to be specified in a form in an HTML-form in the viewer. A GetFeature request (using a filter on the X_AKR_OBJECTNUMMER) is constructed in XML-encoding that retrieves the specified parcel. This request is sent to the web feature server. The bounding box of the returned feature is used for making another GetFeature-request to select data from other feature types as well. This way a GetFeature-request is constructed that retrieves all features that interact with the bounding box of one parcel. Thus, data are filtered using a bounding box. With a filter to select a feature on its feature identifier (fid), these filters are the only implemented filters in XML-encoding. The requests are constructed, sent and processed with Java Server Pages. The returned GML is transformed to SVG again (similar to the way GML is transformed to SVG when the entire dataset is retrieved).

In order to make valid WFS-requests for adding new features, another layer ("WFS-layer") is used. When a new feature is drawn in SVG, the coordinates of this feature are passed to a form where the user has to fill in some more details on the new feature. This is encoded in the correct XML for the corresponding feature type (a point is always of type `draft_parcel`). These XML-data are added to a Transaction-request. The user can add other transaction operations (e.g. creating another new feature) to this request. When finished, the user has to send the transaction to the web feature server, where it is processed. This layer consists of several Java Server Pages (JSPs).

For deleting features, the feature has to be selected in the SVG-layer. The user can then choose to delete the selected feature. Then the WFS-layer is called to create the XML for deleting the feature. This XML is added to the Transaction request.

Constructing the request to add a new draft_boundary

Figure C.1 shows the components (the User, the SVG-layer and the JSPs) that are involved in constructing the XML-request for inserting a new feature instance of type `draft_boundary`. The result of this is the new feature encoded in GML and the WFS-elements for the transaction-request (for an example see Figure C.2). Note that the feature lacks the primary key (FID), because this has to be provided on insert by the server. This string is added to the (potentially) existing WFS-request in the `requeststore.jsp`.

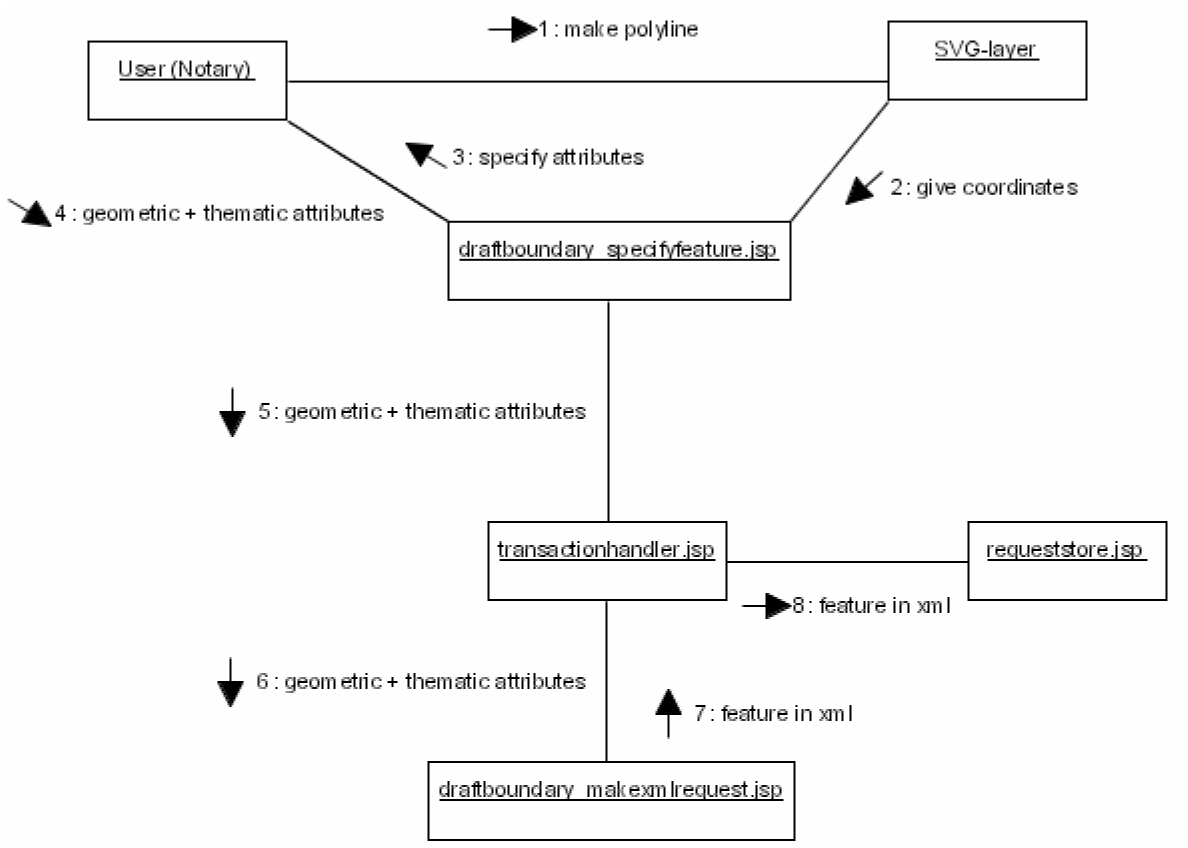

1 : make polyline
User (Notary)
SVG-layer
3 : specify attributes
2 : give coordinates
4 : geometric + thematic attributes
draftboundary_specifyfeature.jsp
5 : geometric + thematic attributes
transactionhandler.jsp
requeststore.jsp
8 : feature in xml
6 : geometric + thematic attributes
7 : feature in xml
draftboundary_makexmlrequest.jsp

Figure C.1 Collaboration diagram for construction of the XML to add a new `draft_boundary`*.*

```
<wfs:Insert>
 <cad:DRAFT_BOUNDARY>
      <cad:SHAPE>
        <gml:MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
          <gml:lineStringMember>
            <gml:LineString>
              <gml:coordinates    decimal="."    cs=","    ts="    ">106616787,448520583
106639566,448504105 106660406,448513798 106678822,448512344</gml:coordinates>
            </gml:LineString>
          </gml:lineStringMember>
        </gml:MultiLineString>
      </cad:SHAPE>
      <cad:OBJECT_ID>341411971</cad:OBJECT_ID>
      <cad:CLASSIF>31</cad:CLASSIF>
      <cad:STATUS_CD>0</cad:STATUS_CD>
      <cad:OBJECT_DT>19920213</cad:OBJECT_DT>
  <TMIN>260218287</TMIN>
  <TMAX>0</TMAX>
  <SOURCE>-</SOURCE>
  <QUALITY>T1</QUALITY>
 </cad:DRAFT_BOUNDARY>
</wfs:Insert>
```

Figure C.2 Example of constructed string to add to the Transaction-request for a new draft_boundary.

