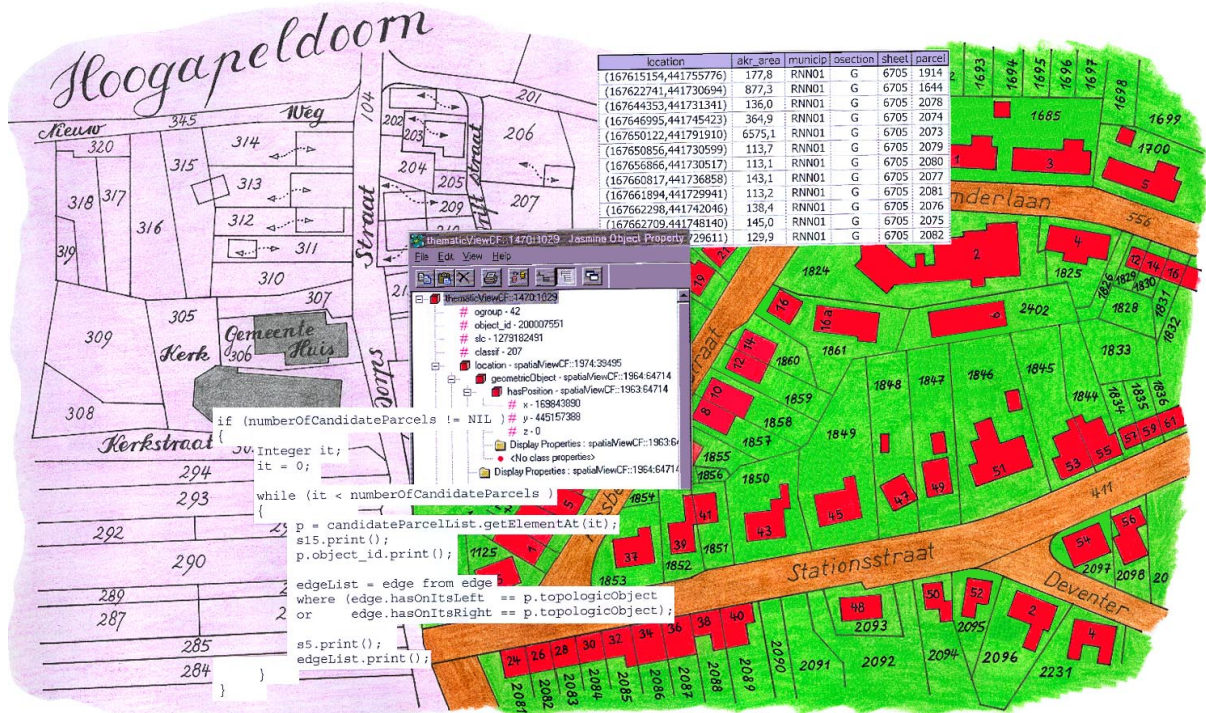


# Jasmine

## Spatial modelling with an object oriented database system



Master thesis

Patrice Wijnands

Supervisor: Prof.dr.ir. Peter van Oosterom

Instructors: Prof.ir. Henri Aalders

Drs. Wilko Quak

Delft University of Technology

Faculty of Civil Engineering and Geosciences

Department of Geodesy

August 2001





Wednesday, September 27<sup>th</sup> 2000, a small girl named Yasmina vanished, five years old. October 3<sup>rd</sup> she was found, but not alive anymore. I did not know her, but since these days I remembered her name every time I was working on this thesis.



## Preface

This is a Master thesis on object orientation, its capabilities, advantages and disadvantages when using it to keep a database holding geo information. The study and research tasks, performed at the Department of Geodesy of the Faculty of Civil Engineering and Geosciences of the Delft University of Technology, started in September 2000 and were completed in July 2001.

Readers interested in the concepts and ideas of object orientation can get an overview in chapter 3. Readers interested in the object oriented database system Jasmine will find a description of the system in chapter 4 and a description of an implemented database in chapter 5. Chapter 6 describes some tests performed and can be understood without knowledge of object orientation.

I choose to write this thesis in the “we” form. Not because I want to use a kind of Pluralis Majestatis. No, I am not that important. This form is allowed to for example our queen. The reason I developed during my study, for which I had to read many books. I always felt it like a sympathetic gesture when authors used the “we” form, trying to involve the reader into his book.

I’d like to thank Wilko Quak, for always being patient in providing help and documentation; Henri Aalders for giving always good advice; Peter van Oosterom for his critical remarks and for signing dozens of papers so I could obtain a key for the building; Theo Tijssen for helping me with the computer facilities; Edward Verbree for his always critical remarks; Franklin Monteiro for his cooperation and for his help on introducing a new programming language named Perl; Herman van der Wal for answering all my questions on the Jasmine system; Mario dePrisco for keeping my computer healthy; Axel Smits for scanning the cover; Wietse Balster for his patience on waiting for me to come home and prepare dinner and for his good cooking in case it was his turn; my parents who always welcomed me home and took care of me after some weeks of working and of course Elke Berger for her love and patience and her never failing ability to make me forget my work which made that I never lost motivation.

Patrice Wijnands



# Contents

Preface .....	v
Contents .....	vii
Abstract .....	ix
Samenvatting .....	xi
1 Introduction .....	1
2 Geo information in relational databases .....	3
2.1 Introduction .....	3
2.2 A relational database .....	3
2.3 The LKI system model .....	4
2.4 Advantages of relational databases .....	6
2.5 Disadvantages of relational databases .....	6
2.6 Towards a better LKI .....	6
2.6 Summary .....	7
3 Geo information in object oriented databases .....	9
3.1 The concepts of object orientation .....	9
3.2 Database capabilities .....	12
3.3 Definition of an object oriented database system .....	13
3.4 The advantages of object orientation .....	13
3.5 The disadvantages of object orientation .....	14
3.6 Summary .....	14
4 The Jasmine object oriented database system .....	15
4.1 Object orientation in Jasmine .....	15
4.2 Database capabilities of Jasmine .....	17
4.3 Is Jasmine an object oriented system? .....	19
4.4 Description of the standard interfaces to the database within Jasmine Studio .....	19
4.5 System requirements .....	24
4.6 Summary .....	24
5 Implementation of an object oriented database in Jasmine: the LKI .....	25
5.1 The LKI test database .....	25
5.2 An object oriented data model for the LKI .....	26
5.3 How to handle objects that change .....	36
5.4 Methods on the LKI object oriented database .....	37
5.5 Summary .....	38
6 Comparison .....	39
6.1 The database volume test .....	39
6.2 The index test .....	41
6.3 The “find boundaries that have only one next or previous boundary” test .....	42
6.4 The “find boundaries with the same parcel at both sides” test .....	43
6.5 The “find all parcels inside a given rectangle” test .....	44
6.6 Summary .....	47
7 Conclusions and recommendations .....	49
7.1 Conclusions on the use of an object oriented database .....	49
7.2 Conclusions on the use of Jasmine for an implementation of an object oriented database .....	49
7.3 Recommendations .....	50
8 References .....	53

Appendix A	The data conversion .....	55
Appendix B	Conversion programs written in Perl .....	59
Appendix C	Storing and structuring of data in Jasmine .....	153
C.1	Creating the stores .....	153
C.2	Creating the class families .....	153
C.3	The object oriented data model .....	154
C.4	Creating the classes .....	160
Appendix D	Methods written in ODQL .....	167
D.1	Calculate the area of a given parcel .....	167
D.2	Count the parcels within a rectangle by centroid .....	173
D.3	Count the parcels within a rectangle by bounding box .....	173
D.4	Create index .....	174
D.5	Find the surrounding boundaries of a parcel .....	175
D.6	Find boundaries that point into other boundaries .....	179
D.7	Find boundaries inside a parcel .....	180
D.8	Find a parcel from a given directPosition (point-in-polygon) .....	181



## Abstract

The database system used by the Dutch Cadastral Office for spatial information, the relational database management system CA OpenIngres LKI (in this thesis abbreviated to LKI), is an administrative application. Such applications are typical relational database systems. Relational database systems have great advantages but suffer from certain disadvantages causing limitations. In this thesis, an object oriented database system called Jasmine is investigated, to find a solution for these limitations. An object oriented data model for the LKI has been implemented and tested. The AKR, the counterpart of the LKI holding the information on rights subjects have on objects for example ownership, is not covered in this thesis.

The database technique is developing towards object orientation, with a trend towards a mixture of relational and object oriented systems. The abilities of relational systems are very well known, but object oriented systems are less well explored. Research on their abilities is very important.

One of the disadvantages of relational systems is the missing ability to query recursively, which makes it in the practise of the LKI impossible to calculate areas of parcels from the topology and geometry of the boundaries within the database system. Another disadvantage is the limited modelling capacity. This forced the designers of the LKI to split up the data on parcels and store the information on island parcels within parcels in another table. Using an object oriented database system, at least in theory can solve such complications.

In this thesis, an experiment is documented on the object oriented database system Jasmine. The main question is:

*Is the object oriented database system Jasmine suitable to store and query spatial data?*

A dataset of the Dutch Cadastral Office, describing the geometry and topology of the boundaries and parcels in and around the Grebbeberg in the municipality of Rhenen in the middle of the Netherlands, is used for testing purposes.

First, an object oriented data model is designed, based the European standard CEN-ENV 12160 for storing spatial data. The test dataset is converted to meet this data model, using several Perl scripts. Then an object oriented database is built up. Its modelling capacities overcome the modelling limitations of relational database systems. Very quickly additional data structures can be implemented. The built in Object Database Query Language (ODQL) makes it possible to build not only queries within a short time, but also methods to perform operations. Implemented are a point in polygon operation and an area calculation operation. Inheritance of data definitions and methods saves time. The compilation of methods however did not succeed due to software problems. Errors were reported on methods that were not present anymore when running the same code in a built in code interpreter. Therefore, no real methods could be implemented. The implemented operations were run and tested by using the code interpreter.

The original dataset is also loaded into a relational database system -MS-Access- for comparisons with Jasmine. Performed tests cover storage space and performance. The storage of the data in Jasmine lead to waste of storage space because of only partly filled pages. The data stored inside Jasmine needed a factor 30 more on storage space than when stored outside Jasmine in a dump file in a text format. This dump file containing object oriented structured data need as much storage space as the original relational structured data stored in MS-Access.

Performed operations on the test data set resulted in long waiting times in Jasmine, up to 30 seconds and more, where MS-Access needed only a few seconds. This is partly caused by the code interpreter. A compiled method will always perform faster. Besides, spatial indexing and spatial clustering are not available in Jasmine.

From this thesis can be concluded that Jasmine is an object oriented database system suitable to store and query spatial data, to build a small GIS-system capable of performing the spatial operations implemented and other operations that share the same basic arithmetic and database operations. Object orientated systems overcome the disadvantages of relational systems. Jasmine has however its own limitations to overcome.



## Samenvatting

Het databanksysteem dat door het Nederlandse Kadaster wordt gebruikt voor ruimtelijke informatie, het relationele *database management* systeem CA OpenIngres LKI (kortweg LKI genoemd in dit verslag), is een administratieve applicatie. Zulke applicaties zijn typisch relationele databanksystemen. Relationele databanksystemen kennen belangrijke voordelen maar hebben te maken met sterk beperkende nadelen. Tijdens deze afstudeeropdracht is het object georiënteerde databanksysteem Jasmine onderzocht om een oplossing voor deze beperkingen te vinden en is een object georiënteerd data model toegepast en getest op de LKI. De AKR, dat de rechten beschrijft die subjecten hebben op objecten (zoals eigendom), valt buiten het kader van dit afstudeeronderzoek.

De databanktechniek ontwikkelt zich in de richting van object oriëntatie, met een trend naar een combinatie van relationele en object georiënteerde systemen. De mogelijkheden van relationele systemen zijn goed bekend, maar object georiënteerde systemen zijn nog minder goed verkend. Onderzoek naar hun mogelijkheden is daarom erg belangrijk.

Eén van de nadelen van relationele systemen is het ontbreken van de mogelijkheid om recursieve *queries* uit te voeren. Dit maakt het voor de praktijk van de LKI onmogelijk om de oppervlakte van percelen uit te rekenen, onder gebruikmaking van de topologie en de geometrie van de grenzen binnen het databank systeem. Een ander nadeel is de beperking op de mogelijkheden tot modelleren. Dit dwong de ontwerpers van de LKI tot het opsplitsen van de gegevens over percelen: de gegevens over eilandvormige percelen binnen andere percelen moesten gedeeltelijk in een aparte tabel worden opgeslagen. Zulke complicaties kunnen worden opgelost door gebruik te maken van object georiënteerde systemen. In theorie tenminste.

In dit afstudeerverslag is een experiment gedocumenteerd met het object georiënteerde databanksysteem Jasmine. The hoofdvraag is:

*Is het object georiënteerde databanksysteem Jasmine geschikt om ruimtelijke gegevens op te slaan en te bevragen?*

Een dataset van het Kadaster is gebruikt om dit te testen. Deze dataset beschrijft de geometrie en topologie van de percelen op en rond de Grebbeberg in de kadastrale gemeente Rhenen.

Eerst is een object georiënteerd datamodel ontworpen waarin de LKI-data kunnen worden opgeslagen. Dit model is gebaseerd op de Europese standaard CEN-ENV 12160, die beschrijft hoe ruimtelijke gegevens zouden moeten worden opgeslagen. De test dataset is geconverteerd naar dit datamodel met behulp van scripts geschreven in *Perl*. Hiermee is een object georiënteerde databank opgebouwd. De modelleercapaciteiten kennen niet meer de beperkingen hierop van relationele databanksystemen. Aanvullende datastructuren kunnen snel worden toegevoegd. De ingebouwde *Object Database Query Language (ODQL)* maakt het niet alleen mogelijk om binnen korte tijd *queries* te vormen, maar ook methoden om operaties uit te voeren. Hiermee zijn een “*point in polygon*”-operatie geïmplementeerd en een operatie om oppervlaktes van percelen te bereken. Overerving van datastructuren en methoden kan veel tijd en programmeerwerk sparen. Het compileren van methoden was echter niet mogelijk ten gevolge van *software* problemen. De compilatie levert foutmeldingen op voor methoden die wel uitgevoerd kon worden door een *code interpreter*. De *interpreter* is gebruikt om de geïmplementeerde operaties uit te voeren en te testen.

De originele test dataset is ook in een relationeel databanksysteem ingeladen: MS-Access. Zo konden vergelijkingen gemaakt worden met Jasmine. Vergeleken werden de benodigde schijfruimte en de *performance*. De opslag van de data in Jasmine verspilte veel schijfruimte omdat *pages* slechts ten dele werden gevuld. De data gebruiken 30 keer zoveel schijfruimte binnen Jasmine als opgeslagen in een *dump file* in een tekstformaat buiten Jasmine. Deze object georiënteerd georganiseerde data in een *dump file* gebruiken net zoveel schijfruimte als de relationele data opgeslagen binnen MS-Access.

Operaties op de testdata in Jasmine duren erg lang, tot 30 seconden en meer. MS-Access heeft voor dezelfde operaties slechts enkele seconden nodig. Dit werd ten dele veroorzaakt door de *code interpreter*. Een gecompileerde methode zal altijd sneller werken. Daarnaast is in Jasmine geen ruimtelijke indexering en ook geen ruimtelijke clustering beschikbaar.

Uit dit afstudeeronderzoek kan worden geconcludeerd dat Jasmine een object georiënteerd databanksysteem is, geschikt voor het opslaan en bevragen van ruimtelijke gegevens, voor het bouwen van een klein GIS-systeem dat in staat is tot het uitvoeren van de geïmplementeerde ruimtelijke operaties en andere operaties op basis van dezelfde aritmetische en databankoperaties. Object georiënteerde systemen kennen niet meer de nadelen van relationele systemen. Jasmine kent echter haar eigen beperkingen.



# 1 Introduction

Relational databases have some disadvantages: limited modelling capacities [Kroha, 1993], no recursive querying, limited possibilities to hold multi media data and large objects [Khoshafian et al., 1999]. Can an object oriented database system overcome these? An object oriented database system, what does that mean? Its market share is only small, compared to that of the relational database systems like Computer Associates Ingres II or Oracle. The database technique is developing towards object orientation, but without completely leaving the relational techniques. The trend tends towards object-relational systems like: [Khoshafian et al., 1999] Intelligent SQL, Oracle 8 or Informix's Illustra. A logical development, because the investments in relational systems were high and they function still properly. On the other hand, pure object oriented systems do not carry the load of older systems anymore. Therefore, it is very important to know the possibilities of object orientation. After one of the first object oriented database systems, Servio Logic's Gemstone, built on Smalltalk, a wide range of companies like Object Design, Versant, Ardent, Fujitsu and Computer Associates offer them [Khoshafian et al., 1999], [Kroha, 1993].

To explore the research field on object oriented databases, a system called Jasmine is tested. Offered by the last two companies, it is a modern commercial system, developed recently and brought onto the market. Of course Jasmine is one out of many systems and I will surely recommend to investigate other object oriented database management systems as well.

In this Master thesis the following question will be answered:

*Is the object oriented database system Jasmine suitable to store and query spatial data?*

To find an answer, the following research questions are investigated:

1. How is an object oriented database system characterized?
2. Is an object oriented database a better solution for conceptualisation and modelling of the real world than a relational database?
3. What are the disadvantages of relational databases and can object oriented databases overcome these?
4. Is Jasmine really an object oriented database system?
5. What system requirements are needed to run the database management system?
6. How much storage space does a database in Jasmine need?
7. Can Jasmine be used to store and query a dataset describing the geometry and topology of a model of the world's surface?
8. Is it possible to implement a data model following an international standard on geographic information?
9. How does Jasmine perform under spatial operations? Testing of the correct functionality and the efficiency.

This is not the first time Jasmine is used for spatial modelling. Van Wijngaarden (1997) already investigated its possibilities to hold spatial data. He implemented another data model following Peng e.a. (1996) but concluded that Jasmine possibly will become suitable for developing GIS-applications when spatial data types, spatial indexing and visualisation of spatial data will be realized. Up to now, these are not realized. This limits this Master thesis.

Chapter 2 of this thesis gives a short overview on relational database techniques, without attempting to be complete. It lists some of the advantages and disadvantages and discusses the database system maintained by the Dutch Cadastral Office: the relational database management system CA OpenIngres LKI, in this thesis abbreviated to LKI.

Chapter 3 introduces object oriented programming and databases, its concepts, advantages and disadvantages. The system Jasmine is described in chapter 4. The implementation of an object oriented data model to hold topological and geometrical data and the implementation of an object oriented database is covered by chapter 5. This database holds the test dataset describing the boundaries and parcels around the Grebbeberg, municipality of Rhenen. It is a part of the LKI.

Chapter 6 describes tests performed on this object oriented database, benchmarked on a simple relational database holding the same data but in the original relational data model, in order to make a comparison. This thesis will not cover the implementation and performance tuning of a relational database. Hardware configurations will also not been taken into account. The comparison between the two systems will therefore be limited.

Chapter 7 finally holds the conclusions of this Master thesis.

Appendix A summarizes all the details of the data conversion and how it is performed. The code of the conversion programs written in Perl is included in appendix B. Appendix C is about the way the data are stored in Jasmine, it holds the complete definition of the data model. Appendix D contains the code of the methods written in ODQL.



## 2 Geo information in relational databases

### 2.1 Introduction

In this chapter we will discuss relational database techniques, without attempting to be complete. Advantages, disadvantages and problems known in the database system maintained by the Dutch Cadastral Office will be covered.

Database techniques are developed from the need to collect, store and use information in a structured way. Let's take an example on this. Anyone collecting data on objects, which share similar properties, wants to compare these. He can write down his results on a piece of text, on paper or in a digital file. This raises two disadvantages:

- he will forget to check for some properties when investigating some objects;
- while reading the collected information, he cannot easily compare the different objects.

One solution is to create a table. The columns give the possibility to hold the properties. This also raises two disadvantages:

- the number of columns cannot be changed from object to object;
- some values are repeated.

The problem of the columns is awkward, but unsolvable for this moment. We will see in the next chapter how we can overcome this problem.

The columns of a table are called the *attributes*. The rows of a table the *tuples*. The cells contain *attribute occurrences* or *values*.

A table can hold only information on one kind of object, called the *entity*. Of course these objects can be separated into types, described by its attributes (table 1).

**Table 2.1: the naming conventions for tables.**

entity	attribute	attribute
tuple	value or attribute	value or attribute
tuple	occurrence	occurrence

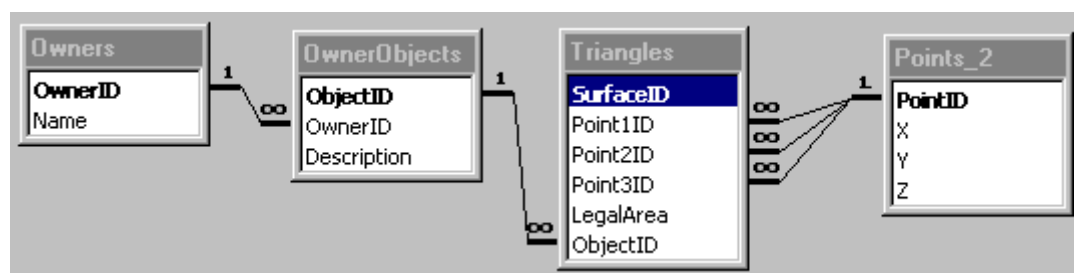
The problem of the repeating properties results into tuples holding repeating values. This raises two disadvantages:

- you have to write down the same information a large number of times;
- you have to change all these entries when you decide to change the description of a property.

The solution is to divide the data over more tables. One main table and several sub tables to hold the repeated property. The property becomes an entity of its own. A *relation* keeps the connection between the two.

### 2.2 A relational database

Codd developed this model of the structure for a database, published in [Codd, 1970]. I will give a short explanation of this model.



**Figure 2.1: schema of a simple relational database for a cadastral system.**

The above schema illustrates a simple *relational database model*. In the entity “Owners” you can keep a registration of the people who own a piece of land. You write down their names, but maybe also their addresses. You can deal with natural persons but also with organisations.

One owner can own one or several pieces of land. These pieces of land are kept in the entity “OwnerObjects”. The identifier of the owner is one of its attributes. This is the connection between the owner and its piece of land.

If you would use one single entity, you were forced to write down the name and address of the owner to every piece of land. This will have to be changed when the owner moves to another town for every tuple describing a piece of land he owns.

Every piece of land is modelled by one or several triangular surfaces defined by their three corner points. These are kept in the entity “Triangles”. The connection is the identifier of the piece of land, which is kept in the attribute “ObjectID” of the entity “Triangles”.

Every triangle is modelled by three points, of which the coordinates are kept in an entity “Points”. The point identifiers are used to connect the triangles to their points. This is very important because neighbouring triangles use the same two points. Now you can avoid registering the coordinates of a point more than one time. However you cannot avoid that two triangles overlap each other partly when a wrong point id is used.

The process of dividing the data over several entities in order to avoid multiple storage and keeping the data consistent is called *normalisation*.

## Normalisation

First normal form: remove fields holding repeating groups of data and create relations instead.

Second normal form: remove all fields that do not depend on the key of the table and create relations instead.

Third normal form: remove all fields that are dependent of other fields and create relations instead.

[Worboys, 1995]

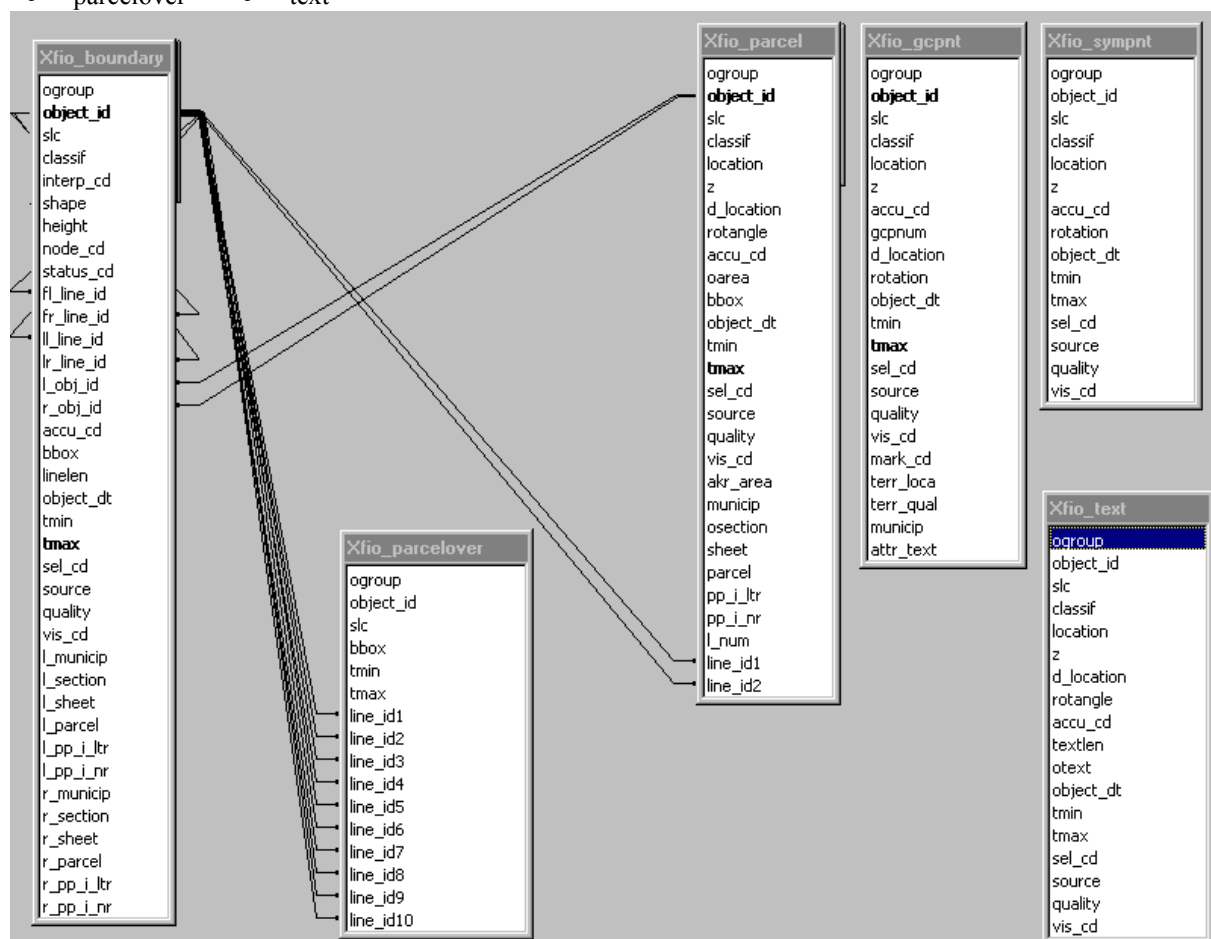
It is important to know that other models are also available. Beside the relational, [Worboys, 1995] mentions also the hierarchical, network, object oriented and deductive database models. The object oriented model will be described in chapter 3.

## 2.3 The LKI system model

The schema beneath (figure 2.2) shows also a model of a cadastral system, which is far more sophisticated and complicated than the schema in figure 2.1. It's not my intention to explain it in all details. A more extended description can be found in [Osch, 1997]. This system holds the geometrical and topological data of the real estate registered by the Dutch Cadastre as in the cadastral maps, divided over the entities:

- boundary
- parcel
- parcelover
- gcpt
- sympnt
- text
- line

**Figure 2.2: schema of the Dutch cadastral system, Landmeetkundig Kartografisch Informatiesysteem.**





The entity “**line**” contains topographic elements from the GBKN (large scale map of the Netherlands 1:5,000) and is not used in this thesis. Therefore it will not be discussed here.

The entity “**boundary**” is the largest one in the number of attributes and -after “line”- in the need of storage space. The upper fields contain identifiers, layer information and classifications. The line ids (fl\_line\_id, ..., ll\_line\_id) refer to the boundaries that are connected to one boundary, according to figure 2.3.

The fields “l\_obj\_id” and “r\_obj\_id” refer to the parcel left and the parcel right of the boundary.

The entity “**parcel**” holds in its two last attributes the identifiers of boundaries. “line\_id1” refers to a boundary of the outer ring (the outer surrounding boundary). “line\_id2” refers to one inner ring (an island in the parcel being a parcel of its own). If a parcel has more than one island, the references to one of their boundaries are stored in the entity “**parcellover**”. This means a parcel id can be listed in both the tables “parcel” and “parcellover”. “parcellover” is only a technical entity, not a logical entity. Here we see very clearly the problem of the number of a table’s columns that cannot be changed. This leads to modelling limitations.

The entity “**gcpnt**” contains information on marked points that can be used to reconstruct or visualize the boundaries in the field. The entity “**sympnt**” holds the positions of cartographic symbols. The entity **text** contains text strings, like house numbers, street names etc.

All entities contain the fields “tmin” and “tmax”. They hold time stamps between which the objects are valid, counted in seconds database time. This is a way to implement versioning (see chapter 3) and keep history data. Figure 2.4 illustrates this: until May 1<sup>st</sup> 1985 the parcel nr 1 exists:

The value registered in his tuple for the “tmax” attribute is zero, indicating they are valid for the current situation. The value registered in its “tmin” attribute

indicates the time point on which the parcel’s tuple was created.

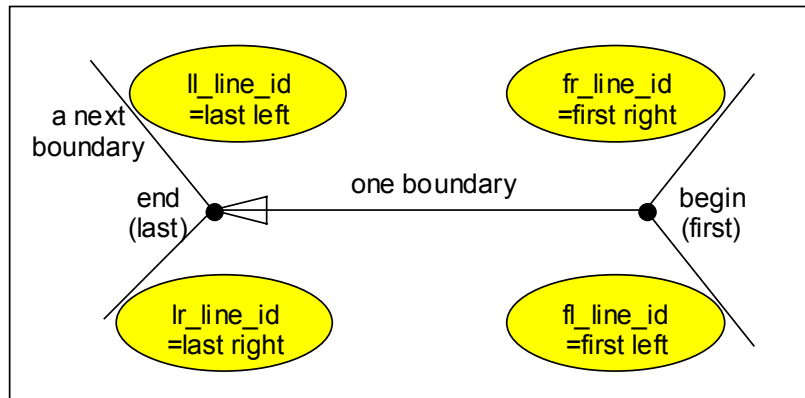


Figure 2.3: the topological relations of the boundaries in the LKI, following the winged edge structure.

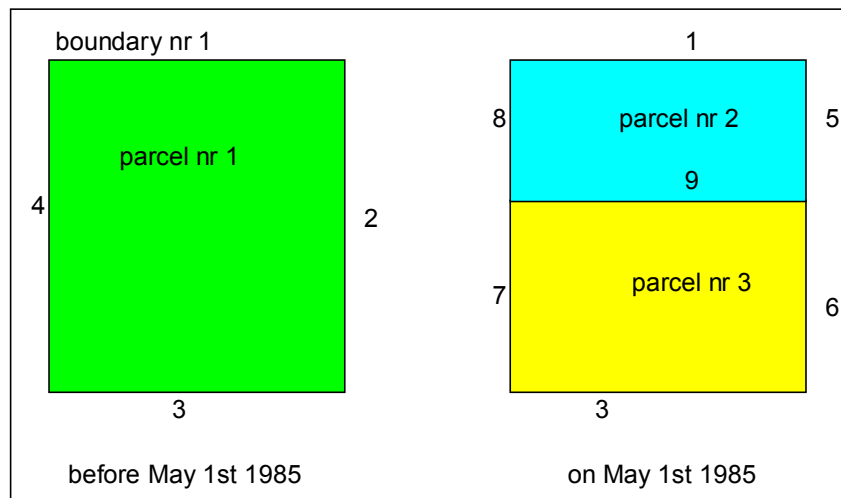


Figure 2.4: example of a change in the LKI-database.

