Ministerie van Verkeer en Waterstaat

Rijkswaterstaat

Specificatie datamodel Beheerkaart Nat

Datamodel conform OGC-standaarden

27 juni 2005

RWS-AGI:
Paul van Asperen
Michel Grothe

TU Delft – OTB:
Sisi Zlatanova
Marian de Vries
Theo Tijssen
Peter van Oosterom

IONIC Software S.A.:
Alain Kabamba

AGI-2005-GPMP-017
GISt report No. 31
IONIC-2005-GISRP-001

ISBN: 9036900204

Colofon

Uitgegeven door: Adviesdienst voor Geoinformatie en ICT
P.O. Box 5023
2600 GA Delft
The Netherlands
Tel: + 31 (0) 15 2757575
Fax: + 31 (0) 15 2757576
E-mail: agi-info@agi.rws.minvenw.nl

Auteurs: Paul van Asperen
Michel Grothe
Sisi Zlatanova
Marian de Vries
Theo Tijssen
Peter van Oosterom
Alain Kabamba

Datum: 27 juni 2005

Status: definitief

Versienummer: 1.0

AGI-rapport AGI-2005-GPMP-017

TU-rapport GISt report No. 31

IONIC-rapport IONIC-2005-GISRP-001

ISBN 9036900204

Inhoudsopgave

Managementsamenvatting 6

Bijlagen:

A. Design of a logical and physical model for the Beheerkaart-Nat
Sisi Zlatanova, Marian de Vries, Theo Tijssen, Peter van Oosterom

B. OpenGIS compliance check of the Beheerkaart-Nat data model
Alain Kabamba

Managementsamenvatting

Inleiding

De Beheerkaart Nat vormt de uniforme geografische vastlegging van de natte beheerobjecten van Rijkswaterstaat.

Het project is uitgevoerd binnen de kaders van het programma geo-informatievoorziening (GIV) van beheer, onderhoud en ontwikkeling in de natte sector, afgekort GIV BOO Nat. Het eindproduct van dit werkprogramma is: "een gestroomlijnde GIV voor de beheerkaart van Rijkswaterstaat door de centrale opslag en ontsluiting van de beheerkaart, waarmee de geo-informatie voor GIV BOO Nat (Beheerkaart-Nat) op een eenduidige en uniforme wijze kan beheerd en gebruikt binnen Rijkswaterstaat". In het kader van het werkprogramma GIV BOO Nat zijn achtereenvolgens in de afgelopen 3 jaar de volgende 'voorbereidingen' getroffen ten behoeve van de realisatie van de Beheerkaart Nat:

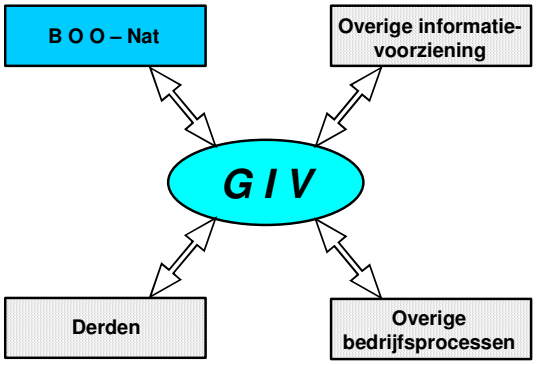
- 2002: Definitiestudie beheerkaart Nat;
- 2003: Architectuurtekening GIV BOO Nat;
- 2004/2005: Inventarisatie beheerkaart Nat en ontwikkeling uniform datamodel en use cases beheerapplicatie Nat;
- 2005: Realisatie Beheerkaart Nat.

Dit rapport beschrijft het uniforme (logische en fysieke) datamodel van de Beheerkaart Nat.

Het datamodel is ontwikkeld door de sectie GIS-technologie van het Onderzoeksinstituut Technische Bestuurskunde (OTB) van de TU Delft. Het datamodel is beoordeeld of haar OGC-compliancy – voldoet het datamodel aan de standaarden van het Open Geospatial Consortium (OGC) - door de firma IONIC Software S.A.. De opmerkingen, die IONIC heeft gemaakt, zijn in de rapportage van de TU Delft verwerkt. Beide rapportages zijn integraal als bijlage A (TU Delft) en B (IONIC) opgenomen.

Context en positionering BKN

GIV BOO Nat is in de modellering van dit onderzoek gedefinieerd als "het geheel van infomatieverwerkende systemen, zowel handmatig als geautomatiseerd, dat BOO van het hoofdvaarwegennet en hoofdwatersystemen moet ondersteunen en waarin de geo-component een significante rol vervult". De plaats of context waarbinnen GIV functioneert, is als onderstaand te schematiseren.



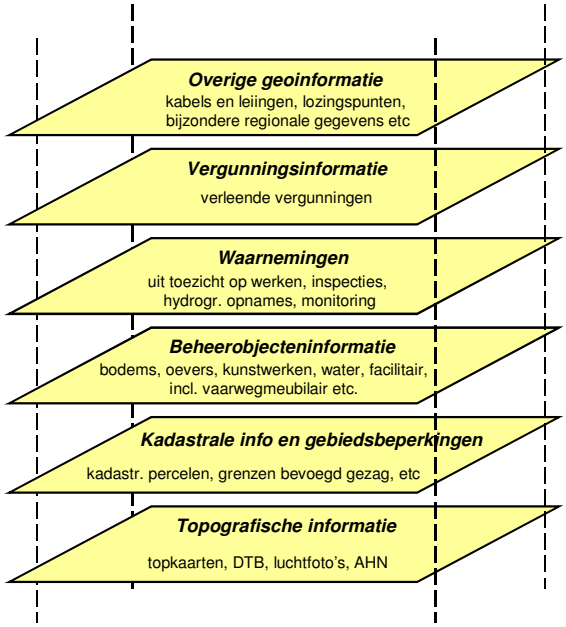
1. De hoofdfuncties van GI voor RWS kunnen als volgt worden gekenschetst: Fungeren als bron van informatie over de ruimtelijke werkelijkheid buiten. Bijvoorbeeld: topografische kaarten, DTB's, luchtfoto's etc.
2. Ruimtelijk beschrijven van ligging, vorm en afmetingen van het RWS-areaal en de onderverdeling van dat areaal in beheerobjecten.
3. Fungeren als interface en integratie-tool voor alle informatie die aan een locatie is te relateren. Immers: met GI is deze informatie beter toegankelijk te maken, beter te ontsluiten, te koppelen c.q. te integreren en in onderlinge ruimtelijke samenhang te beschouwen. Het uitwisselen en delen van informatie wordt gemakkelijker met GIS. Een plaatje zegt meer dan 1000 woorden.
4. Faciliteren van ruimtelijke analyses, modelberekeningen, geografische visualisaties en dergelijke. Bijvoorbeeld: waar en hoeveel moet er gebaggerd worden om een vaargeul op de interventiediepte te brengen.
5. Ondersteunen van mobiele locatiegebonden diensten, bijvoorbeeld ontsluiten van databases 'in het veld' en informatie aanbieden afhankelijk van de actuele positie. Deze toepassing van GIS veelal in combinatie met GPS[1], duidt men vaak aan met Location Based Services, kortweg LBS.

Met andere woorden: GI is op vele manieren zinvol in te zetten voor de werkprocessen van RWS en wordt ook al veel zinvol ingezet, zo is in de bezoeken gebleken.

De geo-informatie en de daaraan gestelde eisen

Wat is de GI die binnen RWS-BOO-Nat wordt gebruikt?
Dit is een verscheidenheid aan informatieverzamelingen, die kan worden ingedeeld in een aantal clusters of categorieën. Deze zijn als volgt te schematiseren.

[1] Global Positioning System.



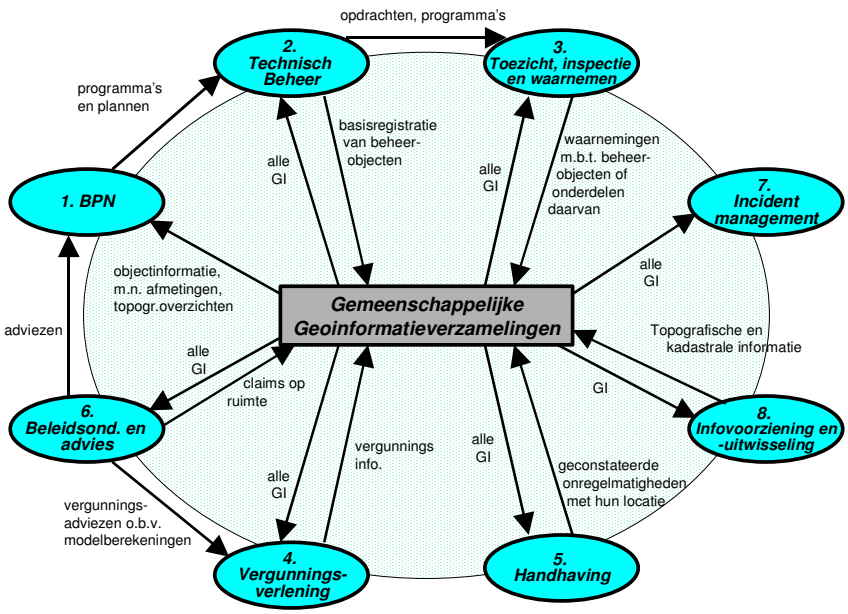
De onderste twee lagen worden gevormd door uitgangsinformatie, informatie die men niet kan veranderen. Boven deze twee lagen komen de deelverzamelingen met de eigen RWS-informatie, om te beginnen de objecten die men beheert (de Beheerkaart Nat), vervolgens de waarnemingen die gekoppeld zijn aan deze objecten, daarna de vergunningsinformatie en bovenaan een laag met overige, moeilijker in te delen groepen van geo-informatie.

De eisen die worden gesteld aan een goede geo-informatievoorziening, kunnen als volgt worden omschreven:

- GIV moet een éénduidige, uniforme, enkelvoudige basisregistratie van beheerobjecten mogelijk maken.
- De GI moet gemeenschappelijk kunnen worden gebruikt (meervoudig gebruik; multi-user; ook qua updates).
- De GI moet transparant toegankelijk zijn, d.w.z. voor alle belanghebbenden, eenvoudig, gebruiksvriendelijk en overal.
- De Gi over de beheerde objecten moet verifieerbaar zijn (in verband met de 'accountability').
- Last but not least: de nauwkeurigheid en actualiteit van GI moet zijn afgestemd op het gebruik.

Dit is nu bij lange na nog niet het geval. Er zijn nu diverse dataverzamelingen, diverse systemen, vaak opgezet voor één specifiek doel, met verschillende opslag- en uitwisselingsformaten. Uniformiteit, gemeenschappelijk gebruik, transparantie en verifieerbaarheid is er nog niet of nauwelijks. nauwkeurigheid en actualiteit zijn doorgaans niet gedocumenteerd.

Naar een gemeenschappelijke geodatabase
De ideale situatie om invulling te geven aan deze eisen wordt bereikt door het opzetten van gemeenschappelijke geodatabase. Het volgende schema illustreert dit:



Bij de inrichting van deze geodatabase is het belangrijk om een aantal principes in acht te nemen:

- Steevast bij informatie ook meta-informatie vastleggen (definitie, bron, actualiteit etc) zodat iedere gebruiker zich ervan kan vergewissen van de bruikbaarheid van de data.
- Een flexibele en uitbreidbare databasestructuur aanleggen, zodat ook specifieke regionale dataverzamelingen daarin kunnen worden ingepast.
- De data zodanig structureren dat die geschikt zijn voor koppelingen met alfanumerieke en andere informatiesystemen.
- Het toepassen van standaards voor opslag en uitwisseling van de GI.

Voor de inrichting van de BPN database is de beheerkaart Nat ontwikkeld. Hieronder volgt een overzicht van de inhoudelijke en technische randvoorwaarden, die zijn gesteld aan het datamodel voor de Beheerkaart Nat.

Gestelde eisen aan het datamodel

Aan het model zijn de volgende inhoudelijke eisen gesteld:

- Inhoud: het datamodel voldoet aan de BPN-systematiek (zie daarvoor Wegwijzer BPN). Het datamodel is zodanig flexibel, dat uitbreiding ervan en mogelijke integratie van andere modellen mogelijk is. De beheerkaart kan daarmee - indien nodig - worden uitgebreid met beheergrenzen uit andere natte sectoren, zoals het waterkwaliteitsbeheer, het nautisch beheer en het verkeersmanagement, de vergunningverlening en – handhaving.
- Koppelbaarheid: door gebruik te maken van de BPN-code kunnen extra gegevens aan de Beheerkaart-Nat-objecten worden gekoppeld.
- Genereren van areaalgegevens: op basis van een selectie ((hoofd)watersysteem(deel), regionale dienst of district) dienen

areaalgegevens per objectsubcategorie gegenereerd te kunnen worden.

- Schaalafhankelijkheid: beide modellen bevatten de mogelijkheden tot aggregatie van de Beheerkaart-objecten naar een hoger schaalniveau. Op die wijze kunnen areaalgegevens (oppervlakte, lengte per categorie, per watersysteemdeel) uit de database worden geselecteerd, kwantitatief geanalyseerd en gevisualiseerd. Aggregatie moet op diverse niveaus mogelijk zijn. Indien mogelijk wordt gebruik gemaakt van andere bestanden, zoals bijvoorbeeld NWB-vaarwegen.
- Editmogelijkheden:uiteindelijk moet de data in de database door eindgebruikers kunnen worden bewerkt:
  - invoegen, muteren en verwijderen van objecten
  - invoegen, muteren en verwijderen van attributen (BPN-code).
- Het model dient dit te faciliteren. Het ontwikkelen van de editmogelijkheden behoort niet tot dit project.
- Historie en mutaties: wijzigingen over een door de gebruiker te specificeren periode en van een gekozen gebied moeten uit de database gefilterd kunnen worden.
- Queries: het datamodel moet het mogelijk maken de vragen bij hierboven vermelde onderwerpen te beantwoorden.

Aan het model zijn de volgende (technische) eisen gesteld:
Ten aanzien van het logisch model:

- Objectgericht;
- Gebaseerd op OGC Simple Feature Specification;
- Notatie gemodelleerd in UML;
- Rapportage met omschrijvingen van de structuur, entiteiten en gemaakte keuzes.

Ten aanzien van het fysiek model:

- Objectgericht;
- Gemodelleerd in Oracle (Spatial) 9i;
- Geschikt voor OGC WFTS en WMS;
- Inclusief rapportage met beschrijvingen van de velden, tabellen en gemaakte keuzes.

Resultaten
Het ontwikkelde datamodel voldoet aan bovenvermelde eisen. Het model is technisch volledig uitgewerkt. Inhoudelijk zijn alle objecten gemodelleerd, het maken van koppelingen, het genereren van areaalgegevens zijn mogelijk, historie en schaalafhankelijkheid zijn gemodelleerd. De OGC-compliancy controle is uitgevoerd en heeft het model op twee aspecten aangescherpt. Het gebruik maken van andere bestanden, zoals NWB-vaarwegen, en het generaliseren van bijvoorbeeld oeverobjecten is lastig gebleken. Hier zal in overleg met een gebruikersafvaardiging het model nader moeten worden gespecificeerd.