Constructing the request to add a new draft_parcel

Figure C.3 shows the components (the User, the SVG-layer and the JSPs) that are involved in constructing the XML-request for inserting a new feature instance of type `draft_parcel`. The result of this is the new feature encoded in GML and the WFS-elements for the transaction-request (for an example see Figure C.4). Note that the feature lacks the primary key (FID), because this has to be provided on insert by the server. This string is added to the (potentially) existing WFS-request in the `requeststore.jsp`.

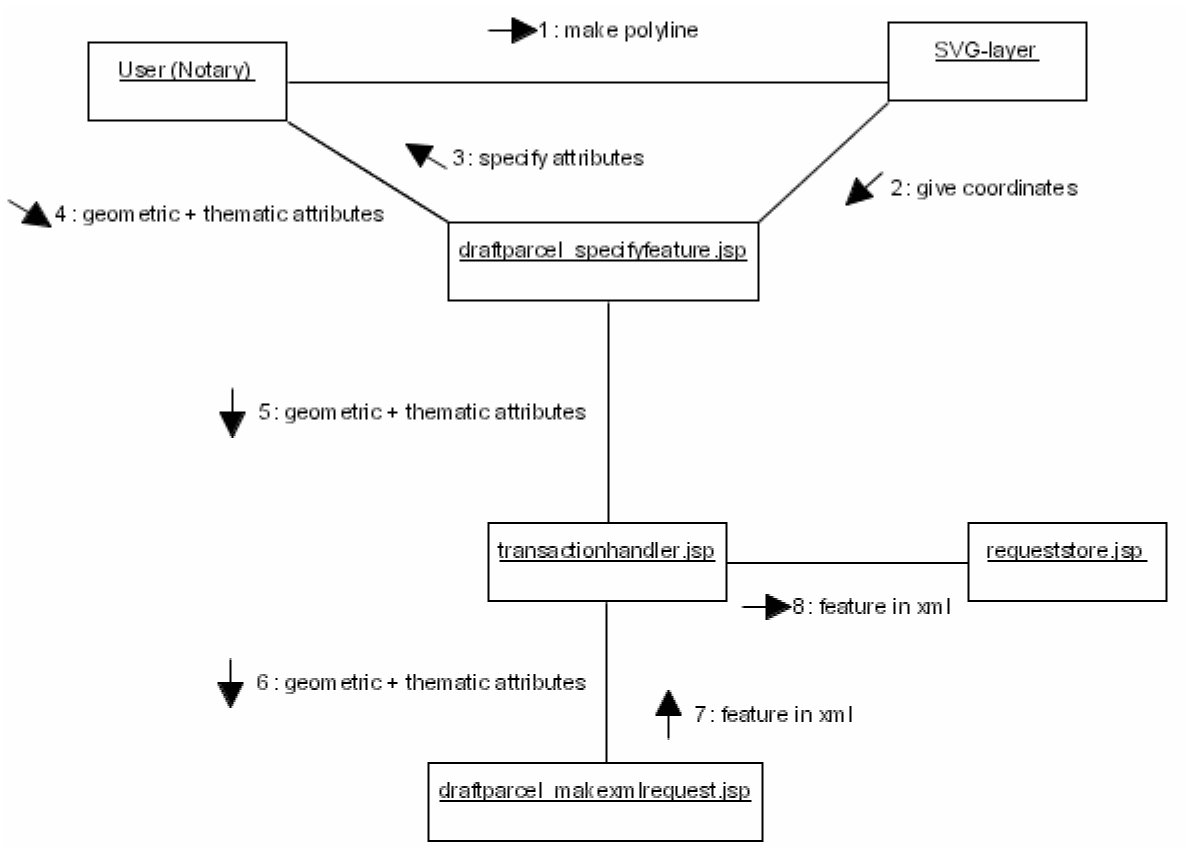

1 : make polyline
User (Notary)
SVG-layer
3 : specify attributes
2 : give coordinates
4 : geometric + thematic attributes
draftparcel_specifyfeature.jsp
5 : geometric + thematic attributes
transactionhandler.jsp
requeststore.jsp
8 : feature in xml
6 : geometric + thematic attributes
7 : feature in xml
draftparcel_makexmlrequest.jsp

Figure C.3 Collaboration diagram for the construction of the XML to add a new `draft_parcel`.

```
<wfs:Insert>
 <DRAFT_PARCEL>
    <TMIN>260218287</TMIN>
    <TMAX>0</TMAX>
    <SOURCE>-</SOURCE>
    <QUALITY>T1</QUALITY>
    <MUNICIP>GDA01</MUNICIP>
    <OSECTION> w</OSECTION>
    <PARCEL>X4489</PARCEL>
    <PP_I_LTR>D</PP_I_LTR>
    <PP_I_NR>0001</PP_I_NR>
    <X_AKR_OBJECTNUMMER>GDA01M 04489G0000</X_AKR_OBJECTNUMMER>
    <LINE_ID1>340505636</LINE_ID1>
    <LOCATION>
       <gml:Point srsName='http://www.opengis.net/gml/srs/epsg.xml%23;28992'>
          <gml:coordinates decimal='.' cs=',' ts=' '>
106634235,448502651
          </gml:coordinates>
       </gml:Point>
    </LOCATION>
    <AKTE_DATUM>0</AKTE_DATUM>
    <LABEL>perceel A</LABEL>
    <STATUS>0</STATUS>
    <RESTPERCEEL>oude eigenaar</RESTPERCEEL>
    <AKTENUMMER>123456</AKTENUMMER>
 </DRAFT_PARCEL>
</wfs:Insert>
```

Figure C.4 Example of constructed string to add to the Transaction-request for a new `draft_parcel`.

Constructing the request to delete a feature

Figure C.5 shows the components (the User, the SVG-layer and the JSPs) that are involved in constructing the XML-request for deleting an already existing feature. The result of this is an XML-string that contains the WFS-elements for a Delete-operation with the feature's identifier (for an example see Figure C.6). This string is added to the (potentially) existing WFS-request in the `requeststore.jsp`.