Table 2.2 (left) and table 2.3 (right): How the entity “parcel” changes when a parcel is being split.

entity parcel before May 1 <sup>st</sup> 1985			entity parcel after May 1 <sup>st</sup> 1985		
object_id	tmin	tmax	object_id	tmin	tmax
1	<some number of seconds>	0	1	<some number of seconds>	<seconds until May 1 <sup>st</sup> 1985>
			2	seconds since May 1 <sup>st</sup> 1985	0
			3	seconds since May 1 <sup>st</sup> 1985	0

On May 1<sup>st</sup> the parcel is being split up. Two new parcels are created and five new boundaries. Now the value registered in the “tmax” attribute of parcel 1 is set to the current database time, indicating that this tuple is not valid anymore for the current situation. Copies of the tuples of the boundaries 1 and 3 are saved again, but with references to the new parcels 2 and 3, so in fact seven new tuples describing boundaries are created. The values registered in the “tmin” attribute of all the new tuples are set to the current system time. The values registered in the attribute “tmax” of all the new tuples are set to zero, indicating that they are now valid for the current situation. [Oosterom, 1997]

The “xfio” in front of all the entity names indicates that attribute occurrences with non-zero values registered in the attribute “tmax” are present. This means the history data are available. The attribute “tmax” is part of the key of the entity, together with the attributes “ogroup” and “object\_id”.

The abbreviation xfio stands for “X-Fingis OpenIngres”.

The system described is not standing on its own. Attached to it exists the AKR (*Automatische Kadastrale Registratie*). This holds the information concerning the natural and juridical persons holding rights on real estate, like ownership.

## 2.4 Advantages of relational databases

- ☑ Relational databases are the product of the demand for better consistency mechanisms, which they fulfil. The duplication of data can be avoided.
- ☑ The data structure is simple, based on matrix operations, and can easily be understood.
- ☑ The underlying mathematical basis is clearly defined.

## 2.5 Disadvantages of relational databases

Relational databases are according to [Ishikawa, 1993] not very suitable for the generation of generic knowledge. Apart from the question whether systems are able to generate knowledge, this is a result of the disadvantages of relational databases:

- ☑ The normalisation forces the user to create a large number of tables [Kroha, 1993].
- ☑ Not only the investments in time and money are high. Looking up the data requires a large number of JOIN-operations, on the cost of memory capacity and processor time. Of course this is application dependant.
- ☑ Aside to this, recursive querying is not possible. Recursive querying is needed for looking up data by following the links to other tuples in the same entity. In fact, this is asking the same questions more than one time. Alternatively formulated, executing the query several times. How often is not known at forehand.
- ☑ The limited possibilities to hold multimedia data and large objects [Ishikawa, 1993].
- ☑ Kroha (1993) concluded that relational databases limit the modelling capacities.

## 2.6 Towards a better LKI

The limitations of the relational database management system of the LKI do not restrict the task execution of the Dutch Cadastral Office at the moment. Applications outside this system take care of the tasks the system is not capable of, for example topology checks. An object oriented system should be able to complete this within its system.

The problem of not possible recursive querying becomes evident when the boundaries of a parcel are needed, sorted in the right order. This is often needed by applications to visualize the parcels on screen or paper, or to calculate the area. It should be remarked that it is *not* mathematically necessary to obtain the boundaries in the right order to calculate the area, following this two-point algorithm:

$$\text{area of polygon} = \frac{1}{2} (x_2 - x_1) (y_1 + y_2)$$

In fact the area of the rectangle underneath two points is calculated, by multiplying the difference between the x-coordinates with the average of the y-coordinates. Only the points per boundary must be followed clockwise or anticlockwise, but the boundaries themselves have not be ordered.

To obtain the boundaries, a user looks up the tuple describing the parcel in the entity parcels, in SQL:

```
SELECT *
FROM Xfio_parcel
WHERE ((Xfio_parcel.object_id)=1));
```

The parcel's attribute "line\_id1" delivers the identifier of one tuple in the entity boundaries:

```
SELECT *
FROM Xfio_parcel INNER JOIN Xfio_boundary ON Xfio_parcel.line_id1 = Xfio_boundary.object_id
WHERE ((Xfio_boundary.object_id)=1));
```

The four identifiers in the attributes "xx\_line\_id" point to two candidates that surround the same parcel. Take for example figure 2.4. The tuple describing boundary number 1 contains the identifiers of the tuples describing the boundaries number 2, 4 and two other ones (if present). A JOIN-operation available in SQL can find parcel number 1 and boundary number 1. However, there is no SQL-statement possible that can also look up boundary 2, 3 and 4, unless the query is reformulated and executed again. Only then, the user can find boundary number 2.

Because no information is available on how often the query should be executed, it is not possible to find the boundaries in the right order. The only solution is to look up all the boundaries that contain the parcel's object identifier listed in the field l\_obj\_id or r\_obj\_id, but the boundaries will not be found in the right order:

```
SELECT *
FROM Xfio_boundary
WHERE (((Xfio_boundary.l_obj_id)=1)) OR
      (((Xfio_boundary.r_obj_id)=1));
```

We will see how an object oriented database management system orders the boundaries of a parcel within its system.

Another example concerns the coordinates of points, which are saved with the entities they belong to. The entity boundary keeps an attribute called shape, holding the points describing the geometry, with a maximum of 50 points. The start point and end point is shared with other boundaries. The coordinates of these points are stored multiple times. This disadvantage can be accepted when considering the better performance when storing the coordinates this way. We will make the experiment in the object oriented data model to store only references to points with the boundaries and store the coordinates only once.

The problem of the existence of islands within parcels is solved by the entity "parcelover". Actually the entity "parcelover" describes the 1:n relationship between parcels and island parcels. The relationship could also have been described by using an entity "parcel-island\_parcel", consisting of a column holding the object\_id of the parcel and of a column holding the object\_id of one island parcel. Each island parcel is related to the parcel by another tuple in this entity. The solution implemented stems however from considerations to save a large amount of storage capacity and obtain a better performance.

The existence of the entity "parcelover" is a typical problem of relational databases. The number of attributes cannot be changed from object to object, and an additional entity is needed. We will see how an object oriented data model can solve this problem.

## 2.6 Summary

In this chapter, we recalled our knowledge on relational databases and introduced the LKI database, which describes the geometry and topology of the cadastral parcels (and topographic elements) in the Netherlands. We saw some advantages of relational databases, but also the disadvantages, illustrated by examples of the LKI. We saw the different entities in this database, especially the entity "parcelover". This entity serves to store islands within parcels, when more than one island is present. A real problem is the failing ability to query recursively, which makes it impossible to sort within the database the boundaries of a parcel in the right order.

In the next chapters we will see how we can overcome these disadvantages.



## 3 Geo information in object oriented databases

In an object oriented database all entities or data entries are viewed as objects. Even digits and characters. It is possible to define collections of objects. A collection of characters can be used to store a word in some language. A collection of digits can be used to store a number. This is the most fundamental aspect of an object oriented database.

Paragraph 3.1 introduces the basic ideas and fundamental aspects of object orientation. Paragraph 3.2 discusses the capabilities any database system should support. Paragraph 3.3 gives a definition of an object oriented database system. The Paragraphs 3.4 and 3.5 finally list the advantages and disadvantages of object orientation.

### 3.1 *The concepts of object orientation*

The words “object” or “object orientation” are often used in a very broad manner and it is not possible to give a definition of it. We will review the ideas presented in three different texts in this paragraph. First an article by Peter Batty representing the point of view of Smallworld Systems. Second a book written by three authors, all employed by Technology Deployment International, Inc: Setrag Khoshafian, Surapol Dasanandra and Norayr Minassian. [Khoshafian e.a., 1999]. These will be compared with Michael Worboys’ discussion in [Worboys, 1995].

According to Batty [Batty, year unknown] there are five areas within the GIS context in which the word “object” is frequently used:

- object based systems;
- object centred systems;
- object oriented user interfaces;
- object oriented programming;
- object oriented databases.

Object based systems avoid splitting up the database into map sheets, on which objects are cut into two pieces when crossing the map’s border. They keep the description of these objects intact by storing them only once and keep a seamless geo data set. Batty points out that object based systems do not have anything to do with object orientation.

Object centred systems [Newell, 1992] describe the world in terms of objects people naturally perceive as objects like houses, streets and bridges. In contrast, a geometry centred model describes the world in terms of points, lines, surfaces and cubes. Also an object centred model does not have to be implemented by an object orientation.

An object oriented interface is often the classification used for a graphical user interface. However, there is by far no agreement on what an object oriented interface should be. One can point out, that any interface that gives the user the possibility to first select an object and then choose an action to be carried out gives right to be named object oriented. However, an object oriented interface doesn’t tell anything about the underlying data structures or functions.

Object oriented programming is rather clearly defined and accepted, at least after Batty. He points out the basic ideas:

- object;
- class;
- message;
- method.

Besides that, he lists the concepts of:

- encapsulation;
- polymorphism;
- inheritance;

He sees these as the main characteristics.

[Khoshafian e.a., 1999] considers three fundamental concepts for object orientation:

- abstract data typing;
- inheritance;

- object identity.

Khoshafian e.a. add abstract data typing and object identity.

[Worboys, 1995] discusses:

- object identity;
- encapsulation;
- inheritance;
- composition;
- polymorphism.

He mentions composition as a separate topic. We will have a close look on all these ideas and concepts in the next sections.

The key notion of the object oriented approach is that an object instance always has a current situation (the state) and that an object is equipped with a certain set of predefined reactions (behaviour) for received requests. The state is the totality of the data values stored within an object. The behaviour is expressed as a set of operations that the object can perform. The underlying ideas of the different authors are the same; it's only the terminology that differs:

object = state + behaviour	[Khoshafian e.a., 1999]
object = state + functionality	[Worboys, 1995]

The level of agreement on object oriented databases is actually really low. Should its implementation be based on an *object oriented programming language*? Should its syntax for querying and data handling being implemented as an object oriented interface? Should its actual data storage and handling use the ideas and concepts of object oriented programming? Should it be able to store Binary Large Objects (so-called “blobs” like images, sounds or movies) in a better manner? We will get to an answer in this chapter.

## Object oriented programming: object, class, message and method

This paragraph will explain the terms object, class, message and method and the next ones the concepts of abstract data typing, encapsulation, polymorphism, inheritance, object identity and composition.

An object instance can be seen as a data item like a variable or a constant in a programming language. A class can be viewed as a data type. It is the description of how a certain category of objects should look like and -very characteristic for object orientation- how it *behaves*. A class prescribes what actions can be carried out on its objects and how the objects will react to this action. An action can be requested by asking for it by means of a message. The message will trigger some piece of programming code to be executed, stored as if it was a program, called a method. Examples of very common actions are creation, deletion and printing. The methods define the behaviour of an object. [Khoshafian e.a., 1999], [Batty], [INT, 2001]

For example, given a database of persons, you might define a class named person. When meeting a new person, you might want to keep his or her name, phone number and address. Once home, you give into your system: `person.new()`

What you actually did was sending a message to the object's class “please create a new object in the class person”. As a result the method called “new” was executed, taking care of the definition of the properties of the new object and assign a unique identifier.

Above I used the word properties, which I did not mention before. What is a property? A property can be an attribute or a relationship [Khoshafian e.a., 1999]. When objects contain other objects this is a relationship. A name is an attribute.

The name is described in characters. A character is an object. The attribute “name” could contain an object of the class “character”. But did we not mention before that when objects contain other objects this is a relationship? Sure, but characters are objects without identity. There is no relationship needed to a class “character”.

But we cannot describe a person's name by only one character. Therefore we can store collections of objects belonging to one class into one property, so we store a collection of characters.

Such a collection of characters is widely used that we can decide to define a class named “string”. Now we can store an object of the class string into the property name.

The properties together contain the data values stored in an object. This defines the state of the object [Worboys, 1995] and makes the object to an object instance.

## Object oriented programming: abstract data typing

Abstract data typing is used “to describe a set of objects with the same representation.” [Khoshafian e.a., 1999, p. 36]. This refers to the basic idea of object instances belonging to classes. It consists of the representation (the data structure) and the operations (the algorithms). Abstract data typing separates the implementation (the internal representation of objects) from the interface to the outside world. This means this author uses the term abstract data typing to refer to the concept of encapsulation, which we will consider in the next paragraph.

## Object oriented programming: encapsulation, polymorphism and inheritance

Encapsulation means that data within an object can only be accessed or changed by the methods defined on the object's class. [Batty] This makes it possible to hide the implementation of an object's data structure and algorithms. The reason is to free a user from the need for exact knowledge of that internal data structure and algorithm(s). This makes the use of the data inside the objects less complicated. [Khoshafian e.a., 1999]

Inheritance means that subclasses at least get the same data structure (properties) and get the same behaviour (methods) of its super classes. It is possible to add properties, and make the subclass more specialized. It is also possible to define new methods in the subclass, even with the same names as methods of the super class. This is called polymorphism of methods [Khoshafian e.a., 1999].

“Polymorphism is the ability to appear in many forms. Polymorphism makes it possible to process objects differently depending on their data type or class. More specifically, it is the ability to redefine methods for derived classes.

For example, given a base class shape, polymorphism enables the programmer to define different circumference methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the circumference method to it will return the correct results.

The type of polymorphism described above is sometimes called parametric polymorphism to distinguish it from another type of polymorphism called overloading.”

“Overloading allows an object to have different meanings depending on its context. The term is used most often in reference to operators that can behave differently depending on the data type, or class, of the operands. For example,  $x+y$  can mean different things depending on whether  $x$  and  $y$  are simple integers or complex data structures.

Not all programming languages support overloading but it is a feature of most object oriented languages, including C++ and Java. Overloading is one type of polymorphism.”  
(After the Webopedia dictionary [INT, 2001].)

## Object oriented programming: object identity

Object identity is according to [Khoshafian e.a., 1999] extremely important. Every object has a unique system generated identity. It makes identity checks possible to check whether two objects are the same. This in contrast to equality checks where the contents of two objects are compared. Object identity alleviates the programmer's task on dynamic memory management, avoids bugs and makes applications more stable and easier to create.

## Object oriented programming: composition

[Worboys, 1995] mentions composition as a special feature. Composition allows the modelling of objects with a complex internal structure. He gives three ways in which objects may be composed:

1. by aggregation: component objects build an aggregated object. For example a house might be composed out of the walls and the roof.
2. by association: a set of objects of the same type form an associated or grouped object. For example an object of type “countries” might be an association of individual countries.
3. by ordered association: a set of component object placed in a certain order form an ordered association. For example an object of type “time sequence” might be structured as a linear ordering of “minute” objects.

### 3.2 Database capabilities

[Khoshafian e.a., 1999] defines the concept of an object oriented database as the sum of fundamental concepts for object orientation and database capabilities. Any database system should support the database capabilities, defined in table 3.1.

**Table 3.1: the database capabilities.**

<ul style="list-style-type: none"> <li>• persistence;</li> <li>• transaction;</li> </ul>	<p>Manipulated data can be transient (“short living”) or persistent (“long living”).</p> <p>A sequence of statements that is executed entirely or not at all in case of a problem. Transient data are only valid within a transaction. Transactions are atomic, consistent, isolated, and durable, also called the ACID properties:</p> <ul style="list-style-type: none"> <li>- “Atomic: a transaction is the smallest unit of recovery.</li> <li>- Consistent: a transaction takes the database from one consistent state to another consistent state.</li> <li>- Isolated: during its execution, a transaction cannot see any changes made concurrently by other transactions.</li> <li>- Durable: once a transaction commits, the system will ensure that its effects persist in the database even if subsequent failures occur.” (“Jasmine ii Database Developer’s Reference 2.0.”)</li> </ul>
<ul style="list-style-type: none"> <li>• concurrency control;</li> </ul>	<p>To keep the database consistent statements should be executed in a serializable order. A multi-user environment makes therefore special strategies necessary:</p> <ul style="list-style-type: none"> <li>- timestamp ordering (“first requested, first executed”);</li> <li>- optimistic algorithms (“solve concurrency conflict when it occurs”);</li> <li>- pessimistic or locking algorithms (“avoid concurrency conflicts”).</li> </ul>
<ul style="list-style-type: none"> <li>• recovery;</li> </ul>	<p>A database management system must protect the persistent part of the database from transactions that fail because of:</p> <ul style="list-style-type: none"> <li>- transaction errors;</li> <li>- system errors;</li> <li>- media errors.</li> </ul>
<ul style="list-style-type: none"> <li>• complex object modelling and querying;</li> <li>• versioning;</li> <li>• integrity constraints;</li> </ul>	<p>To select subsets and sub objects.</p> <p>To keep alternative versions of object instances available.</p> <p>Examples of integrity constraints:</p> <ul style="list-style-type: none"> <li>- unique primary key constraints;</li> <li>- existential constraints (“no dead end references”);</li> <li>- NOT NULL constraints (“data entry mandatory”);</li> <li>- mechanisms for integrity definition and maintenance like triggers: they consist of a condition component and an action component (other authors use the term “daemons” [Ishikawa, 1993]);</li> </ul>
<ul style="list-style-type: none"> <li>• security;</li> </ul>	<p>For example user access regulations, like:</p> <ul style="list-style-type: none"> <li>- read;</li> <li>- write;</li> <li>- erase;</li> <li>- create;</li> <li>- modify;</li> <li>- scan.</li> </ul>
<ul style="list-style-type: none"> <li>• performance issues.</li> </ul>	<p>Data structures and optimisation techniques like:</p> <ul style="list-style-type: none"> <li>- indexes (e.g. B-tree);</li> <li>- clustering (storage of referenced objects in the same block);</li> <li>- query optimisation;</li> <li>- storage structures (e.g. files on media disk).</li> </ul>



To conclude this, I would like to add availability as an important requirement. Availability expresses the time during which the system is restricted available or not available at all, due to maintenance or due to its sensibility for errors caused by:

- bugs in the software of the system itself (causes most of the system errors);
- bugs in the operating system;
- for transaction errors;
- for errors on media;
- for errors in network connections.

For example a system that keeps running and only terminates an operation that fails due to an error can be called robust. To the user an error message is shown. This minimizes the sensibility for errors and adds to the availability. Exception handling is provided with modern programming languages following the try-throw-catch principle [Laan, 1998] and every modern database system should be equipped with it.

### **3.3 Definition of an object oriented database system**

I will assume to this point that an object oriented database should meet the following ideas and concepts of object oriented programming when looking at the way data are stored, retrieved and queried:

- object;
- object identity;
- class;
- message;
- method;
- encapsulation;
- polymorphism;
- inheritance;
- composition.

Besides, the database capabilities should be supported:

- persistence;
- transactions;
- concurrency control;
- recovery;
- complex object modelling and querying;
- versioning of instances;
- integrity constraints;
- security;
- performance issues;
- availability.

### **3.4 The advantages of object orientation**

- ☑ Abstract data typing allows a better conceptualisation and modelling of the real world.
  - ☑ The semantics of types are explicitly defined by properties and operations.
  - ☑ Systems become more robust.
  - ☑ Systems are easier extensible; software components are reusable, easier to create and to maintain.
- [Khoshafian e.a., 1999]

An object oriented approach offers advantages specifically for spatial databases:

- ☑ The topology structure can be kept correct under transactions by the database management system itself.
- ☑ Special standardized classes for spatial entities.
- ☑ Special standardized class for temporal support.
- ☑ Explicit support of geometry and topology.
- ☑ Spatial indexing.
- ☑ Spatial clustering.

### **3.5 *The disadvantages of object orientation***

- ☒ Deep inheritance hierarchies are difficult to manage and understand. The dependence of the subclasses from the super class makes changes in the super class very difficult. [Khoshafian e.a., 1999]
- ☒ The step of leaving a relational database system and convert all the data into an object oriented database system requires high investments.
- ☒ An extra investment risk because no object oriented database system has a serious market share.

### **3.6 *Summary***

In this chapter the paradigm of object orientation is discussed. We saw which database capabilities any database system should support. We defined an object oriented database system and listed the advantages and disadvantages of object orientation. In the next chapter we will see how the issues discussed in this chapter are implemented in Jasmine.

## 4 The Jasmine object oriented database system

In the preceding chapter we saw which concepts, basic ideas and capabilities an object oriented system should support. We will now discuss these issues for Jasmine.

Paragraph 4.1 describes the implementation of the basic ideas and fundamental concepts of object orientation in Jasmine, which were introduced in paragraph 3.1. Paragraph 4.2 discusses the implementation of the database capabilities, which were introduced in paragraph 3.2.

Paragraph 4.3 covers the question “is Jasmine really an object oriented database system”. Paragraph 4.4 gives an overview of the Jasmine interfaces. Paragraph 4.5 lists the system requirements. Paragraph 4.6 finally summarizes the last three chapters. Most of the information is taken from the “Jasmine ii Database Developer’s Reference 2.0” [CA, 2000].

### 4.1 Object orientation in Jasmine

Objects are the fundament for Jasmine. Objects are stored in classes. Classes are again stored in class families. The next section will give a description of the system class family, which is defined at installation.

#### The class “object” in the system class family

The system class family has the following hierarchy (figure 4.1):

**Figure 4.1: the class hierarchy of the system class family.**

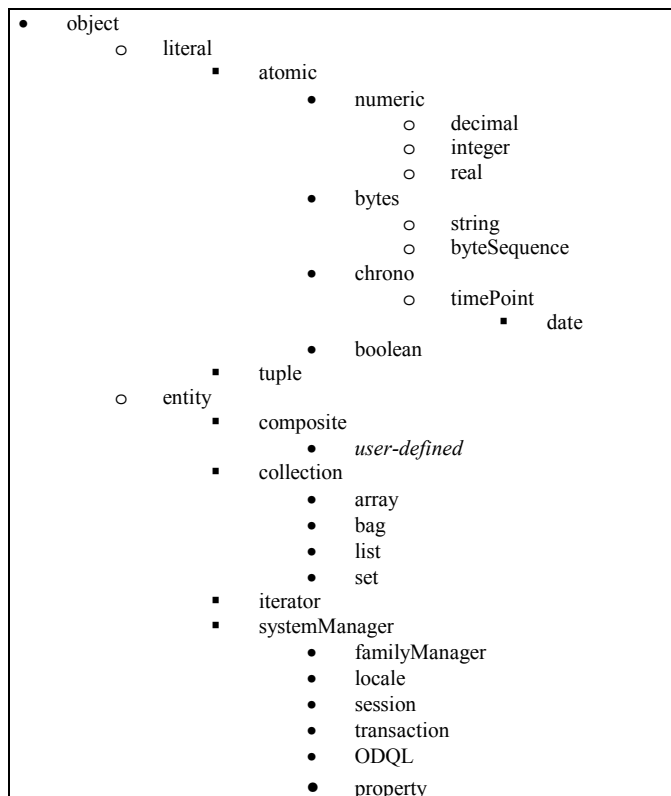
At the top the class “object”. A short explanation will be given here. More details can be found in the documentation.

A literal is a value such as a string, a decimal or an integer. Literals can be atomic or structured. Structured literals are tuples. Tuples serve to retrieve and process results, for example from queries.

An entity represents a real world object in the database. An entity can be user-defined and belong thus to the class composite. Four types of collections are defined and three types of restrictions for them are available (table 4.1):

**Table 4.1: the four collection types defined in the class “collection” of the system class family. The collection types in the rows, the defined restrictions on them in the columns.**

	fixed number of elements	duplicate values not allowed	collection is ordered
array	x		x
list			x
set		x	
bag			



The most strict collection type is array.

The “iterator” class serves to loop through a collection. It replaces the well known for or while statements.

The familyManager contains specific high level operations that act on classes and class families, for example to select classes or print information on a class family. Locale is used to store parameters that describe which time zone, date formats, character sets etc. have to be used. Session supplies the environment each time a Jasmine session is started by an application and provides some methods e.g. Session.clear() to delete all the collections defined during a session.

A transaction is contained within one session and is used to control the operations of starting, ending and rolling back transactions.

The class “ODQL” defines the statements available for the Object Database Query Language. This is the programming and query language provided with Jasmine. The “property” class serves to store information about ODQL statements.

## Object identity

Object identity is automatically maintained by the system. It is used only for object instances and remains unchanged through their lifetimes. Literals do not have any object identity.

## Class

Jasmine’s own structure is organized in classes, as we saw in the preceding discussion. It is possible to define new classes and manipulate them.

## Message

Jasmine supports the basic idea of messages, as illustrated in the following example:

```
Bag<parcel> p;  
Integer numberOfParcels ;  
numberOfParcels = p.count() ;
```

“Count” is the message sent to the object p.

## Method

Methods in Jasmine can be written and compiled using the Method Editor (see figure 4.7 in paragraph 4.4). Methods can be defined on four different levels:

- instance level methods, for example a method that prints the data of an instance;
- instance collection level methods, for example a method that counts instances;
- class level methods, for example a method that creates a new instance;
- class collection level methods, for example a method that counts classes.

See also figure 4.7. Instances are all objects belonging to a specific class. Instance methods work on single instances (“objects”) in the class. Class methods work on the class as a whole. Collection methods work on selected groups of instances or classes.

The message “count” from the previous example invoked an instance collection level method called “count”.

## Encapsulation

Jasmine completely supports the concept of encapsulation.

For example a method “calculateSurface”, which could be defined on the class parcel. This method has thus access to the data inside the objects of this class. But this method cannot directly access the data inside the class symynt describing the symbols on a cadastral map.

## Polymorphism

Jasmine completely supports the concept of polymorphism. For example the method “count”, which can be used to count collections of atomic literals (like integers), collections of tuples, collections of user entities (like parcels) or collections of classes. This means the method can be used to count integers, but also to count parcels.

## Inheritance

Jasmine completely supports the concept of inheritance. For example the method “count” which can be used to count classes, but also to count the classes’ sub classes for which the method is also defined.

It is also possible to implement multiple inheritance. For example a subclass can inherit properties and methods from two classes. Special attention has to be paid in case a method has the same name in both classes.

## 4.2 Database capabilities of Jasmine

In paragraph 3.2 we discussed which capabilities any database system should support. In the following paragraph we will discuss per issue -if it is supported- how it is implemented.

### Persistence

Jasmine can store data permanently in so-called stores, files on a medium. They contain the complete class definitions plus all the instances. These data are persistent.

### Transactions

During a transaction persistent data can be created or deleted, but transient data will always be deleted (if defined) at the transaction's end. Transactions in Jasmine support the ACID properties.

### Concurrency control

Jasmine uses locking to control the concurrency. Different levels of locking are possible:

- the whole database can be locked, for example for archiving the database;
- an entire class can be locked, for example when changing a property of instances fulfilling certain requirements;
- a single instance can be locked.

Locks can escalate from instances to classes.

### Recovery

Jasmine supports journaling ("logging") and provides backup/restore commands. This helps to protect the database against system and media errors. Transaction errors are avoided by rolling back any failing transaction.

### Complex object modelling and querying

The user can define its own classes and hierarchies, and make them almost as complex as wished. Class hierarchies can be made as deep as needed. Instances can contain thousands of instances of other classes. I did not reach any restrictions. The computer's resources restrict the capacity. The querying can be performed ad hoc using a query editor. ODQL scripts can be written and interpreted using the ODQL Interpreter (see figure 3.10).

### Versioning

Jasmine does not support any built-in versioning. The user can however create a versioning system of its own.

### Integrity constraints

Jasmine maintains key constraints automatically. Each new entity gets a unique identifier. Existential constraints are also automatically maintained. When a referenced object is deleted, the reference to this object is changed into "NIL". "NOT NULL" constraints can be defined when creating or modifying a class, and will result in the system refusing to store an object when this constraint is not fulfilled.

### Security

Encapsulation results in a certain level of security. Jasmine has however no built-in security measures. The user can create its own systems.

### Performance issues

Jasmine automatically creates an index on the identifiers of all the instances inside any class. The type of index is a B+ tree. In [Wijngaarden, 1997] it was announced that CA intended to implement a R-tree in Version 2.0 of Jasmine. A good description of the B+ and R tree structures can be found in [Worboys, 1995] and [Oosterom 1999].

When a class is created, a primary index is created automatically, based on the object identifiers. The user can define further secondary indices (of the same type) on properties of the instances. When a query is being executed, it uses at least the primary index.

Clustering is not supported. Query optimisation is not performed by the system.

The physical storage of data is done by creating files on media. These files, so-called stores, have to be created by the user. He has to provide e.g. the total size.

## Stability

The version of Jasmine I used (Jasmine TND) is still sensible for system errors causing the unexpected termination of the application Jasmine Studio. The reason is unclear. In case when changing a method's name into a too long new name, the whole method can even be lost. Other cases are reported when compiling methods written in the method editor. See paragraph 4.4 for more details. Error messages can appear whilst exact the same method was accepted earlier. Often the error message is not clear.

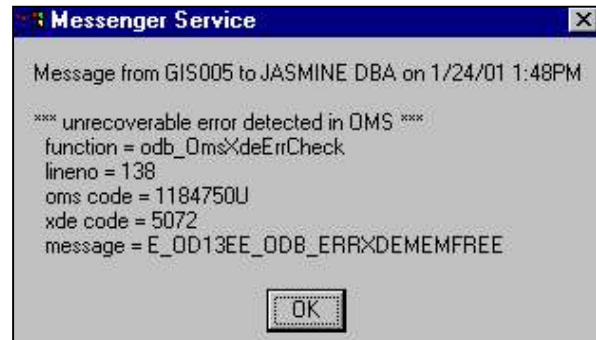
The application Studio of Jasmine ii was also tried, but here the whole application Studio did not run. Therefore, the older version was used instead (Jasmine TND, Studio version 1.2). The existence of two versions of Jasmine beside each other can be a reason for system errors.

In February, Jasmine lost all classes and instances of a user-defined class family including 200 Mb on data as the result of a system error (figure 4.2). The reason was a bug in the application Jasmine Studio. The problem occurred when the class browser within this application was started while the database itself was not running yet. As a result, the data were still present in the store, but could not be found anymore. The class family could not be deleted anymore, and because of this, its store neither. A re-installation was necessary. Computer Associates provided a patch for this bug.

No experience has been gained on sensibility on operating system errors.

Jasmine provides sophisticated error handling procedures for transaction errors. For example, errors in methods result in the termination of only that method, rolling back of the transaction and the displaying of an error message.

Errors on media or in network connections have not occurred.



**Figure 4.2: the database is corrupt due to a system error.**

### 4.3 Is Jasmine an object oriented system?

Batty formulated five questions to check whether a system is object oriented:

1. Is the system object based?
2. Is the system object centred?
3. Does the system support encapsulation, polymorphism and inheritance?
4. Does the system provide a set of standard class libraries, which can be extended by the user?
5. Does the system support the previous concepts within a database system?

Jasmine is object based, because no technical data divisions are necessary. An object crossing the border between two map sheets will be treated as one object. Jasmine is object centred because objects are stored as objects and not as collections of points, lines, surfaces and cubics. Encapsulation, polymorphism and inheritance are supported. During installation a set of standard class libraries are created and available to the user. They can easily be extended. Jasmine does support these concepts within a database system.

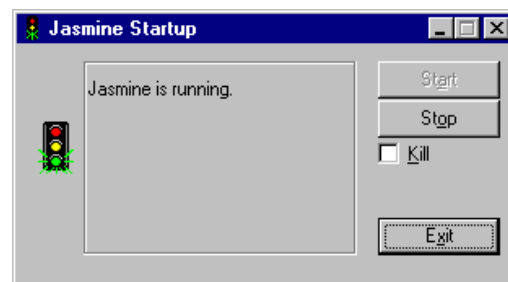
For Jasmine the answer to these questions is positive.