Bijlage A

Design of a logical and physical model for the Beheerkaart-Nat

Dr. dipl.-eng. S. Zlatanova, drs. M.E. de Vries, drs. T.P.M. Tijssen and prof.dr.ir. P.J.M. van Oosterom

GISt report No. 31

Summary:
The goal of this project is to design a data model (logical and physical) for the management of the Beheerkaart –Nat objects. The model is implemented in Oracle Spatial 9i and tested with respect to the needs of RWS users. The proposed model provides the RWS organisation (and more particularly AGI) with extended functionality and improved data consistency that reduces the resources (in terms of time, man-power and money) needed for maintenance of the Beheerkaart-Nat. The consequence of such a new organisation of data should be significant reductions in the production and maintenance cost and improved services to customers.

This report has been prepared under the authority of Rijkswaterstaat – AGI.

© 2005
Rijkswaterstaat – AGI
P.O. Box 5023, 2600 GA Delft, the Netherlands
Tel.: +31 (0)15 2757575; Fax +31 (0)15 2757576;
E-mail: agi-info@agi.rws.minvenw.nl

Section GIS technology
TU Delft, OTB Research Institute for Housing, Urban and Mobility Studies
P.O. Box 5030, 2600 BX Delft, the Netherlands
Tel.: +31 (0)15 278 4548; Fax +31 (0)15 278 2745;
E-mail: e.fendel@otb.tudelft.nl

All rights reserved. No part of this publication may be reproduced or incorporated into any information retrieval system without written permission from the publisher.

The Section GIS technology accepts no liability for possible damage resulting from the findings of this research or the implementation of recommendations.

CONTENTS

1 INTRODUCTION 7

2 DATA IMPORT AND VALIDATION 9

2.1 DATA SETS 9
2.1.1 DATA IMPORT 9
2.1.2 IMPORT OF SHAPE FILES INTO ORACLE WITH FME 10
2.1.3 DATA VALIDATION 12
2.2 CONCLUSION 14

3 LOGICAL MODEL 15

3.1 CURRENT PROBLEMS 15
3.2 LOGICAL DESIGN 17
3.3 CONCLUSSION 19

4 PHYSICAL MODEL AND IMPLEMENTATION 21

4.1 PHYSICAL MODELLING 21
4.2 IMPLEMENTATION AND POPULATION 21
4.2.1 CREATING VIS1_GEOM, VIS2_GEOM FROM ORIGINAL GEOMETRIES 25
4.2.2 CREATING VIS1_GEOM, VIS2_GEOM BY LINKING TO EXISTING DATA SETS 28
4.3 CONCLUSION 29

5 TESTS 33

5.1 TYPES OF QUERIES 33
5.2 EXAMPLES OF QUERIES 33

6 DISCUSSION OF THE MODEL WITH RESPECT TO THE REQUIREMENTS OF AGI37

6.1 MULTIPLE GEOMETRY TYPES 37
6.2 MULTIPLE 'LAYERS' 38
6.3 MULTIPLE GEOMETRY REPRESENTATIONS 38
6.4 AGGREGATION OF OBJECTS BASED ON THEMATIC ATTRIBUTE VALUES 39
6.5 POSSIBILITIES FOR MAINTAINING HISTORY 40
6.5.1 OPTION 1. ONLY MONITORING THE CHANGES 40
6.5.2 OPTION 2. KEEPING A COMPLETE HISTORY OF ALL BEHEEROBJECTS 40
6.5.3 OPTION 2.1 STORING COPY OF ALL INSERTS, UPDATES AND DELETES IN SEPARATE TABLE 41
6.5.4 OPTION 2.2. KEEPING UPDATED AND DELETED RECORDS IN BEHEEROBJECT TABLE 42

7 IMPLEMENTATION STEPS 47

8 OPENGIS COMPLIANCE CHECK 49

8.1 'LAYERS' (LBODEMS ETC.) BECOME SUBCLASSES INSTEAD OF AGGREGATIONS 49
8.2 NO MIXED GEOMETRY TYPES 49

LIST OF FIGURES

FIGURE 1: BEHEERKAART-NAT (LEFT) AND DIENSTKRINGEN BOUNDARIES (RIGHT) 9
FIGURE 2: NAVIGATION CHARTS (LEFT) AND MAN-MADE POINT OBJECTS (RIGHT) 9
FIGURE 3: BEHEERKAART-NAT: LOGICAL MODEL 18
FIGURE 4: BEHEERKAART-NAT: PHYSICAL MODEL 23
FIGURE 5: POLYGON-TO-LINE: EASY (KA) AND DIFFICULT (OO) OBJECTS 25
FIGURE 6: POLYGON-TO-POINT: RELATIVELY EASY (EG AND KH) 26
FIGURE 7: CENTROID POINT OF BEHEEROBJECT KH (IN MICROSTATION GEOGRAPHICS) 27
FIGURE 8: LINE-TO-POINT WITH INTERMEDIATE COMPUTATION OF A BUFFER (VISUALISATION IN MICROSTATION GEOGRAPHICS) 28
FIGURE 9: EXAMPLE OUTPUT OF ARC/INFO CENTERLINE EXTRACTION APPLIED TO BEHEERKAART-NAT SUBCLASS OEVERS (RIVER AND CANAL BANKS, SEA AND LAKE SHORES).....30
FIGURE 10: VISUALISE ALL THE BEHEEROBJECTS WHERE RD_CODE='LIMM' FROM VIEW V_BEHEEROBJECT.....34
FIGURE 11: VISUALISATION OF COLLECTION OF POLYGONS 38

DESIGN OF A LOGICAL AND PHYSICAL MODEL FOR THE BEHEERKAART-NAT

1 INTRODUCTION

This report presents the work completed on developing logical and physical data models for the Beheerkaart-Nat. The report provides an overview of the steps undertaken to import the data in the Oracle Spatial DBMS, provides arguments for the chosen solutions, discusses the results of the tests and concludes on issues which need further development and investigation. A summary of the steps to be followed when implementing the model is given at the end.

The new model for the Beheerkaart-Nat has to be able to deal with (as specified in the Offerte):

- Multiple geometry types, since some Beheerobject subclasses can be represented by different spatial types (point, line or polygon geometry).
- Multiple 'Layers' ('Lagen' in RWS-AGI terminology), because consistency rules (e.g. non-overlap) apply for a subset of objects.
- Multiple representation of objects, since different users may need less detail than others.

The new model (logical and physical) of the Beheerkaart-Nat has to address the above-mentioned problems by:

- Suggesting a solution for management of multiple geometries and an approach for deriving less detailed geometry. Investigating possibilities of using the NWB-Vaarwegen data set (network topology) for visualisation of the BV Beheerobjects in small scales.
- Ensuring consistency checks within particular 'Layers'.
- Investigating possibilities for an aggregation based on thematic (non-spatial) attributes.
- Investigating options for a dynamic segmentation of compound objects to analyse the attributes of the original aggregated objects.
- Possibilities for maintaining the history of all data values and/or monitoring changes in the data.
- Organisation of metadata.

2 DATA IMPORT AND VALIDATION

2.1 Data sets

The data sets available for this project are in the area of Maastricht, with approximate coordinates: X: (175307, 191494), Y: (307293, 353802). The Beheerkaart-Nat for the area is delivered as 5 different shape files namely *Bruggen*, *Winterbed*, *Laag (=Maaiveld)*, *Bodems* and *Ondergronds* (Figure 1, left). Three more data sets, i.e. DTB-Nat, Vaarwegvakken (containing navigation charts) (Figure 2, left), NVN (data set with the boundaries of the Dienstkringen) (Figure 1, right), and a data set with man-made objects represented by points (Figure 2, right).

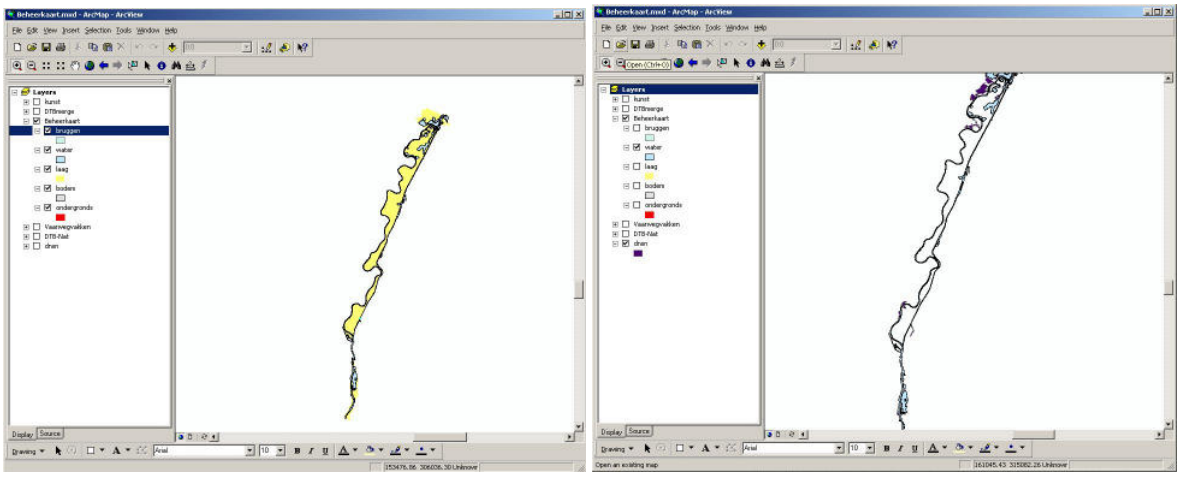


Figure 1: Beheerkaart-Nat (left) and Dienstkringen boundaries (right)

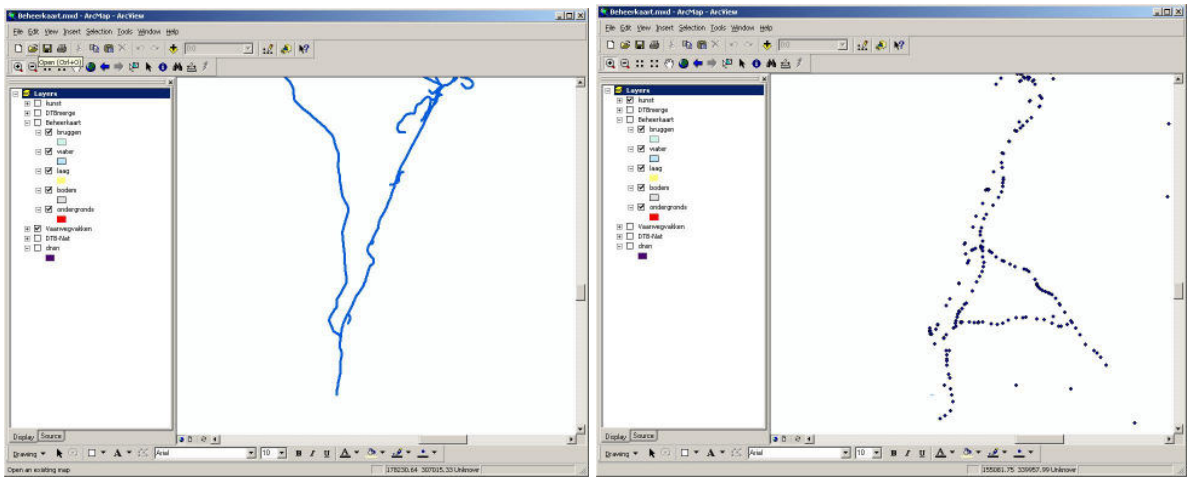


Figure 2: Navigation charts (left) and man-made point objects (right)

2.1.1 Data import

The shape files are imported in Oracle Spatial Object-oriented model using FME. Since all the objects imported in the database need to have a unique ID, this ID has to be created. The shape files as they were given to us do not contain IDs. The ID can be created in two

ways, before import in Oracle Spatial (in ArcGIS) or after importing (in Oracle Spatial using a special function for it). The two methods are briefly described below:

In ArcGIS: ArcGIS maintains and ID but it is internal and is not recognisable by FME. To avoid this problem, a new column ID (with the name: name_layer_ID, i.e. bodem_ID) is generated for all objects in ArcGIS and the ID from the internal column is copied in the new one. The new column is 'visible' for FME and can be used as ID in the Oracle tables.

In Oracle: The second approach is importing all the shapes without ID and creating an ID using Oracle functions. A mechanism exists in Oracle that allows creating sequential values. A '*sequence'* with a name (in this example *myseq*) has to be created and the starting value of the sequence has to be given (in this example 1).

```
drop sequence myseq;

create sequence myseq 1 nocache;
```

This sequence can be used to create an unique ID, e.g. in the table EV as follows:

```
create table tmp_ev as
select id, omschrijvi, oever, kmr, wd, geom
from VAARWEGMEUBILAIR;
--- selects needed values from the original table imported from shape file

drop table ev;

create table ev  as
select myseq.nextval as mslink,
tmp_ev.*
from tmp_ev;

drop table tmp_ev;
```

The sequence of values starting from 1 is created in the column 'mslink' (in this case, other names for ID (primary key) columns are of course also possible). The *sequence* has to be dropped and created again to be able to reset the starting value to 1 (or another starting value). Otherwise, the nextval function keeps increasing the values.

In this project almost all ID were created in ArcGIS, with two exceptions: EV and LAAG (for no particular reason; they were just received a bit later).

2.1.2 Import of shape files into Oracle with FME

An example of the FME mapping file is given in Appendix 1.

An example of the transformation section of all the shape files follows:

```
SHAPE bodem                                     \
    AREA                %AREA                   \
    PERIMETER           %PERIMETER              \
    OBJECTCODE          %OBJECTCODE             \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECT              %OBJECT                 \
    OBJECTNAAM          %OBJECTNAAM             \
    BODEM_ID            %BODEM_ID

ORACLE8I BODEM                                  \
    oracle_type         oracle_area             \
    AREA                %AREA                   \
    PERIMETER           %PERIMETER              \
    OBJECTCODE          %OBJECTCODE             \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECT_             %OBJECT                 \
    OBJECTNAAM          %OBJECTNAAM             \
    BODEM_ID            %BODEM_ID
```

```
-----------------------------------------------------------------------------------------------

SHAPE bruggen                                   \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    NAAM                %NAAM                   \
    IN_BEHEER_          %IN_BEHEER_             \
    BRUGGEN_ID          %BRUGGEN_ID

ORACLE8I BRUGGEN                                \
    oracle_type         oracle_area             \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    NAAM                %NAAM                   \
    IN_BEHEER_          %IN_BEHEER_             \
    BRUGGEN_ID          %BRUGGEN_ID
-----------------------------------------------------------------------------------------------

SHAPE laag                                      \
    AREA                %AREA                   \
    PERIMETER           %PERIMETER              \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    WSD                 %WSD                    \
    OBJECT              %OBJECT                 \
    LAAG_ID             %LAAG_ID

ORACLE8I LAAG                                   \
    oracle_type         oracle_area             \
    AREA                %AREA                   \
    PERIMETER           %PERIMETER              \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    WSD                 %WSD                    \
    OBJECT_             %OBJECT                 \
    LAAG_ID             %LAAG_ID
-----------------------------------------------------------------------------------------------

SHAPE ondergronds                               \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    AV                  %AV                     \
    VW3                 %VW3                    \
    RW                  %RW                     \
    OV                  %OV                     \
    ONDER_ID            %ONDER_ID

ORACLE8I ONDERGRONDS                            \
    oracle_type         oracle_area             \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    AV                  %AV                     \
    VW3                 %VW3                    \
    RW                  %RW                     \
    OV                  %OV                     \
    ONDER_ID            %ONDER_ID
-----------------------------------------------------------------------------------------------

SHAPE water                                     \
    AREA                %AREA                   \
    PERIMETER           %PERIMETER              \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    OBJECTNAAM          %OBJECTNAAM             \
    WATER_ID            %WATER_ID

ORACLE8I WATER                                  \
    oracle_type         oracle_area             \
    AREA                %AREA                   \
    PERIMETER           %PERIMETER              \
    COMPLEXCOD          %COMPLEXCOD             \
    OBJECTCODE          %OBJECTCODE             \
    OBJECTNAAM          %OBJECTNAAM             \
    WATER_ID            %WATER_ID
```

The import of shape files with FME is in general a two step operation: creating a mapping file and running it. The mapping file is a text file and can be edited if needed. For this project two manual corrections were made: 1) the size of the ID field (e.g. BODEM_ID) was set to number (38) to be able to visualise data sets in ArcGIS and 2) The dimensions of the data was changed to 3D to be compatible with the dimension of DTB-Nat.