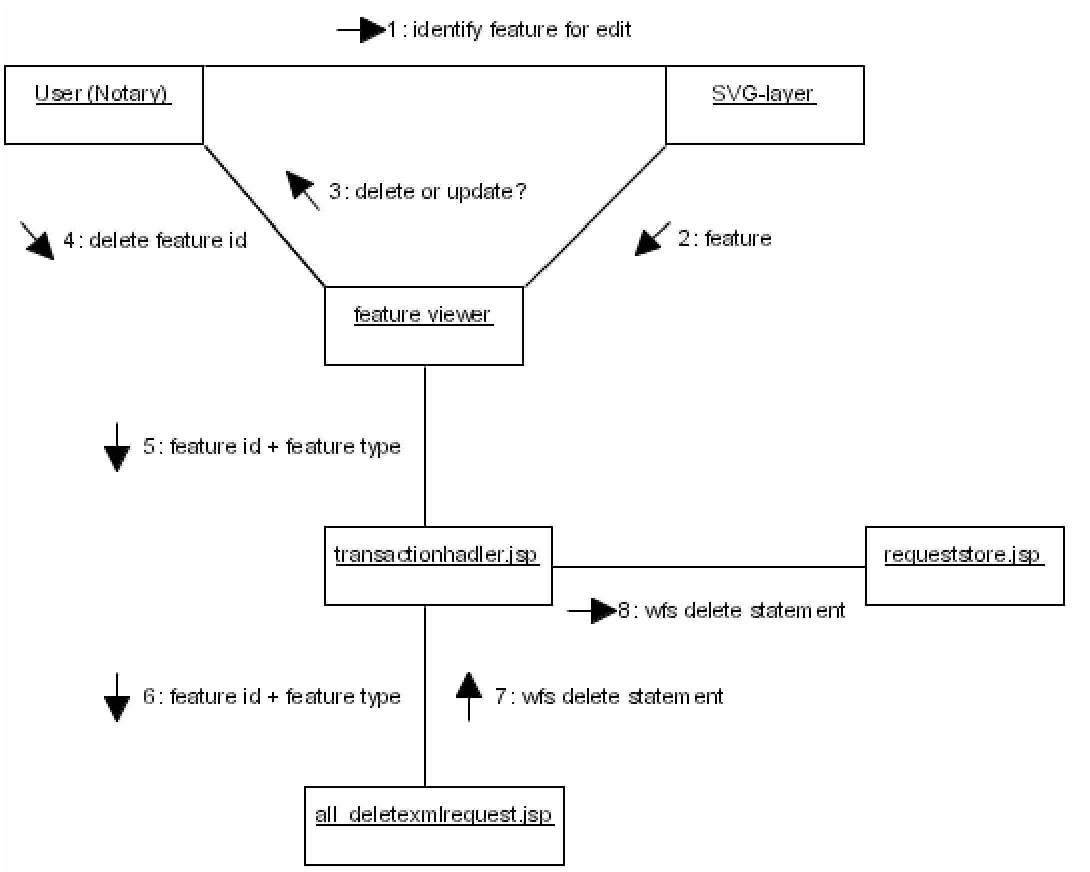


Figure C.5 Collaboration diagram for construction of the XML-request to delete a feature.

```
<wfs:Delete typeName="cad:DRAFT_PARCEL">
  <ogc:Filter>
    <ogc:FeatureId fid="DRAFT_PARCEL.7779161"/>
  </ogc:Filter>
```

```
</wfs:Delete>
```

Figure C.6 Example of constructed string to add to the Transaction-request for deleting the feature `draft_parcel` *with feature identifier 7779161.*

Sending a WFS transaction

When the user wishes to send the transaction to the server, the user can submit the request from the `requeststore.jsp` (Figure C.7). The request is send to the Web Feature Server and the response is send back to `requeststore.jsp` where the response is shown to the user.

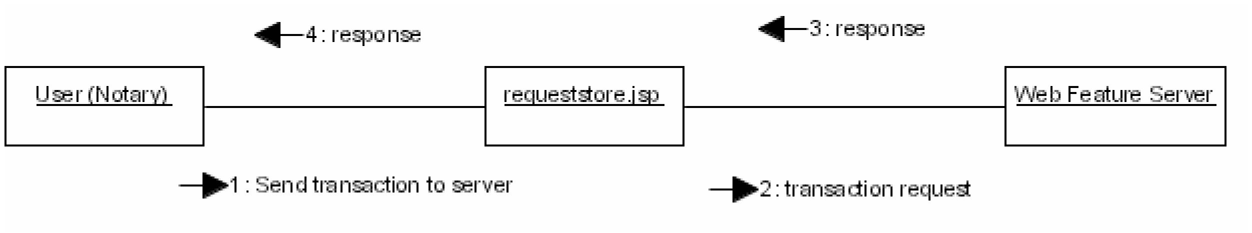


Figure C.7 Sending the request from the client to the web feature server.

Appendix D: Example Transactional WFS

This example shows how the Web Feature Service developed in the case study can be used to insert new features. Let's assume that a client wishes to access the Web Feature Service located at:
`http://130.161.150.109:8080/geoserver/wfs/`

The client sends the KVP-request:
`http://130.161.150.109:8080/geoserver/wfs/wfs?request=GetCapabilities`