### 4.4 Description of the standard interfaces to the database within Jasmine Studio

In this paragraph, the figures 4.3 to 4.11 give a quick overview of the interfaces standard available in Jasmine. This is useful for readers who want to reproduce the test results documented in this thesis, but never worked with Jasmine before.

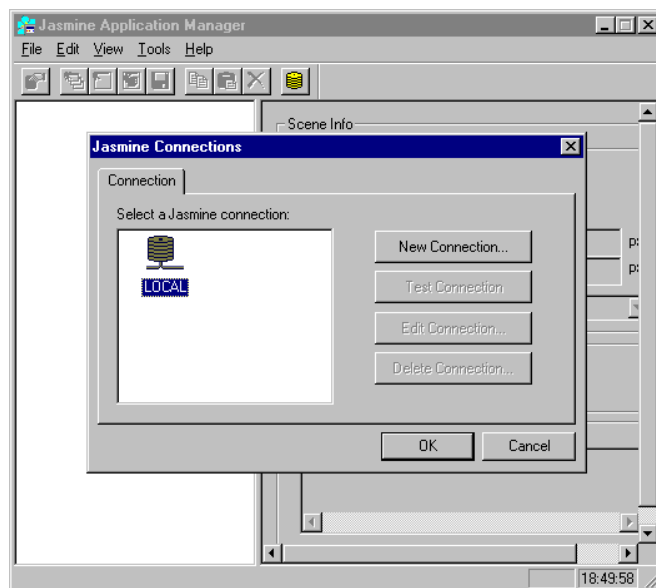
The database can explicitly be started and stopped. This is mandatory when unloading or loading data, because the database has to be shut down. (figure 4.3)

**Figure 4.3: starting and stopping the database system.**



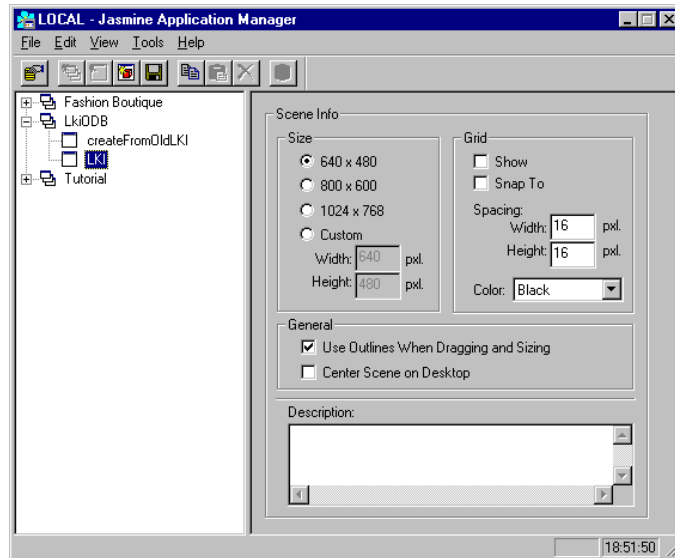
When the database is running, the application Studio can be used (figure 4.4). This is one way of approaching the database. Another way is by the ODQL Interpreter (see figure 4.12).

**Figure 4.4: starting Jasmine studio and setting up the database connection by choosing a connection (here only the “local” server is available). Database connections can be set up over a network or only local.**



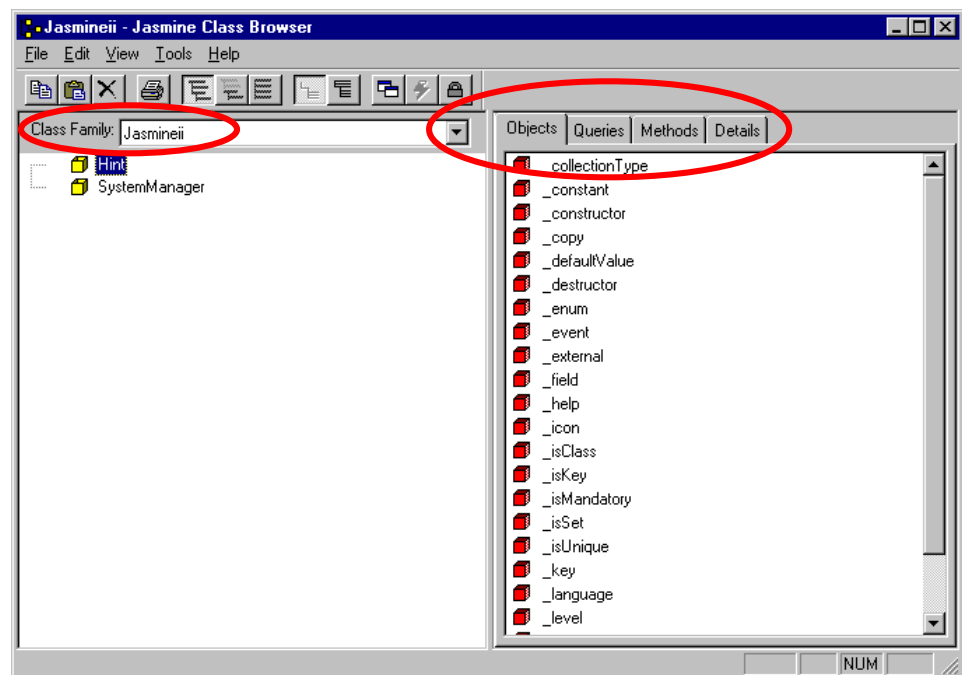
Within Studio (figure 4.5) the Application Manager can be used to view, modify, create or delete applications built on basis of the data structure and the data. These applications are another way to approach the database. These applications can be web based.

**Figure 4.5: the Application Manager.**



The data structure and data can be viewed, modified, created or deleted using the Class Browser, a part of Studio (figure 4.6). Distinction is made between the object instances inside a class, the queries, the methods and the super class (in the tab “details”). All classes in Jasmine are divided over class families (abbreviated as CF). Within a class family the class names are unique.

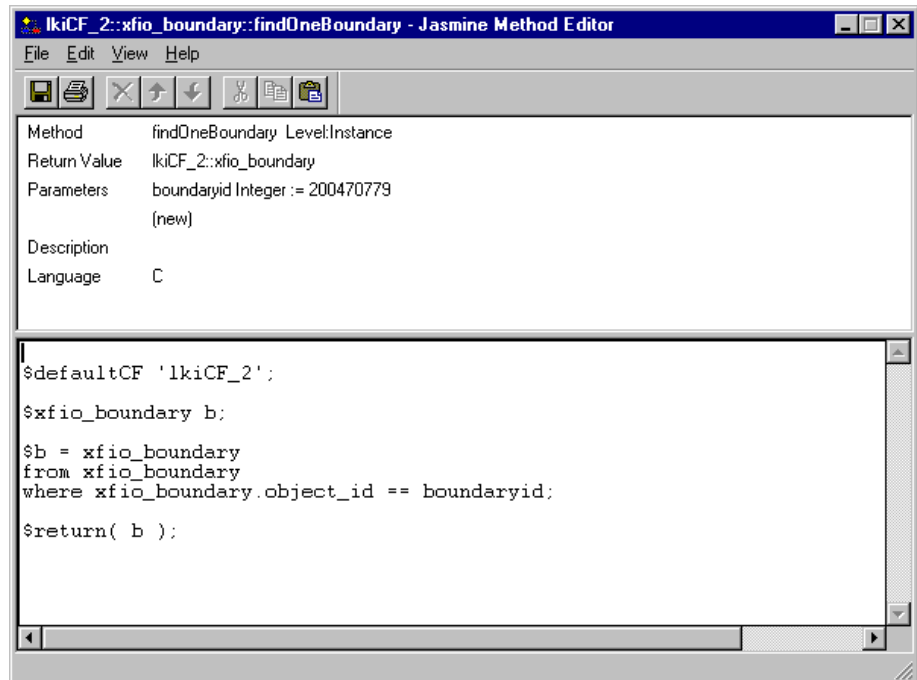
**Figure 4.6: the Class Browser. The left ellipse indicates the class family. The right ellipse indicates the tabs holding objects, queries, methods and details.**





To write a method, the Method Editor can be used (figure 4.7). This is another part of Studio. A method will be compiled in the database, and this imposes strict restrictions. It is possible to mix C or C++ with ODQL. However, the ODQL statements need a “\$” sign at the start of every new line.

**Figure 4.7: The Method Editor.**

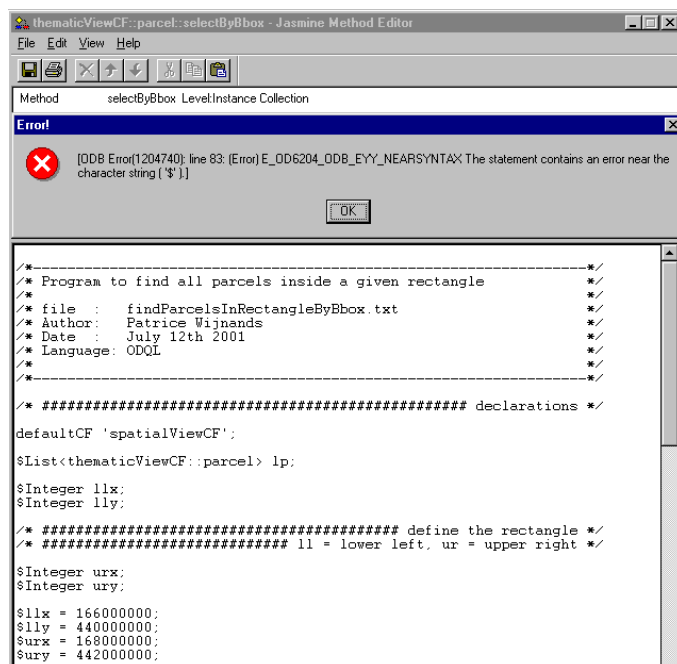


The Method Editor does not function properly. When trying to save a method containing a syntax error, an error message is displayed (figure 4.8). This error message is not very useful. It refers to the line number on which the error is found, but the Method Editor does not indicate any line numbers. The message text mentions that the error is near the character string “\$”, but every code line in ODQL begins with the “\$” sign. Sometimes this error message is not displayed anymore and the whole application Jasmine Studio hangs.

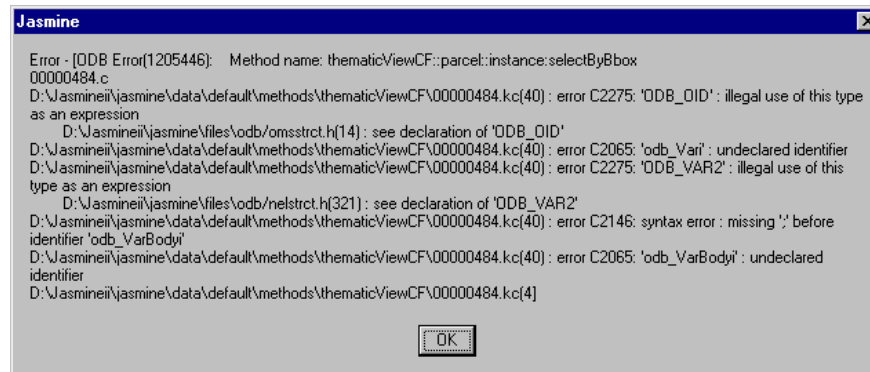
The compilation of methods fails often. This is logical when the code contains an error. But the compilation fails often on the addition of a simple piece of code (see figure 4.9). In this case only an integer was declared. The displayed error message is not clear.

Because of these problems it was not possible to compile methods. The codes were written and tested using the ODQL Interpreter (see figure 4.12).

**Figure 4.8: an error message displayed when trying to save a method containing a syntax error.**



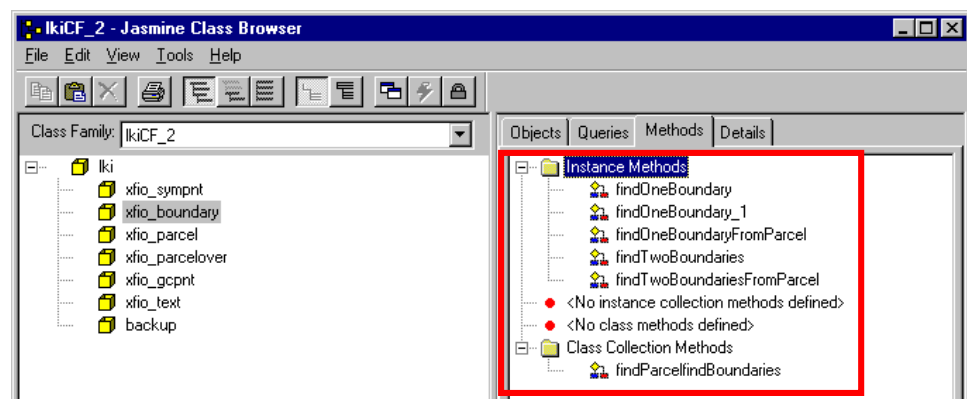
**Figure 4.9: an error message displayed after the compilation of a method failed.**



Four different levels can be defined for the methods as described in paragraph 4.1:  
(figure 4.10)

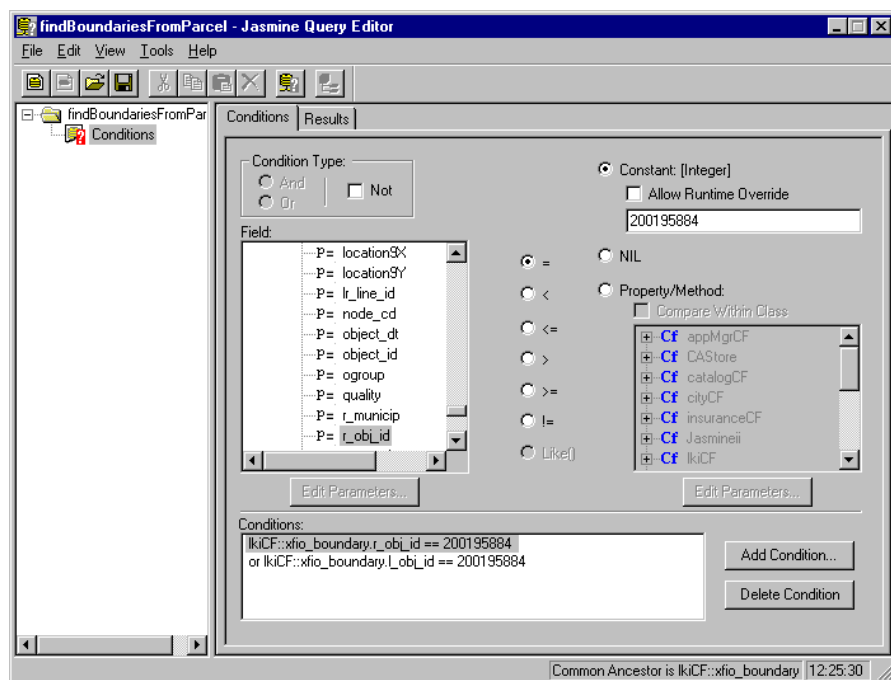
- instance level methods;
- instance collection level methods;
- class level methods;
- class collection level methods.

**Figure 4.10: inside the rectangle the “methods” tab of the class browser.**



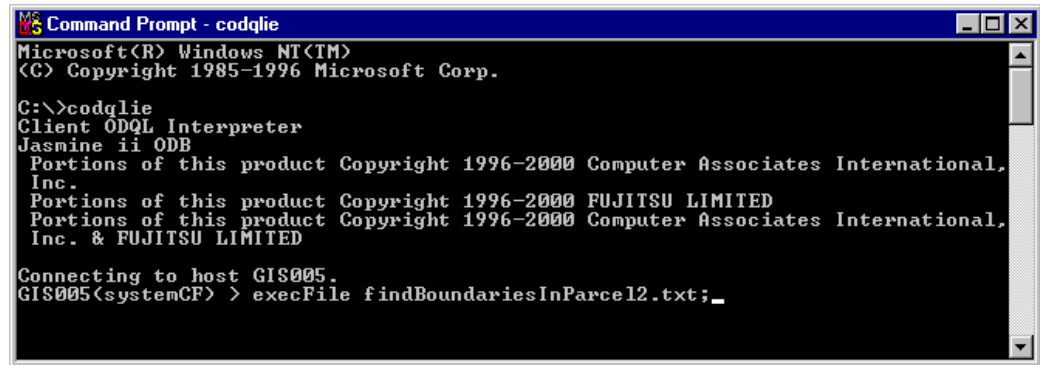
A query can ad hoc be made using the query editor (figure 4.11). However, only a single class together with its related classes can be queried. For example recursive querying is not possible.

**Figure 4.11: the Query Editor.**



In order to query data for questions not possible in the Query Editor, a text file with ODQL statements can be interpreted using the ODQL Interpreter (figure 4.12). Typing “codqlie” at the DOS-prompt starts the Interpreter.

**Figure 4.12:**  
the ODQL  
Interpreter.



This utility can also be used to test and run ODQL code for methods. “\$” signs in front of the statements are not allowed anymore. The advantage over the Method Editor is the lower restriction level of the interpreter.

Because the Method Editor did not function properly, this utility was used to test and run the codes for the methods. (See also paragraph 4.2 in the section on stability). Unfortunately the ODQL Interpreter does not support function calls to other ODQL scripts saved as text files. Encapsulation or polymorphism is naturally only possible for methods inside the database.

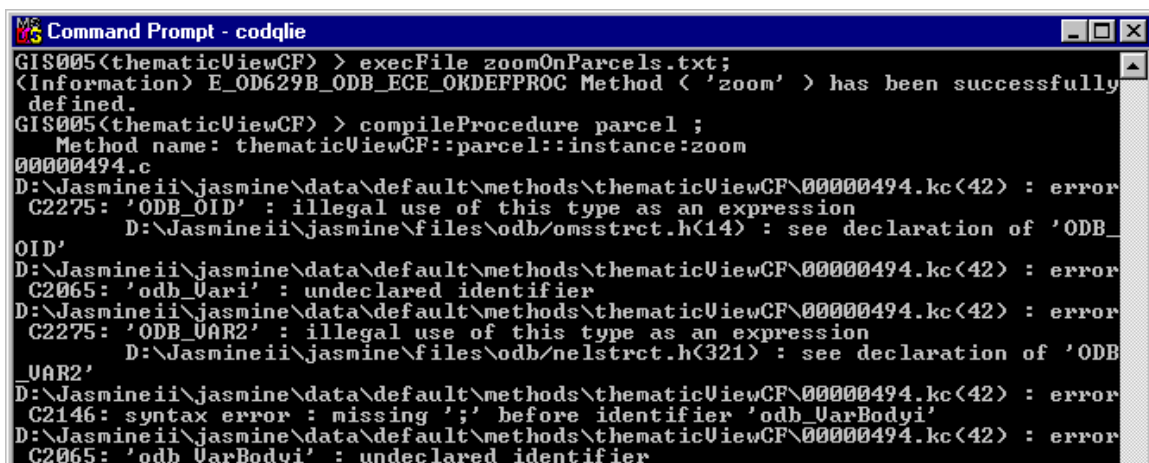
It was also tried to use the Interpreter instead of Studio to define methods. Therefore an ODQL script was prepared, holding the command “addProcedure” and a complete method:

```
defaultCF 'thematicViewCF';

addProcedure List<parcel> parcel::instance:zoom()
{
    $....(other ODQL statements)
    $List<thematicViewCF::parcel> lp2;
    $....
    $return( lp2 );
};
```

The code for this method was successful executed by the Interpreter earlier and assumed to be free of errors. Then this script was interpreted by the ODQL Interpreter resulting in the definition of the method “zoom” and the storage of the code. See figure 4.13, first two lines.

Then the command “compileProcedure parcel;” was given into the ODQL Interpreter in order to compile all the methods of the class “parcel”. This compilation failed, but the error messages are not clear (figure 4.13).



**Figure 4.13:** part of the error message displayed as the compilation of a method in the ODQL Interpreter failed.

## 4.5 System requirements

(Taken from “Jasmine ii System Administrator Guide 2.0” [CA, 2000])

When installing Jasmine ii on a Windows platform:

- a Pentium processor;
- 128 Mb RAM (256 Mb recommended);
- the amount of hard disk space can differ, but the applications requires 90 Mb, to function properly much more; (For Jasmine TND 300 Mb was required and 400 Mb recommended);
- Windows 95, 98, 2000, NT 4.0;

Jasmine TND and Jasmine ii were installed in different directories on a Windows NT 4.0 workstation in single user modus with 256 Mb RAM.

Jasmine TND (“The New Dimension”): release Jasmine 1.2/9804. Jasmine Studio Version 1.2, Build 1254 is a part of this.

Jasmine ii:            Jasmine ii Base 2.0.  
                          Jasmine ii ODB Option

No release number for any Jasmine application was found in any of the files on the CD-ROMs.

When installing Jasmine ii on a Unix platform:

- a SUN SPARCstation 20 is required, a SUN SPARC Ultra 1 is recommended;
- 128 Mb RAM (256 Mb recommended);
- the amount of hard disk space can differ, but the applications requires 30 Mb, to function properly much more; (For Jasmine TND 300 Mb was required and 400 Mb recommended);
- Unix 2.5 or higher;

A Unix version was not available for this thesis.

## 4.6 Summary

In the last two chapters, we introduced the object oriented paradigm, together with its basic ideas and concepts. We saw which database capabilities are necessary for a database system. These two themes together form our definition of an object oriented database system.

We checked the system Jasmine for the basic ideas and concepts of object orientation and for the database capabilities. We concluded that Jasmine meets our definition of an object oriented database system.

In the next chapter we will see an example of an implementation of an object oriented database in Jasmine.

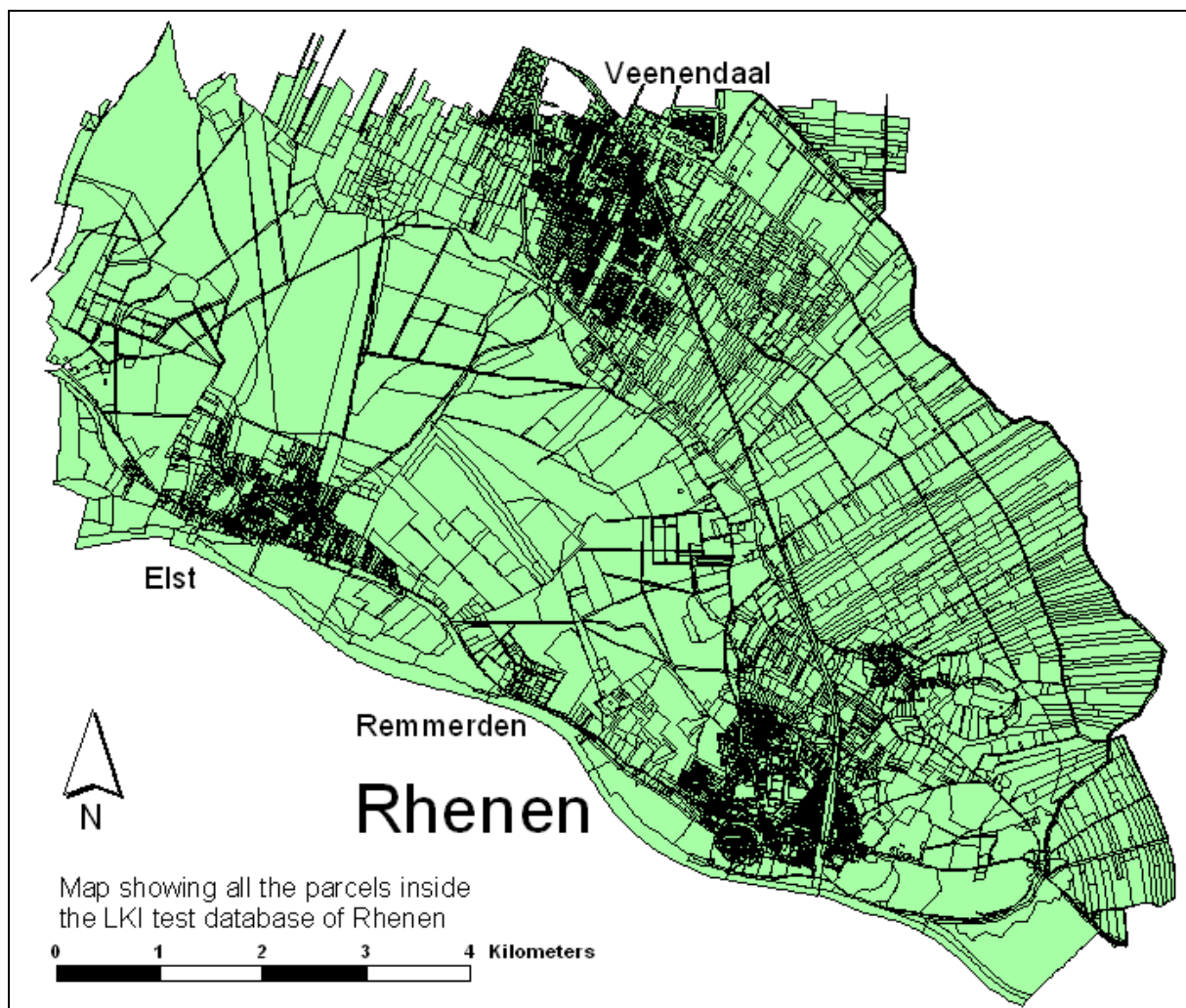
## 5 Implementation of an object oriented database in Jasmine: the LKI

### 5.1 The LKI test database

The LKI (*Landmeetkundig Kartografisch Informatiesysteem*) holds the geometrical and topological data of the Dutch cadastral registration, as described in paragraph 2.3.

Parts from the LKI data are available for testing purposes. One of these test data sets describes the parcels and boundaries around the Grebbeberg, municipality of Rhenen, in the middle of the Netherlands. This includes 16,574 parcels surrounded by 46,072 boundaries. Besides are 25,858 points for symbols and 9,170 points for text strings available.

Figure 5.1 shows a map of these parcels. The map is produced in ArcView after loading the original relational data only converted into the ArcView format.



**Figure 5.14: plot of all the parcels in the LKI test database of Rhenen. (© Kadaster)**

The next paragraph will describe the object oriented data model designed to fit the LKI data. The data of the test data set of Rhenen were converted and loaded into Jasmine. Most of the time on this thesis was spent on this: five out of nine months. The conversion programs written in Perl are given in appendix B.

The counterpart AKR (*Automatische Kadastrale Registratie*), covering the information about the owners of rights on parcels is not taken into account in this thesis. Although its data structure would be very suited for modelling in an object oriented way, an object oriented data model could not be designed nor implemented because the time needed to realise this was not available.

## 5.2 An object oriented data model for the LKI

Of major importance is an object oriented data structure that fulfils the following requirements:

- the geometry, topology and thematics of parcels have to be described completely according the rules for the LKI defined in [Osch, 1997];
- the parcels form a partition: a well defined area has to be covered completely by parcels and the parcels are not allowed to overlap;
- islands of parcels inside other parcels have to be taken into account;
- the first, second and third normal form (see chapter 2) should be respected.

One of the goals of this thesis was to make and test an implementation of a worldwide accepted standard on geographic information. Two standards have been considered:

- The “European prestandard CEN-ENV 12160 – Geographic information – Data description – Spatial schema” published by the Comité Européen de Normalisation (CEN) [CEN, 1997].

- The “CD 19107 Geographic information – Spatial schema” published as draft by the International Standards Organisation (ISO) [ISO, 2000].

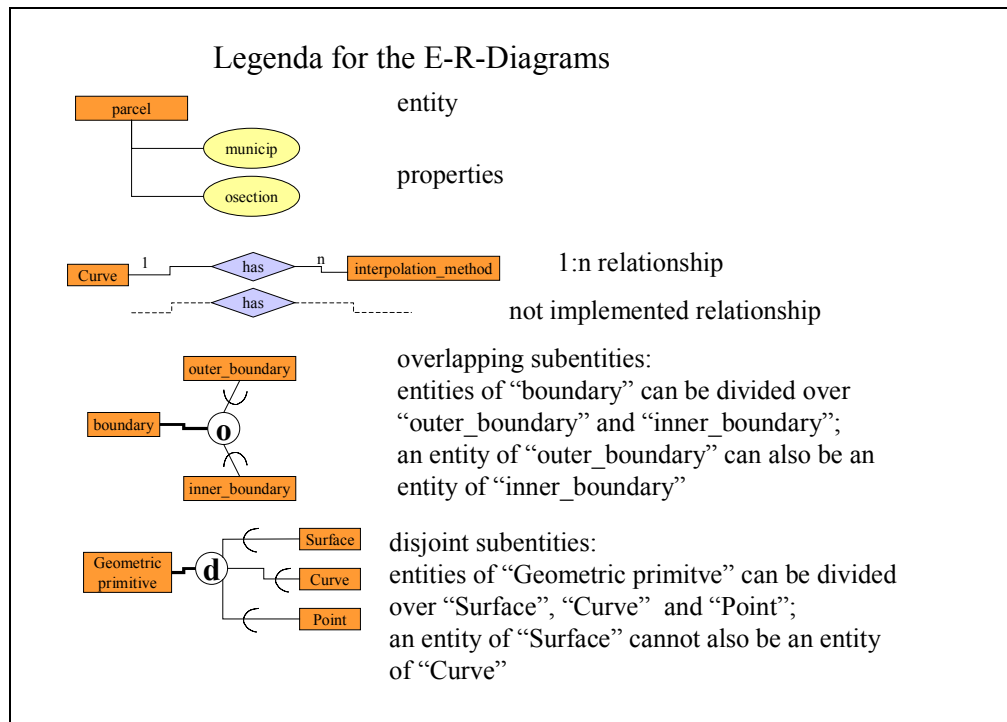
The first one describes a simple, well-defined data model that can be implemented directly, because it includes all the geometric and topologic entities used in the LKI, but is a prestandard.

The latter describes a far more extended model, but was not yet an official standard thus possible subject to change. It covers far more entities to describe geometric and topologic features for example “CubicSpline” or “TriangulatedSurface”. It includes the entities described in the first standard.

Therefore the first one was chosen. ER-diagrams are given in the figures 5.2, 5.3, 5.4 and 5.5.

The data model implemented by van Wijngaarden (1997) is also considered. It includes the geometric and topologic entities needed, includes even a version mechanism but models the edges’ topology without the references to next and previous edges. However, it would be a good alternative to the CEN-ENV 12160.

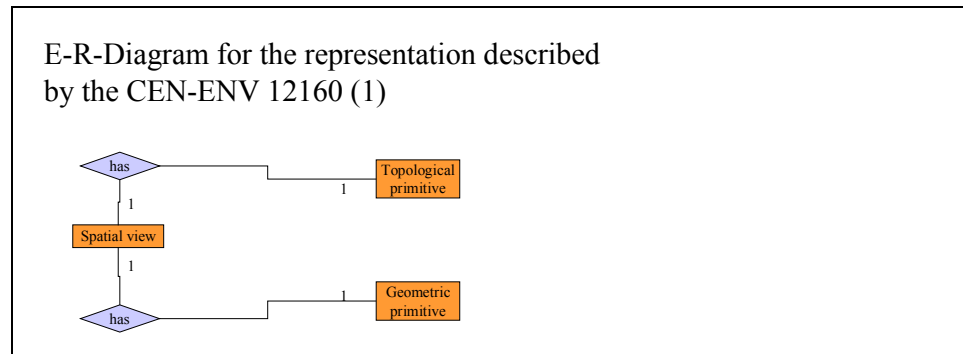
**Figure 5.15:**  
legend for the E-  
R-diagrams in  
this chapter.