The data set DTB-Nat consists of 114 smaller shape files. It is rather inefficient to import all the files into Oracle Spatial one by one. Therefore the shape files were merged in ArcView 3.2. (Note: This operation can also be done in ArcGIS 8 but then the scripts should be re-written in Visual Basic.)

The merged file as well as Dienstkring boundaries and point man-made objects is imported in Oracle Spatial using the approach described above. All the objects are stored using the Oracle Spatial geometry types point, line, polygon and collection (see Zlatanova et al, 2003 regarding the Oracle Spatial geometry model).

A column 'mslink' is created (with a content equal to name_layer_ID) to be able to visualise and edit the data in MicroStation GeoGraphics. This column is used as unique ID later in the scripts checking the validity of the geometries (see below and Appendix 3)

All tables are registered in the Oracle Spatial metadata table 'user_sdo_geom_metadata' and indexed with a 2D spatial index (example of the scripts is given in Appendix 2). These steps are necessary to be able to use the available Oracle Spatial functions.

2.1.3 Data validation

Three types of checks are performed for all spatial data as specified below:

1. validity according to the Oracle Spatial object-oriented models. The validation of the geometries with respect to the Oracle Spatial object-oriented model are performed in a script using the Oracle Spatial function VALIDATE_GEOMETRY. The function checks whether the following constraints are satisfied:
    - Polygons have at least four points, which includes the point that closes the polygon. (The last point is the same as the first.)
    - Polygons are not self-crossing.
    - No two vertices on a line or polygon are the same.
    - Polygons are oriented correctly. (Exterior ring boundaries must be oriented counter clockwise, and interior ring boundaries must be oriented clockwise.)
    - An interior polygon ring touches the exterior polygon ring at no more than one point.
    - If two or more interior polygon rings are in an exterior polygon ring, the interior polygon rings touch at no more than one point.
    - Line strings have at least two points.
    - 1-digit and 4-digit SDO_ETYPE values are not mixed (that is, both used) in defining polygon ring elements.
    - Points on an arc are not collinear (that is, are not on a straight line) and are not the same point.
    - Geometries are within the specified bounds of the applicable DIMINFO column value (from the USER_SDO_GEOM_METADATA view).
    - Geometries are within the extent of the coordinate system.

   The results of the test are given below under **Geometry validation**

2. constraints for non-overlapping within one layer (or table). Some of the layers have rules for non-overlapping objects (e.g. BODEM, LAAG, WATER). The results of this test is given under **Overlap of objects**.

3. two ‘Layers’ (‘Lagen’) have a very specific conditions between objects, i.e. ‘The sum of all objects from category Water has to be equal or larger than the area of Bodems’. These are also tested (see below under **QQ>=BV+BO+BH**)

The results obtained from the three tests are listed below:

Geometry validation. The tables listed below (DTBMERGE, LAAG and BODEM) contain invalid geometries. As mentioned above the column ‘mslink’ contains the unique ID of the object. The script performing this test is given in Appendix 3.

```
DTBMERGE:
MSLINK STATUS
--------------------------------------------------------------------------------
28894   13349 [Element <1>] [Ring <3>][Edge <9>][Edge <7>]
127440  13351 [Element <1>] [Ring 1][Edge <5>] [Element <3>] [Ring 1][Edge <3>]
160625  13349 [Element <1>] [Ring <1>][Edge <78>][Edge <289>]
170880  13349 [Element <1>] [Ring <1>][Edge <78>][Edge <289>]

LAAG:
MSLINK STATUS
---------- ------------------------------------------------
15 13356 [Element <1>] [Coordinate <72>][Ring <1>]
31 13356 [Element <1>] [Coordinate <2>][Ring <2>]

BODEM:
MSLINK STATUS
------ ------------------------------------------------
10 13356 [Element <1>] [Coordinate <5>][Ring <1>]
21 13356 [Element <1>] [Coordinate <24>][Ring <156>]
```

These geometries have to be examined (visually investigated) in the shape file and manually corrected. Oracle Spatial does not give clear message which of the rules is violated. Most often this is the self overlapping constraint. Since this investigation is outside the scope of the project, these objects were simply deleted. The geometries (correct and incorrect) are anyhow copied to new tables (DTBMERGE_ORIG, LAAG_ORIG, BODEM_ORIG) to be checked if necessary:

```
spool remove_errors.log
set echo on

create table DTBMERGE_ORIG as select * from DTBMERGE;
create table LAAG_ORIG     as select * from LAAG;
create table BODEM_ORIG    as select * from BODEM;

delete from DTBMERGE where MSLINK in (28894,127440,160625,170880);
delete from LAAG where MSLINK in (15,31);
delete from BODEM where MSLINK in (10,21);

commit;
spool off
```

Overlap of objects. The script for overlap is based on Oracle Spatial function SDO_RELATE with parameter ‘anyoverlap’. In the script, the objects are tested for ‘overlap’, the overlapping area is computed (i.e. AREA12) and the overlapping objects are listed in a new table. The script is given in Appendix 4. In the ‘Layers’ LAAG (=MAAIVELD) and BODEM were found the following objects with overlapping boundaries. These objects has to be again corrected in the original shape files or edited in a front-end that makes connection to the tables, e.g. Microstation Geographics or ArcGIS.

```
laag_ovrlap:
OID_1 OID_2 NUM_1 NUM_2          AREA_1          AREA_2    AREA_12
----- ----- ----- ----- --------------- --------------- ----------
   16    44     4     2    366887.33545      6858.34975   29.98383
   16    45     4     2    366887.33545      5324.93029   64.07861

bodem_ovrlap:
OID_1 OID_2 NUM_1 NUM_2          AREA_1          AREA_2    AREA_12
----- ----- ----- ----- --------------- --------------- ----------
    1     4     4     2     24707.20015      3989.40144    0.13089
    1    13     4     2     24707.20015   3563982.89172   16.98034
    2    29     2     8     30159.23455   1093999.73714   13.32447
```

```
    9    29     2     8     49542.22015   1093999.73714   19.90614
   11    23     2     2      6674.36933     48389.51654   20.30005
   13    30     2     2   3563982.89172   1076855.42634    0.84703
   16    29     2     8    903938.14559   1093999.73714   16.16681
   23    29     2     8     48389.51654   1093999.73714   16.94905
```

QQ>=BO+BV+BH. This test has shown that the data are correct. The area of all the objects in the corresponding 'Layers' is computed (Appendix 5) and apparently the constraint is fulfilled:

```
SQL> select sum(area) from bodem;

  SUM(AREA)
-----------
 7889013.05

SQL> select sum(area) from water;

  SUM(AREA)
-----------
16880573.16
```

2.2 Conclusion

The test described above has shown existence of inconsistent geometry, i.e. some of the polygons do not fulfil requirements either of Oracle Spatial geometry model or Beheerkaart-Nat model (non-overlapping objects in one layer). The errors are insignificant for DTB-Nat (3 out of 172771) and more significant for Beheerkaart –Nat (LAAG 4 out 58; BODEM 10 out of 30). The polygons of the Beheerkaart-Nat are relatively simple (compared to DTB-Nat) and therefore we expect that the corrections should not be a problem.

3 LOGICAL MODEL

3.1 Current problems

The Beheerkaart-Nat contains objects of interest for 10 different RWS Regionale Directies (RD) and 27 Dienstkringen (DK). The scale of Beheerkaart-Nat is usually 1:1000 (with some exceptions for some regions where the scale is 1:5000). In general the Beheerkaart-Nat is created in ArcView and ArcInfo, as boundaries of the objects are mostly closed polygons (stored as shape files). The coordinates are 2D. Short description of the current organisation and existing way of maintenance are given below.

Maintenance and type geometry: The data are created and maintained by individual DK. Although agreements on the format of the data and the number of 'Layers' exist, some DK maintain slightly different information per object. For example DK Maastricht maintains 'complexcod' which is modification of the object code (according to the classification in BPN). Some of the DK maintain information about the hoofdwatersysteem (HWS) and others not. The geometries are mostly polygons, but other types of objects (points and lines) may also exist. Moreover when less-detailed visualisations are needed, some polygon beheerobjects may change to lines and points (see Table 2).

Code: Each Beheerkaart-Nat object is assigned a code. The code consists of four sections: 1) WaterSysteemDeel (WSD) the objects falls in (four characters), 2) category of the objects (two characters) according to a classification, 3) position of the object (left, right) with respect to the river stream and 4) distance in km along the rivers (given with numbers). This coding allows links to the TISBO administrative system to be made. The code is currently maintained in one field as a string.

Hierarchy: It can be distinguished between four types of hierarchy (grouping) of Beheerobjects:

- Water systems: In principle the objects has to be organised a hierarchy within the water management system in Netherlands, which consists of 6 levels (*hoofdwatersysteem*, *watersysteem*, *watersysteemdeel*, *complex*, *beheerobject* and *onderdeel*). Practically only 5 levels are in use, i.e. *onderdeel* is not maintained. Since complex is also specific type of hierarchy (applicable for a subset of objects), only the four levels (HWS, WS, WSD and BO) are of interest for all users. As mentioned above, WSD is given in the code. The links between other water subdivisions in stored in a separate database (in MSAccess).
- The beheerobjecta can be also grouped with respect to the DK and RD.
- Layers: The Beheerobjects can be grouped in 5 'Layers': BODEM, BRUGGEN, LAAG/MAAIVELD, ONDERGRONDS and WATER. The obeheerobjects with one layer should not overlap. The beheerobjects in one layer are selected with respect to the two-character category. This grouping is used to check mostly data consistency.
- Category: The beheerobjects can be also grouped according to the category in 6 large groups namely kunstwerk (K), bodems (B), oevers (O), water (Q), exploitatie (E) and scheepvaartbegeleiding (V). This grouping is used for visualisation purposes. The categories are given in Table 2, first column.

Metadata: Metadata has to be maintained in three levels: per beheerobject (object), related to DK (dienstkring) and for the entire Beheerkaart-Nat (product). Table 1 outlines the most important metadata information:

Table 1: Metadata (P.v.Asperen)

Metadataveld	Niveau	Invulling
CEN:		
Dataset naam	Product	Beheerkaart-Nat
Samenvatting	Product	Geografische vastlegging BPN-objecten
Dataset producent	dienstkring	naam dienstkring
Ruimtelijk schema	Product	Volledige topologie (G1)
Taal van de dataset	Product	Nederlands
Dataset tekenset	Product	ISO 8859-1
Dataset herkomst	dienstkring	bronvermelding opbouw Beheerkaart: gedigitaliseerd vanaf DTB-Nat, TOP10vector, etc.
Inwinningsdatum	dienstkring	Datum opbouw Beheerkaart-Nat
Metadata invoerdatum	Product	gelijk aan datum invoer in geodatabase
Status van de decking	Product	Actueel
Actualiteit van de decking	dienstkring	Datum laatst ingevoerde wijziging
Datatype	Product	Oracle spatial
Toepassing datatype	Product	ArcGIS, GeoServices
Toe te voegen:		
Wijzigingsdatum	Object	Datum invoer wijziging
Naam contactpersoon	Object	naam contactpersoon wijziging
Dataset/object herkomst	Object	bronvermelding wijziging

Visualisation and editing: Beheerobjects have to be available for editing and visualisation for users in DK and only for visualisation for other users. Some of the users may require less detailed information and simplification of objects. Table 2 gives an idea of the visualisation requirements. Some of the geometries for less-detailed visualisation can be linked to already existing data sets. Such examples are Beheerobjects with category BV and the NWD data set; the man-made point object and the Beheerobject with category K.

Table 2: Visualisation requirements (P. v. Asperen)

Code	Objectsubcategorie	Niveau	Beheerkaart	> 1:25000	1:25000-1:250000	<1:250000
	user			**DK (WSD)**	**RD (WS, HWS)**	**RWS (HWS)**
KA	Aanleginrichting	3	vlak	vlak	lijn	-
KB	Beweegbare brug	1	vlak	vlak	vlak	punt
KG	Gemaal	3	vlak	vlak	punt	-
KH	Hoogwaterkering	3	lijn/vlak	Lijn/vlak	lijn	punt
KN	Kunstwerken natuur	3	lijn/vlak	Lijn/vlak	punt	-
KQ	Aquaduct	5	vlak	vlak	vlak	-
KR	Waterreguleringswerk	3	lijn/vlak	Lijn/vlak	punt	-
KS	Schutsluis	3	vlak	vlak	vlak	punt
KT	Tunnel	5	vlak	vlak	vlak	-
KU	Uitwateringssluis	3	vlak	vlak	vlak	-
KV	Vaste brug	1	vlak	vlak	vlak	punt
KW	Stuw	3	vlak	vlak	lijn	punt
KZ	Sifon/duiker/hevel	5	lijn/vlak	Lijn/vlak	lijn	-
BV	Bodem vaarweg	4	vlak	vlak	vlak	Lijn (NWB)
BO	Bodem overig	4	vlak	vlak	vlak	-
BH	Bodem haven	4	vlak	vlak	vlak	-
OD	Strekdam	3	vlak	vlak	lijn	-
OH	Oever haven	3	lijn/vlak	Lijn/vlak	lijn	-
OK	Kribben	3	vlak	vlak	lijn	-
OL	Leidam/leikade	3	lijn/vlak	Lijn/vlak	lijn	-
OO	Oevers/dijken	3	lijn/vlak	Lijn/vlak	lijn	-
OS	Strandhoofden	3	Vlak	vlak	lijn	-
OU	Uiterwaarden	3	Vlak	vlak	vlak	vlak
OV	Kribvakken	3	Vlak	vlak	vlak	vlak
OW	Waddenzeekwelders	3	Vlak	vlak	Vlak	vlak
QQ	Water	2	Vlak	vlak	Vlak	vlak
QX	Meetsysteem	2	punt/vlak	Punt/vlak	Punt	-
EG	Gebouwen	3	Vlak	vlak	Punt	-
ET	Terreinen	3	Vlak	vlak	Vlak	-
VS	Scheepvaartbegeleiding	3	punt/vlak	Punt/vlak	Punt	-

Queries: Besides the queries for visualisation (with respect to a particular attribute) and editing, no specific true spatial operations (i.e. buffer, neighbouring objects, area, length) are required. Area and length in the current situation (shape files) are explicitly stored. However, in many cases the length (in stead of the area) is required for polygon objects. A typical example of such objects is dikes (long polygons). The length of these objects can be computed on the basis of the line approximation of the polygon.