The server responds with the capabilities document:

```
<?xml version="1.0" encoding="UTF-8"?>
<WFS_Capabilities
 version="1.0.0"
 xmlns="http://www.opengis.net/wfs"
 xmlns:cad="http://130.161.150.109:8080/geoserver"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wfs
http://130.161.150.109:8080/geoserver/data/capabilities/wfs/1.0.0/WFS-capabilities.xsd">
  <Service>
    <Name>Cadastre</Name>
    <Title>Splitting parcels</Title>
    <Abstract>This is a test server for Cadastral data. It contains some LKI-data from
the city of Gouda.</Abstract>
    <Keywords>WFS, TEST, Cadastre, Notary, Gouda, splitting parcel</Keywords>
  <OnlineResource>http://130.161.150.109:8080/</OnlineResource>
  <Fees>NONE</Fees>
  <AccessConstraints>NONE</AccessConstraints>
  </Service>
  <Capability>
    <Request>
      <GetCapabilities>
        <DCPType><HTTP><Get
onlineResource="http://130.161.150.109:8080/geoserver/wfs/GetCapabilities?"/></HTTP></DCP
Type>
        <DCPType><HTTP><Post
onlineResource="http://130.161.150.109:8080/geoserver/wfs/GetCapabilities"/></HTTP></DCPT
ype>
      </GetCapabilities>
      <DescribeFeatureType>
<SchemaDescriptionLanguage><XMLSCHEMA/></SchemaDescriptionLanguage>
        <DCPType><HTTP><Get
onlineResource="http://130.161.150.109:8080/geoserver/wfs/DescribeFeatureType?"/></HTTP><
/DCPType>
        <DCPType><HTTP><Post
onlineResource="http://130.161.150.109:8080/geoserver/wfs/DescribeFeatureType"/></HTTP></
DCPType>
      </DescribeFeatureType>
      <GetFeature>
        <ResultFormat><GML2/></ResultFormat>
        <DCPType><HTTP><Get
onlineResource="http://130.161.150.109:8080/geoserver/wfs/GetFeature?"/></HTTP></DCPType>
        <DCPType><HTTP><Post
onlineResource="http://130.161.150.109:8080/geoserver/wfs/GetFeature"/></HTTP></DCPType>
      </GetFeature>

      <Transaction>
        <DCPType><HTTP><Get
onlineResource="http://130.161.150.109:8080/geoserver/wfs/Transaction?"/></HTTP></DCPType
>
        <DCPType><HTTP><Post
onlineResource="http://130.161.150.109:8080/geoserver/wfs/Transaction"/></HTTP></DCPType>
```

```
      </Transaction>

      <LockFeature>
        <DCPType><HTTP><Get
onlineResource="http://130.161.150.109:8080/geoserver/wfs/LockFeature?"/></HTTP></DCPType
>
        <DCPType><HTTP><Post
onlineResource="http://130.161.150.109:8080/geoserver/wfs/LockFeature"/></HTTP></DCPType>
      </LockFeature>

      <GetFeatureWithLock>
        <ResultFormat><GML2/></ResultFormat>
        <DCPType><HTTP><Get
onlineResource="http://130.161.150.109:8080/geoserver/wfs/GetFeatureWithLock?"/></HTTP></
DCPType>
        <DCPType><HTTP><Post
onlineResource="http://130.161.150.109:8080/geoserver/wfs/GetFeatureWithLock"/></HTTP></D
CPType>
      </GetFeatureWithLock>
    </Request>
  </Capability>
  <FeatureTypeList>
    <Operations>
      <Query/><Insert/><Update/><Delete/><Lock/>
    </Operations>
    <FeatureType>
      <Name>cad:CL_PARCEL2</Name>
      <Title>CL_PARCEL2</Title>
      <Abstract>Parcel location data</Abstract>
      <Keywords/>
      <SRS>EPSG:28992</SRS>
      <LatLongBoundingBox minx="4.5" miny="51.8" maxx="5.0" maxy="52.3"/>
    </FeatureType>
    <FeatureType>
      <Name>cad:DRAFT_BOUNDARY</Name>
      <Title>DRAFT_BOUNDARY</Title>
      <Abstract>draft_boundary where drafts of boundaries should be placed in</Abstract>
      <Keywords/>
      <SRS>EPSG:28992</SRS>
      <LatLongBoundingBox minx="4.5" miny="51.8" maxx="5.0" maxy="52.3"/>
    </FeatureType>
    <FeatureType>
      <Name>cad:DRAFT_PARCEL</Name>
      <Title>DRAFT_PARCEL</Title>
      <Abstract>DRAFT_PARCEL, having 1 geometrie, where drafts of parcels should be
placed in</Abstract>
      <Keywords/>
      <SRS>EPSG:28992</SRS>
      <LatLongBoundingBox minx="4.5" miny="51.8" maxx="5.0" maxy="52.3"/>
    </FeatureType>
    <FeatureType>
      <Name>cad:TEST_BOUNDARY</Name>
      <Title>TEST_BOUNDARY</Title>
      <Abstract>Parcel boundaries</Abstract>
      <Keywords/>
      <SRS>EPSG:28992</SRS>
      <LatLongBoundingBox minx="4.5" miny="51.8" maxx="5.0" maxy="52.3"/>
    </FeatureType>
  </FeatureTypeList>
  <ogc:Filter_Capabilities>
    <ogc:Spatial_Capabilities>
      <ogc:Spatial_Operators>
        <ogc:Disjoint/>
        <ogc:Equals/>
        <ogc:DWithin/>
        <ogc:Beyond/>
        <ogc:Intersect/>
        <ogc:Touches/>
        <ogc:Crosses/>
        <ogc:Within/>
        <ogc:Contains/>
        <ogc:Overlaps/>
        <ogc:BBOX/>
      </ogc:Spatial_Operators>
    </ogc:Spatial_Capabilities>
    <ogc:Scalar_Capabilities>
      <ogc:Logical_Operators/>
      <ogc:Comparison_Operators>
        <ogc:Simple_Comparisons/>
        <ogc:Between/>
        <ogc:Like/>
        <ogc:NullCheck/>
      </ogc:Comparison_Operators>
      <ogc:Arithmetic_Operators>
        <ogc:Simple_Arithmetic/>
      </ogc:Arithmetic_Operators>
    </ogc:Scalar_Capabilities>
  </ogc:Filter_Capabilities>
</WFS_Capabilities>
```

Now the client wishes to insert a new feature instance of `draft_boundary` and 2 new instances of type `draft_parcel`, but first needs to know how such a feature is defined. The client requests the server to send the XML-schema of `draft_boundary` and `draft_parcel`, by sending this DescribeFeatureType request:

```
http://130.161.150.109:8080/geoserver/wfs/wfs?request=DescribeFeatureTy
pe&typeName=DRAFT_BOUNDARY,DRAFT_PARCEL
```