The main feature of this model (figure 5.3) is the distinction between geometric and topological data.

The name “Spatial View” is not well chosen because a view refers to the representation of data. This is not the case for the entity “Spatial view”. “Spatial Theme” or “Spatial Layer” would be the better choice. The words “Topological” and “Geometric” are not consequent. “Topological” and “Geometrical” or “Topologic” and “Geometric” would be a better choice.

**Figure 5.16: the root of the CEN model.**



The geometric primitives (figure 5.4) are surface, curve and point. The coordinates in x, y and z are stored as one entity, called “direct\_position”.

The entity “Point” describes all 0-dimensional geometric primitives.

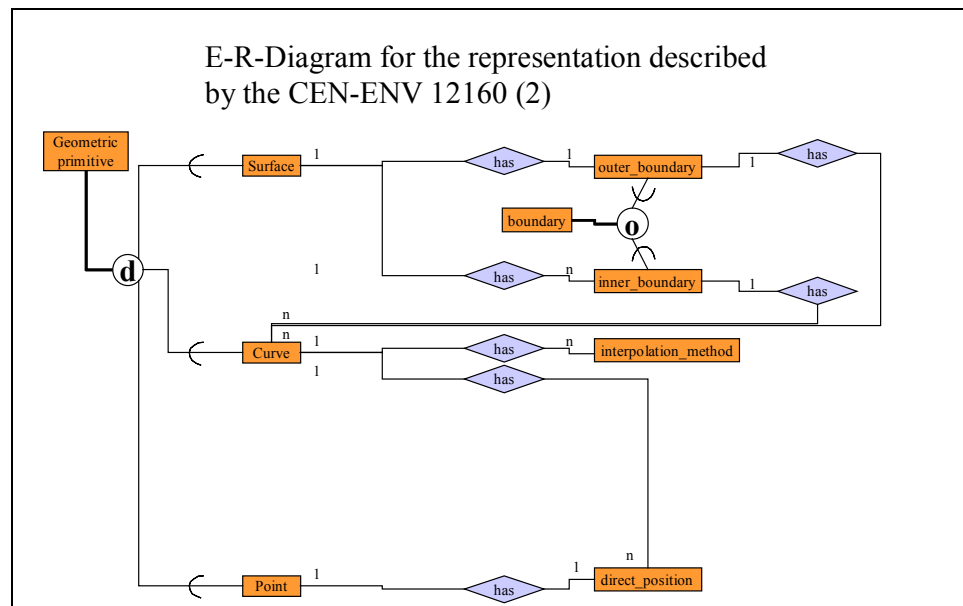
The entity “Curve” describes all 1-dimensional geometric primitives. It keeps a (defined as ordered) list of the “direct\_positions” it contains. The entity “interpolation\_method” indicates whether the curve is a straight line or an arc. The entity “boundary” describes the collections of curves, which surround each surface. It uses a list also. No attention is paid to the direction of the curves. There seems to be redundancy present for the direct\_position that connects two curves.

The entity “Surface” describes all 2-dimensional geometric primitives.

Part of the geometric primitives is a description of a data model for raster data. This part is not used in this thesis and will not be covered here.

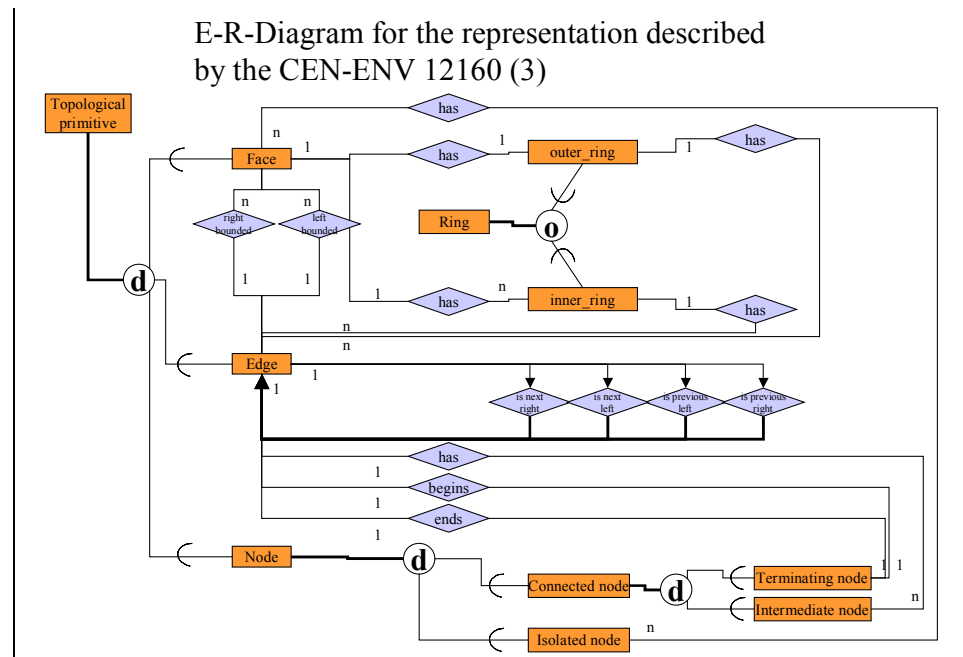
The use of capitals in the entities’ names is taken from figure 33 in [CEN, 1997]. The actual prestandard text mentions no capital use anymore.

**Figure 5.17: the geometric part of the CEN model.**





**Figure 5.18: the topologic part of the CEN model.**



The topologic primitives (figure 5.5) are face, edge and node. They are the topologic “mirror image” of the geometric primitives. They refer also to the geometric primitives. However, the prestandard does not prescribe how this reference should be implemented.

The entity “Node” is decomposed in the entities “ConnectedNode” (connected to each other in an edge) and “IsolatedNode” (without connected neighbours).

The entity “ConnectedNode” is again decomposed in the entities “TerminatingNode” (at the edges’ end) and “IntermediateNode” (between the edges’ ends).

The entity “Edge” forms the central topology. It refers to four neighbouring edges (like in figure 2.3), to the faces left and right of the edge and to the end node and start node.

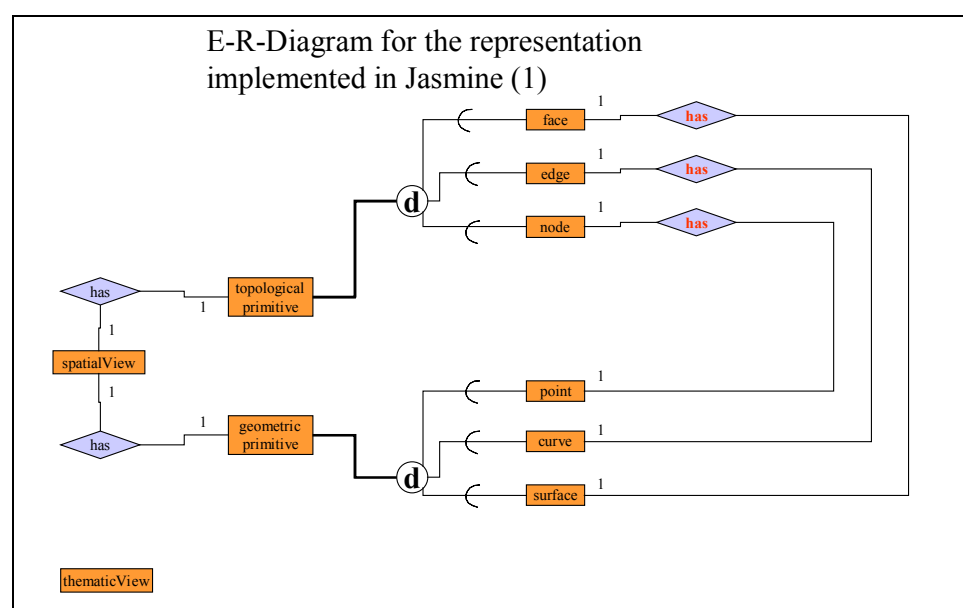
The entity “Face” forms no partition without overlapping faces. Therefore it keeps a n:n relationship to the entity “IsolatedNode”.

The actual implementation of the CEN-ENV 12160 [CEN, 1997] is adapted to fit the requirements of the LKI. Inconsequent capital use has been corrected, incorrect terms not.

First, every topologic primitive will contain a geometric primitive (figure 5.6). The entities “Topologic primitive” and “Geometric primitive” are described in more detail in the figures 5.7 and 5.8.

**Figure 5.19: the root of the implemented model. Compare with figure 5.3.**

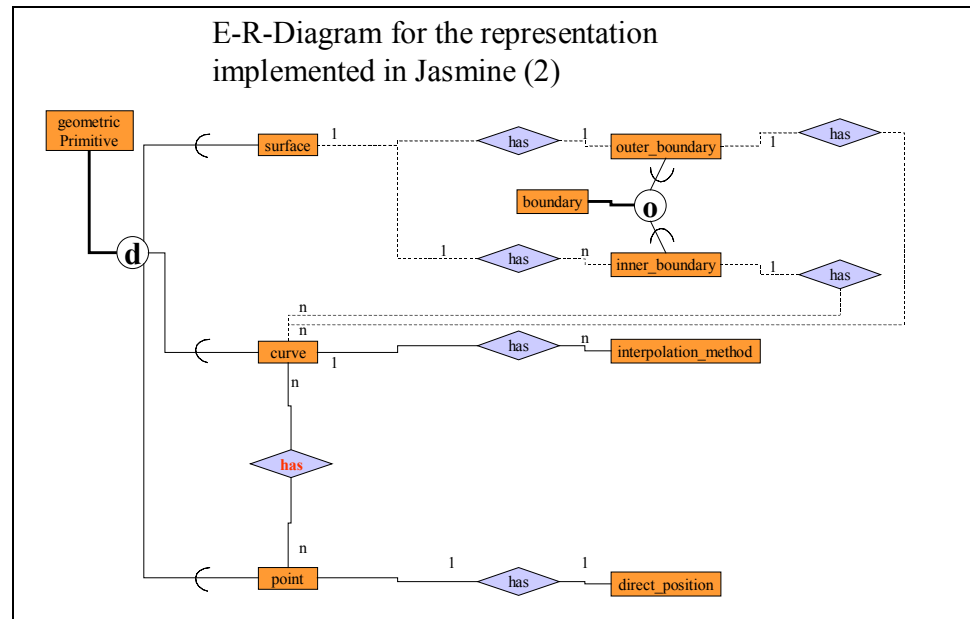
The new entity “Thematic View” will be discussed later.





The geometric part is adapted for the curves (figure 5.7). They will no longer contain direct positions, but they will refer to points. This is according the first normal form (see chapter 2).

**Figure 5.20: the geometric part of the implemented model. Compare with figure 5.4.**

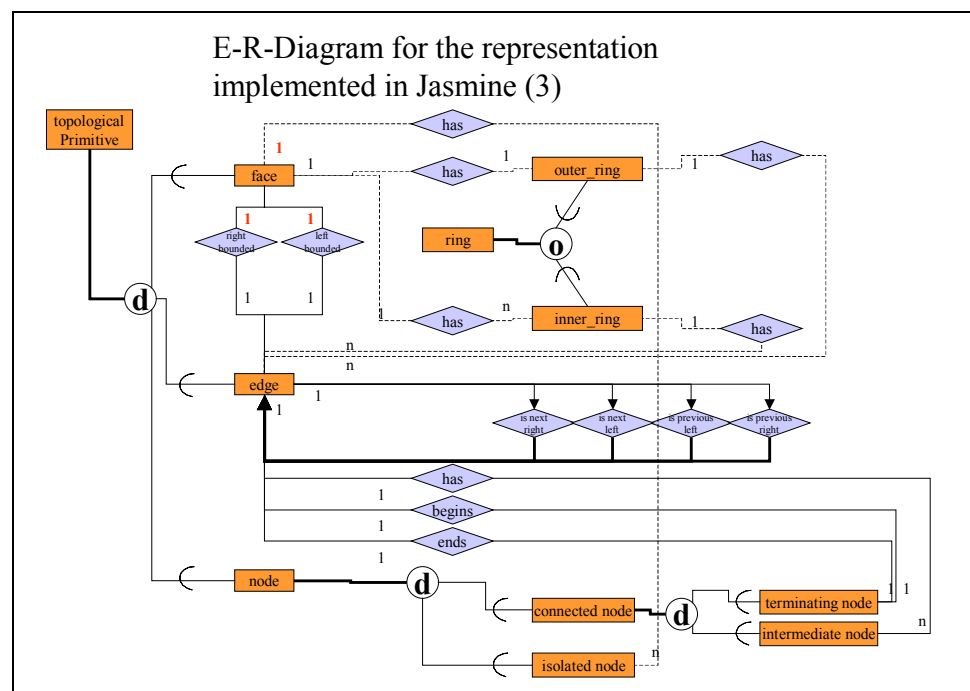


The missing direction is not corrected. This information is actually included in the entity “Edge”: in the properties that describe which face is on the left and which face is on the right and in the properties that describe the start node and the end node (figure 5.8).

Not all the properties have been implemented. The reason is that certain relationships can be maintained by methods instead of attributes. The codes for these methods have been written, but it has not been possible to implement them as real methods. See paragraph 5.4.

The topologic part is also adapted (figure 5.8). In the original model it is possible to have several faces to one side of an edge (figure 5.5). This will be restricted to only one face (figure 5.8) because a boundary may only have one parcel on either side. This is a special implementation of the prestandard CEN-ENV 12160. [CEN, 1997]

**Figure 5.21: the topologic part of the implemented model. Compare with figure 5.5.**



How is the relation between the faces and the coordinates of its boundaries implemented? In figure 5.6 we can see that every face has a surface. From figure 5.8 follows that every surface has an outer boundary. An outer boundary consists of curves. Curves again have references to points. The points finally refer to the coordinates stored in the entity “direct position”. In the actual situation, no data on boundaries are available. Only the data model is implemented. The dashed lines visualize this. The connection from faces to points is made by a method, as we will see further on in this paragraph.

Another relation between the face and the coordinates of its borders could also be possible. From figure 5.8 follows that every face has an outer ring. Every ring consists of edges. From figure 5.6 follows that every edge has a curve. Curves again have references to points. The points finally refer to the coordinates stored in the entity “direct position”. This relation is implemented in the data model, but currently there are no data on rings in the database.

So far, we have a model for the geometric and topologic data. The entities in the ER-diagram can be identified with classes in an object oriented approach. For the thematic data holding classifications, layers and time points, there is no standard available. In the LKI the data are structured as follows (figure 5.9):

**Figure 5.22: the ER-diagram of the LKI. (Compare with figure 2.1.)**

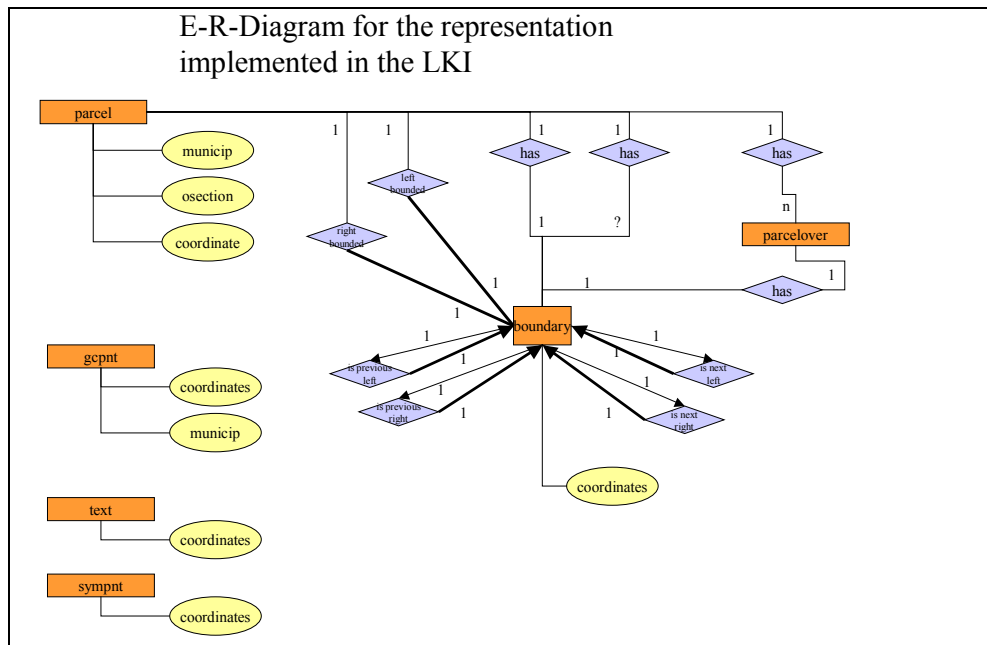


Figure 5.10 shows a possible object oriented data model for the LKI. Instead of a primitive, an entity “landRegistration” is chosen. We could have chosen as well an entity “thematic primitives”, that could contain entities like “landRegistration”, “topographical”, “trafficSignalisation”, “electricCable” etc.

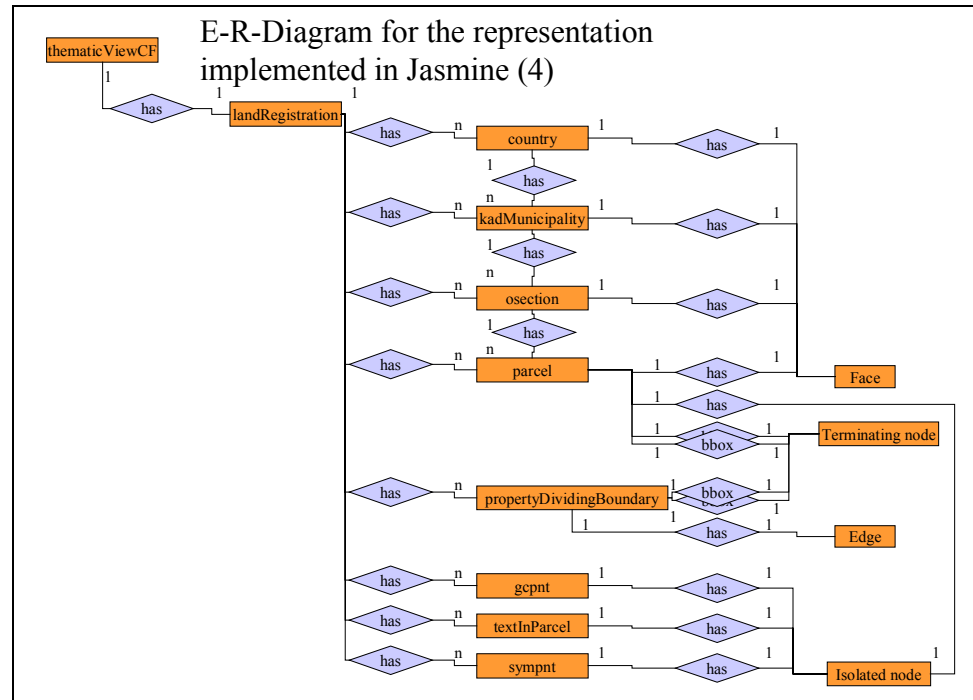
The entity “landRegistration” contains the primitives needed to model a cadastral situation: parcels and boundaries. The latter are called “propertyDividingBoundaries” in order to avoid confusion with the boundaries in the geometric primitives. In Jasmine this would not lead to confusion, because the entities “boundary” and “propertyDividingBoundaries” belong to different class families as we will see further on. In Jasmine class names are only unique within a class family. The combination of the class families’ name and the class’ name makes the class globally unique.

The names for the entities “gcpnt” (base points for measurements), “textInParcel” (in the LKI “text”) and “sympnt” (symbolpoint) stem from the LKI and are maintained in this object oriented model.

A main feature is the hierarchical structure up from the country. The country has cadastral municipalities (in the LKI “municips”, in Jasmine called “kadMunicips”). These are again divided into sections (in the LKI “osections”). The sections are finally divided into parcels.

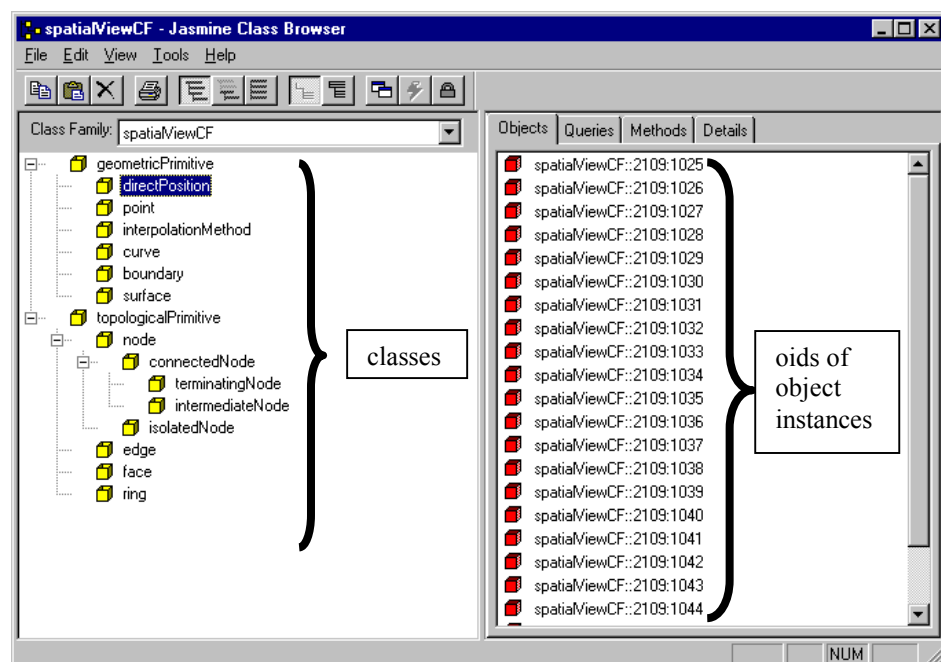
Note that the entities “country”, “kadMunicipality”, “osection” and “parcel” have a face. The entity “propertyDividingBoundaries” has an edge. TerminatingNodes are provided for the bounding boxes of the “parcel” and “edge” entities. IsolatedNodes are available for the entities “gcpnt”, “textInParcel” and “sympnt”.

**Figure 5.23: ER-diagram for an object oriented data model for the thematic information in the LKI.**



Figures 5.11 and 5.12 summarise the implementation of the prestandard CEN-ENV 12160 [CEN, 1997]. A full description of the data structure can be found in appendix C.

**Figure 5.24: the spatialView class family (left). This class family can be used to describe any geometry or topology. To the right the representation of the directPosition objects. They hold all the coordinates in the database.**



The terms spatialView, geometricPrimitive, directPosition and all the other class names stem from the prestandard CEN-ENV 12160 [CEN, 1997].

The classes are defined in a hierarchic way in order to use inheritance:

- The `geometricPrimitive` class holds no instances of objects itself. It is an abstract class. It contains all the classes that describe geometric objects:
  - The `directPosition` class holds all the coordinates in the database in x, y and z.
  - The `point` class holds all zero dimensional geometric objects; they keep each one `directPosition` object.
  - The `interpolationMethod` class can hold objects that define the possible interpolation methods, like `shortestWay` (line) and `circularArc` (curved line). Only `shortestWay` is implemented.
  - The `curve` class holds all one-dimensional geometric objects, they keep each a collection of `Points`. Every curve object has also an `interpolationMethod` object.
  - The `boundary` class holds collections of curves that surround a surface (see later on). The name of this class is confusing, because in the LKI the boundaries are the line strings that separate two parcels. In the object oriented version of LKI the boundaries are replaced by the curves (and by the edges, which we will see later on).
  - The `surface` class holds all two-dimensional geometric objects. They keep each one boundary object as their *outer* boundary. Possibly, they keep also a collection of boundary objects for their *inner* boundaries. In this way, islands are modelled geometrically.
- The `topologicPrimitive` class holds no instances of objects itself and is again an abstract class. It contains all the classes that describe topologic objects:
  - The `node` class holds also no instances of objects itself. But every node object keeps a point object to describe its geometry. It contains all the classes that describe zero dimensional topologic objects:
    - The `connectedNode` class holds also no instances of objects itself. It contains two classes that describe zero dimensional topologic objects which are part of edges (see later on):
      - The `terminatingNode` class holds zero dimensional topologic objects at the end of edges.
      - The `intermediateNode` class holds zero dimensional topologic objects between the ends of edges.
    - The `isolatedNode` class holds zero dimensional topologic objects, which are no part of any edges.
  - The `edge` class holds all one-dimensional topologic objects. They keep each a curve object to describe its geometry. Every edge holds two `terminatingNode` objects: `startsAt` and `EndsAt`; two face objects (see later on): `hasOnItsLeft` and `hasOnItsRight`; four neighbouring edges: `nextLeft`, `nextRight`, `previousLeft` and `previousRight`.
  - The `ring` class holds collections of edges that surround a face.
  - The `face` class holds all two-dimensional topologic objects. They keep each a surface object to describe its geometry. They keep each one ring object as their *outer* ring. Possibly, they keep also a collection of ring objects for their *inner* rings. In this way, islands are modelled topologically.

`IsolatedNodes` are not allowed on edges. When an `isolatedNode` appears to be on an edge, it should become an `intermediateNode`.

The model according CEN-ENV 12160 [CEN, 1997] was adapted on the following points:

- the curves contain points instead of `directPositions`, to avoid double storage of `directPositions` at the curve's ends.
- an edge has only one face to its left and only one face to its right, instead of sets of faces to its left and to its right. Of course actually more faces are present to either side, but they belong to another partition. The osection's face, the kadMunicipalities' face and the countries' face (see figure 5.10). But it's more important to force the system to avoid possible overlappings of parcels, then to have easier ways of selecting the edges of the face of an osection.

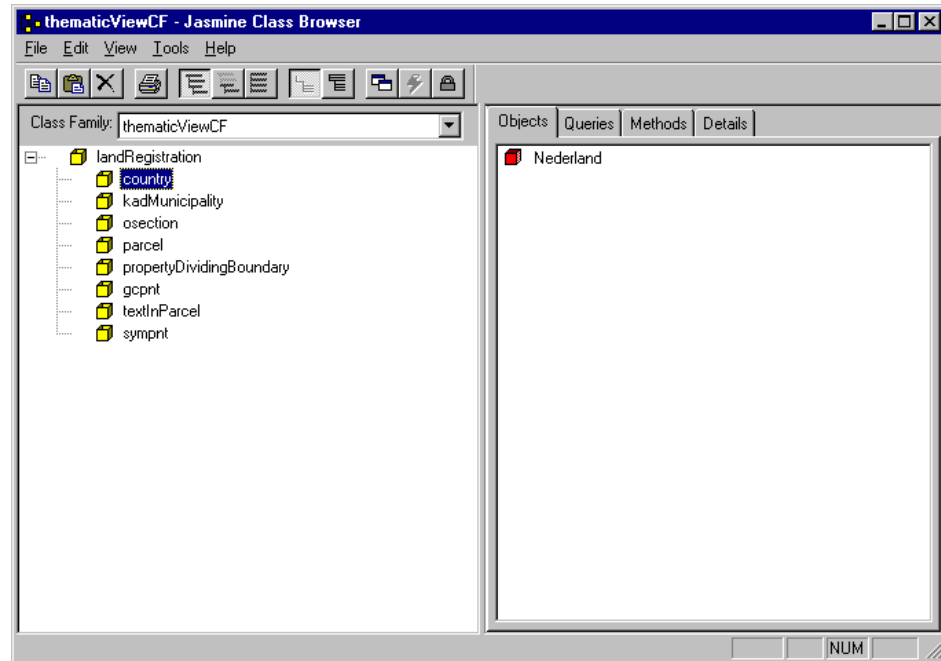
Sometimes a method can combine data from the database instead of storing extra data. Certain relationships can be maintained by methods instead of attributes. The codes for these methods have been written, but it has not been possible to implement them as real methods. See paragraph 5.4.

Such methods are foreseen in the following cases:

- in the `surface` class to find the boundary objects;
- in the `boundary` class to collect the curve objects for the boundary objects;
- in the `intermediateNode` class to find the edge object the `intermediateNode` object belongs to;
- in the `isolatedNode` class to find the face object the `isolatedNode` object belongs to;
- in the `face` class to find the ring objects;
- in the `ring` class to collect the edge objects for the ring objects.

**Figure 5.25: the thematicView class family (left). This class family describes the thematics of the Dutch cadastral registration.**

**To the right the representation of a country object. The country holds all the cadastral municipalities.**



The term thematicView does not belong to any standard anymore. This is the thematic counterpart of the spatial information in the database.

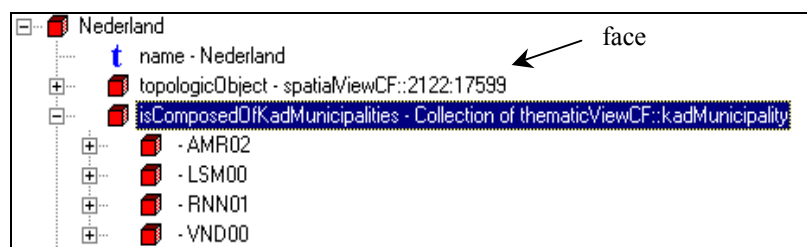
The classes within the thematicView class family are defined as follows:

- The landRegistration class holds no instances of objects itself. It contains all the classes that describe objects containing thematic information on spatial objects:
  - The country class describes countries of which land registration data are available. Every object keeps a face object describing the whole country topologically in the property “topologicObject” (figure 5.13).
  - The country object “Nederland” (figures 5.13 and 5.14) is divided into cadastral municipalities. Therefore, a class kadMunicipality (figure 5.12) holds all the objects describing them. Every object keeps a face object describing the whole municipality topologically.

**Figure 5.26: the object “Nederland”.**

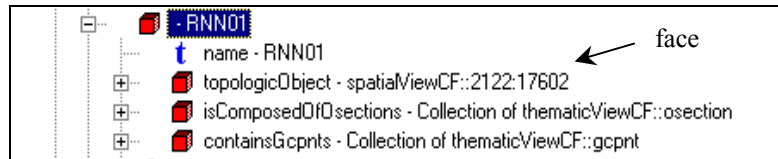


**Figure 5.27: the object “Nederland” divided into kadMunicipality objects.**

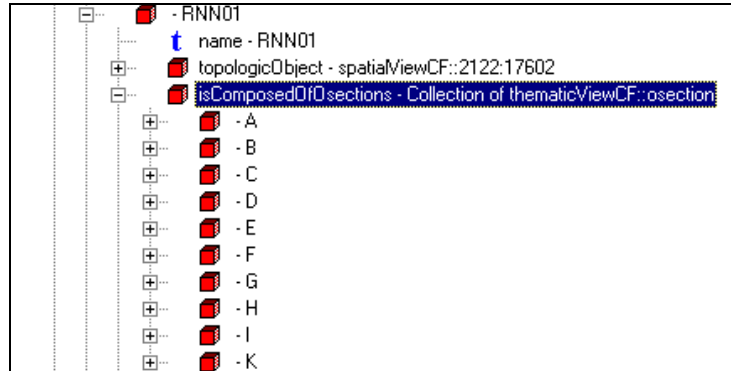


- The kadMunicipality-objects (figure 5.15) are divided into sections (figure 5.16). In the LKI the term “osection” is used. Therefore a class osection (figure 5.12) holds all the objects describing them. Every object keeps a face object describing the whole osection topologically.