The problems with the current management of the Beheerkaart-Nat can be specified as follows:

- One category Beheerobject can be represented with different geometries: There are a number of beheerobjects represented by both line and polygons (*hoogwaterkering*, *waterreguleringswerk*, *oevers*, etc) and point and polygon (*scheepvaartbegeleiding*). Furthermore, the representation may need to change as the level of detail for visualisation is needed. The different users of RWS are interested in different scales. For example some managers may need to have only a map with the *watersysteem*, while Dienstkringen will need to work at level *beheerobject.* An approach for maintenance of multiple representations has to be developed.
- There are several hierarchies: The Beheerkaart-Nat objects can be grouped in different hierarchy levels with different constraints. The general question is how to both: ensure data consistency and maintain all hierarchies.
- History has to be maintained: Currently, the history is maintained within the different variants of shape files. An appropriate solution is needed for registering and tracking changes.
- Metadata information. The metadata has to be organised.

A new integrated model has to be developed to be able to resolve multiplicities in object geometries, 'Layers' and hierarchies, time changes and metadata.

3.2 Logical design

Based on analysis of the data, discussions with AGI specialists and experience of TUDelft gained with other geo-database projects we suggest an organisation as follows**:**

A class named Beheerobject is created which will represent all the objects of the Beheerkaart-Nat. Attributes of this object are object_ID, geometry (multiple geometries), code, time and the part of the metadata information related to an individual object.

BEH_ID (i.e. the object_ID) gives identification of the object and it is preserved even if the object is changed (see the time attribute below). The BEH_ID is automatically created when the object is imported in the model.

We propose the three representations of one object (as given in Table 2) to be explicitly maintained. This approach is chosen because the three geometries vary significantly per category and no efficient classification based on geometry could be defined. The first spatial attribute is GEOM and it contains geometry available for DK (visualisation and editing). Vis1_Geom and Vis2_Geom contain geometries as they are specified per category and per visualisation level. Some of the values in Vis2_Geom will be NULL. Constraints on editing geometries in Vis1_Geom and Vis2_Geom have to be created. In many cases changes in base geometries will require reconsidering the geometries for visualisation. Maintaining consistency between different representations as well as possibilities for automatic generation of some of them is topic of further research and investigation.

The original 17character Code is suggested to be split into four parts Code_WSD, Code_CAT, Code_ZIJDE and Code_KM containing the codes of WSD, category of objects,

zijde (links or right on the river) and km along the river. The separation into four individual groups is suggested to prevent introducing typing errors. Previous tests with Beheerkaart-Nat have clearly shown the risk of introducing errors in the numerical part of the Code. For example: often instead of a decimal point, a comma is typed as separator (Zlatanova et al, 2003).

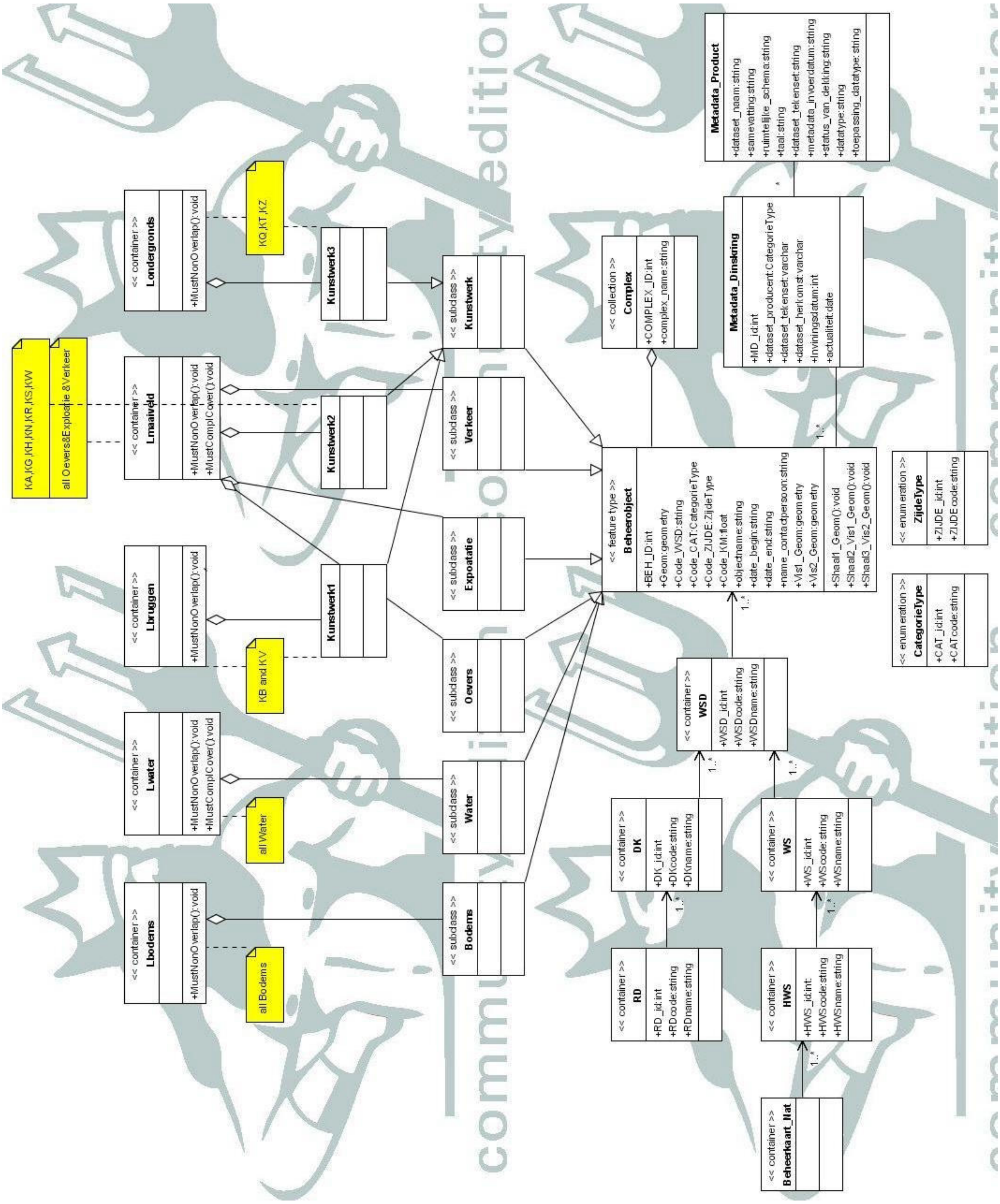


Figure 3: Beheerkaart-Nat: Logical model

Keeping track of modifications (edits) in the database (maintenance of history). The history is maintained by introducing *begin_date* and *end_date* containing begin and end of a certain status of an object. When an object is inserted in the database, the begin_date is automatically filled, the end_date has value NULL. When an attribute is modified, the end_date of the object is filled (the object 'dies') and a new object (new status of the same object) is created. Having the same ID for all the previous variants of an object, it will be very easy to get oriented how a certain object has changed over time.

Metadata information is represented as two separate classes (related to Dienstkring and to the Beheerkaart-Nat). The metadata per beheerobject is actually organised as attributes in beheerobject (i.e. end_date and begin_date). Therefore the name of the person made the changes is also maintained there.

The four hierarchies based on the Beheerobject are modelled either as separated classes or as generalisations. The five 'Layers' for consistency check (Lbodems, Lwater, Lbruggen, Lmaaiveld and Londergronds) are given in the UML model with their constrains (dependent on the category of the objects). Kunstwerk is subdivided into three different subclasses Kunstwerk1, Kunstwerk2, and Kunstwerk3 for readability.

3.3 Conclussion

The integrated model suggested above has the following characteristics:

- Beheerobject is the core class
- Time changes are maintained
- Metadata about object, DK and Beheerkaart-Nat are included
- Four hierarchies of interest for Beheerobjects are clearly showed. Analysis and visualisation on the basis of these can be easily specified (see physical model and testing).
- Multiple geometries are explicitly stored (further research is needed for consistency check). Some of the geometries can be linked with existing data sets (see physical model and testing)

4 PHYSICAL MODEL AND IMPLEMENTATION

4.1 Physical modelling

The physical modelling is done with respect to the functionality offered by Oracle Spatial. The logical model is mapped to 10 tables (7 of which are Look-Up-Tables, with codes and descriptions of codes) and 5 views (Figure 4). All the tables have unique IDs defined as primary key. In the case of the Look-Up-Tables the primary key is used (via foreign key constraints of the Beheerobject table) in joins with the Beheerobject table to find the descriptions for the codes in the Look-Up-Table and to act as validity check during editing. None of the codes is used as primary key, which will allow future refinement of the code abbreviations.

The *Beheerobject* class is implemented as the table *Beheerobject* and all the class attributes are implemented as table fields. When all versions of a Beheerobject are kept in the table (when history is maintained, see Section 6.5) the BEH_ID will not be unique. Therefore a new ID is introduced: MSLINK with data type NUMBER(38). The name and the size are selected to make possible visualisation in MicroStation GeoGraphics and ArcGIS/ArcSDE. MicroStation needs a column named MSLINK and ArcGIS requires a size of exactly NUMBER(38) when a connection to Oracle is created with 'ArcSDE direct connect'. (Remark: in some of the scripts in this Report the reader will see the column REC-ID as primary key. This needs to be replaced by MSLINK when the decision is taken to have MicroStation as editing and/or viewing application.)

The fields WSD, Category and Zijde from Code are defined as foreign keys and linked with the WSD table and the two Look-Up-Tables CategoryType and ZijdeType. A link to the Metadata_DK table is included, also via a foreign key to the MD_ID.

The tables representing the hierarchies (RD, DK, HWS, WS and WSD) are linked by their IDs (using the structure primary-foreign key). The table WSD has references by two foreign keys to the water system hierarchy and administrative division of RWS.

The 'Layers' *Lbodems*, *Lwater*, *Lbruggen*, *Lmaaiveld* and *Londergronds* are defined as views having the same attributes as *Beheerobject* table.

Table CategoryType and ZijdeTupe are LUP tables containg the values of the categories and Zijde (part of the BPN code).

Metadata are organised in one table *Metadata_DK* (in contract to the logical model) because the Matadata_product class contains only one record. The *dataset_producent* (in the Logical model) is the Dienstkring and therefore DK_ID is used that makes link to the table with all the data of Dienstkring.

4.2 Implementation and population

Beheerobject table is created using a PL/SQL script that unites all the records from all the original tables (coming from the shape files). The operation performed is actually simple UNION, but because UNION does not work with nested tables and varrays (Oracle Spatial data types are implemented by varays), a script (reading and inserting records) has to be created. The script is given in Appendix 7. As it can be realised with this script only the data available in the original tables are copied in the new table. The column date_begin is filled in with the system time, data_end and name_contact person are empty. The two columns related to the visualisation will be partially filled with data after performing some tests (see next section)

The table is indexed and registered in the Oracle user_metadata table.

The tables RD, DK, WSD, HWS and WS are imported from MSAccess database as they were provided by AGI.

The table with the Metadata is created according to the schema.