The server responds with:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema
  targetNamespace="http://130.161.150.109:8080/geoserver"
  xmlns:cad="http://130.161.150.109:8080/geoserver"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="1.0">
<xs:import
  namespace="http://www.opengis.net/gml"
schemaLocation="http://130.161.150.109:8080/geoserver/data/capabilities/gml/2.1.2/feature
.xsd"/>
<xs:complexType name="DRAFT_BOUNDARY_Type">
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
        <xs:element name="FID" nillable="‶false″ minOccurs="1" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
       <xs:element    name="SHAPE"    type="gml:LineStringPropertyType"    minOccurs="0"
maxOccurs="1"/>
          <xs:element name="OBJECT_ID" nillable="‶true″ minOccurs="1" maxOccurs="1">
            <xs:simpleType>
              <xs:restriction base="xs:integer">
                <xs:maxLength value="11"/>
               </xs:restriction>
            </xs:simpleType>
          </xs:element>
           <xs:element name="CLASSIF" nillable="‶false″ minOccurs="1" maxOccurs="1">
            <xs:simpleType>
               <xs:restriction base="xs:integer">
                <xs:maxLength value="11"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="STATUS_CD" nillable="false" minOccurs="1" maxOccurs="1">
            <xs:simpleType>
              <xs:restriction base="xs:integer">
                <xs:maxLength value="11"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="OBJECT_DT" nillable="false" minOccurs="1" maxOccurs="1">
            <xs:simpleType>
              <xs:restriction base="xs:integer">
               <xs:maxLength value="11"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="TMIN" nillable="false" minOccurs="1" maxOccurs="1">
            <xs:simpleType>
              <xs:restriction base="xs:integer">
                <xs:maxLength value="11"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="TMAX" nillable="false" minOccurs="1" maxOccurs="1">
            <xs:simpleType>
              <xs:restriction base="xs:integer">
               <xs:maxLength value="11"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="SOURCE" nillable="false" minOccurs="1" maxOccurs="1">
            <xs:simpleType>
              <xs:restriction base="xs:string">
               <xs:maxLength value="5"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="QUALITY" nillable="false" minOccurs="1" maxOccurs="1">
            <xs:simpleType>
              <xs:restriction base="xs:string">
               <xs:maxLength value="2"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<xs:complexType name="DRAFT_PARCEL_Type">
  <xs:complexContent>
    <xs:extension base="gml:AbstractFeatureType">
      <xs:sequence>
        <xs:element name="OGROUP" nillable="false" minOccurs="0" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="OBJECT_ID" nillable="false" minOccurs="0" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="SLC" nillable="false" minOccurs="0" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="CLASSIF" nillable="false" minOccurs="0" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element      name="SHAPE"      type="gml:PointPropertyType"      minOccurs="1"
maxOccurs="1"/>
        <xs:element name="Z" nillable="true" minOccurs="0" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:maxLength value="11"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element              name='DRAFT_BOUNDARY'              type='cad:DRAFT_BOUNDARY_Type'
substitutionGroup='gml:_Feature'/>
<xs:element                name='DRAFT_PARCEL'                type='cad:DRAFT_PARCEL_Type'
substitutionGroup='gml:_Feature'/>
</xs:schema>
```

Now the client constructs two new feature instances for `draft_parcel` and one for `draft_boundary` and sends following transaction request to the server using HTTP POST:

```
<?xml version='1.0'?>
<wfs:Transaction version='1.0.0' service='WFS'
    xmlns='http://130.161.150.109:8080/geoserver'
    xmlns:wfs='http://www.opengis.net/wfs'
    xmlns:cad='http://130.161.150.109:8080/geoserver'
    xmlns:gml='http://www.opengis.net/gml'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:ogc='http://www.opengis.net/ogc'
    xsi:schemaLocation='http://130.161.150.109:8080/geoserver
    http://130.161.150.109:8080/geoserver/wfs/DescribeFeatureType?typeName=DRAFT_BOUNDARY
    http://130.161.150.109:8080/geoserver/wfs/DescribeFeatureType?typeName=draft_parcel
    http://www.opengis.net/wfs
    http://130.161.150.109:8080/geoserver/data/capabilities/wfs/1.0.0/WFS-
transaction.xsd'>
<wfs:Insert>
 <cad:DRAFT_BOUNDARY>
  <cad:SHAPE>
    <gml:MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
      <gml:lineStringMember>
        <gml:LineString>
          <gml:coordinates decimal="." cs="," ts=" ">
            106417204,448719275 106367184,448675614
          </gml:coordinates>
        </gml:LineString>
      </gml:lineStringMember>
    </gml:MultiLineString>
  </cad:SHAPE>
  <cad:OBJECT_ID>341411971</cad:OBJECT_ID>
  <cad:CLASSIF>31</cad:CLASSIF>
  <cad:STATUS_CD>0</cad:STATUS_CD>
  <cad:OBJECT_DT>19920213</cad:OBJECT_DT>
  <cad:TMIN>260218287</cad:TMIN>
  <cad:TMAX>0</cad:TMAX>
  <cad:SOURCE>-</cad:SOURCE>
  <cad:QUALITY>T1</cad:QUALITY>
 </cad:DRAFT_BOUNDARY>
</wfs:Insert>
```

```
<wfs:Insert>
 <cad:DRAFT_PARCEL>
  <cad:TMIN>260218287</cad:TMIN>
  <cad:TMAX>0</cad:TMAX>
  <cad:SOURCE>-</cad:SOURCE>
  <cad:QUALITY>T1</cad:QUALITY>
  <cad:MUNICIP>GDA01</cad:MUNICIP>
  <cad:OSECTION> w</cad:OSECTION>
  <cad:PARCEL>X4489</cad:PARCEL>
  <cad:PP_I_LTR>D</cad:PP_I_LTR>
  <cad:PP_I_NR>0001</cad:PP_I_NR>
  <cad:X_AKR_OBJECTNUMMER>GDA01M 04489G0000</cad:X_AKR_OBJECTNUMMER>
  <cad:LINE_ID1>340505636</cad:LINE_ID1>
  <cad:LOCATION>
    <gml:Point srsName='http://www.opengis.net/gml/srs/epsg.xml%23;28992'>
      <gml:coordinates decimal='.' cs=',' ts=' '>
        106410845,448674342
      </gml:coordinates>
    </gml:Point>
  </cad:LOCATION>
  <cad:AKTE_DATUM>0</cad:AKTE_DATUM>
  <cad:LABEL>perceel A</cad:LABEL>
  <cad:STATUS>0</cad:STATUS>
  <cad:RESTPERCEEL>oude eigenaar</cad:RESTPERCEEL>
  <cad:AKTENUMMER>123456</cad:AKTENUMMER>
 </cad:DRAFT_PARCEL>
</wfs:Insert>
<wfs:Insert>
 <cad:DRAFT_PARCEL>
  <cad:TMIN>260218287</cad:TMIN>
  <cad:TMAX>0</cad:TMAX>
  <cad:SOURCE>-</cad:SOURCE>
  <cad:QUALITY>T1</cad:QUALITY>
  <cad:MUNICIP>GDA01</cad:MUNICIP>
  <cad:OSECTION> w</cad:OSECTION>
  <cad:PARCEL>X4489</cad:PARCEL>
  <cad:PP_I_LTR>D</cad:PP_I_LTR>
  <cad:PP_I_NR>0001</cad:PP_I_NR>
  <cad:X_AKR_OBJECTNUMMER>GDA01M 04489G0000</cad:X_AKR_OBJECTNUMMER>
  <cad:LINE_ID1>340505636</cad:LINE_ID1>
  <cad:LOCATION>
    <gml:Point srsName='http://www.opengis.net/gml/srs/epsg.xml%23;28992'>
      <gml:coordinates decimal='.' cs=',' ts=' '>
        106376934,448708677
      </gml:coordinates>
    </gml:Point>
  </cad:LOCATION>
  <cad:AKTE_DATUM>0</cad:AKTE_DATUM>
  <cad:LABEL>perceel B</cad:LABEL>
  <cad:STATUS>0</cad:STATUS>
  <cad:RESTPERCEEL>nieuwe eigenaar</cad:RESTPERCEEL>
  <cad:AKTENUMMER>123456</cad:AKTENUMMER>
 </cad:DRAFT_PARCEL>
</wfs:Insert>
</wfs:Transaction>
```

The server responds with the result of the transaction and specifies the new id (fid) of the feature instance:

```
<wfs:WFS_TransactionResponse
   version="1.0.0"
   xmlns:wfs="http://www.opengis.net/wfs"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wfs
   http://130.161.150.109:8080/geoserver/data/capabilities/wfs/1.0.0/WFS-
transaction.xsd">
   <wfs:InsertResult handle="null cad:DRAFT_BOUNDARY 1">
      <ogc:FeatureId fid="feature-13261823"/>
   </wfs:InsertResult>
   <wfs:InsertResult handle="null cad:DRAFT_PARCEL 2">
      <ogc:FeatureId fid="feature-30940873"/>
   </wfs:InsertResult>
   <wfs:InsertResult handle="null cad:DRAFT_PARCEL 3">
      <ogc:FeatureId fid="feature-19736274"/>
   </wfs:InsertResult>
   <wfs:TransactionResult>
      <wfs:Status>
         <wfs:SUCCESS/>
      </wfs:Status>
   </wfs:TransactionResult>
</wfs:WFS_TransactionResponse>
```

This concludes the transaction, since it has been completed successfully.

Appendix E: Open source development GeoServer

The GeoServer project aims to be the Apache for web feature servers. More information on GeoServer can be found at http://geoserver.sourceforge.net.

GeoServer 1.0.0 and 1.1.0 beta contained several bugs that troubled especially processing transactions. In the MSc-project time has been spent on debugging and testing this. Most problems have been solved in version 1.1.0 so that transactions can be performed. The box below contains a discussion with a GeoServer developer on one of these one of these issues.

Thijs:
Hi,
…, here are some things on inserts with 1.1beta (and an Oracle database).

1. …, this problem is known in JIRA already (as GEOS-33), but now I finally have a good example. With inserts, FID's are returned as: <ogc:FeatureId fid="feature-17227669"/>. This is GeoServer's internal FID, not the one that should come from the sequence in the (Oracle) database.
…

Sean:
This is a known issue. Do you have any suggestions on how you would like GeoServer to perform here? The JIRA task has a couple ideas on how to fix it, your input would be appreciated.

Well, if you mean how GeoServer should get its value for the FID, I think the best would be that GeoServer picks the next value from the sequence in Oracle. I assume Postgis/postgres has a similar mechanism to generate values for id's e.g. Anyway, the specs say that the server must generate the id and I think using a sequence would be the most elegant way. I do not know any other solutions in Oracle than that (I'm not an Oracle expert, just some new user :)).

I've seen in the code and from a test that GeoServer does determine the next value for the Oracle sequence correctly, but the only thing is that this value is not used/put in the database. And how is GeoServer's internal fid determined now? It seems to be some random value, but is it checked to the existing ids to make sure it is unique?

To make it short, maybe this is an acceptable solution for Oracle (and Postgis as well?):
- let GeoServer check for a sequence on the fidcol (don't know if this is possible however)
- in case there is a sequence: get the nextvalue and use that as ID
- in case there is no sequence: use a random value. The thing is that this value has to be checked for uniqueness.

Sean (Oracle expert for GeoServer)
Just a couple things:

* Ideally the FID should be generated from a sequence. However a sequence is not always present and if it is it is there are a couple of different ways it can be used depending on how the database is set up. So what I am asking for is opinions on how to allow users to specify a FID generation strategy.

* We have come up with a way of handling this that added a configuration property for each feature type. This property will be called something like fidGenerationType. It will have at least the following options:
AUTO (means that the database will automatically generate the fid so geotools should insert NULL into the fid column. This will be used when you have a trigger or an auto-incrementing column),