**Figure 5.28: a kadMunicipality object.**

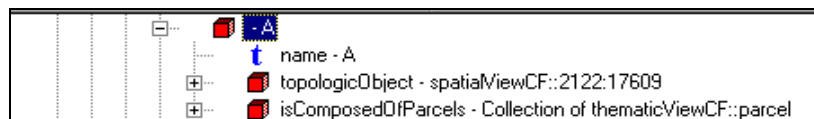


**Figure 5.29: kadMunicipality objects contain osection objects.**

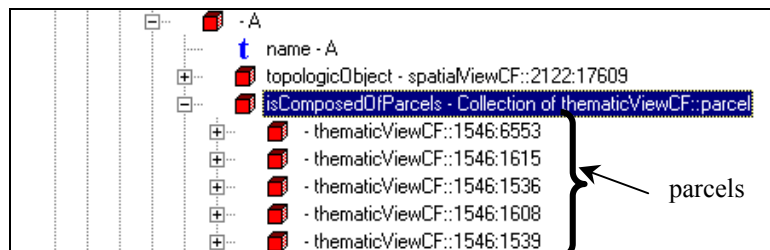


- The osections (figure 5.17) are divided into parcels (figure 5.18). Therefore a class parcel (figures 5.12 and 5.19) holds all the objects describing them.

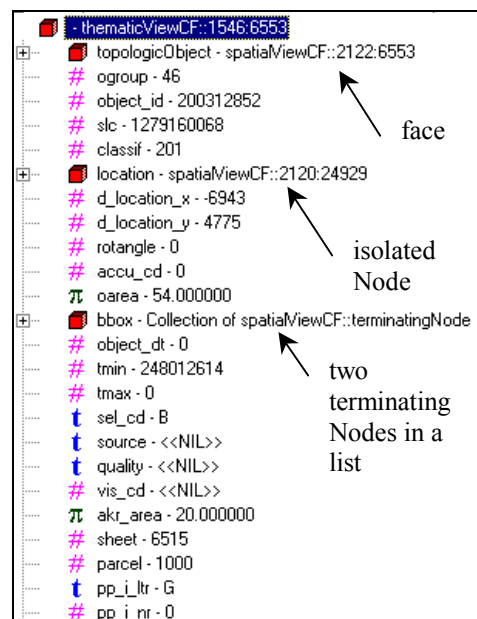
**Figure 5.30: an osection object.**



**Figure 5.31: section objects contain parcel objects.**

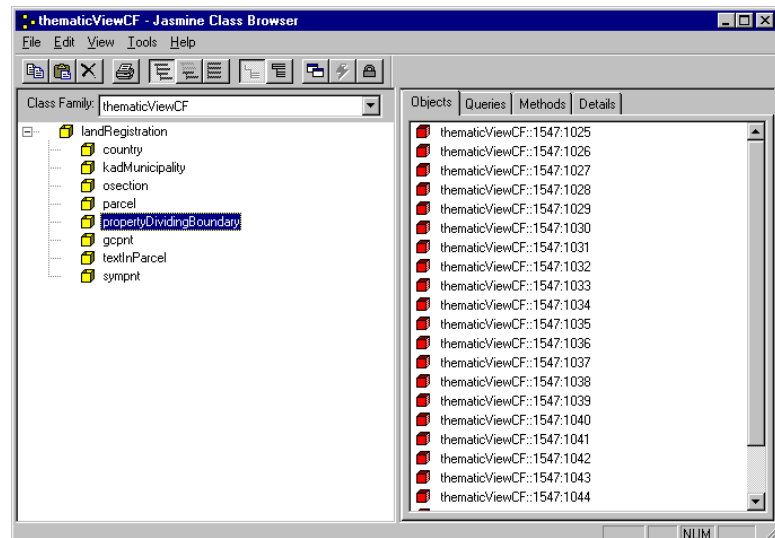


**Figure 5.32: a parcel object. Every parcel object keeps a face object describing the whole parcel topologically in the property “topologicObject”. Further, a bounding box is available described by two terminatingNode objects and a centroid described by an isolatedNode object.**

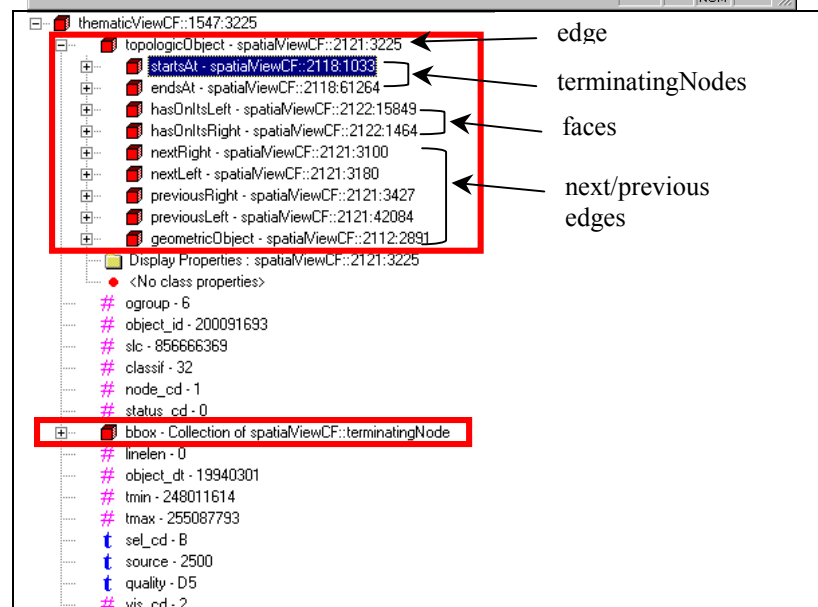


- Every parcel object is surrounded by propertyDividing-Boundaries (figures 5.20, 5.21 and 5.22).

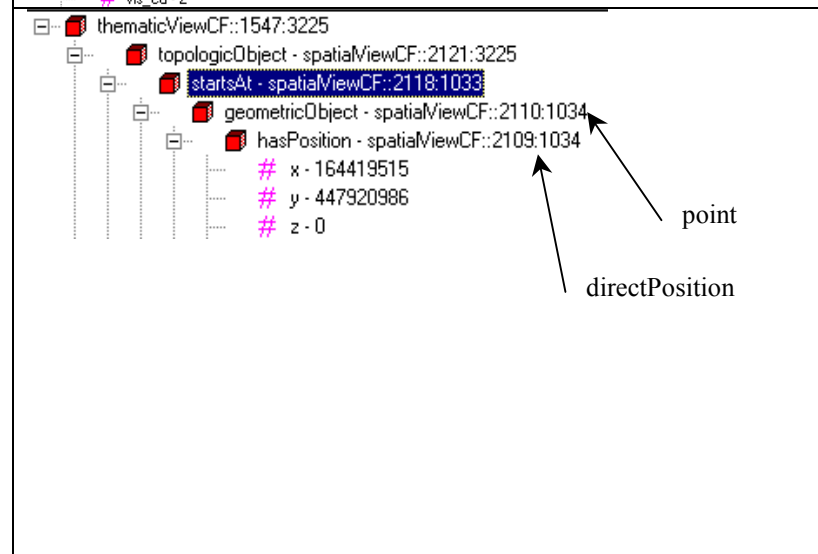
**Figure 5.33:**  
propertyDividingBoundary  
objects.



**Figure 5.34:** a  
propertyDividingBoundary object.  
Every object keeps an edge object  
describing the  
propertyDividingBoundary  
topologically (upper rectangle).  
Further a bounding box is  
available described by two  
terminatingNode objects (lower  
rectangle).



**Figure 5.35:** the hierarchy of  
objects: a  
propertyDividingBoundary  
(object id 1547::3225)  
contains an edge :  
object id 2121::3225.  
This edge contains a  
terminatingNode in the property  
“startsAt”:  
object id 2118::1033.  
This terminatingNode contains a  
point to describe its geometry:  
object id 2110::1034.  
This point contains a  
directPosition:  
object id 2109::1034.  
This directPosition has coordinates  
in x, y and z.



- The class `gcpt` describes base points for surveying measurements. Every object keeps an `isolatedNode` object describing the position of the `gcpt` topologically.
- The class `textInParcel` describes text strings. Every object keeps an `isolatedNode` object describing the position of the `textInParcel` topologically.
- The class `sympnt` describes symbols. Every object keeps an `isolatedNode` object describing the position of the `sympnt` topologically.

Not all methods of the classes have been implemented yet:

- in the surface class to find the boundaries;
- in the boundary class to find the curves;
- in the `intermediateNode` class to find the corresponding edge which any `intermediateNode` is part of;
- in the `isolatedNode` class to find the face surrounding any `isolatedNode`.

Not all the data are available yet:

- the surfaces for the country, `kadMunicipality` and `osection` classes have no surrounding curves;
- the list of `gcpt` objects in the `kadMunicipality` objects is not present.

The model implemented contains also an error:

- an `isolatedNode` can only be within one single face, instead of a set of faces. But it can be within a parcel's face, within an `osection`'s face, within an `kadMunicipalities`' face and of course within a countries' face.

The missing information in the direction of edges causes complications, as we will see in paragraph 5.5. These complications can be a reason to change the data model.

### 5.3 *How to handle objects that change*

Owners of rights can change the topological situation of parcels. For example a parcel can be split up. A new parcel emerges together with some new boundaries. How are these changes recorded in the database?

In the relational LKI system a new tuple is created in the table `xfio_parcel` each time something changes in the situation of a parcel. The property "tmax" is set to zero and the property "tmax" of the older tuple is set to the current database time.

How are these changes recorded in an object oriented database?

- In the `Parcel` class a new object is created.
- The old object describing the parcel gets a value  $> 0$  for the property "tmax".

Also new objects are created to describe new boundaries in the `Boundaries` class of the `thematicView` class family, in the edge and curve classes and in the boundary class of the `spatialView` class family.

In the classes of the `spatialView` class family no property "tmax" is available. This is logical, because this class family describes only geometry and topology, no time or thematics. The CEN-ENV 12160 [CEN, 1997] describes no data models for temporal data.

But what happens to the face and surface classes?

Two alternatives:

1. We can simply create two new objects in each of these classes.
2. We can create just one new object in each of these classes and keep the old one unchanged.

The first possibility seems to be according to the policy practised in the LKI. The second possibility not. The old parcel changes, its geometry and topology changes. The face and surface objects change also, because their outer ring and outer boundary change.

Less storage space is an advantage of the second possibility. The data model becomes however less logical. Historical situations are not reconstructable anymore; this means versioning is not possible. Therefore the first alternative is the better one.

The best solution would be to create a generic temporal class from which the temporal information is derived. An object oriented system would be very suited for this. Van Wijngaarden (1997) implemented this already. However, this is outside the scope of this thesis.

During the conversion of the LKI data the historical data are also converted. This means there are faces and surfaces available for valid parcels, but also for the invalid (historical) parcels.



## 5.4 Methods on the LKI object oriented database

This paragraph introduces some operations implemented for the LKI test database. The implementations are not complete but serve to test the capabilities of Jasmine to handle spatial operations. Because of the compilation problems (see paragraphs 4.2 and 4.4) these operations are *not* implemented as real methods. The operations are therefore written as ODQL scripts, tested and performed using the ODQL Interpreter (paragraph 4.4). This means the operations will perform slower. Unfortunately the ODQL Interpreter accepts no parameters, allows no function calls and returns no objects. To change the scripts into methods for future applications, the following adaptations have to be made:

- replacement of the built in default value for the parameter by a value given by the system or the user;
- replacement of pieces of code by messages to invoke another method;
- replacement of the output to the screen by a return statement.

We will cover in this paragraph the implementations of the “point in polygon” operation and the “calculate area” operation. First, the “find boundaries from parcel” operation is presented. The code of the methods covers several pages and is included in appendix D.

### The “find boundaries from parcel” operation

The method “findBoundariesFromParcel” serves to find the edges surrounding any parcel in the right order. In fact the method tries to reconstruct an outer ring. This makes the storage of ring objects avoidable, and thus the storage of extra data. Of course, it is always possible to change this situation. The data structures for storing rings is available. The method is not able to handle inner rings.

Currently the method contains a value taken from the LKI attribute “object\_id” to replace the parameter. Naturally the method should accept an object instance of class parcel as parameter. The method outputs to the screen a list containing the parcel’s edges sorted in the right order. This should be changed into a return statement.

The method “findBoundariesFromParcel” needs 10 seconds to find a parcel and its surrounding edges and reconstruct the correct order of the edges from the edge’s topology.

The method’s code is included in appendix D.5.

### The “point in polygon” operation

The method was written using the algorithm developed by Prof. Alfred Schmitt [Kreylos, 2000]. This algorithm uses a horizontal line starting at the given point. The amount of crossings with the surrounding polygon is counted. If this amount is odd, the point is inside the polygon. In this way the method can handle holes (island parcels).

The method written uses a directPosition as point and a parcel as polygon. First all bounding boxes of the parcels are checked and only parcels whose bounding box contains the given directPosition are selected. This group of parcels is checked following Schmitt’s algorithm.

The method should use any description of a 2D-point as parameter. Currently this is a new directPosition, but this can be changed into a node, a point or simply two integers. The method outputs an object instance of the class “parcel” to the screen. This should become the return value.

The method “findParcelFromDirectPosition” needs approximately 1 minute and 45 seconds to complete its task. Most of the time (1 minute and 15 seconds) is needed to check the bounding boxes of the 16,574 parcels in the test data, because no spatial indexing is used on them.

The method’s code is included in appendix D.8

### The “calculate area” operation

Inside the LKI database it is not possible to calculate the area of a parcel. The reason is that SQL cannot recursively run the same query to find the next boundary from the last one found. The amount of boundaries is unknown at forehand.

The method uses the code of the method “findBoundariesFromParcel”. This should be replaced by a message to invoke this method. Not that this method cannot handle islands. This method is not needed for the actual area calculation, because a two-point algorithm is used (see paragraph 2.6). This method is only needed to find out what direction the edges have.

The data model implemented offers no explicit information on the direction of the edges: clockwise or anti clockwise. The method compares therefore the current edge's terminatingNode in the property "EndsAt" with the next edge's terminatingNode in the property "startsAt". If these are the same, the next edge has the same direction as the current edge. The points of the edge's underlying curve are used from start to end. If these are not the same, the points of the edge's curve are used in the opposite direction. Remains the problem that the direction of the first edge is unknown. This is solved by checking the sign of the area at the end of the calculation. This inelegant solution can be a reason to change the data model.

Currently the method contains a value taken from the LKI attribute "object\_id" to replace the parameter. Naturally the method should accept an object instance of class parcel as parameter. The method outputs to the screen the calculated area. This should be changed into a return value.

The code of the method "calculateAreaOfParcel" implemented in ODQL delivers a result after 45 seconds the first time. A second run delivered the same result after again 45 seconds. We will see in the next chapter that repeated runs could differ in needed time.

The method pays no attention to the "interpolationMethod" object. Arcs are thus treated as lines. This should be changed.

The method's code is included in appendix D.1.

## 5.5 Summary

In this chapter the object oriented data model for the LKI test database is described and we discussed its current state of implementation. We discussed how parcels and boundaries are modelled using a hierarchy: a parcel has a face describing its topology, a face has a surface describing its geometry, a surface is surrounded by a boundary, a boundary is composed of curves, curves contain points and points finally have directPositions holding the coordinates. We discussed also the hierarchic structure from parcels, sections, and cadastral municipalities up to the country, which is a better conceptualisation than storing references to these entities with every parcel or boundary. However, this data model is not optimal.

We also covered the implemented operations. The "point in polygon" operation shows us that Jasmine is capable of handling spatial operations. Of major importance is the developed "calculate area" operation, because area calculations are not possible within the relational data model of the LKI. The implemented object oriented model makes area calculations possible. This is a great advantage of an object oriented database system.

## 6 Comparison

In this chapter the tests are documented that were performed to compare the object oriented database in Jasmine with a relational database holding the same data. This is a relational database of type MS-Access version 2000.

The used SQL scripts to perform the tests in MS-Access are given in their original form. The scripts were run using the query editor inside MS-Access.

The used ODQL scripts to perform the tests in Jasmine are given in a shortened version. The complete scripts can be found in appendix D. The scripts were run using the ODQL Interpreter. Again the ODQL code has to be interpreted, but the SQL code also.

The tests are performed on a WindowsNT 4.0 machine with 256 RAM in single user modus. Every test consists of minimal two executions to take hot/cold effects into account. Hot/cold effects refer here to the possibility that the second execution takes less time because the system “remembered” information from the first execution. Tests are repeated at least on two different days to check the results.

The processes that started automatically during start up were not stopped. Their consumption of memory and processor time kept stable and it is assumed they did not influence the test results.

These tests have been performed using different sizes for the workStore (default 20480 Kb), for the transactionStore (default 10240 Kb) (together called the workSize parameter) specified in the client environment file, stored in the env-directory inside the Jasmine directory. “A store is a physical container for stored objects. Each store is implemented using files that are managed by Jasmine.[...] The work store is used to store temporary objects that outlive a transaction. The transaction store is used to store collections that exist within a transaction.” (“Database Developer’s Reference”)

These tests have been performed also using different sizes for the bufferSize parameter (default 1024 Kb). “The bufferSize client environment parameter specifies the size of the page cache for the work store and transaction store within the server process. This buffer holds a local cache of pages transferred between the server process and the disk storage where the work and transaction stores reside in the course of a transaction. The parameter is specified in the client environment file used when starting a session.” (“Database Developer’s Reference”)

### 6.1 The database volume test

The amount of storage space needed to store the object oriented structured data are dependent of the used data model and the used database management system.

#### **Comparing relational to object oriented structured data: dumped text files**

The LKI dump data, structured in a relational way, stored as text files need a volume of 82 Mb. They contain enormous amounts of spaces.

When the data of the object oriented LKI test database in Jasmine are dumped into a text file, its volume is 59 Mb.

The amount of spaces in the LKI dump data is so high, that the amount of storage space needed is higher than the amount of storage space needed to store the object oriented structured data. Although the object oriented data contain by far more object instances than there are tuples in the relational data.

The Jasmine database stores its data in stores, which are divided into pages, see figure 6.1.

#### **Comparing relational to object oriented structured data: inside Jasmine**

In the two stores LKISore1 and LKISore2 the original relational LKI data are stored only converted into a format that can be loaded into Jasmine. 7449 and 7429 pages are used. The chosen page size is 8 Kb, which makes 116 Mb.

In the stores DCStore1 and DCStore2 the object oriented LKI test database is stored. 218,181 and 7,788 pages are used. The chosen page size is again 8 Kb, which makes 1,765 Mb.

#### **Comparing relational to object oriented structured data: MS-Access against Jasmine**

The original relational structured data loaded into MS-Access need a storage space of 56 Mb.

The object oriented LKI test database needs 1,765 Mb.

It is clear that Jasmine really needs a lot of storage room, a factor 30 in comparison to MS-Access or its own dump data. It is especially the spatialViewCF that needs the most storage space. This is caused by the classes “directPosition” and “point”.

The only difference between the dumped data in the text files and the data inside Jasmine is the maintenance of the relationships. But these cannot make a difference that large.

The file containing the store DCStore1 was zipped using WinZip 7.0 to estimate the real information content. WinZip compressed the store to 214,864 Kb, which is still a large file.

When the contents of the stores were visualized, the large amount of zeros was striking. Most of the pages are only partially used. Possibly the default page size of 8 kB is not optimal.

```
===== S T O R E   C O N T E N T S =====
Store Name: DCStore1
Class Families:
    spatialViewCF
Locations:
    D:\Jasmineii\Jasmine\data\DCStore1
Page size: 8192
Total pages: 218192
Used pages: 218181

===== S T O R E   C O N T E N T S =====
Store Name: DCStore2
Class Families:
    thematicViewCF
Locations:
    D:\Jasmineii\Jasmine\data\DCStore2
Page size: 8192
Total pages: 32000
Used pages: 7788

===== S T O R E   C O N T E N T S =====
Store Name: DCStore3
Class Families:
    LkiCF
Locations:
    D:\Jasmineii\Jasmine\data\DCStore3
Page size: 8192
Total pages: 32000
Used pages: 8

===== S T O R E   C O N T E N T S =====
Store Name: LKISore1
Class Families:
    LkiCF
Locations:
    D:\Jasmineii\Jasmine\data\LKISore1
Page size: 8192
Total pages: 32000
Used pages: 7449

===== S T O R E   C O N T E N T S =====
Store Name: LKISore2
Class Families:
    LkiCF_2
Locations:
    D:\Jasmineii\Jasmine\data\LKISore2
Page size: 8192
Total pages: 32000
Used pages: 7429
```

Figure 6.1: the result of the “listStore” system command showing the data stores, which class families are contained in them, its page size (in byte), how many pages are available and how many used.

## 6.2 The index test

Jasmine uses B+ trees for indexing, see paragraph 4.2.

### Comparing a query with and without index: MS-Access against Jasmine

One of the most simple queries is: find the parcel with object id XXX.

In MS-Access the parcel is found immediately. The use of an index makes no difference.

The same query implemented in ODQL delivers the parcel after 5 seconds. A second run repeats the result after 4 seconds.

When using an index on the property “object\_id”, the answer comes immediately. A second run makes no difference.

This result shows clearly the effect of an index, is however not surprising.

To create the index on the parcel’s attribute “object\_id”, the following ODQL-script was run:

```
defaultCF 'thematicViewCF';

Bag<Composite class> CCS;
Composite class CC;
CC = parcel.getClass();
CCS = parcel.getSubClasses(TRUE);
CCS = CCS.add(CC);
scan(CCS, CC)
{
    CC.createIndex("object_id", "object_id");
};

undefVar CCS;
undefVar CC;
```

### 6.3 The “find boundaries that have only one next or previous boundary” test

In the LKI every boundary has four attributes that tell to which other four boundaries it is connected: two at the start point, two at the end point. In a certain number of cases the identifiers of the two boundaries at one side are equal. It seems that the topology is incorrect. However, the reason is the limitation of 50 points that can be stored with each boundary in the LKI. When more than 50 points are needed, a next boundary is used, and the attributes at the end point refer both to that next boundary.

#### Find boundaries that have only one next boundary or only one previous boundary

The test data in MS-Access were queried in SQL for this feature. After 10 seconds, a set of 4,718 records was presented. When the same query was run again, the result was displayed almost immediately.

The same question was implemented in ODQL and queried in the test data in Jasmine. After 45 seconds, 4,798 records were counted. The difference in number is explained by records of boundaries at the border of the test area. These records point to boundaries outside the test area, and during the conversion these occurrences were set to “NIL”.

When the same query was run again, the execution time dropped to 27 seconds. A third run did not change the results anymore.

These tests were run using a size of 20480 Kb for the workStore, 10,240 Kb for the transactionStore and a size of 1,024 Kb for the bufferSize parameter.

In order to obtain better performance results these tests were repeated using the maximum values for the three specified parameters. The size of the workStore became 2,097,152 Kb, the size of the transactionStore became also 2,097,152 Kb and the bufferSize became 131,072 Kb.

The query was run again, after restarting of Jasmine. After 28 seconds, the result was displayed. The immediate repetition of the query delivered the result after 25 seconds.

The difference in syntax between SQL and ODQL is of course not surprising. But the difference in length is remarkable. In SQL this query can be formulated like this:

```
SELECT object_id, fl_line_id, fr_line_id, ll_line_id, lr_line_id
FROM Xfio_boundary
WHERE ((fl_line_id = fr_line_id) OR (ll_line_id = lr_line_id));
```

In Jasmine it was not possible to use the query editor for this question, because this editor did not allow the comparison of two attributes from the same class. To solve this problem, a small program was written and interpreted by the ODQL Interpreter:

```
/* ##### declarations */
defaultCF 'thematicViewCF';
List <propertyDividingBoundary> b;
/* ##### find boundaries */

b =   propertyDividingBoundary
from   propertyDividingBoundary
where  (propertyDividingBoundary.topologicObject.nextLeft    ==
        propertyDividingBoundary.topologicObject.nextRight)
or     (propertyDividingBoundary.topologicObject.previousLeft ==
        propertyDividingBoundary.topologicObject.previousRight);

Integer numberOfBoundaries;
numberOfBoundaries = b.count();
numberOfBoundaries.print();

/* ##### clear memory from variables */

undefVar b;
undefVar numberOfBoundaries;

/* ##### end of program */
```

In this program a list is defined to store the results from the query. The number of objects in this list is explicitly counted. In MS-Access this is done automatically. This explains the difference in code length.

The long waiting time is of more importance. The database size can be a reason for this, but also the use of interpreted code instead of compiled code.

We can conclude that enlarging the bufferSize and workSize parameters results in a better performance by speeding up the process. However, the larger sizes result in a longer waiting time before the ODQL Interpreter is ready: 10 seconds instead of only 1 second. The same happens when the application Studio is started. The larger values are especially useful when performing more than one operation. For the following, we use only the maximum settings of these parameters.

## 6.4 The “find boundaries with the same parcel at both sides” test

In the LKI every boundary “knows” which parcel is at his left and which at his right side. For the whole Netherlands there are some erroneous boundaries stored that have the same parcel at both sides.

### Find boundaries with the same parcel at both sides: MS-Access against Jasmine

The test data in MS-Access were queried in SQL for this error. After 7 seconds the result was displayed: no records found. A second run delivered the result almost immediately.

The same question was implemented in ODQL and queried in the test data in Jasmine. After 35 seconds 768 boundaries were found. But this were boundaries with the entry NIL for both the parcels due to its position at the border of the test area. The same result was displayed after the second run taking 25 seconds.

After enlarging the BufferSize and WorkSize parameters these two runtimes remained both on 25 seconds.

In SQL the query was formulated as follows:

```
SELECT object_id
FROM Xfio_boundary
WHERE (l_obj_id = r_obj_id);
```

In ODQL the same result was obtained with:

```
/* ##### declarations */
defaultCF 'thematicViewCF';
List <propertyDividingBoundary> b1;

/* ##### find boundaries */
b1 = propertyDividingBoundary
from propertyDividingBoundary
where propertyDividingBoundary.topologicObject.hasOnItsRight ==
      propertyDividingBoundary.topologicObject.hasOnItsLeft;

String s2;
s2 = "boundary";
s2.print();

b2.count().print();

/* ##### clear memory from variables */
undefVar b1;
undefVar s2;
```

After these two tests we can conclude that Jasmine clearly shows some difference between “cold” and “hot” results due to caching of data in the transactionStore. MS-Access shows also such differences, probably due to caching.

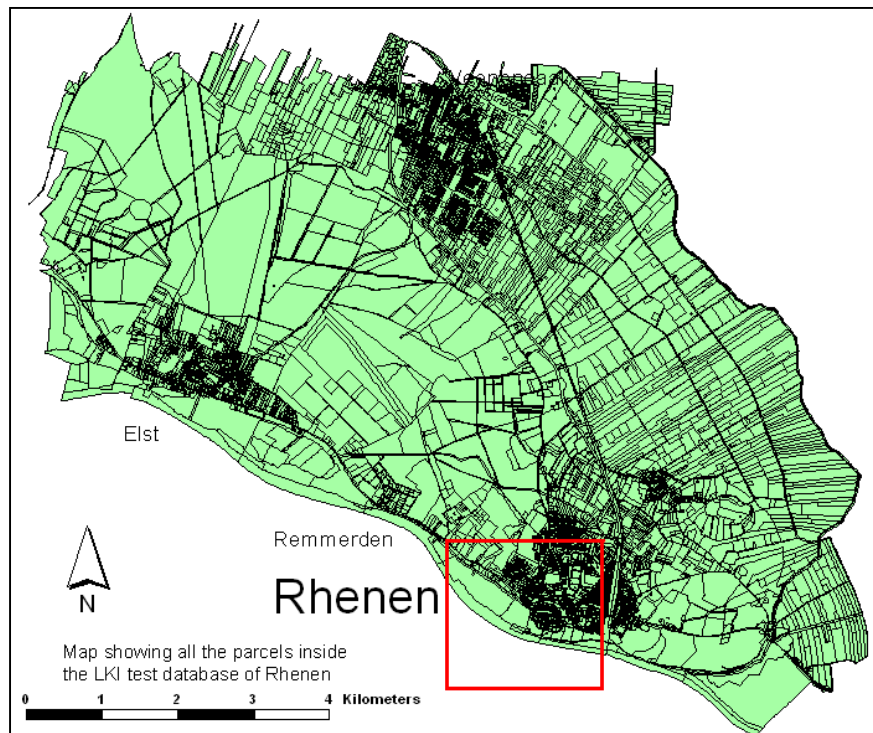
## 6.5 The “find all parcels inside a given rectangle” test

Every GIS-system needs a visualisation tool. One operation often used in visualisation tools is zooming. In figure 6.2 a possible zoom operation is illustrated. We have the map showing all the parcels in the LKI test database. Now we want to zoom in on Rhenen. To perform a zoom operation on a cadastral map, a set of parcels has to be selected in order to show them on the screen.

**Figure 6.2: inside the rectangle the parcels that were selected.**  
(© Kadaster)

The selection can be done on basis of a centroid, a bounding box or the boundaries of the parcels. The last option results in a waste of processor time, and will be left out here. We will show how Jasmine performs when selecting parcels on basis of centroids and bounding boxes.

In the LKI a centroid (“location”) is available for every parcel, implemented by a point stored in one field. In the relational LKI test database inside MS-Access the x- and y-coordinates of the point are stored in separate fields. In the object oriented LKI test database inside Jasmine, this centroid is implemented as an isolatedNode.



### Select parcels in a rectangle by centroid: MS-Access against Jasmine

The test data in MS-Access were queried in SQL for a rectangle of 2 by 2 km using the centroid. Only valid parcels (tmax = 0) are selected. After two seconds, 2,117 records were found and displayed. A second run delivered the same result after one second.

The question implemented in ODQL and queried delivered also a result of 2,117 parcels but after 32 seconds the first time. A second run delivered the same result after 20 seconds.

In SQL the query was formulated as follows:

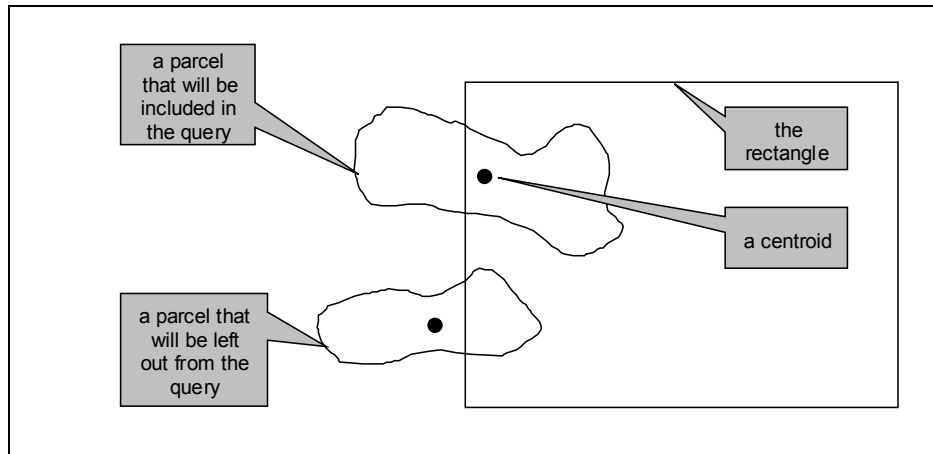
```
SELECT object_id, centroid_x, centroid_y
FROM Parcel_formatted
WHERE (tmax=0)
AND (((centroid_x > 166000000) AND (centroid_y > 440000000)) AND
      ((centroid_x < 168000000) AND (centroid_y < 442000000)));
```

In ODQL the same result was obtained with:

```
defaultCF 'thematicViewCF';
List<parcel> lp;
lp = parcel from parcel
where (parcel.location.geometricObject.hasPosition.x > 166000000
and parcel.location.geometricObject.hasPosition.y > 440000000
and parcel.location.geometricObject.hasPosition.x < 168000000
and parcel.location.geometricObject.hasPosition.y < 442000000)
and parcel.tmax == 0);
lp.count().print();
undefVar lp;
```



When using centroids to obtain all parcels within a certain rectangle not all parcels will be found, because centroids from parcels partly within the rectangle are not taken into account (figure 6.3).