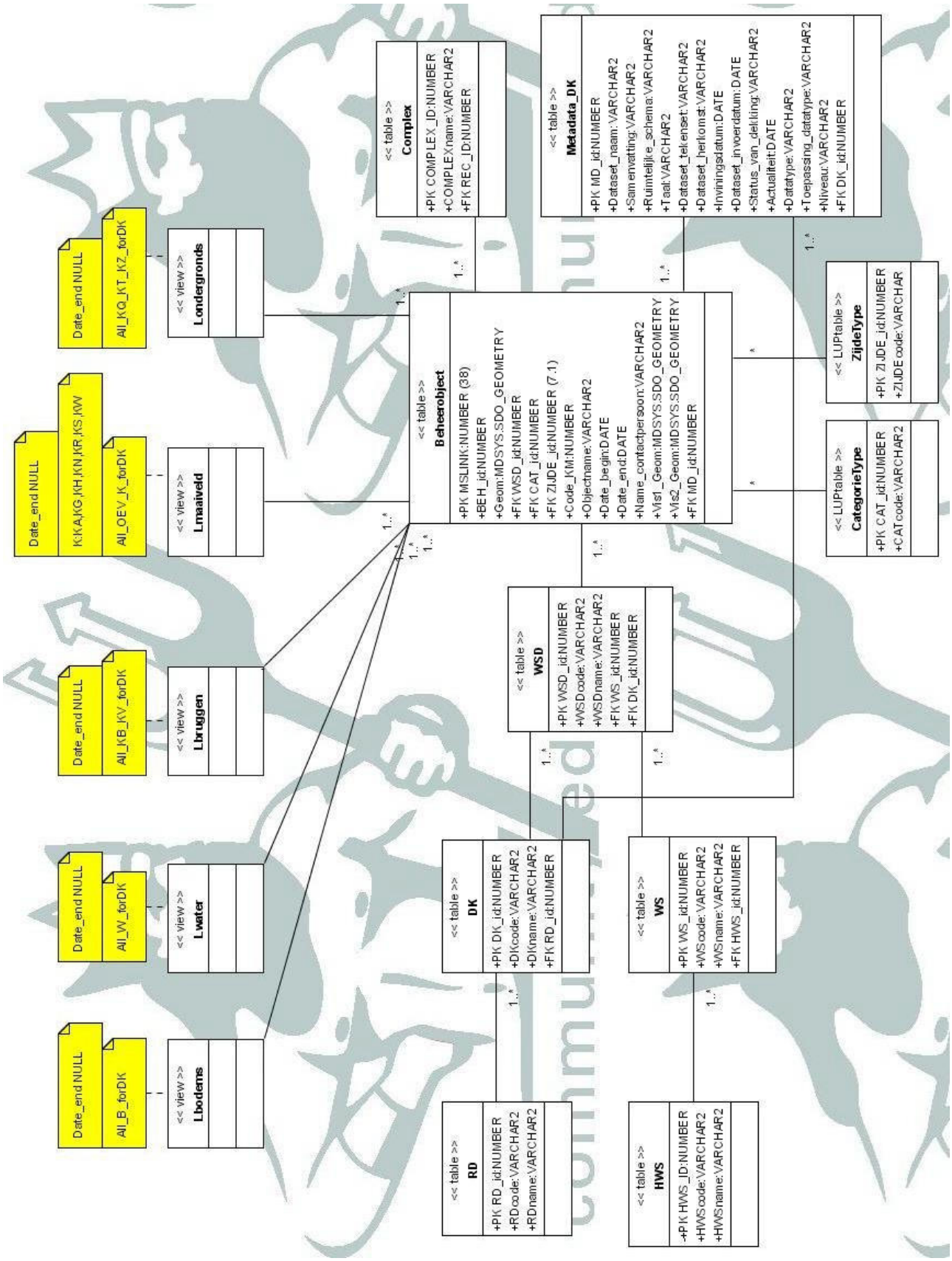


Figure 4: Beheerkaart-Nat: Physical model

The implementation of tables BEHEEROBJECT, METADATA_DK, ZIJDETYPE and CATEGORIETYPE and the view V_BEHEEROBJECT is as follows:

```
SQL> descr beheerobject;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 REC_ID                                             NUMBER(38)
 BEH_ID                                             NUMBER
 OBJECTNAME                                         VARCHAR2(40)
 WSD_ID                                             NUMBER
 CAT_ID                                             NUMBER
 ZIJDE_ID                                           NUMBER
 CODE_KM                                            VARCHAR2(21)
 DATE_BEGIN                                         DATE
 DATE_END                                           DATE
 NAME_CONTACTPERSON                                 VARCHAR2(30)
 GEOM                                               MDSYS.SDO_GEOMETRY
 VIS1_GEOM                                          MDSYS.SDO_GEOMETRY
 VIS2_GEOM                                          MDSYS.SDO_GEOMETRY
 MD_ID                                              NUMBER


SQL> descr metadata_dk;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MD_ID                                     NOT NULL NUMBER
 DATASET_NAAM                                       CHAR(11)
 SAMENVATTING                                       CHAR(15)
 DATASET_PRODUCENT                                  CHAR(50)
 RUIMTELIJK_SCHEMA                                  CHAR(3)
 DATASET_TEKENSET                                   CHAR(3)
 DATASET_HERKOMST                                   CHAR(3)
 DATASET_INVOERDATUM                                DATE
 STATUS_VAN_DEKKING                                 CHAR(3)
 ACTUALITEIT                                        CHAR(3)
 DATATYPE                                           CHAR(3)
 TOEPASSING_DATATYPE                                CHAR(3)


SQL> descr zijdetype;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ZIJDE_ID                                  NOT NULL NUMBER
 ZIJDECODE                                          VARCHAR2(1


SQL> descr categorietype;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CAT_ID                                    NOT NULL NUMBER
 CATCODE                                            VARCHAR2(2)


SQL> descr v_beheerobject;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 REC_ID                                    NOT NULL NUMBER(38)
 BEH_ID                                             NUMBER
 OBJECTNAME                                         VARCHAR2(40)
 WSD_CODE                                           CHAR(4)
 WSD_NAAM                                           CHAR(40)
 CATCODE                                            VARCHAR2(2)
 ZIJDECODE                                          VARCHAR2(1)
 CODE_KM                                            VARCHAR2(21)
 DATE_BEGIN                                         DATE
 DATE_END                                           DATE
 NAME_CONTACTPERSON                                 VARCHAR2(30)
 GEOM                                               MDSYS.SDO_GEOMETRY
 VIS1_GEOM                                          MDSYS.SDO_GEOMETRY
 VIS2_GEOM                                          MDSYS.SDO_GEOMETRY
 MD_ID                                              NUMBER
 WS_CODE                                            CHAR(2)
 WS_NAAM                                            CHAR(35)
 HWS_CODE                                           CHAR(3)
 HWS_NAAM                                           CHAR(30)
 DK_CODE                                            CHAR(50)
 DK_NAAM                                            CHAR(50)
 RD_CODE                                            CHAR(2)
 RD_NAAM                                            CHAR(25)
```

The view joins six tables namely RD, DK, HWS, WS, ZIJDETYPE, CATEGORIETYPE and BEHEEROBEJCT. This view is convenient for query because all the required codes (used for aggregation of data) are available to the user at once. Such views however may have effect on the performance. If a query needs data from less than all 6 tables, joining the tables in the select statement performs better (see Section Tests, 5.2 Examples of queries).

Yet another approach to avoid join of many tables is to keep some of the tables non-normalised. For example the data for RD and DK, as well as, HWS and WS can be stored in one table. Thus the number of tables and consequently the joins will be reduced. Theoretically this is not recommended, since the control of the codes (maintained by the foreign keys) will be violated. The codes of RD, DK, HWS and WS are not that often modified and this consideration is not that critical. In our implementation, we keep the normalised tables as it is specified in the UML model.

4.2.1 Creating Vis1_Geom, Vis2_Geom from original geometries

There are different approaches to create these geometries from already existing ones depending on the type of the geometry and the required new representation. The following three combinations are possible for the Beheerkaart-Nat objects:

- polygon to line (**KA**,**KW**,**KH**,KN,**KZ**,**BV**,OD,OK,OS, OH,OL,**OO** and QX)
- polygon to point (KB,**KS**,**KG, KR**,QX,**EG**,VS)
- line to point (**KW**,**KH**,KN,**KR**)

The optimal solution will be indeed an automatic creation. In general, the objects from the different categories have to be studied and individual approach has to be selected for each category. In the test data set were only the categories given in bold. The visual examination has shown that in most of the cases (KA, KW, KH, KZ and BV) polygon-to-line generalisation should not be a problem in terms of shape. The objects are thin and the direction of the line would be well defined. Figure 5 shows an example of category KA. Such objects can be easily 'shrink' to lines (algorithms for this exist, perhaps also PL/SQL scripts).

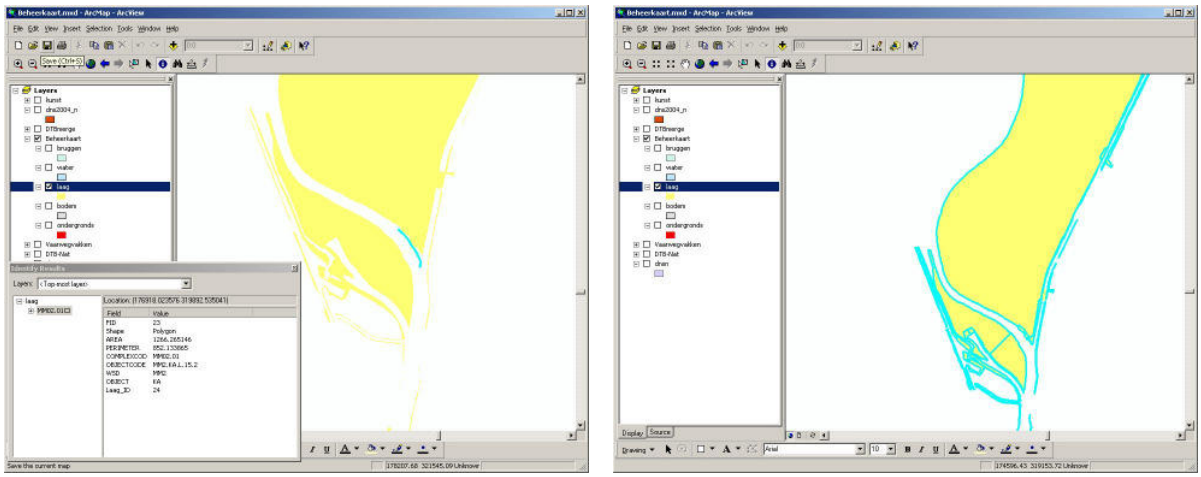


Figure 5: Polygon-to-line: easy (KA) and difficult (OO) objects

The most difficult group is OO. Many objects of this category are large areas with undefined shape.

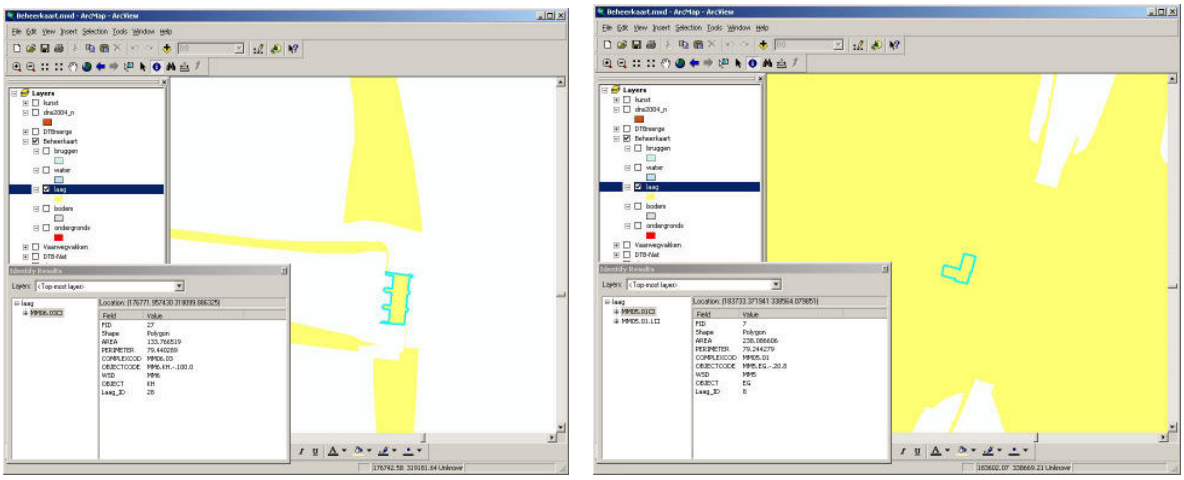


Figure 6: Polygon-to-point: relatively easy (EG and KH)

Polygon-to-point is a relatively easy operation because the middle point of the object can be considered as a new representation. It can be computed with Oracle Spatial function SDO_GEOM.SDO_CENTROID. For example the centroid point of the KH object (only one) hown in Figure 6 will be computed with the following select statements (the query uses the extended view V_BEHEEROBJECT):

```
select c.rec_id as id, sdo_geom.sdo_centroid (c.geom, m.diminfo) as point
from v_beheerobject c, user_sdo_geom_metadata m
where m.table_name='V_BEHEEROBJECT' and m.column_name='GEOM' and
c.catcode='KH';
```

or if the tolerance is explicitly mentioned (e.g. 0.0005):

```
select rec_id as id, sdo_geom.sdo_centroid (c.geom, 0.0005) as point
from v_beheerobject where catcode='KH';
```

The result is:

```
 ID        POINT(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
 102     SDO_GEOMETRY(3001, NULL, SDO_POINT_TYPE(176772.231, 319098.172, 0), NULL, NULL)
```

The object is stored in a view CENTROID. Figure 7 shows the original object and the computed point. Most of the objects that have to be represented as points are compact and the centroid point represents the object optimally.

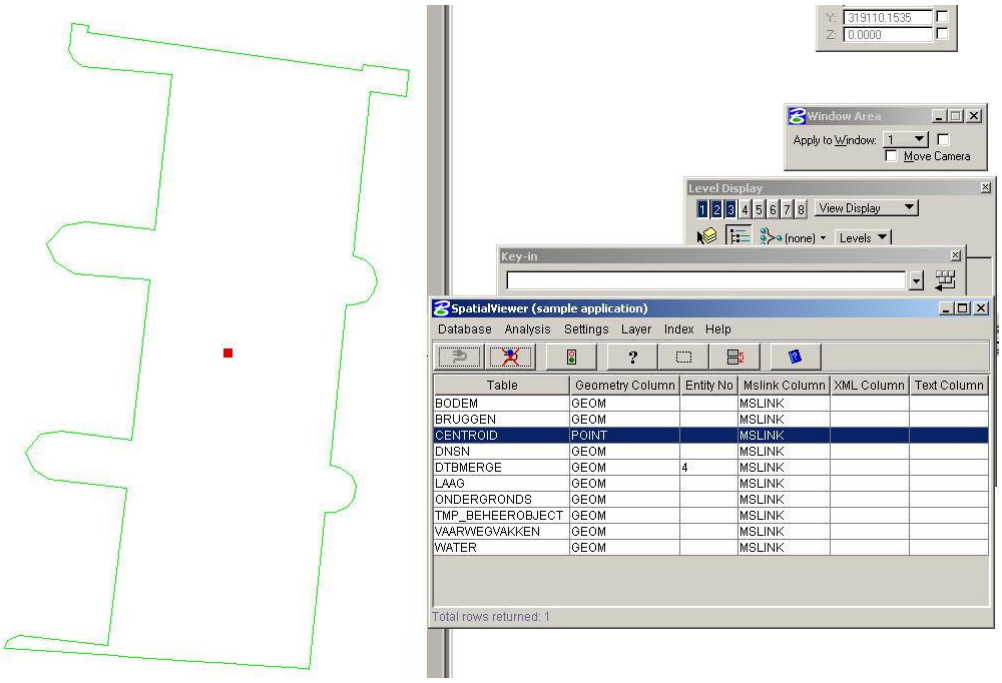


Figure 7: Centroid point of beheerobject KH (in MicroStation GeoGraphics)

The last case line-to-point needs more attention. The function SDO_GEOM.SDO_CENTROID does not works with lines, but only with polygons and points. One option will be to create a buffer polygon around the line (SDO_GEOM.SDO_BUFFER) and then find the centroid point of the buffer. SDO_GEOM.SDO_BUFFER returns always a polygon data type. Since the test data set does not contain line object, a line object from the NWB data set was used for testing. Figure 4 shows the results of the modelling (the script is in Appendix 7). It might happen that the line has a very complex shape and the centroid point will fall outside the line. The centroid point has to be 'projected' to the line. Oracle Spatial offers spatial functions (within the Linear Referencing System) that allow this operation to be completed relatively easily. The Linear Referencing System operates only with lines and points. The function that is needed in this case is SDO_LRS.PROJECT_PT. Given input parameters (e.g. a line and a point), it returns the projection of the point on the line. An example how the function works (Oracle Spatial 9i manuals) is given below:

The function returns the point (9,4,9) on the geometric segment representing Route 1 that is closest to the specified point (9,3,NULL).

```
-- Point 9,3,NULL is off the road; should return 9,4,9

SELECT  SDO_LRS.PROJECT_PT(route_geometry,
  MDSYS.SDO_GEOMETRY(3301, NULL, NULL,       ----line
     MDSYS.SDO_ELEM_INFO_ARRAY(1, 1, 1),
     MDSYS.SDO_ORDINATE_ARRAY(9, 3, NULL)) )  ---point
  FROM lrs_routes WHERE route_id = 1;

SDO_LRS.PROJECT_PT(ROUTE_GEOMETRY,MDSYS.SDO_GEOMETRY(3301,NULL,NULL,MDSYS.SDO_EL
--------------------------------------------------------------------------------
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
9, 4, 9))
```

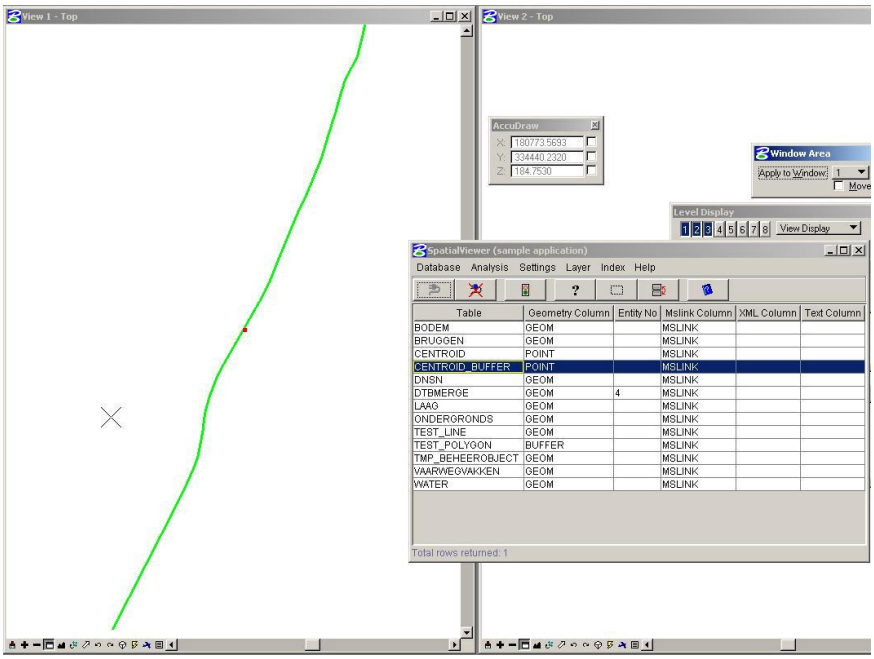


Figure 8: Line-to-point with intermediate computation of a buffer (Visualisation in MicroStation GeoGraphics)

4.2.2 Creating Vis1_Geom, Vis2_Geom by linking to existing data sets

The second way to obtain geometries is by linking them to existing data sets. Two such data sets were provided for testing namely, point data set for all the kunstwerken (for the entire area of the Netherlands) and NWB data set. Tests should clarify whether the accuracy of the data set is sufficient to link point data sets to the Beheerobjects of catecory Kunstwerk ('K') and NWB to BodemVaargeul ('BV'). The test were completed in several steps:

1. The two data sets were imported in Oracle Spatial from shape files using FME. The corresponding tables (KUNST and VAARWEGVAKKEN) were registered in the metadata tables and indexed.
2. In the table KUNST a new column KUNST_ID with unique ID was created. In the VAARWEGVAKKEN the column VAAR_ID was used for a unique ID.
3. The two tables were validated. It appeared that the tolerance for validation has to be made smaller (0.0005) to avoid duplication of points.
4. Since the two data sets contain data for the whole Netherlands, a selection within only the test area was made. Thus only 206 points (KUNST) and 39 lines (NWB) were considered for the tests. The subsets were stored in new tables KUNSTWERKEN and VAARWEGEN. The new tables were also registered in the metadata and indexed.
5. A PL/SQL script (Appendix 8 and 9) was used to check Point-in-Polygon and Line-in-Polygon relations between the data sets.