the other option is MANUAL (This will use the OracleDataSource method of incrementing the max value. It should only be used if there are no sequences or autoincrement options). The OracleDataStore will also add an option that allows you to specify a sequence which will be used, this option can be used if you have a sequence but no trigger set up.

* Having thought about this, there also needs to be a way to get the last insert id when the auto method is being used so it can be reported back. There is not really any vendor independant way of handling this though, e.g. Oracle allows you to select curr_val from the sequence, but I don't know how other DBs do it. This might mean that this functionality needs to be pushed up into DataStore subclasses and a sequence name will need to be provided if the AUTO method is used.

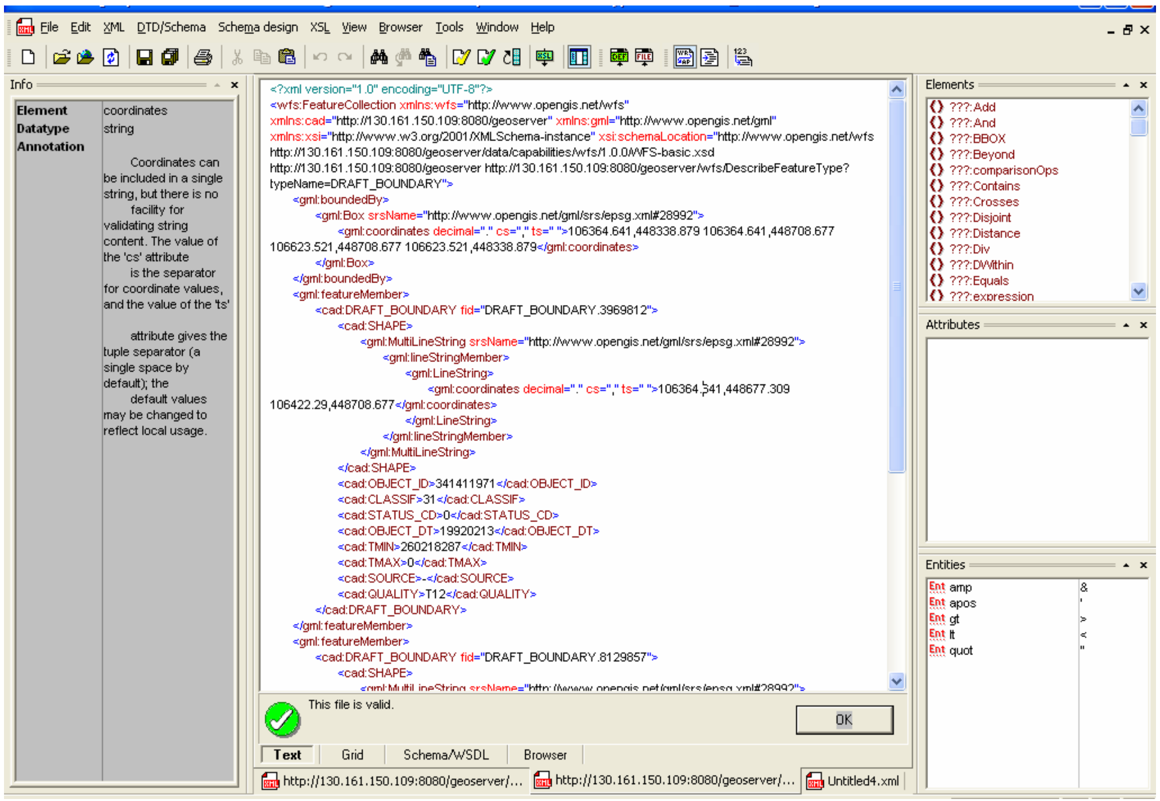
Besides the issued mentioned above, other problems that have been signalised and tested in this research project are:

- problems with attributes that are allowed to be NULL but were not returned on a *GetFeature* request, even if the attribute was not NULL;
- a problem with filtering on feature identifiers, that didn't succeed;
- a problem with matching of data types (strings only containing numbers were treated as integers or doubles)
- a problem in converting geometry from GeoServer to Oracle's geometry type SDO_geometry;
- problems with the structure of transaction request in XML. For example, if two features should be inserted, two insert elements (one for each feature) had to be used in GeoServer, while (according the WFS specification) these features are allowed to be in one insert-element.

These problems have been solved in cooperation with developers contributing to the GeoServer project.

Appendix F: Validation of web feature server responses

This appendix describes the validation of responses of the GeoServer web feature server. Validation has been done using XMLSpy Home edition version 2004 (http://www.xmlspy.com, see figure F.1 for a screenshot).



Element	coordinates
Datatype	string
Annotation	Coordinates can be included in a single string, but there is no facility for validating string content. The value of the 'cs' attribute is the separator for coordinate values, and the value of the 'ts' attribute gives the tuple separator (a single space by default); the default values may be changed to reflect local usage.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection xmlns:wfs="http://www.opengis.net/wfs"
xmlns:cad="http://130.161.150.109:8080/geoserver" xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wfs
http://130.161.150.109:8080/geoserver/data/capabilities/wfs/1.0.0/WFS-basic.xsd
http://130.161.150.109:8080/geoserver http://130.161.150.109:8080/geoserver/wfs/DescribeFeatureType?
typeName=DRAFT_BOUNDARY">
    <gml:boundedBy>
        <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
            <gml:coordinates decimal="." cs="," ts=" ">106364.641,448338.879 106364.641,448708.677
106623.521,448708.677 106623.521,448338.879</gml:coordinates>
        </gml:Box>
    </gml:boundedBy>
    <gml:featureMember>
        <cad:DRAFT_BOUNDARY fid="DRAFT_BOUNDARY.3969812">
            <cad:SHAPE>
                <gml:MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
                    <gml:lineStringMember>
                        <gml:LineString>
                            <gml:coordinates decimal="." cs="," ts=" ">106364.541,448677.309
106422.29,448708.677</gml:coordinates>
                        </gml:LineString>
                    </gml:lineStringMember>
                </gml:MultiLineString>
            </cad:SHAPE>
            <cad:OBJECT_ID>341411971</cad:OBJECT_ID>
            <cad:CLASSIF>31</cad:CLASSIF>
            <cad:STATUS_CD>0</cad:STATUS_CD>
            <cad:OBJECT_DT>19920213</cad:OBJECT_DT>
            <cad:TMIN>260218287</cad:TMIN>
            <cad:TMAX>0</cad:TMAX>
            <cad:SOURCE>-</cad:SOURCE>
            <cad:QUALITY>T12</cad:QUALITY>
        </cad:DRAFT_BOUNDARY>
    </gml:featureMember>
    <gml:featureMember>
        <cad:DRAFT_BOUNDARY fid="DRAFT_BOUNDARY.8129857">
            <cad:SHAPE>
```