**Figure 6.3: not all parcels will be found when using centroids to select parcels.**

Therefore we will repeat the test, now using the bounding box of every parcel. In the LKI this bounding box is implemented as two points stored in one field. One on the lower left corner, one on the upper right corner. If only one of the two points is inside the rectangle, we will select the parcel. In the relational LKI test database inside MS-Access the two x-coordinates and the two y-coordinates of the points are stored in separate fields. In the object oriented LKI test database inside Jasmine, the bounding box is implemented as a list of two terminatingNodes stored in one attribute.

#### Select parcels in a rectangle by bounding box: MS-Access against Jasmine

The test data in MS-Access were queried in SQL for a rectangle of 2 by 2 km. Only valid parcels ( $t_{max} = 0$ ) are selected. After three seconds 2,185 records were found and displayed. A second run delivered the same result after two seconds.

The question implemented in ODQL and queried delivered a result of 2,185 parcels but after 1 minute and 45 seconds the first time. A second run delivered the same result after 2 minutes 15 seconds.

Note that the second time the query is performed in Jasmine even more time is needed than for the first time.

In SQL the query was formulated as follows:

```
SELECT object_id, bbox_ll_x, bbox_ll_y, bbox_ur_x, bbox_ur_y
FROM Parcel_formatted
WHERE
(
    tmax = 0
    AND
    (
        (
            (
                bbox_x2 >= 168000000
                AND bbox_x1 <= 168000000
            )
            OR
            (
                168000000 >= bbox_x2
                AND 166000000 <= bbox_x2
            )
        )
        AND
        (
            (
                bbox_y2 >= 442000000
                AND bbox_x1 <= 442000000
            )
            OR
            (
                442000000 >= bbox_x2
                AND 440000000 <= bbox_x2
            )
        )
    )
);
```

In ODQL the same result was obtained with:

```

defaultCF 'spatialViewCF';
List<thematicViewCF::parcel> lp;
Integer llx;
Integer lly;
/* ##### define the rectangle */
/* ##### ll = lower left, ur = upper right */
Integer urx;
Integer ury;
llx = 166000000;
lly = 440000000;
urx = 168000000;
ury = 442000000;
/* ##### find all valid parcels */
lp = thematicViewCF::parcel from thematicViewCF::parcel
where (thematicViewCF::parcel.tmax == 0);
Integer i;
Integer numberOfParcels;
i = 0;
numberOfParcels = lp.count();
thematicViewCF::parcel p;
List<terminatingNode> lon;
terminatingNode tn1;
terminatingNode tn2;
List<thematicViewCF::parcel> lp2;
while (i < numberOfParcels)
{
    p = lp.getElementAt(i);
    lon = p.bbox;
    tn1 = lon.getElementAt(0);
    tn2 = lon.getElementAt(1);
    /* if the sides of the bbox overlap with the sides of the rectangle */
    if( ( ( (tn2.geometricObject.hasPosition.x >= urx
              and tn1.geometricObject.hasPosition.x <= urx
            )
          or (urx >= tn2.geometricObject.hasPosition.x
              and llx <= tn2.geometricObject.hasPosition.x
            )
        )
      and ( (tn2.geometricObject.hasPosition.y >= ury
              and tn1.geometricObject.hasPosition.y <= ury
            )
          or (ury >= tn2.geometricObject.hasPosition.y
              and lly <= tn2.geometricObject.hasPosition.y
            )
        )
    )
    {
        /* ##### then add the parcel to the list */
        /* ##### put the first parcel into the list */
        if (lp2.count() == NIL)
        {
            lp2 = List{ p };
        }
        /* ##### and then the other parcels */
        else
        {
            lp2.directAdd( p );
        }
    };
    i = i + 1;
};
lp2.count().print();
/* ##### undefine variable */
undefVar llx;
undefVar lly;
undefVar lp;
undefVar i;
undefVar numberOfParcels;
undefVar p;
undefVar lon;
undefVar tn1;
undefVar tn2;
undefVar lp2;
undefVar urx;
undefVar ury;
/* ##### end of program */

```

Note that this is a different operation. The valid ( $t_{max} == 0$ ) parcels are put into a list and from this list the parcels are taken one by one. The terminatingNodes forming the bounding box, stored in the property “bbox”, are kept in a list. They have to be taken out of that list and stored in a temporary list, called “lon” (“list of nodes”). Then the terminatingNodes are checked for overlap with the rectangle.

The long time needed by Jasmine to perform this operation is clearly not acceptable, when keeping in mind that this operation is part of a zooming operation.

## 6.6 Summary

From the performed tests, we can say that Jasmine needs much more storage room than relational database systems to store its data. One reason is that the pages are only partly filled. Of course, the default page size of the stores can be questioned. Another reason can be the object oriented data model. The data model contains a very deep hierarchy, thus forcing the system to maintain an enormous amount of relations.

The waiting times when performing operations on the LKI test database are unacceptable long. A reason can be that the code is being interpreted, which takes time. A compiled method would perform better. Another reason might be the lack of spatial indexing and spatial clustering. However, MS-Access performs much better without compiled methods and without any efforts to create spatial indexes.

When an operation is performed the second time, the waiting time is *mostly* shorter by some seconds caused by hot/cold effects. A third time makes no difference anymore. In one case (the last test) the waiting time became longer when performing the operation the second time.

Larger sizes of the bufferSize and workSize minimize the hot/cold effects and the waiting times. This is however a trade off, because these larger sizes result in longer waiting times when starting the Jasmine environment.

The code lengths to implement a query in ODQL are often much longer than in SQL, because the declarations and countings have to be stated explicitly.



## 7 Conclusions and recommendations

The conclusions can be divided into two parts: conclusions on the use of an object oriented database (paragraph 7.1) for spatial data, and conclusions on the use of Jasmine for an implementation of an object oriented database (paragraph 7.2). A conclusion gives an answer to a question listed in chapter 1. The recommendations are presented in paragraph 7.3.

### 7.1 *Conclusions on the use of an object oriented database*

1. How is an object oriented database system characterized? In chapter 3 we saw that any object oriented system should support the basic ideas of object, class, message and method. Any object oriented system should also support the concepts of encapsulation, polymorphism, inheritance and object identity.  
Besides, any database system should be capable of holding persistent data, performing transactions, supporting concurrency control, recovery, complex object modelling and querying, supporting versioning, integrity constraints, security and handling performance issues.
2. Is an object oriented database a better solution for conceptualisation and modelling of the real world than a relational database? We can say this is true. Object oriented database systems are a more natural way of describing the real world. Chapter 5 shows how the object oriented data model for the Dutch cadastral system LKI partly resulted in a better conceptualisation and modelling of the geometry and topology of parcels.
3. Can object oriented databases overcome the disadvantages of relational databases? Object oriented databases can overcome some of the disadvantages of relational databases but not all of them. According to the disadvantages listed in paragraph 2.5, the number of tables in the LKI is not very high, but it is replaced by a large number of classes, as we see in chapter 5. This results in still high investments in time and money to create or convert the data (see Appendix A). This is due to the implemented data model and not to the concept of object orientation. Large numbers of JOIN-operations on the cost of memory capacity and processor time are not necessary anymore. It was expected that the inner relations of hierarchical structured objects would result in a better performance. This could not be confirmed: the tests performed and documented in chapter 6 show long reaction times. Recursive querying is very well possible, for example the area calculation operation in paragraph 5.4. The possibilities to hold multimedia data and large objects are not tested in this thesis. The modelling capacities are clearly better according to the former conclusion.  
Besides, object orientation offers strong advantages, especially for spatial databases (paragraph 3.4): the topology structure can be kept correct under transactions by the database management system itself. It is able to perform complex operations, which in the context of relational database management systems are performed by exterior applications. Systems become more powerful and robust. Systems are easier to extend and to maintain than relational systems. This leads to economical advantages.  
However, the investments needed to make it possible to leave a relational system and convert all the data into an object oriented database system depend highly on the size and complexity of the data and can back out the advantages. An extra threat is that no object oriented database system has a serious market share

### 7.2 *Conclusions on the use of Jasmine for an implementation of an object oriented database*

4. Is Jasmine really an object oriented database system? Chapter 4 gives an overview of the basic ideas and concepts implemented and supported by Jasmine. These comprise all the basic ideas and concepts any object oriented system should support according to chapter 3.  
Besides, Jasmine supports also the database capabilities listed in paragraph 3.2. From this point we can say Jasmine is an object oriented database system.
5. What system requirements are needed? The system requirements listed in paragraph 4.5 are high, demand a modern desktop computer equipped with a 256 Mb RAM. The system requirements for Jasmine are much more higher than for relational database systems.

6. How much storage space does a database in Jasmine need? Jasmine demands a very large amount of storage space to store a database. The LKI test database in an object oriented data structure, dumped 59 Mb, requires 1765 Mb inside the Jasmine environment (paragraph 6.1). These data are not efficiently stored because storage space is wasted. Jasmine needs much more storage space for its databases than relational systems.
7. Can Jasmine be used to store and query a dataset describing the geometry and topology of a model of the world's surface? The LKI test database describes a model of the geometry and topology of a part of the Dutch parcels. For this small part of the world's surface the model's data are stored in an object oriented structure. The implemented data model following the CEN-ENV 12160 standard is however not optimal. The terminology needs reconsideration, the amount of classes is high and temporal aspects are not included.

During this thesis no queries proved to be impossible. The query language ODQL proved to be very powerful, easy to use and meets the basic ideas of object orientation. Advantage is taken from the basic idea of inheritance of data definitions and methods to save time. ODQL cannot only be used to create queries. ODQL proved to be a complete programming language and can easily be used to implement methods. The compilation caused however serious problems, as unclear errors were reported that are not present anymore when the same code is run using the ODQL Interpreter. As a result of this, no methods have been implemented.
8. How does Jasmine perform under spatial operations? Jasmine is able to perform the spatial operations implemented and tested in paragraph 5.4 and chapter 6. But the waiting time is higher than acceptable. Performed operations on the test data set resulted in long waiting times in Jasmine, up to 30 seconds and more, where MS-Access needed only a few seconds. Even for simple queries. This is partly caused by the interpreted code, which takes more time to perform than for compiled code. The implemented data model also weakened the performance. Besides, it is expected that spatial indexing and spatial clustering will result in a much better performance. The operations chosen to implement (point in polygon, area calculations) will never give a complete overview of the capabilities of Jasmine performing spatial operations.

The conclusion of this thesis is that Jasmine is an object oriented database system equipped with the necessary standard database capabilities. It is suitable to store and query spatial data, to build a small GIS-system capable of performing the spatial operations implemented and other operations that share the same basic arithmetic and database operations. Object orientated systems overcome the disadvantages of relational systems. Jasmine has however its own limitations to overcome.

## 7.3 Recommendations

### Recommendations for the TU-Delft

The recommendations for the TU-Delft comprise further research themes:

- on a mixture of relational and object oriented techniques, in order to combine the technical advantages of the two and to reduce the investments;
- on other object oriented database systems;
- on the implementation of a better object oriented data model for the LKI;
- on the implementation of an object oriented data model for the AKR;
- on the implementation and testing of methods in Jasmine performing spatial operations e.g. to split a parcel or to combine two parcels;
- on the implementation of integrity rules to check topology;
- on the implementation of a generic time class;
- on the implementation of a viewer to make the spatial data in Jasmine visible. This can culminate in performing the spatial operations in this viewer.

## **Recommendations for CA**

The recommendations for Computer Associates Inc. comprise extensions to Jasmine to make this object oriented database system more suitable to develop GIS-applications:

- implementation of spatial data types, spatial clustering and spatial indexing based on standards or specifications accepted by the CEN, ISO or the OpenGis Consortium;
- implementation of a visualisation tool for spatial data;
- check the standard page size of the stores of 8 kB to find out whether it is really optimal;

Van Wijngaarden (1997) announced that CA worked on spatial data types.

## **Recommendations for the Dutch Cadastre**

The recommendations for the Dutch Cadastre comprise:

- continue the research on object oriented database systems;
- research in cooperation with the TU-Delft;
- do not implement systems based on Jasmine as long as spatial data types, spatial clustering and spatial indexing are not available.

## **Recommendations for the CEN**

The recommendations for the CEN comprise adaptations on the CEN-ENV 12160 prestandard:

- the entity name “Spatial View” should be reconsidered. Proposals: “Spatial Theme” or “Spatial Layer”;
- development of implementation standards for geographic information.





## 8 References

- [Batty, year unknown] Batty, P., "Smallworld GIS: Object-orientation – some objectivity please!" In: *Smallworld technical paper number 7*. Available at: <http://www.smallworld.co.uk/english/products/whitepapers/OOobjectivity.pdf>  
Year of publication unknown. Last time visited: May 25<sup>th</sup> 2001.
- [Booch e.a., 1999] Booch, G.; Rumbaugh, J.; Jacobson, I., *The unified modelling language user guide*. Boston: Addison Wesley Longman, Inc., 1999.
- [Codd, 1970] Codd, E., "A relational model for large shared data banks." In: *Communications of the Association of Computing Machinery*. 1970
- [CEN, 1997] Comité Européen de Normalisation (CEN), "European prestandard ENV 12160 – Geographic information – Data description – Spatial schema". In: *Voornorm NVN-ENV 12160 Geografische informatie – Gegevensbeschrijving – Ruimtelijk schema*. Delft: Nederlands Normalisatie Instituut, 1997.
- [CA, 1999] Computer Associates International, Inc.; FUJITSU LIMITED, "Jasmine TND Database Developer's Reference 2.0."  
"Jasmine TND Database Administrator's Guide 2.0."  
"Jasmine TND System Administrator's Guide 2.0."  
In: *Jasmine*. Developer's edition (Installation CD-ROM) 1996-1999.
- [CA, 2000] "Jasmine ODB Database Developer's Reference 2.0."  
"Jasmine ODB Database Administrator's Guide 2.0."  
"Jasmine ODB System Administrator's Guide 2.0."  
In: *Jasmine ii The eBusiness Platform*. (Installation CD-ROM) 1996-2000.
- [Fowler e.a., 2000] Fowler, M.; Scott, K., *UML distilled: a brief guide to the standard object modelling language*. Second edition. Boston: Addison Wesley Longman, Inc., 2000.
- [INT, 2001] INT Media Group, Inc., *Webopedia*. Available at: <http://webopedia.internet.com/>  
Last time visited: July 11<sup>th</sup> 2001.
- [ISO, 2000] International Standards Organisation (ISO), "Final text of CD 19107 Geographic information – Spatial schema." In: *ISO/TC 211 Geographic information/Geomatics*. Majorstua, Norway: ISO/TC 211, 2000.
- [Ishikawa e.a., 1993] Ishikawa, H., *Object-Oriented Database System*. Tokyo: Springer-Verlag, 1993.
- [Kreylos, 2000] Kreylos, O., *When is a Point Inside a Polygon?* Available at: <http://graphics.cs.ucdavis.edu/~okreylos/TAship/Spring2000/PointInPolygon.html>  
Last time visited: July 11<sup>th</sup> 2001.
- [Kroha, 1993] Kroha, P., "Object and databases." *The McGraw-Hill International Series in Software Engineering*. Berkshire: McGraw-Hill Book Company Europe, 1993.
- [Koshafian e.a., 1999] Khoshafian, S.; Dasananda, S.; Minassian, N., *The Jasmine Object Database. Multimedia Applications for the Web*. San Francisco: Morgan Kaufmann Publishers, Inc, 1999.
- [Laan, 1998] Laan, G., *Aan de slag met C++*. Schoonhoven, the Netherlands: Academic Service, 1998.
- [Newell, 1992] Newell, R. G., "Practical experiences of using object-orientation to implement a GIS." In: *Proceedings of GIS/LIS 1992*. Volume 2, p 624-629.

- [Molenaar, 1989] Molenaar, M, "Een formele gegevensstructuur voor enkelvoudige vectorkaarten." In: *NGT Geodesia*, 1989, pages 392-401.
- [Oosterom, 1997] Oosterom, P. van, "Maintaining Consistent Topology including Historical Data in a Large Spatial Database". *Auto-Carto 13*, April 1997, pages 327-336.
- [Oosterom, 1999] Oosterom, P. van, "Spatial access methods." In: Longley, P.; Goodchild, M.; Maguire, D.; Rhind, D., *Geographical information systems*. Volume 1, second edition. New York: John Wiley & Sons, Inc. 1999. pages 385-400.
- [Osch, 1997] Osch, B. van, *Fysiek Data Model KVS-Informatie-database*. Apeldoorn: Kadaster, 1997.
- [Peng e.a., 1996] Peng, W.; Tempfli, K., "An Object-oriented design for automated database generalisation." In: Kraak, M.J.; Molenaar, M. (eds), *Proceedings of the 7th international symposium on spatial data handling*. Delft: 1996, pages 4B.15-4B.29.
- [Wijngaarden, 1997] Wijngaarden, F.A. van, *Ontwerp en Implementatie van een Kaartintegrator. Verwerking van GBKN-mutaties in de TOP10Vector*. Master thesis. Apeldoorn: Universiteit Twente, 1997.
- [Worboys, 1995] Worboys, M. F., *GIS A Computing Perspective*. London: Taylor & Francis, 1995.

## Appendix A The data conversion

The cadastral data that were used to build up an object oriented database in Jasmine stem from the Dutch Cadastral Office. The data comprehend the cadastral community of Rhenen, a town in the middle of the Netherlands, and were handed to the TU-Delft for testing purposes.

The relevant data for this thesis held by the Dutch Cadastral Office are separated into two systems:

- *Automatische Kadastrale Registratie* (AKR) holding data on subjects, having rights on objects, e.g. ownership.
- *Landmeetkundig Kartografisch Informatiesysteem* (LKI) holding the data on geometry and topology.

These data are organized in a relational structure and divided over several tables.

From the LKI the following tables are used to build up the object oriented database:

- x fio \_ parcel . dat
- x fio \_ boundary . dat
- x fio \_ gcpnt . dat
- x fio \_ text . dat
- x fio \_ sympnt . dat

For a description of these tables, see section 1.3.

These datasets are converted by use of perl scripts running under a UNIX operating system. Perl (Practical Extraction and Report Language) is very suitable for handling large volumes of data that have to be reformatted. The following list of perl scripts were written, listed with their respective in- and output files:

inputfile	perlscript	outputfile
xfio_boundary.dat xfio_parcel.dat osection_step1.txt	1 boundary_shape_kad2jas.pl 2 osection_kad2jas.pl 3 createOsectionsFile2.pl	boundary_formatted_step1.txt osection_step1.txt municips.txt osections.txt faces_osection.txt surfaces_osection.txt
xfio_gcpnt.dat	4 gcpnts_kad8jas.pl	gcpnt_formatted.txt directPositions.txt
xfio_text.dat directPositions.txt	5 text_kad8jas.pl	text_formatted.txt directPositions.txt
xfio_sympnt.dat directPositions.txt	6 sympnt_kad8jas.pl	sympnt_formatted.txt directPositions.txt
xfio_parcel.dat directPositions.txt	7 parcel_kad8jas.pl	parcel_formatted.txt directPositions.txt
xfio_boundary.dat directPositions.txt	8 boundary_bbox_kad8jas.pl	boundary_formatted.txt directPositions.txt
parcel_formatted.txt points.txt	9 findNodes_parcel.pl	terminatingNodes.txt points.txt isolatedNodes.txt
boundary_formatted.txt points.txt	10 findNodes_boundary.pl	terminatingNodes.txt points.txt
gcpnt_formatted.txt points.txt	11 findNodes_gcpnt.pl	isolatedNodes.txt points.txt
text_formatted.txt points.txt	12 findNodes_text.pl	isolatedNodes.txt points.txt
sympnt_formatted.txt points.txt	13 findNodes_sympnt.pl	isolatedNodes.txt points.txt
directPositions.txt  terminatingNodes.txt boundary_formatted_step1.txt	14 createEdgeTopology6step1.pl	directPositions.txt points.txt intermediateNodes_shape_bou.txt terminatingNodes.txt curves.txt edges_step1.txt
parcel_formatted.txt	15 createFaceTopology1step1.pl	faces_step1.txt surfaces_parcel.txt
edges_step1.txt faces_step1.txt	16 createEdgeTopology3step2.pl	edges.txt
faces_step1.txt	17 createFaceTopology1step2.pl	faces_parcel.txt
faces_parcel.txt faces_osection.txt	18 createFacesFile.pl	faces.txt
surfaces_parcel.txt surfaces_osection.txt	19 createSurfacesFile.pl	surfaces.txt
directPositions.txt points.txt curves.txt surfaces.txt terminatingNodes.txt intermediateNodes_shape_bou.txt isolatedNodes.txt edges.txt faces.txt parcel_formatted.txt boundary_formatted.txt gcpnt_formatted.txt text_formatted.txt sympnt_formatted.txt lkiodb09.txt	20 createJasmineUnloadFile.pl 21 createTemplateJasmineUnloadFile.pl	lkiodb12.txt lkiodb10.txt

The numbers in front of the perl scripts refer to their corresponding paragraph in appendix B where the source code can be found.

	perlscript	purpose	shared characteristics
1	boundary_shape_kad2jas.pl	Partly converting the entity xfo_boundaries from the LKI format into the Jasmine format concerning the coordinates of the attribute "shape".	<p>Removing the coordinates and storing them into a separate file. This files contains the data for the class directPosition (see Appendix C). The object identifiers of the directPositions are left in the converted data set. One type of coordinates is only removed and not stored into a separate file. These concern the coordinates of the attribute "shape" in the entity xfo_boundary.dat. All the converted objects get a Jasmine object id</p> <p>From the object identifiers of the directPositions, create the data for the classes point, terminatingNodes and IsolatedNodes.</p>
2	osection_kad2jas.pl	Partly converting the entity xfo_parcel from the LKI format into the Jasmine format: only the information on cadastral municipalities, sections and parcel numbers.	
3	createOsectionsFile2.pl	Creation of the hierarchy of the parcels, sections and cadastral municipalities up to the country. Creation of the surfaces and faces for these entities but not for the entity "parcel".	
4	gcpnts_kad8jas.pl	Converting the entity xfo_gcpnts from the LKI format into the Jasmine format.	
5	text_kad8jas.pl	Converting the entity xfo_text from the LKI format into the Jasmine format.	
6	sympnt_kad8jas.pl	Converting the entity xfo_sympnt from the LKI format into the Jasmine format.	
7	parcel_kad8jas.pl	Converting the entity xfo_parcel from the LKI format into the Jasmine format.	
8	boundary_bbox_kad8jas.pl	Converting the entity xfo_boundary from the LKI format into the Jasmine format.	
9	findNodes_parcel.pl	Create the Nodes and Points for the entity "parcel".	
10	findNodes_boundary.pl	Create the Nodes and Points for the entity "propertyDividingBoundary".	
11	findNodes_gcpnt.pl	Create the Nodes and Points for the entity "gcpnt".	
12	findNodes_text.pl	Create the Nodes and Points for the entity "text".	
13	findNodes_sympnt.pl	Create the Nodes and Points for the entity "sympnt".	
14	createEdgeTopology6step1.pl	Removing the coordinates of the attribute "shape" from the entity xfo_boundary.dat and storing them into a separate file. This file contains the data for the class directPosition (see Appendix C). The object identifiers of the directPositions are left in the converted data set. Creation of the curves and the edges. The edges keep the LKI identifiers for the faces.	
15	createFaceTopology1step1.pl	Creation of the surfaces and faces for the entity "parcel". The faces keep also the LKI identifier.	
16	createEdgeTopology3step2.pl	Completing the edges: replacement of the LKI identifiers for the faces by the new Jasmine object identifiers.	
17	createFaceTopology1step2.pl	Completing the faces : removal of the LKI identifier.	
18	createFacesFile.pl	Combining the faces from the entities "parcel", "osection", "KadMunicipaliyti" and "country".	
19	createSurfacesFile.pl	Combining the surfaces from the entities "parcel", "osection", "KadMunicipaliyti" and "country".	
20	createJasmineUnloadFile.pl	Combine all the files into one "unload file" (dump file) that can be loaded into Jasmine.	
21	createTemplateJasmineUnloadFile.pl	Creation of a template for the perl script "createJasmineUnloadFile.pl". Has to be used only when the class definitions are changed.	

These perl scripts are available in appendix B. These perl scripts are accessible over the program perlConversionUtility<version>.pl for easier use (see appendix B.22):

```

/export/home/patrice/Dutch_Cadastre/data_lki> perl perlConversionUtility4.pl
#####
UTILITY TO CONVERT LKI-DUMP-DATA INTO AN UNLOADFILE FOR A JASMINE DATABASE
#####
**** Key instructions: ****
pressing <0> exits this program

pressing <1> runs boundary_shape_kad2jas.pl for the first conversion of the boundaries
pressing <2> runs osection_kad2jas.pl for the first conversion of the municipalities and the osections
pressing <3> runs createOsectionsFile2.pl for the creation of the municipalities and the osections

pressing <4> runs parcel_kad8jas.pl for the first conversion of the parcels
pressing <5> runs boundary_bbox_kad8jas.pl for the second conversion of the boundaries
pressing <6> runs gcptnt_kad8jas.pl for the first conversion of the gcpts
pressing <7> runs text_kad8jas.pl for the first conversion of the texts
pressing <8> runs sympnts_kad8jas.pl for the first conversion of the sympnts

pressing <9> runs findNodes_parcel2.pl
pressing <10> runs findNodes_boundary2.pl
pressing <11> runs findNodes_gcptnt2.pl
pressing <12> runs findNodes_text2.pl
pressing <13> runs findNodes_sympnt2.pl

pressing <14> runs createEdgeTopology6step1.pl
pressing <15> runs createFaceTopology1step1.pl
pressing <16> runs createEdgeTopology3step2.pl
pressing <17> runs createFaceTopology1step2.pl

pressing <18> runs createFacesFile.pl
pressing <19> runs createSurfacesFile.pl

pressing <20> runs createJasmineUnloadFile.pl

pressing <0> exits this program

Enter you choice:

```

Syntax to run this program:

```
perl perlConversionUtility4.pl
```

Make sure all the files and scripts are available in the same directory.

It is important to note that the object ids for Jasmine are given by these programs. These ids should be unique for each class (Jasmine makes them unique to the whole database). Check therefore the following when using these programs on a new dataset:

- the ids for the directPositions per \*\*\*kad#jas.pl;
- createEdgeTopology#step1.pl;
- the ids for the faces in createOsectionsFile#.pl;

The last program delivers the unload file that can be imported into a windows version of Jasmine. Therefore, the file has to be copied from UNIX to a windows environment. The different storage system concerning line endings proved not to be any problem for Jasmine. When the stores and class families have been defined, the unload file can be loaded into Jasmine using the command “load” (see appendix C).

## Appendix B Conversion programs written in Perl

The perl scripts included in this appendix are listed with their page numbers:

The purposes, input and outputfiles are explained in appendix A.

B.1	boundary_shape_kad2jas.pl.....	61
B.2	osection_kad2jas.pl.....	65
B.3	createOsectionsFile2.pl.....	69
B.4	gcptnt_kad8jas.pl.....	74
B.5	text_kad8jas.pl.....	80
B.6	sympnt_kad8jas.pl.....	86
B.7	parcel_kad8jas.pl.....	92
B.8	boundary_bbox_kad8jas.pl.....	99
B.9	findNodes_parcel2.pl.....	106
B.10	findNodes_boundary2.pl.....	108
B.11	findNodes_gcptnt2.pl.....	110
B.12	findNodes_text2.pl.....	112
B.13	findNodes_sympnt2.pl.....	114
B.14	createEdgeTopology6step1.pl.....	116
B.15	createFaceTopology1step1.pl.....	123
B.16	createEdgeTopology3step2.pl.....	125
B.17	createFaceTopology1step2.pl.....	129
B.18	createFacesFile.pl.....	131
B.19	createSurfacesFile.pl.....	132
B.20	createJasmineUnloadFile.pl.....	133
B.21	createTemplateJasmineUnloadFile.pl.....	145
B.22	perlConversionUtility4.pl.....	147





## Appendix C Storing and structuring of data in Jasmine

After a successful installation of Jasmine the following steps have been taken:

1. creating physical files on a disk, called *stores* where the class families, the classes, the methods and the data will be stored (section C.1);
2. creating the class families that will hold an unique set of classes and subclasses with their methods (section C.2);
3. design of the data structure (section C.3);
4. creating the classes and subclasses (section C.4);
5. loading the data into these classes (using the command load <input file>);
6. defining the methods (appendix D).

When an unload file is available the steps 1, 2 and 5 are necessary to build up a database.

### C.1 Creating the stores

The listed commands can be typed at the DOS prompt or can be copied into a batchfile, for example called c:\jas.bat and run from there. To create a store the following syntax has to be used:

#### createStore system command

```
createStore [-h] [-dbName database] [-userName user]
[-passwd password] [-envFile envFile]
[-numberOfPages pages] [-pageSize pageSize]
[-wait] storeName {fileName}...
```

Source: “Jasmine ODB Database Developer’s Reference 2.0”.

Remark: a 8 Kb page size is the optimal choice “for most of the cases”.

Source: “Jasmine ODB Database Administrator’s Guide 2.0”.

The following commands were applied, copied and run one by one from jas.bat:

```
createStore -numberOfPages 32000 LKISore1 D:\Jasmineii\Jasmine\data\LKISore1
createStore -numberOfPages 32000 LKISore2 D:\Jasmineii\Jasmine\data\LKISore2
createStore -numberOfPages 32000 DCStore1 D:\Jasmineii\Jasmine\data\DCStore1
createStore -numberOfPages 32000 DCStore2 D:\Jasmineii\Jasmine\data\DCStore2
createStore -numberOfPages 32000 DCStore3 D:\Jasmineii\Jasmine\data\DCStore3
```

The first two stores were created for testing purposes after the first conversion step of the LKI data (see Appendix A). The three last stores are created to hold the new object oriented database.

### C.2 Creating the class families

The listed commands can be typed at the dosprompt or can be copied into a batchfile, for example called c:\jas.bat and run from there. To create a class family the following syntax has to be used:

#### createcf system command

```
createcf [-h] [-dbName database] [-userName user]
[-passwd password] [-envFile envFile]
[-CFAlias aliasName] CFName storeName
```

Source: “Jasmine ODB Database Developer’s Reference 2.0”.