The results of the matches Point-in-Polygon are stored in a table. Only 5.34 % (11of 206) of the objects can be matched using overlap.

```
select * from beheer.kunstwerken_match order by oid_pnt,oid_area;

   OID_PNT   OID_AREA RELATION
---------- ---------- -------------------------------
      4425        115 INSIDE
      4445         85 INSIDE
      4451         97 INSIDE
      4452         97 INSIDE
      4453         97 INSIDE
      4765         82 INSIDE
```

```
      4778         86 INSIDE
      4782         87 INSIDE
      5004         99 INSIDE
      5005         99 INSIDE
      5006        100 INSIDE

11 rows selected.
```

The results clearly show that a large amount of points are outside of the polygons. In addition all kind of possibilities can be also seen. For example more than one point can be found in one polygon, or a point lies in the border of two polygons. Apparently, the simple spatial overlap is not sufficient to take a decision if two object are the same. In addition to the geometry check, the description of the points in the KUNSTWERKEN has to be considered and a additional search in a given area has to be performed. The query should be of the type: 'find all the points with a particular description type that are within a given distance from the polygons of interest'. Unfortunately, the points of KUNSTWERKEN do not have BPN codes. Some manual pre-referencing of KUNSTWERK descriptions and BPN codes, however is possible.

The results of checking Line-in-Polygon are a bit better (28.21%) but the results are more difficult for interpretation. The matches are also stored in a table:

```
select * from beheer.vaarwegen_match order by oid_line,oid_area;

  OID_LINE   OID_AREA RELATION                          PCT_INSIDE
---------- ---------- -------------------------------- ----------
      1259        159 OVERLAPBDYDISJOINT                   4.6166
      1473        159 OVERLAPBDYDISJOINT                   2.4521
      1657        146 OVERLAPBDYDISJOINT                   1.3835
      1657        158 OVERLAPBDYDISJOINT                   8.1486
      2193        159 INSIDE                             100.0000
      2274        141 OVERLAPBDYDISJOINT                   0.1016
      2274        142 OVERLAPBDYDISJOINT                   3.3279
      2318        144 OVERLAPBDYDISJOINT                  32.0711
      2318        159 OVERLAPBDYDISJOINT                  46.5604
      2320        159 INSIDE                             100.0000
      2661        141 OVERLAPBDYDISJOINT                  17.8487
      2661        152 OVERLAPBDYDISJOINT                  69.9627
      2661        158 OVERLAPBDYDISJOINT                  12.3407
      2731        158 INSIDE                             100.0000
      3050        158 INSIDE                             100.0000
      3051        140 OVERLAPBDYDISJOINT                  27.9569
      3051        158 OVERLAPBDYDISJOINT                  17.0485

17 rows selected.
```

The NWB lines are not related to the Beheerobjects in category BV and therefore the NWB lines that match can be either completely inside or partially inside. For completely inside lines, it can be easily concluded that they can be linked to the corresponding beheerobject. More complicated is the decision how to link NWB lines that are partially matched with BV objects. The relationship is in principle m:m as all the combinations are possible. For example line 2661 is matched to thee beheerobjects (141,152,158) while beheerobject 159 is matched to two NWB-lines (1259 and 1473). Prior suggesting appropriate organisation of these two objects, a larger subset of matched objects has to be tested and further analysed.

4.3 Conclusion

Based on the tests, it is beneficial to consider automatic generation of geometries. As mentioned in the previous Section the automatic generation of points from polygons or lines can be easily done using available functions. The more serious task to be resolved is the conversion from Polygon-to -Line.

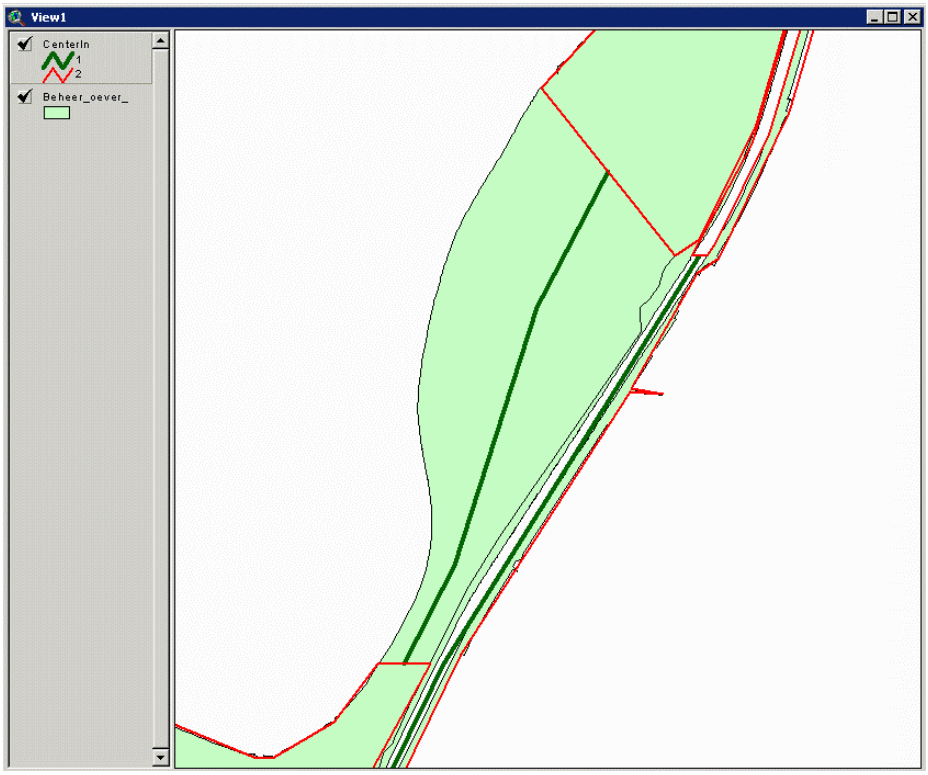


Figure 9: Example output of Arc/Info centerline extraction applied to Beheerkaart-Nat subclass Oevers (river and canal banks, sea and lake shores)

This is one of those 'problems' for which a multitude of algorithms exist but hardly any implementations (e.g. Chithambaram, R. et al, 1991, DeLucia, A., en T. Black, 1987, Oosterom, P.J.M. van, 1999, Uitermark, H. et al, 1999, F. Penninga, 2003, Berg, M. de, et al, 1997, Steven A. Roberts, G. Brent Hall, Barry Boots, 2004). All vector-based 'skeletonization' or 'centerline extraction' methods take as their starting point a Delaunay-triangulation, or a Voronoi-diagram, or the Thiessen polygons, or the Medial Axis Transformation of the relevant areas. These are closely related, or even complementary, structures. But the centerline generated from the structures normally does not 'look' right from the perspective of the user, e.g. at T-shape 'junctions' and at the end of the areas unwanted line segments result. So following the initial calculations a second phase of cartographic generalization is necessary before a satisfactory result can be expected. During this phase all the difficulties that have hampered the progress of 'intelligent' generalization in the past decades are occurring. ESRI's Arc/Info software contains a 'centerline' command to automatically generate centerlines from 'casings' (e.g. road polygons with straight, nearly parallel boundaries). It is not very well suited for irregular polygons like the 'oever' areas of Beheerkaart-Nat (Figure 9: red line color means that the command cannot calculate a centerline; green means a correct centerline according to the command, but in reality most of these lines are incorrect).

The data sets with point KUSNTWERKEN and NWB can be in principle linked to the corresponding objects but a more elaborated study is required to clarify the best sequence of steps.

Bearing in mind the results of the tests, the geometries in columns Vis1_Geom and Vis2_Geom where created from existing geometries according to the Table 2 as follows:

- All the polygons that remain polygons were copied from the basic geometry (in column GEOM)

- All the polygons that change to points were created automatically with the help of CENTROID function
- Lines are not created (the geometry is empty).

The script is based on the following update commands:

```
update beheer2 set vis1_geom=geom where substr(objectcode,5,2)='EV'; ----copy geometry

update beheer2 set vis1_geom= sdo_geom.sdo_centroid (geom, 0.0005) where
substr(objectcode,5,2)='VS';                                  ----- centroid point
```

5 TESTS

5.1 Types of queries

The queries performed most often within AGI can be specified as:

- Query (with respect to attributes) and visualisation of objects for different users (DK, RD or RWS). The visualisation for the different users may require use of different geometries (as given in Vis1_Geom and Vis2_Geom).
- Query and visualisation of history.
- Quantity, area and length computation for Beheerobjects of a given category
- Editing (also deleting and adding new) Beheerobjects and their attributes. Most of the cases is only editing text data, but it could be possible changes of the geometry to be required.
- Query of metadata.

An important aspect of the tests is the type of the front-end engine that will be used to visualise the result of the queries. It can be distinguished between three general levels of visualisation:

Database level: This means that the user will execute SQL statements (and/or scripts) from an Oracle client or directly at the server. The result will be in the form of tabular data.

GIS or CAD software: Currently many GIS and CAD software packages make use of the Oracle Spatial object-oriented model. Each package provides different functionality depending on the origin. For example GIS based packages (e.g. ArcGIS) provide good tools to visualise and edit 2D geometries and attributes. CAD packages (e.g. MicroStation GeoGraphics) have better 3D editing tools but are a bit restricted in editing attributes. In most of the cases both types of software supply means to type SQL queries and visualise the geometries according to a specific clause.

For example using the Spatial Viewer (a viewer for visualisation geometries) of MicroStation Geographic the following query can be specified: 'Show all the beheerobjects that have code 'KV' (). To be able to see also the attributes, the user has to use the more elaborated application Geospatial, which allows joins between tables to be performed. Note that with the Spatial Viewer only one table can be queried at a time.

Web services: It is also possible to retrieve the Beheerkaart data that is stored in Oracle Spatial via Web services. These can be OpenGIS Web services (WMS, WFS, WTS or WCS), but also more specific Oracle technology can be used, e.g. the Oracle XSQL servlet (see e.g. De Vries and Stoter, 2003).

5.2 Examples of queries

Depending on the environment for visualisation and the types of queries, views have to be created, which will supply the necessary basis for queries. Discussion on how the specified queries can be visualised in the different front-ends follows:

1. The BEHEEROBJECT table cannot be used directly by ESRI software. ESRI software cannot work with more than one geometry column per table. Therefore, for ESRI users views (containing only one geometry column) have to be created.

2. The first type of queries, i.e. visualisation of spatial information based on attribute restriction can be easily specified in all kinds of front-ends. As mentioned above, the attributes (e.g. Bentley Spatial Viewer) can be small exceptions. The situation differs if a query requires join of tables. An example of such a query is 'visualise all the beheerobject in a given RD'. According to the implementation the RD are organised in a separate table. It is possible to perform join operation on the fly, but it can violate the performance (especially in case of large data sets). It would be more appropriate to create views with the tables often joined. Example of such a view is V_BEHEEROBJECT. The views can be accessed after that from the front-ends in the same way as the original tables.

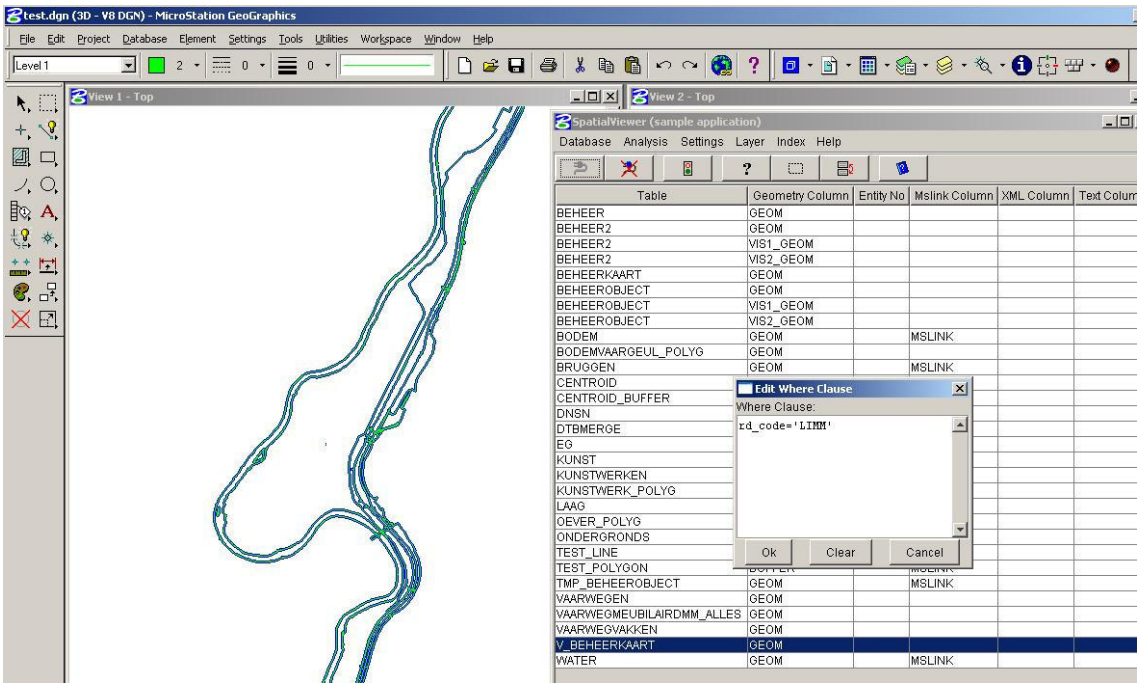


Figure 10: Visualise all the Beheerobjects where RD_CODE='LIMM' from view V_BEHEEROBJECT.