This file is valid.

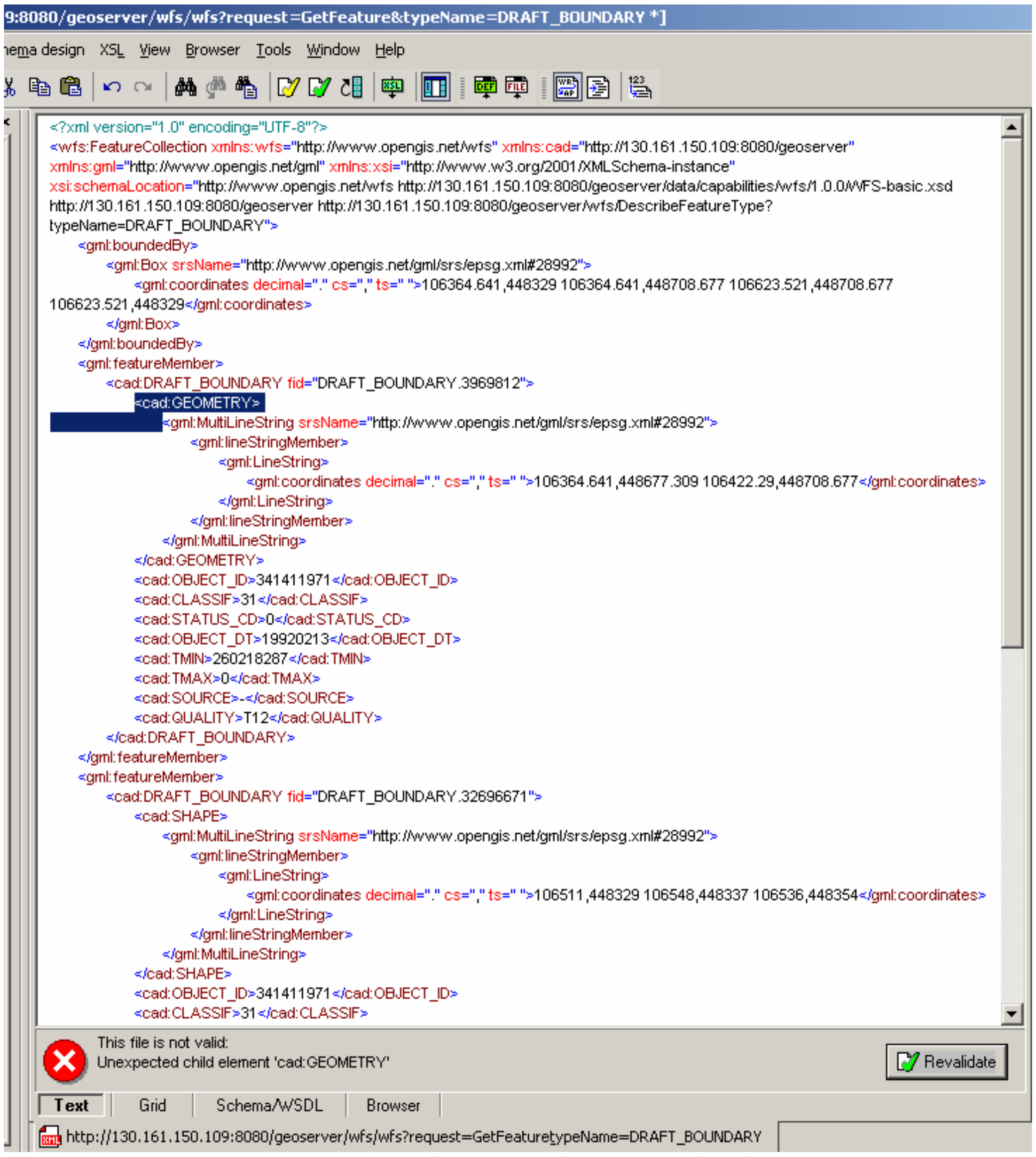
http://130.161.150.109:8080/geoserver/... | http://130.161.150.109:8080/geoserver/... | Untitled4.xml

Figure F.1 Screenshot of XMLSpy. The GML is validated against it's schema and is tested valid.

For GeoServer one can define application schemas by hand. This has been done for all feature types in the case study. During testing of GeoMedia Viewer as a WFS client, it became clear that errors occurred when retrieving features from the web feature server (GeoServer). GeoMedia Viewer uses the XML Schema documents for each feature type when connecting to a web feature server. The features it received for two feature types did not validate against the schemas and therefore were not visualized.

Because of this error, it was decided to validate the GML output stream of GeoServer. Validating this GML did not succeed. However, the GML itself seemed to be correct. Therefore the hand made XML Schema documents were checked. These contained errors in some attribute and element names, for two feature types. Because the schema was not correct, the GML did not validate (Figure F.2).

After correcting these errors (i.e. matching the element names to the exact column names in Oracle of the corresponding feature type), the GML output stream was validated again. Now the GML validated against the schemas.


9:8080/geoserver/wfs/wfs?request=GetFeature&typeName=DRAFT_BOUNDARY *]
hema design XSL View Browser Tools Window Help
Revalidate
Text Grid Schema/WSDL Browser

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection xmlns:wfs="http://www.opengis.net/wfs" xmlns:cad="http://130.161.150.109:8080/geoserver"
xmlns:gml="http://www.opengis.net/gml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wfs http://130.161.150.109:8080/geoserver/data/capabilities/wfs/1.0.0/WFS-basic.xsd
http://130.161.150.109:8080/geoserver http://130.161.150.109:8080/geoserver/wfs/DescribeFeatureType?
typeName=DRAFT_BOUNDARY">
    <gml:boundedBy>
        <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
            <gml:coordinates decimal="." cs="," ts=" ">106364.641,448329 106364.641,448708.677 106623.521,448708.677
106623.521,448329</gml:coordinates>
        </gml:Box>
    </gml:boundedBy>
    <gml:featureMember>
        <cad:DRAFT_BOUNDARY fid="DRAFT_BOUNDARY.3969812">
            <cad:GEOMETRY>
                <gml:MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
                    <gml:lineStringMember>
                        <gml:LineString>
                            <gml:coordinates decimal="." cs="," ts=" ">106364.641,448677.309 106422.29,448708.677</gml:coordinates>
                        </gml:LineString>
                    </gml:lineStringMember>
                </gml:MultiLineString>
            </cad:GEOMETRY>
            <cad:OBJECT_ID>341411971</cad:OBJECT_ID>
            <cad:CLASSIF>31</cad:CLASSIF>
            <cad:STATUS_CD>0</cad:STATUS_CD>
            <cad:OBJECT_DT>19920213</cad:OBJECT_DT>
            <cad:TMIN>260218287</cad:TMIN>
            <cad:TMAX>0</cad:TMAX>
            <cad:SOURCE>-</cad:SOURCE>
            <cad:QUALITY>T12</cad:QUALITY>
        </cad:DRAFT_BOUNDARY>
    </gml:featureMember>
    <gml:featureMember>
        <cad:DRAFT_BOUNDARY fid="DRAFT_BOUNDARY.32696671">
            <cad:SHAPE>
                <gml:MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
                    <gml:lineStringMember>
                        <gml:LineString>
                            <gml:coordinates decimal="." cs="," ts=" ">106511,448329 106548,448337 106536,448354</gml:coordinates>
                        </gml:LineString>
                    </gml:lineStringMember>
                </gml:MultiLineString>
            </cad:SHAPE>
            <cad:OBJECT_ID>341411971</cad:OBJECT_ID>
            <cad:CLASSIF>31</cad:CLASSIF>
```

```
This file is not valid:
Unexpected child element 'cad:GEOMETRY'
```

http://130.161.150.109:8080/geoserver/wfs/wfs?request=GetFeaturetypeName=DRAFT_BOUNDARY

Figure F.2 GML that is not valid against it's schema.

This example shows that validation of XML is essential to make sure that the XML output (in this case GML) is compliant with the specification. If the input or output of the XML is not compliant with the service specification, the service is not truly interoperable. This results in generic clients (as GeoMedia Viewer) that can't use the service correctly, for example.