The following commands were applied, copied and run one by one from jas.bat:

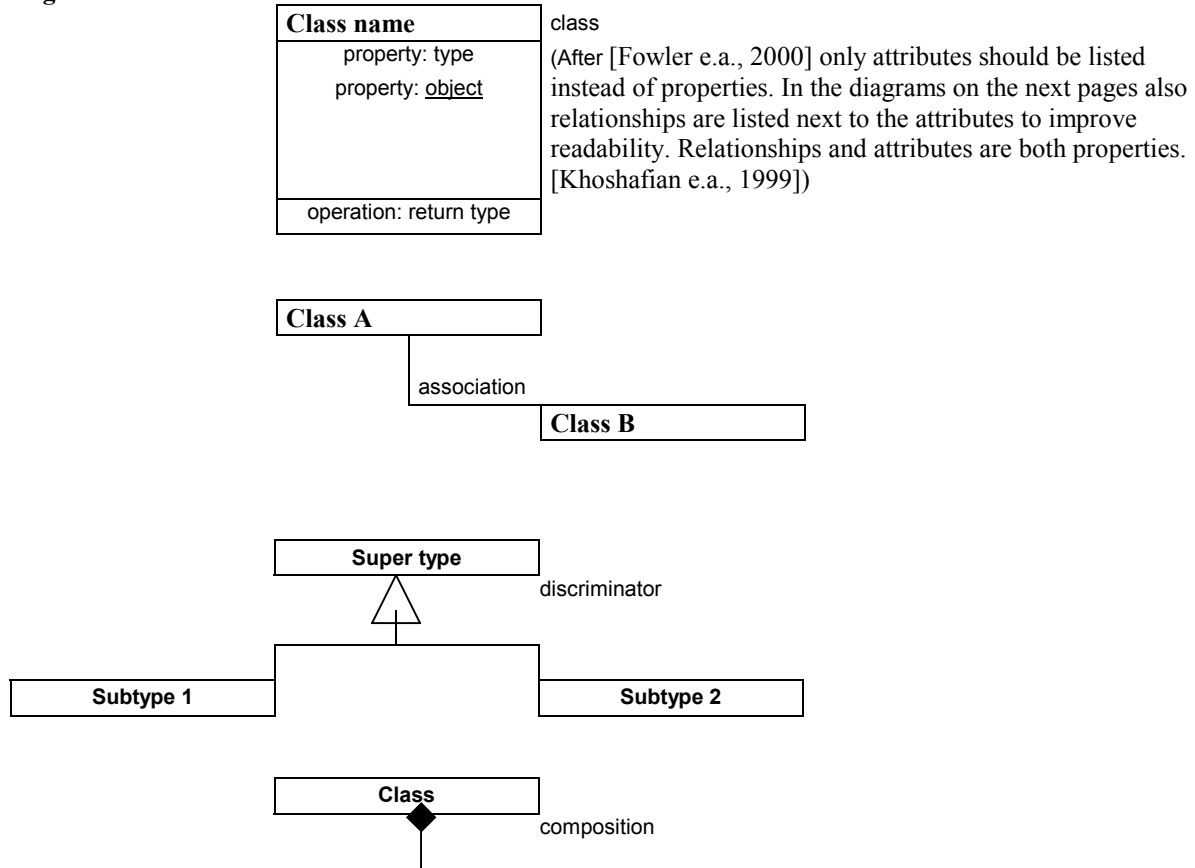
```
createcf lkiCF LKISore1
createcf lkiCF_2 LKISore2
createcf spatialViewCF DCStore1
createcf thematicViewCF DCStore2
```

After the above operation the command “listStore” typed at the DOS prompt gives an overview of the stores and class families (see figure 6.1).

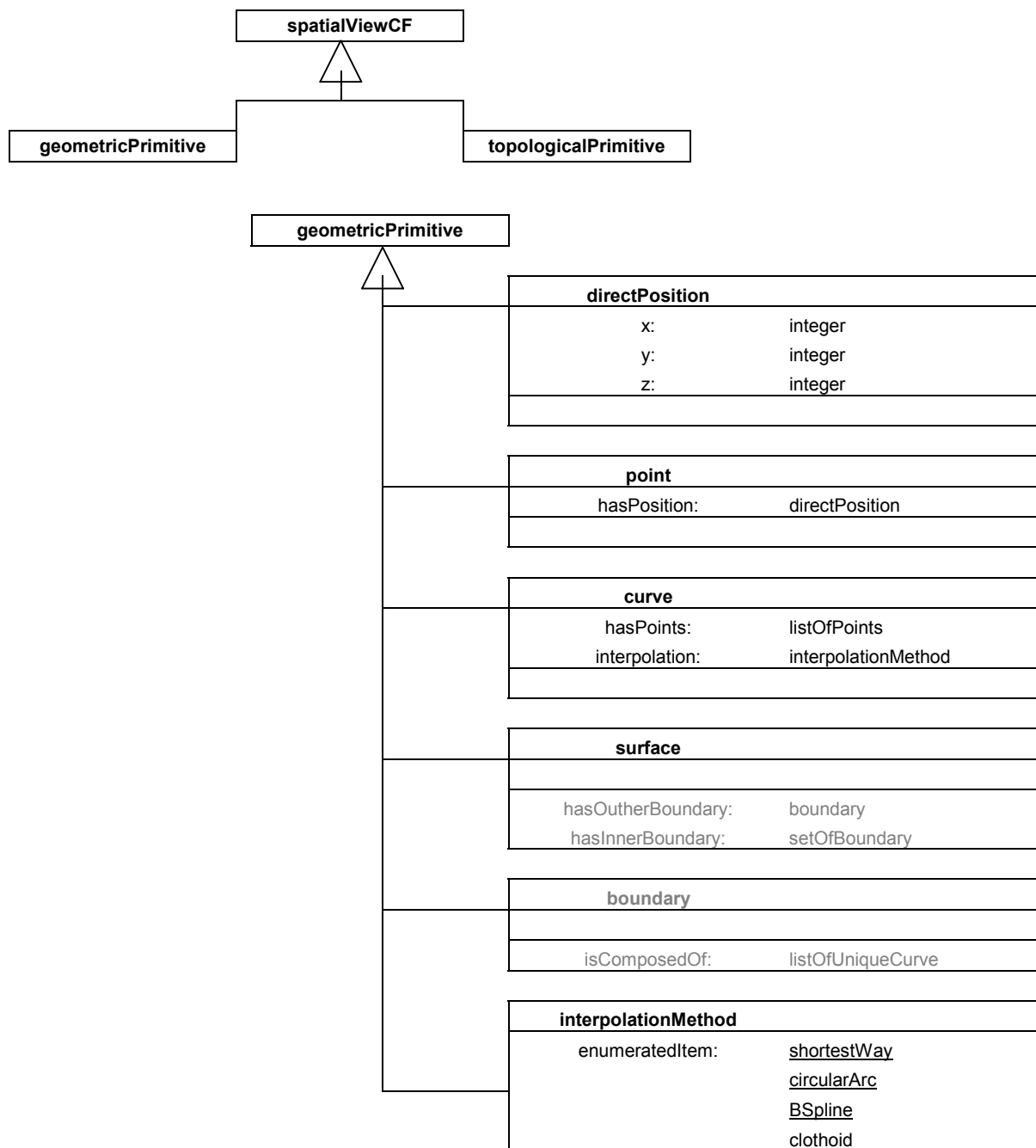
### C.3 The object oriented data model

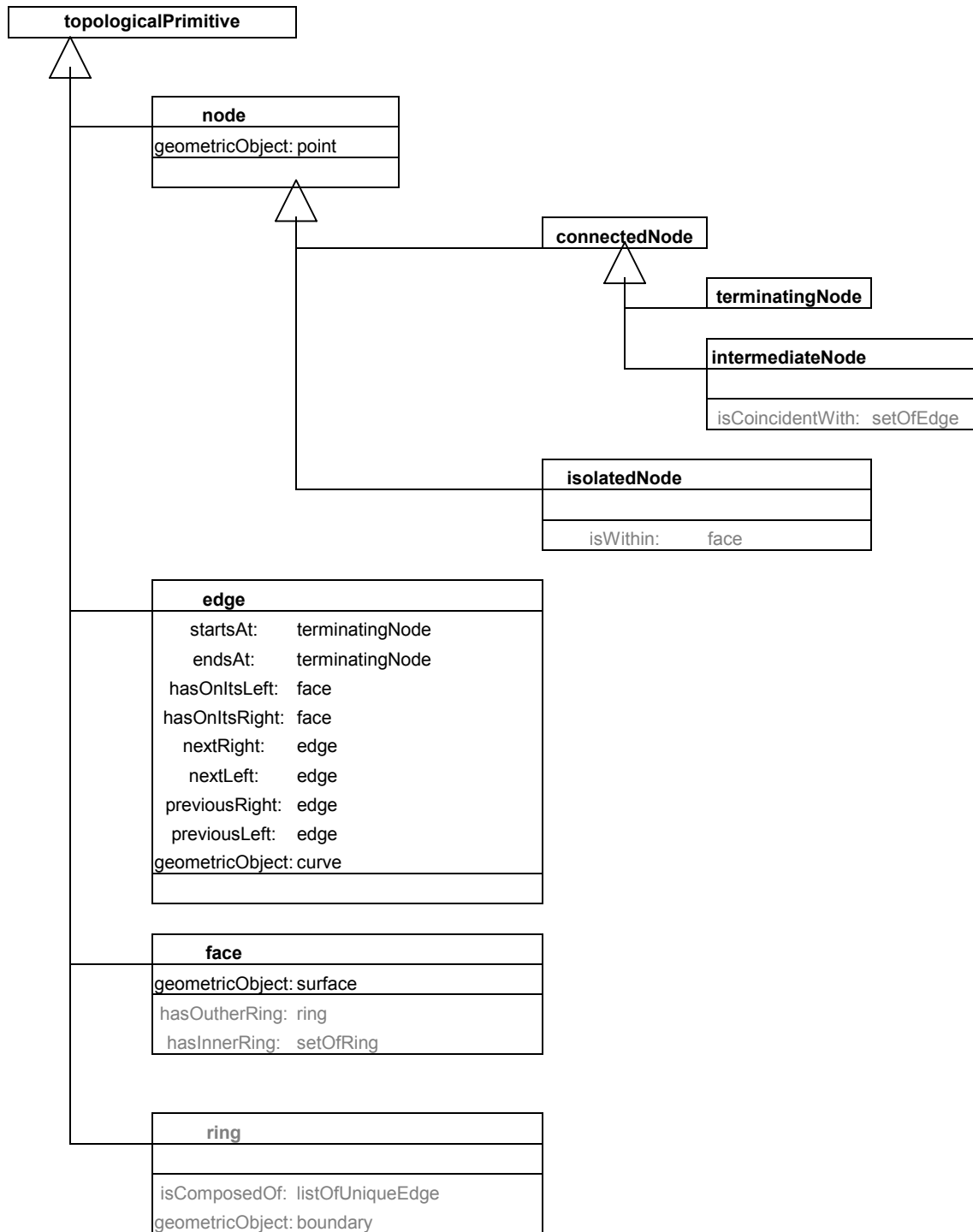
The data model designed to hold the LKI test database in an object oriented structure is given in the UML class diagrams on the next pages. [Booch e.a., 1999], [Fowler e.a., 2000]

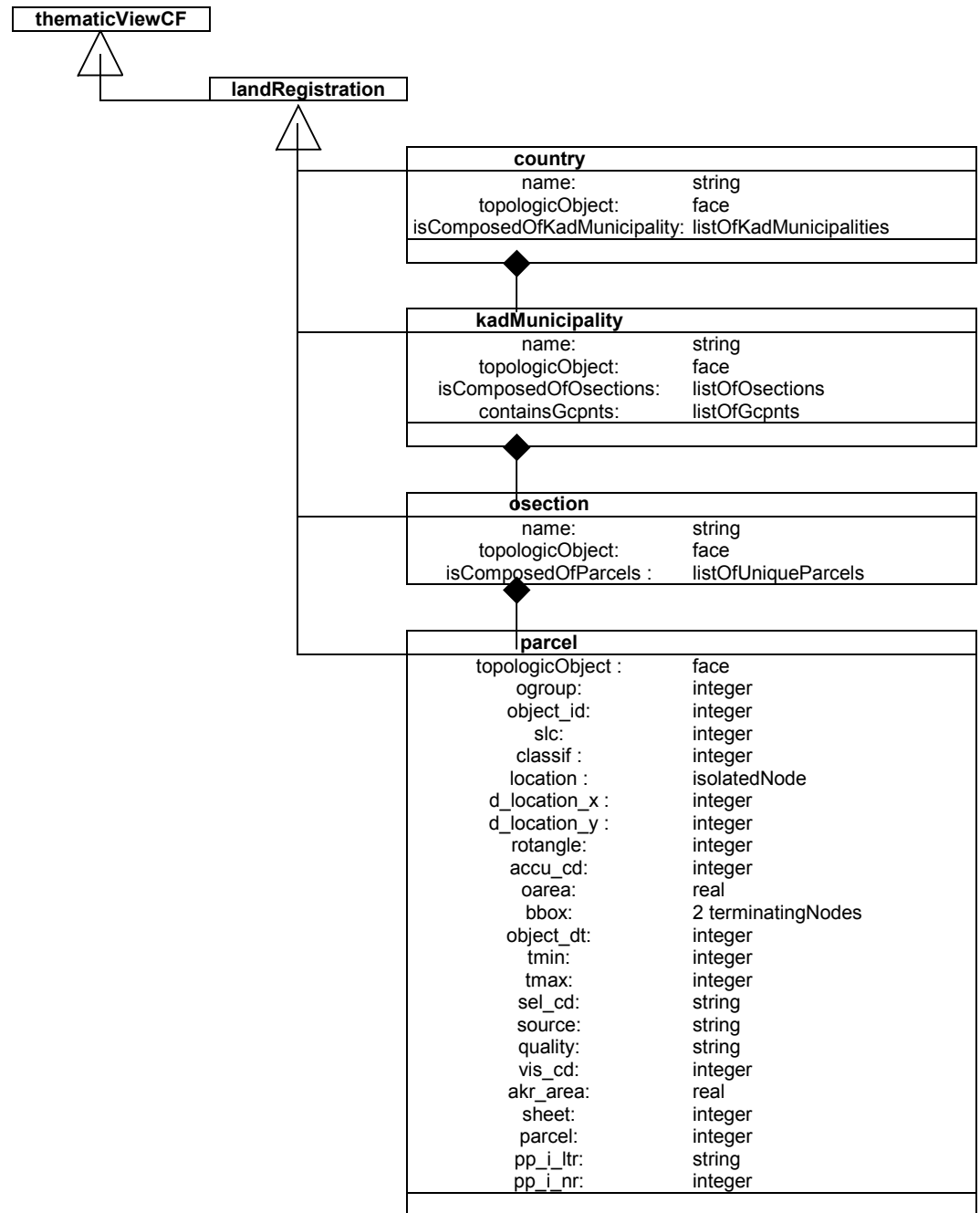
#### Legend:

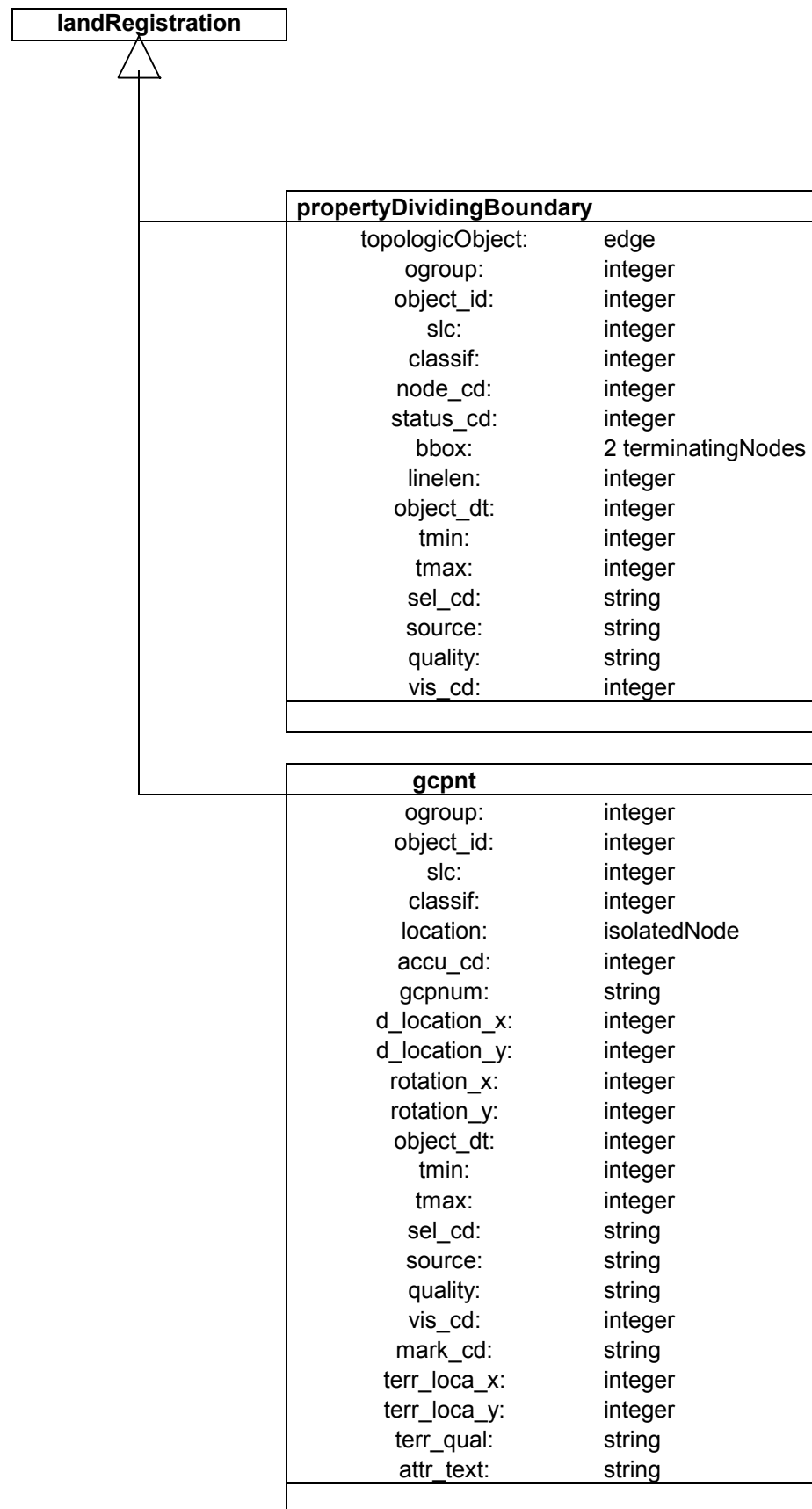


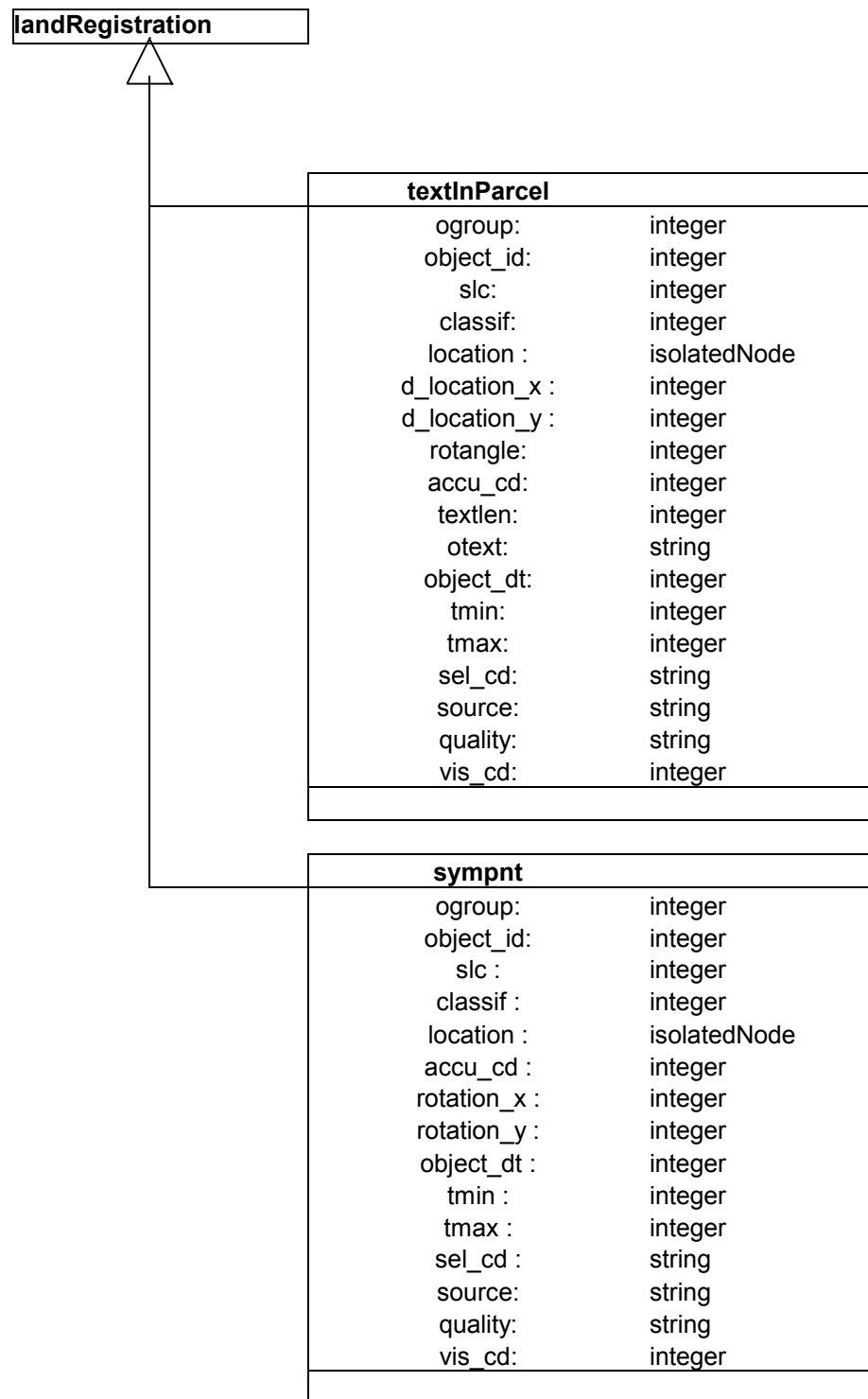
In grey text the parts that have not been implemented.











## C.4 Creating the classes

Before classes can be created, make sure the stores and class families are created (see appendix C.1 and C.2). The classes are created using the ODQL script cc.txt. This script is run under the ODQL Interpreter. This interpreter is started from the DOS prompt by typing:

codqlie

This interpreter is then given the statement:

execFile cc.txt

The contents of cc.txt are given below:

```
*****
defaultCF spatialViewCF;

/* Define a new class */
defineClass spatialViewCF::geometricPrimitive
description: "root class of the geometrical classes"
{
};
buildClass geometricPrimitive;

/* Define a new class */
defineClass spatialViewCF::directPosition
description: "class for storing coordinates, geometrical model for OD-
objects"
super: spatialViewCF::geometricPrimitive
{
    instance:
        systemCF::Integer    x    ;
        systemCF::Integer    y    ;
        systemCF::Integer    z    ;
};
buildClass directPosition;

/* Define a new class */
defineClass spatialViewCF::point
description: "class for storing references to directPositions"
super: spatialViewCF::geometricPrimitive
{
    instance:
        spatialViewCF::directPosition    hasPosition    ;
};
buildClass point;

/* Define a new class */
defineClass spatialViewCF::interpolationMethod
description: "class for storing the descriptions of the interpolation
methods"
super: spatialViewCF::geometricPrimitive
{
    instance:
        systemCF::String[20]    enumeratedItem    ;
};
buildClass interpolationMethod;
```



```
/* Define a new class */
defineClass spatialViewCF::curve
description: "class for storing curves, geometrical model for 1D-
objects, a curve with interpolationMethod = "lijnketen" becomes a line
string"
super: spatialViewCF::geometricPrimitive
{
    instance:
        List<spatialViewCF::point>      hasPoints      ;
        spatialViewCF::interpolationMethod interpolation;
};
buildClass curve;

/* Define a new class */
defineClass spatialViewCF::boundary
description: "class for storing objects holding the curves around a
surface forming a closed polygon"
super: spatialViewCF::geometricPrimitive
{
    instance:
        List<spatialViewCF::curve>      isComposedOf      ;
};
buildClass boundary;

/* Define a new class */
defineClass spatialViewCF::surface
description: "class for surfaces, geometric description for 2D-object"
super: spatialViewCF::geometricPrimitive
{
    instance:
        spatialViewCF::boundary      hasOuterBoundary      ;
        Set<spatialViewCF::boundary> hasInnerBoundary      ;
};
buildClass surface;
```

```
defaultCF spatialViewCF;

/* Define a new class */
defineClass spatialViewCF::topologicalPrimitive
description: "root class of the topological classes"
{
};

/* Define a new class */
defineClass spatialViewCF::node
description: "class for nodes, topological model for OD-objects"
super: spatialViewCF::topologicalPrimitive
{
    instance:
        spatialViewCF::point    geometricObject    ;
};

/* Define a new class */
defineClass spatialViewCF::connectedNode
description: "class for nodes connected to other nodes"
super: spatialViewCF::node
{
};

/* Define a new class */
defineClass spatialViewCF::terminatingNode
description: "class for nodes at the edge's endings"
super: spatialViewCF::connectedNode
{
};

/* Define a new class */
defineClass spatialViewCF::intermediateNode
description: "class for nodes between the edge's endings"
super: spatialViewCF::connectedNode
{
    instance:
        Set<spatialViewCF::edge>    isCoincidentWith    ;
};

/* Define a new class */
defineClass spatialViewCF::isolatedNode
description: "class for nodes without neighbouring nodes"
super: spatialViewCF::node
{
    instance:
        spatialViewCF::face    isWithin    ;
};

/* Define a new class */
/* This class refers to other classes */
/* that have not been defined yet */
/* This is solved by building classes at the end */
defineClass spatialViewCF::edge
description: "class for topological 1D-objects"
super: spatialViewCF::topologicalPrimitive
{
    instance:
        spatialViewCF::terminatingNode    startsAt    ;
        spatialViewCF::terminatingNode    endsAt    ;
        spatialViewCF::face    hasOnItsLeft    ;
        spatialViewCF::face    hasOnItsRight    ;
        spatialViewCF::edge    nextRight    ;
        spatialViewCF::edge    nextLeft    ;
        spatialViewCF::edge    previousRight    ;
        spatialViewCF::edge    previousLeft    ;
        spatialViewCF::curve    geometricObject    ;
};
```

```
/* Define a new class */
defineClass spatialViewCF::face
description: "class for topological 2D-objects"
super: spatialViewCF::topologicalPrimitive
{
    instance:
        spatialViewCF::ring      hasOuterRing      ;
        Set<spatialViewCF::ring> hasInnerRing      ;
        spatialViewCF::surface    geometricObject ;
};

/* Define a new class */
defineClass spatialViewCF::ring
description: "class for objects holding all the edges forming a closed
ring around a face"
super: spatialViewCF::topologicalPrimitive
{
    instance:
        List<spatialViewCF::edge> isComposedOf      ;
        spatialViewCF::boundary    geometricObject ;
};

/* Built the new classes. */
/* This is only possible at the end, because they point to each other */

buildClass ring;
buildClass node;
buildClass connectedNode;
buildClass terminatingNode;
buildClass intermediateNode;
buildClass isolatedNode;
buildClass edge;
buildClass face;
buildClass topologicalPrimitive;

/* Define a new class */
defineClass thematicViewCF::landRegistration
description: "root class for all the classes needed to hold the thematic
data on spatial objects modelling a cadastral situation"
{
};
buildClass landRegistration;

/* Define a new class */
defineClass thematicViewCF::country
description: "class to hold the information concerning countries of
which cadastral information is included"
super: thematicViewCF::landRegistration
{
    instance:
        systemCF::String[20]    name                ;
        spatialViewCF::face      topologicObject      ;
        List<thematicViewCF::kadMunicipality>
isComposedOfKadMunicipalities                ;
};

/* Define a new class */
defineClass thematicViewCF::kadMunicipality
description: "class for cadastral municipalities"
super: thematicViewCF::landRegistration
{
    instance:
        systemCF::String[10]    name                ;
        spatialViewCF::face      topologicObject      ;
        List<thematicViewCF::osection>
isComposedOfOsections                ;
        List<thematicViewCF::gcpnt> containsGcpnts    ;
};
```

```
/* Define a new class */
defineClass thematicViewCF::osection
description: "class for sections"
super: thematicViewCF::landRegistration
{
    maxInstanceSize:    256;
    instance:
        systemCF::String[5]    name                ;
        spatialViewCF::face    topologicObject      ;
        List<thematicViewCF::parcel> isComposedOfParcels;
};

defineClass thematicViewCF::parcel
description: "class for parcels"
super: thematicViewCF::landRegistration
{
    instance:
        spatialViewCF::face    topologicObject;
        systemCF::Integer      ogroup         ;
        systemCF::Integer      object_id       ;
        systemCF::Integer      slc             ;
        systemCF::Integer      classif         ;
        spatialViewCF::isolatedNode location     ;
        systemCF::Integer      d_location_x    ;
        systemCF::Integer      d_location_y    ;
        systemCF::Integer      rotangle        ;
        systemCF::Real          oarea          ;
        List<spatialViewCF::terminatingNode> box ;
        systemCF::Integer      object_dt       ;
        systemCF::Integer      tmin            ;
        systemCF::Integer      tmax            ;
        systemCF::String[3]    sel_cd          ;
        systemCF::String[6]    source          ;
        systemCF::String[3]    quality         ;
        systemCF::Integer      vis_cd          ;
        systemCF::Real          akr_area        ;
        systemCF::Integer      sheet           ;
        systemCF::Integer      parcel          ;
        systemCF::String[2]    pp_i_ltr        ;
        systemCF::Integer      pp_i_nr         ;
};
buildClass parcel;

/* Built a class that points to above class */
buildClass osection;

/* Define a new class */
defineClass thematicViewCF::propertyDividingBoundary
description: "class for boundaries between parcels"
super: thematicViewCF::landRegistration
{
    instance:
        spatialViewCF::edge    topologicObject;
        systemCF::Integer      ogroup         ;
        systemCF::Integer      object_id       ;
        systemCF::Integer      slc             ;
        systemCF::Integer      classif         ;
        systemCF::Integer      node_cd         ;
        systemCF::Integer      status_cd        ;
        List<spatialViewCF::terminatingNode> bbox ;
        systemCF::Integer      linelen         ;
        systemCF::Integer      object_dt       ;
        systemCF::Integer      tmin            ;
        systemCF::Integer      tmax            ;
        systemCF::String[3]    sel_cd          ;
        systemCF::String[6]    source          ;
        systemCF::String[3]    quality         ;
        systemCF::Integer      vis_cd          ;
};
buildClass propertyDividingBoundary;
```

```
/* Define a new class */
defineClass thematicViewCF::gcpnt
description: "class for geodetical points"
super: thematicViewCF::landRegistration
{
    instance:
        systemCF::Integer          ogroup          ;
        systemCF::Integer          object_id        ;
        systemCF::Integer          slc              ;
        systemCF::Integer          classif          ;
        spatialViewCF::isolatedNode location        ;
        systemCF::Integer          accu_cd          ;
        systemCF::String[6]        gcpnum          ;
        systemCF::Integer          d_location_x     ;
        systemCF::Integer          d_location_y     ;
        systemCF::Integer          rotangle         ;
        systemCF::Integer          object_dt        ;
        systemCF::Integer          tmin             ;
        systemCF::Integer          tmax             ;
        systemCF::String[3]        sel_cd          ;
        systemCF::String[6]        source          ;
        systemCF::String[3]        quality         ;
        systemCF::Integer          vis_cd          ;
        systemCF::String[2]        mark_cd         ;
        systemCF::Integer          terr_loca_x      ;
        systemCF::Integer          terr_loca_y      ;
        systemCF::String[2]        terr_qual       ;
        systemCF::String[20]       attr_text       ;
};
buildClass gcpnt;

/* Define a new class */
defineClass thematicViewCF::textInParcel
description: "class for text strings on the cadastral map"
super: thematicViewCF::landRegistration
{
    instance:
        systemCF::Integer          ogroup          ;
        systemCF::Integer          object_id        ;
        systemCF::Integer          slc              ;
        systemCF::Integer          classif          ;
        spatialViewCF::isolatedNode location        ;
        systemCF::Integer          d_location_x     ;
        systemCF::Integer          d_location_y     ;
        systemCF::Integer          rotangle         ;
        systemCF::Integer          accu_cd          ;
        systemCF::Integer          textlen          ;
        systemCF::String[80]       otext           ;
        systemCF::Integer          object_dt        ;
        systemCF::Integer          tmin             ;
        systemCF::Integer          tmax             ;
        systemCF::String[3]        sel_cd          ;
        systemCF::String[6]        source          ;
        systemCF::String[3]        quality         ;
        systemCF::Integer          vis_cd          ;
};
buildClass textInParcel;
```

```
/* Define a new class */
defineClass thematicViewCF::sympnt
description: "class for symbol point on the cadastral map"
super: thematicViewCF::landRegistration
{
  instance:
    systemCF::Integer      ogroup      ;
    systemCF::Integer      object_id   ;
    systemCF::Integer      slc         ;
    systemCF::Integer      classif     ;
    spatialViewCF::isolatedNode location ;
    systemCF::Integer      accu_cd     ;
    systemCF::Integer      rotation    ;
    systemCF::Integer      object_dt   ;
    systemCF::Integer      tmin        ;
    systemCF::Integer      tmax        ;
    systemCF::String[3]    sel_cd      ;
    systemCF::String[6]    source      ;
    systemCF::String[3]    quality     ;
    systemCF::Integer      vis_cd      ;
};
buildClass sympnt;

/* Built new classes that point to other classes */

buildClass kadMunicipality;
buildClass Nederland;
```