3. The visualisation of low-resolution geometry, i.e. switching between the different geometries is indeed best to be done automatically. The selection which geometry has to be used depends on the size of the area for visualisation. The size can be defined either indicating min-max boundaries or defining the type of user. For example, if the user is RWS then Beheerobjects in the category 'Bodem' have to be visualised as lines (i.e. geometries available in Vis2_Geom) and most of the Beheerobjects 'Kunstwerk' should not be visualised. Automatic switch between different geometries can be implemented in two ways: 1) According to specified parameters (e.g. user or visualisation area) defined by the user, an appropriate SQL statement has to be defined by the front-end that will extract the needed geometry. 2) The parameters can be forwarded to the database and the selection can be done applying triggers. In any case the interface for this type of query has to be developed (e.g. in addition to the OpenGIS Web services). Commercial software (e.g. ESRI or Bentley) does not have means for automatic switching between geometries stored in the same table in a DBMS. The geometries can be only manually selected from the tables. To be accessible for selection they have to be registered in the user_metadata tables in the same way as the basic geometry (in the column GEOM).

4. Query and visualisation of history is easy to perform using the begin and end date stored in the Beheerobject table. A selection by BEH_ID will give all previous sates of an object

with all the changes (geometry or attributes). Since the data are in one table they can be queried by any commercial software in the way specified above.

5. Example of quantity and area computations

"Show the BPN code of all the objects that are in WSD MM1 and have ID<100"

```
select beh_id, watersysteemdeel.code||'.'||catcode||'.'||zijdecode||'.'||code_km
from beheerobject, watersysteemdeel,categorietype,zijdetype
where beh_id<100 and watersysteemdeel.code='MM1'
  and watersysteemdeel.wsd_id=beheerobject.wsd_id
  and categorietype.cat_id=beheerobject.cat_id
  and zijdetype.zijde_id=beheerobject.zijde_id


    BEH_ID WATERSYSTEEMDEEL.CODE||'.'||CAT
---------- ------------------------------
        62 MM1 .QQ.-.6.0
        63 MM1 .QQ.-.2.5
        64 MM1 .QQ.-.6.7
        77 MM1 .OO.L.2.5
        78 MM1 .OO.L.7.5
        81 MM1 .OO.L.6.8

6 rows selected

Elapsed: 00:00:00.01

The same query from the view V_BEHEEROBJECT:

select beh_id,wsd_code||'.'||catcode||'.'||zijdecode||'.'||code_km
from v_beheerobject
where beh_id<100 and wsd_code='MM1'


   BEH_ID WSD_CODE||'.'||CATCODE||'.'||ZI
---------- ------------------------------
        77 MM1 .OO.L.2.5
        78 MM1 .OO.L.7.5
        81 MM1 .OO.L.6.8
        62 MM1 .QQ.-.6.0
        63 MM1 .QQ.-.2.5
        64 MM1 .QQ.-.6.7

6 rows selected

Elapsed: 00:00:00.08
```

The first query is a bit faster than the second one because in the first query joins only three tables. As mentioned above, the view is a join of six tables.

Area computation:

"What are the areas of all the objects of category BH"

```
SQL> select beh_id, sdo_geom.sdo_area (geom, 0.005) from beheerobject, categorietype
 where beheerobject.cat_id=categorietype.cat_id and catcode='BH'

  BEH_ID SDO_GEOM.SDO_AREA(GEOM,0.005)
---------- ----------------------------
       133                   30159.2346
       136                   14637.1652
       150                   134520.016
       151                   67743.4428
       137                   61208.1781
       139                    39423.062
       149                   72919.9017
       150
7 rows selected.
```

"What is the total area of all the objects in category BH"

```
SQL> select sum(sdo_geom.sdo_area (geom, 0.005)) from beheerobject, categorietype
 where beheerobject.cat_id=categorietype.cat_id and catcode='BH'

SUM(SDO_GEOM.SDO_AREA(GEOM,0.0005))
-----------------------------------
                          420611.001
```

6. Editing of objects require development of two types of triggers. The first type should maintain the history of objects (register begin and end date). This type of triggers have to work for changes in both the geometry and the thematic attributes. The second type has to be used only in case of editing (deleting and adding) of geometry. The second type of triggers has to check the consistency rules (non-overlap) of the Beheerkaart-Nat specified in the five 'Layers' (implemented as 5 views). A third type of triggers can be used to derive the second and/or third geometry (VIS1_GEOM and VIS2_GEOM) from GEOM.

Within this project, we have developed only triggers for maintenance of history. This will be further elaborated in section 6.5.

6 DISCUSSION OF THE MODEL WITH RESPECT TO THE REQUIREMENTS OF AGI

The objective of this project was to develop a new data model for the Beheerkaart-Nat. The model has to be able to provide an efficient method for storage, query and editing of Beheerobjects, their attributes and relationships. The major characteristics of the model with respect to the requirements of the users are discussed below

6.1 Multiple geometry types

As mentioned before (see Chapter 1, Introduction) some categories of objects can be represented by multiple geometry types: sometimes by a line and sometimes by a polygon (*hoogwaterkering*, *waterreguleringswerk*, *oevers*, etc) or sometimes by a point and in other cases by a polygon (*scheepvaartbegeleiding*). We suggest these objects to be maintained in the same table (Beheerobject) with no explicit classification about type of geometry. The underlying motivation is that Oracle Spatial can maintain different spatial data types in one geometry column. The type of geometry (point, line or polygon) can always be found looking at the GTYPE (using the Oracle Spatial function GET_GTYPE) of the geometry of the objects:

'List all line Beheerobjects' :

```
>select c.beh_id from beheerobject c where c.geom..get_gtype()=2; --GTYPE=2 is a line

no row selected
```

'List all objects that are a collection of polygons (have more than one disjoint polygon)'

```
SQL> select c.beh_id from beheerobject c where c.geom.get_gtype()=7;

    BEH_ID
----------
        61
        65
       110
       113
       119
       121
       125
       126
       127
       128
       129

    BEH_ID
----------
       130
       131
       108
       109
       114
        60

17 rows selected.
```

Knowing the ID of the object, it can be retrieved for visualisation, e.g. in MicroStation (Figure 11):

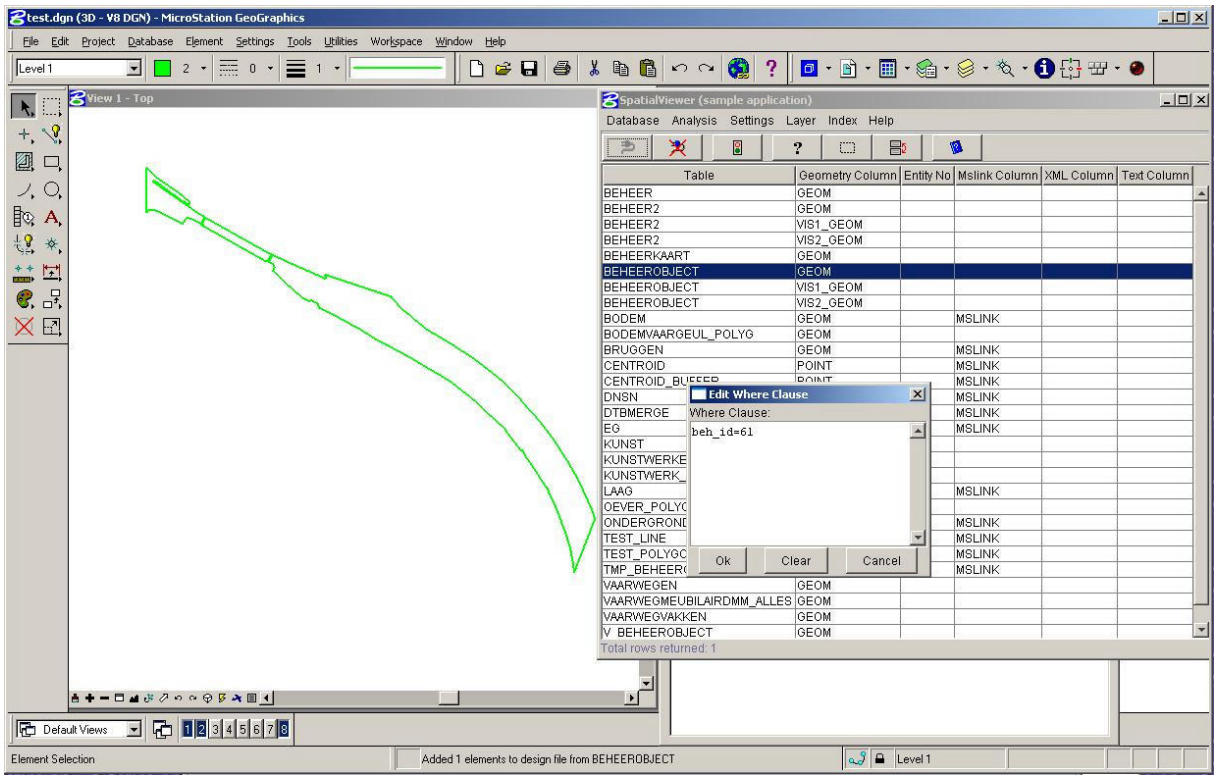


Figure 11: Visualisation of collection of polygons

6.2 Multiple 'Layers'

One important requirement is that most of the Beheerobjects have to create a non-overlapping coverage. As discussed in the first section, there are 5 such 'Layers' in which the objects should not overlap. These layers are represented as views. Thus the geometry is stored only once in the Beheerobject table and triggers are used to check relationships between particular objects within one view. It is important to realise that every change of geometry of objects has to be performed only after a consistency check within the objects of the corresponding view. This check has to be organised by triggers similar to the ones described for maintaining history (see Section 6.5 below). If a change of geometry is made during editing by the end-user, a database trigger has to automatically initiate a control test for non-overlapping within the view the object belongs to, before the new status of the object is stored in the Beheerobject table. These triggers are outside the scope of this project.

6.3 Multiple geometry representations

Within the project was clarified that a reasonable number of geometry representations is maximum three. We propose geometries to be stored in the database explicitly to improve the performance of the model. We consider a creation on the fly not very appropriate for beheerkaartobjects because most of the objects are very long polygons that have to be converted into lines. As discussed in the Test section, trivial approach for polygon-to-line conversion (for a database application) could not be found.

The three geometry representations are maintained in bheerobject table. Some of this representations can be created automatically (line-to-point and polygon-to-point), but polygon-to-line will require development of routines. Another approach, i.e. establishing links to other datasets (NWB) is in general possible, but multiple relationships between objects require further attention.

6.4 Aggregation of objects based on thematic attribute values

Aggregation of objects based on the value of one or more attributes is to be understood as visualisation of objects that have the same attribute values and not as aggregation of geometries. These types of queries can be easily performed within the developed model with select statements giving in the where clause the required attribute restrictions.

Depending on the types of the objects different type of information can be requested for visualisation (Table 3)

Table 3: Required data with respect to the different categories

Tabel 3-3: Objectcategorieën met bijbehorende grootheden en eenheden

Objectcategorie	Object-subcategorie	Afkorting	Grootheid	Eenheid	Voorbeeld
Kunstwerk	Aanleginrichting	KA	aantal	stuks	steigers, meerpalen
	Beweegbare brug	KB	aantal	stuks	brug te A
	Gemaal	KG	aantal	stuks	gemaal X
	Hoogwaterkering	KH	aantal	stuks	stormvloedkering
	Kunstwerken t.b.v. natuur	KN	aantal	stuks	vistrap
	Aquaducten	KQ	aantal	stuks	aquaduct te B
	Waterreguleringswerk	KR	aantal	stuks	overlaat
	Schutsluis	KS	aantal	stuks	grote schutsluis Y
	Tunnel	KT	aantal	stuks	tunnel te A
	Spui/uitwateringsluis	KU	aantal	stuks	spui bij kunstwerk X
	Vaste brug	KV	aantal	stuks	brug Z
	Stuw	KW	aantal	stuks	stuw X
	Sifon/duiker/hevel	KZ	aantal	stuks	grondduiker
Bodems	Vaargeulbodem (vak)	BV	lengte	km	kanaalpand
	Havenbodem	BH	oppervlakte	ha	overnachtingshaven
	Bodems (overig)	BO	oppervlakte	ha	tussen kribben
Oevers	Strekdam	OD	lengte	km	strekdam Y
	Oever haven	OH	lengte	km	kade
	Kribben	OK	aantal	stuks	kribben te A
	Leidam/leikade	OL	lengte	km	leidam X
	Oevers/dijken	OO	lengte	km	kustlijn, linkeroever
	Strandhoofd	OS	lengte	km	strandhoofd Vlieland
	Uiterwaarden	OU	oppervlakte	ha	uiterwaarden te B
	Kribvakken	OV	lengte	km	Tiel 1
	Waddenzeekwelders	OW	oppervlakte	ha	kwelderareaal te A
Water	Water	QQ	oppervlakte	km^2	watervak
	Algemeen	QX	aantal	stuks	algemeen meetsysteem
Verkeersvoorzieningen	Scheepvaartbegeleiding	VS	n.v.t.	n.v.t.	betonning, verkeerstekens, vuurtoren, verkeerspost
Exploitatie	Gebouwen	EG	aantal	stuks	dienstkringkantoor, werkplaats
	Terreinen	ET	oppervlakte	ha	opslagterrein, zanddepot
	Vaar- en voertuigen	EV	aantal	stuks	patrouillevaartuig, ponton, dienstauto's
	Juridisch	EJ	n.v.t.	n.v.t.	RO-zaken, overdrachten
	Crisismanagement	EC	n.v.t.	n.v.t.	crisisbeheersing, oliebestrijding
	Algemeen	EX	n.v.t.	n.v.t.	monumenten, beheerplannen

As it can be seen most of the objects have to be only listed. These queries can be easily performed within the developed model (as demonstrated in Section Tests). Similar to obtaining line from polygons, the most problematic queries remain the ones related to computing length of polygon objects. These objects have to be first approximated to lines and then the length of the line has to be computed.

Spatial aggregation of objects is also possible at a database level. Spatial aggregation has to be understand as computing the outer boundaries of the aggregation object. This can be done using the standard Oracle Spatial functions SDO_AGGR. There are four different aggregation of geometries applicable for beheeobjects :

SDO_AGGR_CENTROID Returns a geometry object that is the centroid ("center of gravity") of the specified geometry objects.
SDO_AGGR_CONVEXHULL Returns a geometry object that is the convex hull of the specified geometry objects.
SDO_AGGR_MBR Returns the minimum bounding rectangle of the specified geometry objects

SDO_AGGR_UNION Returns a geometry object that is the topological union (OR operation) of the specified geometry objects.

Note the last function requires objects to have at least one common point. Otherwise only one polygon (the first) is considered. This condition can be ensured with the help of other Oracle spatial functions, i.e. writing a PL/SQL script.

Example of a statement giving a convexhul of the first 10 objects is shown below:

```
SQL> select sdo_aggr_convexhull(mdsys.sdoaggrtype(c.geom, 0.005)) from beheerobject c where
c.beh_id<10;

SDO_AGGR_CONVEXHULL(MDSYS.SDOAGGRTYPE(C.GEOM,0.005))(SDO_GTYPE, SDO_SRID, SDO_PO
--------------------------------------------------------------------------------
SDO_GEOMETRY(3003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(176076.112, 319781.917, 0, 177976.943, 322467.565, 0, 179918.047, 326023.843, 0, 181325.11,
329074.51, 0, 181328.197, 329081.554, 0, 181328.269, 329081.719,0, 187424.045, 346436.071, 0,
187316.996, 346494.222, 0, 185663.356, 343427.471, 0, 182161.745, 335033.543, 0, 176063.101,
319786.843, 0, 176063.774, 319786.467, 0, 176076.112, 319781.917, 0))
```

6.5 Possibilities for maintaining history

There are several technical solutions for maintaining history in the database. It depends on the purpose of doing this and also on the environment in which the system has to function which of these options is the most optimal solution.