## Appendix D Methods written in ODQL

In this appendix the following methods are included:

D.1	Calculate the area of a given parcel	(see paragraph 5.4)
D.2	Count the parcels within a rectangle by centroid	(see paragraph 6.5)
D.3	Count the parcels within a rectangle by bounding box	(see paragraph 6.5)
D.4	Create index	(see paragraph 6.2)
D.5	Find the surrounding boundaries of a parcel	(see paragraph 5.4)
D.6	Find boundaries that point into other boundaries	(see paragraph 6.3)
D.7	Find boundaries inside a parcel	(see paragraph 6.4)
D.8	Find a parcel from a given directPosition (point-in-polygon)	(see paragraph 5.4)

### D.1 Calculate the area of a given parcel

```
/*-----*/
/* Program to calculate the area of a given parcel */
/* */
/* */
/* file      : calculateAreaOfParcel4.txt */
/* Author    : Patrice Wijnands */
/* Date      : July 10th 2001 */
/* Language: ODQL */
/* */
/*-----*/

/* ##### declarations */

defaultCF 'spatialViewCF';

Real          area;
Real          areaPart;
terminatingNode comparingNode;
edge          e;
Integer       i;
Integer       it2;
List<terminatingNode> ltn;
Integer       m;
Integer       numberOfPoints;
point        pnt1;
point        pnt2;
point        pnt3;
List<point>   pntList;
Real          roughArea;
terminatingNode tn1;
terminatingNode tn2;

edge         e1;
edge         e2;
List<edge>   edgeList;
Integer      ibp;
Integer      j;
Integer      k;
Integer      l;
Integer      numberOfEdges;
List<edge>   sortedEdgeList;

String s1;
s1 = "object:";
String s2;
s2 = "parcel:";
String s3;
s3 = "First edge:";
String s4;
s4 = "Number of";
String s5;
s5 = "edges:";
String s6;
s6 = "This is the next one:";
String s7;
s7 = "No edge found.";
String s8;
```

```
s8 = "Parcel's edges, sorted following topology:";
String s9;
s9 = "Parcel's boundaries, checking topology...";
String s10;
s10 = "j =";
String s11;
s11 = "i =";
String s19;
s19 = "Skipping parcel, topology incorrect.";
String s20;
s20 = "Area is [m2]:";
String s21;
s21 = "Calculating area...";
String s22;
s22 = "Rough estimation of the area [m2]:";
String s23;
s23 = "Attention, the result might be erroneous, there are inner rings present!";
String s24;
s24 = "Area stored in the attribute akr_area:";
String s25;
s25 = "Area stored in the attribute oarea:";

/* ##### find parcel */

List<thematicViewCF::parcel> lp;
thematicViewCF::parcel p;

lp = thematicViewCF::parcel from thematicViewCF::parcel
where (thematicViewCF::parcel.object_id == 200145889
and thematicViewCF::parcel.tmax == 0);

Integer numberOfParcels;

numberOfParcels = lp.count();
s2.print();
numberOfParcels.print();

if ( numberOfParcels != NIL and numberOfParcels != 0 )
{
    p = lp.elementAt(0);
    p.object_id.print();
}

/* ##### estimate the area from the bounding box */

ltn = p.bbox;

tn1 = ltn.elementAt(0);
tn2 = ltn.elementAt(1);

tn1.geometricObject.hasPosition.print();
tn2.geometricObject.hasPosition.print();

roughArea = (tn2.geometricObject.hasPosition.x/1000 -
              tn1.geometricObject.hasPosition.x/1000)
            * (tn2.geometricObject.hasPosition.y/1000 -
              tn1.geometricObject.hasPosition.y/1000);
s22.print();
roughArea.print();
s24.print();
p.akr_area.print();
s25.print();
p.oarea.print();

/* ##### findBoundariesFromParcel2.txt */
/* ##### find all edges surrounding that parcel */

edgeList = edge from edge
where (edge.hasOnItsLeft == p.topologicObject
or edge.hasOnItsRight == p.topologicObject);

s5.print();
edgeList.print();

/* ##### count the edges */

numberOfEdges = edgeList.count();
```



```
s4.print();
s5.print();
numberOfEdges.print();

/* ##### get the first edge in the list and remove it */

if ( numberOfEdges != NIL and numberOfEdges != 0 )
{
    e1 = edgeList.getElementAt(0);

    sortedEdgeList = List{ e1 };

    sortedEdgeList.print();

    edgeList.removeElementAt(0);

/* ##### count the edges */

    numberOfEdges = edgeList.count();
    s4.print();
    s5.print();
    numberOfEdges.print();

/* ##### find from the first edge the next one */
/* ##### until the ring is rounded */

    j = 0;
    k = numberOfEdges;

    while (j < k)
    {
        ibp = 0;
        while (ibp < numberOfEdges) {
            e2 = edgeList.getElementAt(ibp);
            e1 = sortedEdgeList.getElementAt(j);

/* ##### comparing with the parcel at the left side */

            if ((e2 == e1.nextLeft) and (p.topologicObject == e2.hasOnItsLeft)) {
                s6.print();
                sortedEdgeList.insertElementAt(e2, j+1);
                l = ibp;
                ibp = numberOfEdges;
            }
            else {
                if ((e2 == e1.nextRight) and (p.topologicObject == e2.hasOnItsLeft)) {
                    s6.print();
                    sortedEdgeList.insertElementAt(e2, j+1);
                    l = ibp;
                    ibp = numberOfEdges;
                }
                else {
                    if ((e2 == e1.previousLeft) and (p.topologicObject == e2.hasOnItsLeft)) {
                        s6.print();
                        sortedEdgeList.insertElementAt(e2, j+1);
                        l = ibp;
                        ibp = numberOfEdges;
                    }
                    else {
                        if ((e2 == e1.previousRight) and (p.topologicObject == e2.hasOnItsLeft)) {
                            s6.print();
                            sortedEdgeList.insertElementAt(e2, j+1);
                            l = ibp;
                            ibp = numberOfEdges;
                        }
                    }
                }
            }

/* ##### comparing with the parcel at the right side */

            else {
                if ((e2 == e1.nextLeft) and (p.topologicObject == e2.hasOnItsRight)) {
                    s6.print();
                    sortedEdgeList.insertElementAt(e2, j+1);
                    l = ibp;
                    ibp = numberOfEdges;
                }
                else {
                    if ((e2 == e1.nextRight) and (p.topologicObject == e2.hasOnItsRight)) {
                        s6.print();
```

```

        sortedEdgeList.insertElementAt(e2, j+1);
        l = ibp;
        ibp = numberOfEdges;
    }
    else {
        if ((e2 == e1.previousLeft) and (p.topologicObject == e2.hasOnItsRight)) {
            s6.print();
            sortedEdgeList.insertElementAt(e2, j+1);
            l = ibp;
            ibp = numberOfEdges;
        }
        else {
            if ((e2 == e1.previousRight) and (p.topologicObject == e2.hasOnItsRight)) {
                s6.print();
                sortedEdgeList.insertElementAt(e2, j+1);
                l = ibp;
                ibp = numberOfEdges;
            }
            else {
                s19.print();
                l = 0;
                ibp = numberOfEdges;
                j = k;
            };};};};};};};};};
        };
        ibp = ibp + 1;
    };

    edgeList.removeElementAt(l);

    numberOfEdges = numberOfEdges - 1;
    s4.print();
    s5.print();

    numberOfEdges.print();

    j = j + 1;
};

/* ##### count the edges */

    numberOfEdges = edgeList.count();

    s8.print();
    sortedEdgeList.print();

}; /*if numberOfEdges != NIL */
}; /*if numberOfParcel != NIL */

/* ##### if the list still contains edges, there are more rings */
/* ##### not only a outer ring but also one or more inner rings */

if (numberOfEdges != 0) {

    s4.print();
    s5.print();
    numberOfEdges.print();
    s23.print();

};

/* ##### end of subprogram */

it2 = 0;
numberOfEdges = sortedEdgeList.count();

s4.print();
s5.print();
numberOfEdges.print();

s21.print();

area = 0;

if ( numberOfEdges != NIL )
{
    e = sortedEdgeList.getElementAt( 0 );

```

```
e.print();
e.geometricObject.print();
pntList = e.geometricObject.hasPoints;
numberOfPoints = pntList.count();
i = 0;
while (i < numberOfPoints-1)
{
    pnt1 = pntList.getElementAt( i );
    pnt2 = pntList.getElementAt( i+1 );
    areaPart = (pnt2.hasPosition.x/1000 - pnt1.hasPosition.x/1000)
                * (pnt2.hasPosition.y/1000 + pnt1.hasPosition.y/1000);
    area = areaPart;
    i = i + 1;
};
comparingNode = e.endsAt;
m = 1;
while (m < numberOfEdges)
{
    e = sortedEdgeList.getElementAt( m );
    pntList = e.geometricObject.hasPoints;
    if (comparingNode == e.startsAt)
    {
        numberOfPoints = pntList.count();
        i = 0;
        while (i < numberOfPoints-1)
        {
            pnt1 = pntList.getElementAt( i );
            pnt2 = pntList.getElementAt( i+1 );
            areaPart = (pnt2.hasPosition.x/1000 - pnt1.hasPosition.x/1000)
                        * (pnt2.hasPosition.y/1000 + pnt1.hasPosition.y/1000);
            area = areaPart;
            i = i + 1;
        };
        comparingNode = e.endsAt;
    }
    else
    {
        numberOfPoints = pntList.count();
        i = numberOfPoints - 1;
        while (i > 0)
        {
            pnt1 = pntList.getElementAt( i );
            pnt2 = pntList.getElementAt( i-1 );
            areaPart = (pnt2.hasPosition.x/1000 - pnt1.hasPosition.x/1000)
                        * (pnt2.hasPosition.y/1000 + pnt1.hasPosition.y/1000);
            area = areaPart;
            i = i - 1;
        };
        comparingNode = e.endsAt;
    };
    m = m + 1;
};

if ( area < 0 )
{
    area = area * -1/2;
}
else
{
    area = area * 1/2;
};

s20.print();
area.print();
};

/* ##### clear memory from variables */

undefVar m;
undefVar comparingNode;
undefVar ltn;
undefVar tn1;
undefVar tn2;
undefVar roughArea;

undefVar e;
```

```
undefVar e1;
undefVar e2;
undefVar edgeList;
undefVar ibp;
undefVar it2;
undefVar i;
undefVar j;
undefVar k;
undefVar l;
undefVar lp;
undefVar numberOfEdges;
undefVar numberOfPoints;
undefVar numberOfParcels;
undefVar p;
undefVar pnt1;
undefVar pnt2;
undefVar pnt3;
undefVar pntList;
undefVar sortedEdgeList;
undefVar area;
undefVar areaPart;
undefVar s1;
undefVar s2;
undefVar s3;
undefVar s4;
undefVar s5;
undefVar s6;
undefVar s7;
undefVar s8;
undefVar s9;
undefVar s10;
undefVar s11;
undefVar s19;
undefVar s20;
undefVar s21;
undefVar s22;
undefVar s23;
undefVar s24;
undefVar s25;

/* ##### end of program */
```

## D.2 Count the parcels within a rectangle by centroid

```
/*-----*/
/* Program to find all parcels inside a given rectangle */
/*
/* file : findParcelsInRectangle.txt */
/* Author: Patrice Wijnands */
/* Date : June 28th 2001 */
/* Language: ODQL */
/*-----*/
defaultCF 'thematicViewCF';
List<parcel> lp;
lp = parcel from parcel
where (parcel.location.geometricObject.hasPosition.x > 166000000
and parcel.location.geometricObject.hasPosition.y > 440000000
and parcel.location.geometricObject.hasPosition.x < 168000000
and parcel.location.geometricObject.hasPosition.y < 442000000
and parcel.tmax == 0);
lp.count().print();

undefVar lp;
/* ##### end of program */
```

## D.3 Count the parcels within a rectangle by bounding box

```
/*-----*/
/* Program to find all parcels inside a given rectangle */
/*
/* file : findParcelsInRectangleByBbox.txt */
/* Author: Patrice Wijnands */
/* Date : July 12th 2001 */
/* Language: ODQL */
/*-----*/

/* ##### declarations */

defaultCF 'spatialViewCF';

List<thematicViewCF::parcel> lp;

Integer llx;
Integer lly;

/* ##### define the rectangle */
/* ##### ll = lower left, ur = upper right */

Integer urx;
Integer ury;

llx = 166000000;
lly = 440000000;
urx = 168000000;
ury = 442000000;

/* ##### find all valid parcels */

lp = thematicViewCF::parcel from thematicViewCF::parcel
where (thematicViewCF::parcel.tmax == 0);

Integer i;
Integer numberOfParcels;

i = 0;
numberOfParcels = lp.count();

thematicViewCF::parcel p;
List<terminatingNode> lon;
terminatingNode tn1;
terminatingNode tn2;
List<thematicViewCF::parcel> lp2;

while (i < numberOfParcels)
{
```

```
p = lp.getElementAt(i);
lon = p.bbox;
tn1 = lon.getElementAt(0);
tn2 = lon.getElementAt(1);

/* if the sides of the bbox overlap with the sides of the rectangle */

if( ( ( (tn2.geometricObject.hasPosition.x >= urx
        and tn1.geometricObject.hasPosition.x <= urx
        )
    or (urx >= tn2.geometricObject.hasPosition.x
        and llx <= tn2.geometricObject.hasPosition.x
        )
    )
    and ( (tn2.geometricObject.hasPosition.y >= ury
        and tn1.geometricObject.hasPosition.y <= ury
        )
    or (ury >= tn2.geometricObject.hasPosition.y
        and llx <= tn2.geometricObject.hasPosition.y
        )
    )
    )
{
/* ##### then add the parcel to the list */
/* ##### put the first parcel into the list */

    if (lp2.count() == NIL)
    {
        lp2 = List{ p };
    }
/* ##### and then the other parcels */

    else
    {
        lp2.directAdd( p );
    };
};
i = i + 1;
};

lp2.count().print();

/* ##### undefine variable */
undefVar llx;
undefVar lly;
undefVar lp;
undefVar i;
undefVar numberOfParcels;
undefVar p;
undefVar lon;
undefVar tn1;
undefVar tn2;
undefVar lp2;
undefVar urx;
undefVar ury;
/* ##### end of program */
```

## D.4 Create index

```
defaultCF 'thematicViewCF';

Bag<Composite class> CCS;
Composite class CC;
CC = parcel.getClass();
CCS = parcel.getSubClasses(TRUE);
CCS = CCS.add(CC);
scan(CCS, CC)
{
    CC.createIndex("object_id", "object_id");
};

undefVar CCS;
undefVar CC;
```



```
p = lp.getElementAt(0);
p.object_id.print();

/* ##### find all edges surrounding that parcel */

List<edge> edgeList;
edgeList = edge from edge
where (edge.hasOnItsLeft == p.topologicObject
or      edge.hasOnItsRight == p.topologicObject);

s5.print();
edgeList.print();

/* ##### count the edges */

Integer numberOfEdges;
numberOfEdges = edgeList.count();
s4.print();
s5.print();
numberOfEdges.print();

/* ##### get the first edge in the list and remove it */

edge e1;
e1 = edgeList.getElementAt(0);
List<edge> sortedEdgeList;

sortedEdgeList = List{ e1 };

sortedEdgeList.print();

edgeList.removeElementAt(0);

/* ##### count the edges */

numberOfEdges = edgeList.count();
s4.print();
s5.print();
numberOfEdges.print();

/* ##### find from the first edge the next one */
/* ##### until the ring is rounded */

Integer i;
Integer j;
Integer k;
Integer l;
edge e2;
s10.print();
j = 0;
k = numberOfEdges;

while (j < k) {
    i = 0;
    while (i < numberOfEdges) {
        e2 = edgeList.getElementAt(i);
        e1 = sortedEdgeList.getElementAt(j);
/* ##### comparing with the parcel at the left side */

        if ((e2 == e1.nextLeft) and (p.topologicObject == e2.hasOnItsLeft)) {

            s6.print();
            sortedEdgeList.insertElementAt(e2, j+1);
            l = i;
            i = numberOfEdges;
        }
        else {
            if ((e2 == e1.nextRight) and (p.topologicObject ==
            e2.hasOnItsLeft)) {

                s6.print();
                sortedEdgeList.insertElementAt(e2, j+1);
                l = i;
                i = numberOfEdges;
            }
        }
        else {
            if ((e2 == e1.previousLeft) and (p.topologicObject ==
```



```
        e2.hasOnItsLeft)) {

                                s6.print();
                                sortedEdgeList.insertElementAt(e2, j+1);
                                l = i;
                                i = numberOfEdges;
        }
    else {
        if ((e2 == e1.previousRight) and (p.topologicObject ==
        e2.hasOnItsLeft)) {

                                s6.print();
                                sortedEdgeList.insertElementAt(e2, j+1);
                                l = i;
                                i = numberOfEdges;
        }
    }
/* ##### comparing with the parcel at the right side */
    else {
        if ((e2 == e1.nextLeft) and (p.topologicObject ==
        e2.hasOnItsRight)) {

                                s6.print();
                                sortedEdgeList.insertElementAt(e2, j+1);
                                l = i;
                                i = numberOfEdges;
        }
    }
    else {
        if ((e2 == e1.nextRight) and (p.topologicObject == e2.hasOnItsRight)) {

                                s6.print();
                                sortedEdgeList.insertElementAt(e2, j+1);
                                l = i;
                                i = numberOfEdges;
        }
    }
    else {
        if ((e2 == e1.previousLeft) and (p.topologicObject ==
        e2.hasOnItsRight)) {

                                s6.print();
                                sortedEdgeList.insertElementAt(e2, j+1);
                                l = i;
                                i = numberOfEdges;
        }
    }
    else {
        if ((e2 == e1.previousRight) and (p.topologicObject ==
        e2.hasOnItsRight)) {

                                s6.print();
                                sortedEdgeList.insertElementAt(e2, j+1);
                                l = i;
                                i = numberOfEdges;
        }
    }
};
};};};};};};};};

    i = i + 1;
};

edgeList.removeElementAt(l);

numberOfEdges = numberOfEdges - 1;
s4.print();
s5.print();

numberOfEdges.print();

j = j + 1;
};

/* ##### count the edges */

numberOfEdges = edgeList.count();

s8.print();
sortedEdgeList.print();

/* ##### if the list still contains edges, there are more rings */
```

```
/* ##### not only a outer ring but also one or more inner rings */  
if (numberOfEdges != 0) {  
    numberOfEdges.print();  
};  
  
/* ##### clear memory from variables */  
  
undefVar e1;  
undefVar e2;  
undefVar edgeList;  
undefVar i;  
undefVar j;  
undefVar k;  
undefVar l;  
undefVar lp;  
undefVar numberOfEdges;  
undefVar numberOfParcels;  
undefVar p;  
undefVar s1;  
undefVar s2;  
undefVar s3;  
undefVar s4;  
undefVar s5;  
undefVar s6;  
undefVar s7;  
undefVar s8;  
undefVar s9;  
undefVar s10;  
undefVar s11;  
undefVar sortedEdgeList;  
  
/* ##### end of program */
```

### D.6 Find boundaries that point into other boundaries

```

/* Program to find a boundary with the same boundary to its left */
/* and to its right */
/* both for its starting node and to its ending node */
/*
/* file : findBoundariesPointingIntoOtherBoundaries.txt
/* Author: Patrice Wijnands
/* Date : May 18th 2001
/* Language: ODQL
/*
/*
/*          boundary a      boundary b
/*          _____|_____
/*                      |
/*                    central boundary
/*                      |
/*          _____|_____
/*          boundary c      boundary d
/*
/*        find central boundary where a=b and c=d
/*
/* -----
/* ##### declarations #####
defaultCF 'thematicViewCF';
List <propertyDividingBoundary> b;

/* ##### find boundaries #####
b =   propertyDividingBoundary
from   propertyDividingBoundary
where (propertyDividingBoundary.topologicObject.nextLeft ==
       propertyDividingBoundary.topologicObject.nextRight)
or     (propertyDividingBoundary.topologicObject.previousLeft ==
       propertyDividingBoundary.topologicObject.previousRight);

Integer numberOfBoundaries;
numberOfBoundaries = b.count();
numberOfBoundaries.print();

/* ##### clear memory from variables #####

undefVar b;
undefVar numberOfBoundaries;
/* ##### end of program #####

```

## D.7 Find boundaries inside a parcel

```
/*-----*/
/* Program to find a boundary with the same parcel to its left */
/* and to its right */
/*
/* file : findBoundaryInParcel2.txt
/* Author: Patrice Wijnands
/* Date : May 22th 2001
/* Language: ODQL
/*
/*
/*
/*


|          |            |
|----------|------------|
|          | <-boundary |
| parcel a | parcel a   |


/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*-----*/

/* ##### declarations */
defaultCF 'thematicViewCF';
List <propertyDividingBoundary> b1;

/* ##### find boundaries */
b1 = propertyDividingBoundary
from propertyDividingBoundary
where propertyDividingBoundary.topologicObject.hasOnItsRight ==
       propertyDividingBoundary.topologicObject.hasOnItsLeft;

String s2;
s2 = "boundary";
s2.print();

b1.count().print();

/* ##### clear memory from variables */
undefVar b1;
undefVar s2;
```

## D.8 Find a parcel from a given directPosition (point-in-polygon)

```
/*-----*/
/* Program to find a parcel */
/* from a given set coordinates */
/* */
/* */
/* file      : findParcelFromDirectPosition */
/* Author    : Patrice Wijnands */
/* Date      : May 24th 2001 */
/* Language: ODQL */
/*-----*/

/* ##### declarations */

defaultCF 'spatialViewCF';

terminatingNode    bottomLeft;
edge               e;
Integer            i;
Integer            intersectionx;
Integer            intersectionCount;
Integer            it2;
Integer            numberOfPoints;
Integer            oddCheck;
thematicViewCF::parcel p;
Integer            part;
point              pnt1;
point              pnt2;
List<point>         pntList;
List<terminatingNode> tN;
terminatingNode    topRight;

edge              e1;
edge              e2;
List<edge>         edgeList;
Integer            ibp;
Integer            j;
Integer            k;
Integer            l;
Integer            numberOfEdges;
List<edge>         sortedEdgeList;

String s1;
s1 = "object";
String s2;
s2 = "parcel";
String s3;
s3 = "first edge";
String s4;
s4 = "number of";
String s5;
s5 = "edges";
String s6;
s6 = "this is the next one";
String s7;
s7 = "no edge found";
String s8;
s8 = "parcel's edges, sorted following topology";
String s9;
s9 = "parcel's boundaries, checking topology...";
String s10;
s10 = "j =";
String s11;
s11 = "i =";
String s12;
s12 = "Point is inside parcel.";
String s13;
s13 = "Point is not inside any parcel.";
String s14;
s14 = "Looking for parcel surrounding this point.";
String s15;
s15 = "Checking candidate parcel.";
String s16;
```

```
s16 = "Collecting candidate parcels...";
String s17;
s17 = "Finished.";
String s18;
s18 = "Number of candidate parcels:";
String s19;
s19 = "Skipping parcel, topology incorrect.";

/* ##### find this directPosition */

directPosition dP;
dP = directPosition.new( x := 162250270, y := 445286590, z := 0 );
s14.print();
dP.print();

/* ##### list all still existing parcels */

List<thematicViewCF::parcel> lp;
lp = thematicViewCF::parcel from thematicViewCF::parcel
where thematicViewCF::parcel.tmax == 0;

/* ##### find in this list all parcels owning a bounding box */
/* ##### containing the given directPosition */

List<thematicViewCF::parcel> candidateParcelList;

Integer it;
Integer numberOfParcels;
numberOfParcels = lp.count();

s16.print();

it = 0;
while (it < numberOfParcels)
{
    p = lp.getElementAt(it);
    tN = p.bbox;
    tN.head();
    bottomLeft = tN.getElement();
    tN.next();
    topRight = tN.getElement();
    if ( bottomLeft.geometricObject.hasPosition.x <= dP.x )
    {
        if ( bottomLeft.geometricObject.hasPosition.y <= dP.y )
        {
            if ( topRight.geometricObject.hasPosition.x >= dP.x )
            {
                if ( topRight.geometricObject.hasPosition.y >= dP.y )
                {
                    if ( candidateParcelList.count() == NIL)
                    {
                        candidateParcelList = List{ p };
                    }
                    else
                    {
                        candidateParcelList.directAdd( p );
                    }
                }
                candidateParcelList.count().print();
            }
        }
    }
    it = it + 1;
};

s17.print();

/* ##### if such parcels are found, calculate from their */
/* ##### curves which parcel contains the given directPosition */

Integer numberOfCandidateParcels;
numberOfCandidateParcels = candidateParcelList.count();

s18.print();
```

```
numberOfCandidateParcels.print();

if (numberOfCandidateParcels != NIL )
{
    Integer it;
    it = 0;

    while (it < numberOfCandidateParcels )
    {
        p = candidateParcelList.getElementAt(it);
        s15.print();
        p.object_id.print();

/* ##### findBoundariesFromParcel2.txt */
/* ##### find all edges surrounding that parcel */

        edgeList = edge from edge
        where (edge.hasOnItsLeft == p.topologicObject
        or      edge.hasOnItsRight == p.topologicObject);

        s5.print();
        edgeList.print();

/* ##### count the edges */

        numberOfEdges = edgeList.count();
        s4.print();
        s5.print();
        numberOfEdges.print();

/* ##### get the first edge in the list and remove it */

        if ( numberOfEdges != NIL and numberOfEdges != 0 )
        {

            e1 = edgeList.getElementAt(0);
            sortedEdgeList = List{ e1 };
            sortedEdgeList.print();
            edgeList.removeElementAt(0);

/* ##### count the edges */

            numberOfEdges = edgeList.count();
            s4.print();
            s5.print();
            numberOfEdges.print();

/* ##### find from the first edge the next one */
/* ##### until the ring is rounded */

            s10.print();
            j = 0;
            k = numberOfEdges;

            while (j < k)
            {
                ibp = 0;
                while (ibp<numberOfEdges)
                {
                    e2 = edgeList.getElementAt(ibp);
                    e1 = sortedEdgeList.getElementAt(j);

/* ##### comparing with the parcel at the left side */

                    if      ((e2 == e1.nextLeft) and (p.topologicObject == e2.hasOnItsLeft)) {

                        s6.print();
                        sortedEdgeList.insertElementAt(e2, j+1);
                        l = ibp;
                        ibp = numberOfEdges;

                    }
                    else {
                        if ((e2 == e1.nextRight) and (p.topologicObject == e2.hasOnItsLeft)) {

                            s6.print();
                            sortedEdgeList.insertElementAt(e2, j+1);
                            l = ibp;

                        }
                    }
                }
                j++;
            }
        }
    }
}
```





```
/* ##### count the edges */
numberOfEdges = edgeList.count();

s8.print();
sortedEdgeList.print();

}; /*if numberOfEdges != NIL */

/* ##### if the list still contains edges, there are more rings */
/* ##### not only a outer ring but also one or more inner rings */

if (numberOfEdges != 0) {

    s4.print();
    s5.print();
    numberOfEdges.print();
};

/* ##### end of subprogram */

it2 = 0;
numberOfEdges = sortedEdgeList.count();

if ( numberOfEdges != NIL)
{
    intersectionCount = 0;

    while (it2 < numberOfEdges )
    {
        e = sortedEdgeList.getElementAt( it2 );
        pntList = e.geometricObject.hasPoints;
        numberOfPoints = pntList.count();
        i = 0;
        while (i < numberOfPoints - 1)
        {
            pnt1 = pntList.getElementAt( i );
            pnt2 = pntList.getElementAt( i+1 );

/* ##### central part of pointInPolygon algoritm */
/* ##### after Prof. Alfred Schmitt */

            if          (          (pnt1.hasPosition.y < dP.y)
                                and    (pnt2.hasPosition.y < dP.y)   )
            {
            }
            else
            {
                if          (          (pnt1.hasPosition.y >= dP.y)
                                and    (pnt2.hasPosition.y >= dP.y)   )
                {
                }
                else
                {
                    intersectionx = (dP.y - pnt1.hasPosition.y)
                        * ( (pnt2.hasPosition.x - pnt1.hasPosition.x)
                          /  (pnt2.hasPosition.y - pnt1.hasPosition.y) )
                    +    pnt1.hasPosition.x;

                    if ( intersectionx >= dP.x )
                    {
                        intersectionCount = intersectionCount + 1;
                    }
                }
            };

            i = i + 1;
        };
    };

/* ##### end of pointInPolygon algoritm */
/* ##### get next edge */
    sortedEdgeList.next();
    it2 = it2 + 1;
};
```

```
};

part      = intersectionCount / 2;
oddCheck = intersectionCount - 2*part;

if (oddCheck > 0)
{
    it = numberOfCandidateParcels;
    s12.print();
    p.print();
}
else
{
    s13.print();
};
it = it + 1;
};

};
/* ##### clear memory from variables */
undefVar bottomLeft;
undefVar candidateParcelList;
undefVar dP;
undefVar e;
undefVar i;
undefVar intersectionx;
undefVar intersectionCount;
undefVar it;
undefVar it2;
undefVar lp;
undefVar numberOfCandidateParcels;
undefVar numberOfParcels;;
undefVar numberOfPoints;
undefVar oddCheck;
undefVar pntList;
undefVar p;
undefVar part;
undefVar pnt1;
undefVar pnt2;
undefVar sortedEdgeList;
undefVar tN;
undefVar topRight;
undefVar e1;
undefVar e2;
undefVar edgeList;
undefVar ibp;
undefVar j;
undefVar k;
undefVar l;
undefVar numberOfEdges;
undefVar s1;
undefVar s2;
undefVar s3;
undefVar s4;
undefVar s5;
undefVar s6;
undefVar s7;
undefVar s8;
undefVar s9;
undefVar s10;
undefVar s11;
undefVar s12;
undefVar s13;
undefVar s14;
undefVar s15;
undefVar s16;
undefVar s17;
undefVar s18;
undefVar s19;

/* ##### end of program */
```