We will first present the options and after that give an indication of which option to choose.
The purpose of maintaining history / keeping track of inserts, updates and deletes can be:

- **Option 1:** to monitor only the changes so that someone (database administrator) can take action on these changes;

- **Option2:** to keep past versions of objects (after an update or delete) in the database in order to:
  - visualize changes over time (for example: show all situations between 1 January 2000 and 1 January 2001);
  - look at the situation on one specific moment in time (for example: show me the situation on 1 July 2001).

If the purpose is to monitor changes so that a database administrator can take (manual) action, Option 1 is probably enough. Option 1 does not give the possibility however to visualize changes over time or visualize the situation on one specific moment in time.

6.5.1 Option 1. Only monitoring the changes

In this solution there is a separate table Beheerobject_changes. In this table all changes (inserts, updates, deletes) or logged, together with time, username and type of change.

This can be technically implemented by using triggers that only register: when a change has occurred, by whom the insert, update or delete is carried out and what kind of change is involved (stored in an attribute called 'status'). Additionally it could be monitored which attribute has changed (the geometry or one or more of the thematic attributes). In that case one update could result in several log-records in the Beheerobject_changes table.

6.5.2 Option 2. Keeping a complete history of all Beheerobjects

In this solution there is always a complete set of the present version of Beheerobject objects plus all previous versions. So the previous version before an update is saved and also all deleted records are kept.
This can be technically implemented in two ways: storing a copy of all inserts, updates and deletes in a separate table Beheerobject_mirror (option 2.1) or storing all history in Beheerobject itself (option 2.2).

6.5.3 Option 2.1 Storing copy of all inserts, updates and deletes in separate table

A table Beheerobject_mirror is created with the same structure as the Beheerobject table. An extra attribute 'status' can be added to hold the values 'I' (inserted), 'U' (updated) or 'D' (deleted), but this is not necessary.

```
create table beheerobject_mirror (
 REC_ID                                                NUMBER(38),
 BEH_ID                                                NUMBER,
 OBJECTNAME                                            VARCHAR2(40),
 WSD_ID                                                NUMBER,
 CAT_ID                                                NUMBER,
 ZIJDE_ID                                              NUMBER,
 CODE_KM                                               VARCHAR2(21),
 DATE_BEGIN                                            DATE,
 DATE_END                                              DATE,
 NAME_CONTACTPERSON                                    VARCHAR2(30),
status                                                 char(1),
 GEOM                                                  MDSYS.SDO_GEOMETRY,
 VIS1_GEOM                                             MDSYS.SDO_GEOMETRY,
 VIS2_GEOM                                             MDSYS.SDO_GEOMETRY,
 MD_ID                                                 NUMBER )
/
```

When Beheerobject is created and filled with the initial set of records, these records are also imported in the Beheerobject_mirror table.

Below are the triggers that ensure that every time a record is inserted, updated or deleted in Beheerobject this action is followed by inserts and updates in the Beheerobject_mirror table.

```
/*
History is copied to separate table beheerobject_mirror: copy of current record plus status:
I, U or D
*/


create or replace trigger BEHEER_mirror_insupd after insert or update
of
 BEH_ID,
 OBJECTNAME,
 WSD_ID,
 CAT_ID,
 ZIJDE_ID,
 CODE_KM,
 GEOM,
 VIS1_GEOM,
 VIS2_GEOM,
 MD_ID
on BEHEEROBJECT
for each row
declare

   t_status char(1) ;

begin

if updating then

   update beheerobject_mirror h
   set date_end = :new.DATE_BEGIN
   where beh_id = :old.BEH_ID and date_end is null ;

   t_status := 'U' ;
```

```
end if;
if inserting then

   t_status := 'I' ;

end if;

   insert into beheerobject_mirror (
 REC_ID,
 BEH_ID,
 OBJECTNAME,
 WSD_ID,
 CAT_ID,
 ZIJDE_ID,
 CODE_KM,
 DATE_BEGIN,
 NAME_CONTACTPERSON,
status,
 GEOM,
 VIS1_GEOM,
 VIS2_GEOM,
 MD_ID
   )
   values (
 :new.REC_ID,
 :new.BEH_ID,
 :new.OBJECTNAME,
 :new.WSD_ID,
 :new.CAT_ID,
 :new.ZIJDE_ID,
 :new.CODE_KM,
 :new.DATE_BEGIN,
 -- of: sysdate, als beheerobject zelf geen date_begin kent
 :new.NAME_CONTACTPERSON,
 t_status,
 :new.GEOM,
 :new.VIS1_GEOM,
 :new.VIS2_GEOM,
 :new.MD_ID
   );
end;
/

create or replace trigger BEHEER_mirror_delete after delete
on BEHEEROBJECT
for each row
declare
   t_timestamp date;
begin
   select sysdate into t_timestamp from dual ;

   update beheerobject_mirror h
   set date_end = t_timestamp ,
       status = 'D'
   where beh_id = :old.BEH_ID and date_end is null ;

end;
/
```

6.5.4 Option 2.2. Keeping updated and deleted records in Beheerobject table

The triggers that are necessary to accomplish this are more complicated than is the case in strategy 2.1. This has to do with the fact that Oracle doesn't permit insert actions in a table that is at that moment already 'touched' by an update, delete or insert. In other words, the triggers that are created for strategy 2.1 can not simply be changed so that now the inserts are not stored in the separate Beheerobject_mirror table, but in the Beheerobject table itself. The error that is raised by Oracle would be:

```
ORA-04091: table BEHEEROBJECT is mutating, trigger/function may not see it.
```

There are ways to get around this situation.

One solution involves creating a view also for editing: all edits have to made on the view, not on the Beheerobject table itself. We will not further elaborate on this because this seems error prone (maybe someone forgets that edits should always be done via the view) (2.2 with ‘instead of’ trigger).

The other solution is shown in the next set of triggers. It is based on: 'Avoiding Mutating Tables' at http://osi.oracle.com/~tkyte/Mutate/index.html.

We will call it: option 2.2 ‘with postponed action’, because the triggers first keep the records that are updated or deleted in a temporary array in memory before they are actually stored in the table Beheerobject.

```
/*
History is stored in the same table (beheerobject):
*/

create or replace package beheer_hist
as

  type recArray is table of beheerobject%rowtype index by binary_integer ;

  oldvals recArray ;
  empty recArray ;

end;
/
show errors


create or replace trigger beheer_update_before_prep1
before update or delete on beheerobject
begin
  beheer_hist.oldvals := beheer_hist.empty ;
end;
/
show errors

create or replace trigger beheer_update_before_prep2
before update or delete
of
 BEH_ID,
 OBJECTNAME,
 WSD_ID,
 CAT_ID,
 ZIJDE_ID,
 CODE_KM,
 GEOM,
 VIS1_GEOM,
 VIS2_GEOM,
 MD_ID
on beheerobject
for each row
declare

   t_rec_id  number(38) ;
   t_timestamp  date;
   t_username varchar2(16) ;

   i number default beheer_hist.oldvals.count+1 ;

begin

   select sysdate into t_timestamp from dual ;
   select user into t_username from dual ;

 /* save the old values and fill date_end with timestamp */
 beheer_hist.oldvals( i ).REC_ID := :old.rec_id ;
 beheer_hist.oldvals( i ).BEH_ID := :old.beh_id ;
 beheer_hist.oldvals( i ).OBJECTNAME := :old.objectname ;
 beheer_hist.oldvals( i ).WSD_ID := :old.wsd_id ;
 beheer_hist.oldvals( i ).CAT_ID := :old.cat_id ;
 beheer_hist.oldvals( i ).ZIJDE_ID := :old.zijde_id ;
 beheer_hist.oldvals( i ).CODE_KM := :old.code_km ;
 beheer_hist.oldvals( i ).DATE_BEGIN := :old.date_begin ;
 beheer_hist.oldvals( i ).DATE_END := t_timestamp ;
 beheer_hist.oldvals( i ).NAME_CONTACTPERSON := :old.name_contactperson ;
```

```
 beheer_hist.oldvals( i ).GEOM := :old.geom ;
 beheer_hist.oldvals( i ).VIS1_GEOM := :old.vis1_geom ;
 beheer_hist.oldvals( i ).VIS2_GEOM := :old.vis2_geom ;
 beheer_hist.oldvals( i ).MD_ID := :old.md_id ;

if updating then
   /* update the updated record*/

   /* get new rec_id for updated record */
   select seq_beheerobject_rec_id.nextval into t_rec_id from dual ;

   :new.REC_ID := t_rec_id ;

   /* fill date_begin with timestamp and set name_contacperson to username */

   :new.DATE_BEGIN := t_timestamp ;
   :new.NAME_CONTACTPERSON := t_username ;


end if;
end;
/
show errors

create or replace trigger beheer_update_after
after update or delete
on beheerobject
begin

  /* insert old record back in */
  for i in 1 .. beheer_hist.oldvals.count loop

    insert into beheer.beheerobject (
 REC_ID,
 BEH_ID,
 OBJECTNAME,
 WSD_ID,
 CAT_ID,
 ZIJDE_ID,
 CODE_KM,
 DATE_BEGIN,
 DATE_END,
 NAME_CONTACTPERSON,
 GEOM,
 VIS1_GEOM,
 VIS2_GEOM,
 MD_ID
    )
    values
    (
 beheer_hist.oldvals( i ).REC_ID ,
 beheer_hist.oldvals( i ).BEH_ID ,
 beheer_hist.oldvals( i ).OBJECTNAME ,
 beheer_hist.oldvals( i ).WSD_ID ,
 beheer_hist.oldvals( i ).CAT_ID ,
 beheer_hist.oldvals( i ).ZIJDE_ID ,
 beheer_hist.oldvals( i ).CODE_KM ,
 beheer_hist.oldvals( i ).DATE_BEGIN ,
 beheer_hist.oldvals( i ).DATE_END ,
 beheer_hist.oldvals( i ).NAME_CONTACTPERSON ,
 beheer_hist.oldvals( i ).GEOM ,
 beheer_hist.oldvals( i ).VIS1_GEOM ,
 beheer_hist.oldvals( i ).VIS2_GEOM ,
 beheer_hist.oldvals( i ).MD_ID ) ;

  end loop ;

end ;
/
show errors
```

In case of an update or delete of a Beheerobject these triggers take care of keeping the old version of the record in the table. The date_end attribute of the record is filled with the system time, to indicate that the record has ‘expired’. In case of an update, a new record is inserted, which is now the current record. The date_begin of this record has exactly the same value (date+time in seconds) as the date_end of the ‘expired’ record.

In the example below there has been a delete of the Beheerobject with Beheer-id = 6 (by a user action). Instead of really deleting the record the triggers only fill date_end with the system time, and it is not deleted.

There also have been 2 updates of the Beheerobject with Beheer-id = 5 (the 'objectname' attribute is updated 2 times). The triggers automatically insert a record for each update and put the right date_end in the 'old' record. Also a change a geometry (which in Oracle is 'just' another attribute) would be handled this way: by keeping the old record and inserting a new record with the new geometry.
The objects with id 7 and 8 have not been changed.

BEH_ID	DATE_BEGI	DATE_END	OBJECTNAME	REC_ID
5	02-SEP-04	08-SEP-04	Brug Itteren	5
5	08-SEP-04	08-SEP-04	mut Brug Itteren	9004
5	08-SEP-04		mut mut Brug Itteren	9009
6	02-SEP-04	08-SEP-04	Brug Roosteren	6
7	02-SEP-04		Brug Elsloo	7
8	02-SEP-04		Brug Geulle	8

This solution (2.2 with postponed action) does not involve a view, but can still be hard to maintain. The triggers are rather complicated and not so easy to adjust. In addition care has to be taken that the Beheerobject table is only edited with the triggers enabled, otherwise the table Beheerobject itself could become inconsistent.

This disadvantage is less so in case of option 2.1 (keeping the history in a separate table). Option 2.1 however is easier to maintain (the triggers are simpler) and when the triggers are for some reason or another disabled or dropped, the Beheerobject table itself is not affected, only the mirror-table.

The functionality is the same in both cases (2.1 and 2.2): all history is kept. Because the Beheerkaart data set is relatively small, keeping a separate table Beheerobject_mirror is in our opinion the safest option.

7 IMPLEMENTATION STEPS

The general steps to be completed when implementing this model are:

- Importing shape files into Oracle Spatial tables using FME. Perhaps it would be wise some of the shape files to be merged. FME imports one shape file in one table.
- Registering in metadata table and indexing (2D index)
- Validation and correction of geometries. The corrections eventually will be done in the shape files. The corrected shape files have to be again imported in Oracle Spatial Tables and again validated.
- Check for overlap within layers
- Creating all the supporting administrative (importing from MSAccess) and LUP tables.
- Creating and population the metada table.
- Creating and indexing the Beheerobject tables
- Import of all the objects in one Beheerobject table
- Creating links with the metadata table
- Populating the fields with Vis1_geom. and Vis2_geom.
- Creating views according to the UML schema (i.e. the five layers)
- Creating views with respect to the users
- Developing triggers for data consistency check and maintenance of history.

One aspect of the Beheerkaart data models is not so 'visible' in the logical and physical model: that of maintaining 'history' (keeping a track of modifications of the Beheerobject data in the editing process). The physical model as it is now, is based on option 2.2.: keeping updated and deleted records in the Beheerobject table itself. When the choice is made for option 1 or 2.1 (having a separate table) an extra feature class is needed in the logical and physical UML model.

8 OPENGIS COMPLIANCE CHECK

Both the logical and the physical model that were developed in this project have been checked for OpenGIS compliance by IONIC Software (see elsewhere in this publication).
Based on the results of the check and the recommendations in chapter 10 of the OpenGIS compliance report the following adaptations are necessary:

8.1 'Layers' (Lbodems etc.) become subclasses instead of aggregations

In the logical and physical model the 'Layer' classes are now: aggregations (collections). Although it is possible to have collections in GML and WFS (there is a gml:FeatureCollection class in the GML data model), these feature collections are not supported by most WFS/GML software. Therefore it is better to change the 'Layer' classes from aggregation classes to subtype classes, in both logical and physical model. For the database implementation in Oracle Spatial this does not have consequences: the 'Layer' classes have also in the current solution been implemented as Oracle database views. This will stay that way in the OpenGIS compliant solution.

Figures 12 and 13 show the adapted logical and physical model (see next pages).

8.2 No mixed geometry types

In the logical model so far, it was possible to have several geometry types for one subcategory, see Table 2 of this report ('Visualisation requirements'). For example: for a Hoogwaterkering (Code 'KH') at scale level 1:25000 a specific object could have a line or a polygon geometry. In Oracle Spatial this is no problem. For visualization in OpenGIS WFS services however it is necessary to 'stick to' one geometry type for a feature class. Otherwise it is very hard for a Web client application to know how to represent the feature, as a polygon or as a linestring. Therefore it is necessary to look again at Table 2, and decide for each subcategory and visualization level what the geometry type should be. This can then be checked in the editing environment, before a record is stored in the database (one of the consistency checks that is proposed in this report).

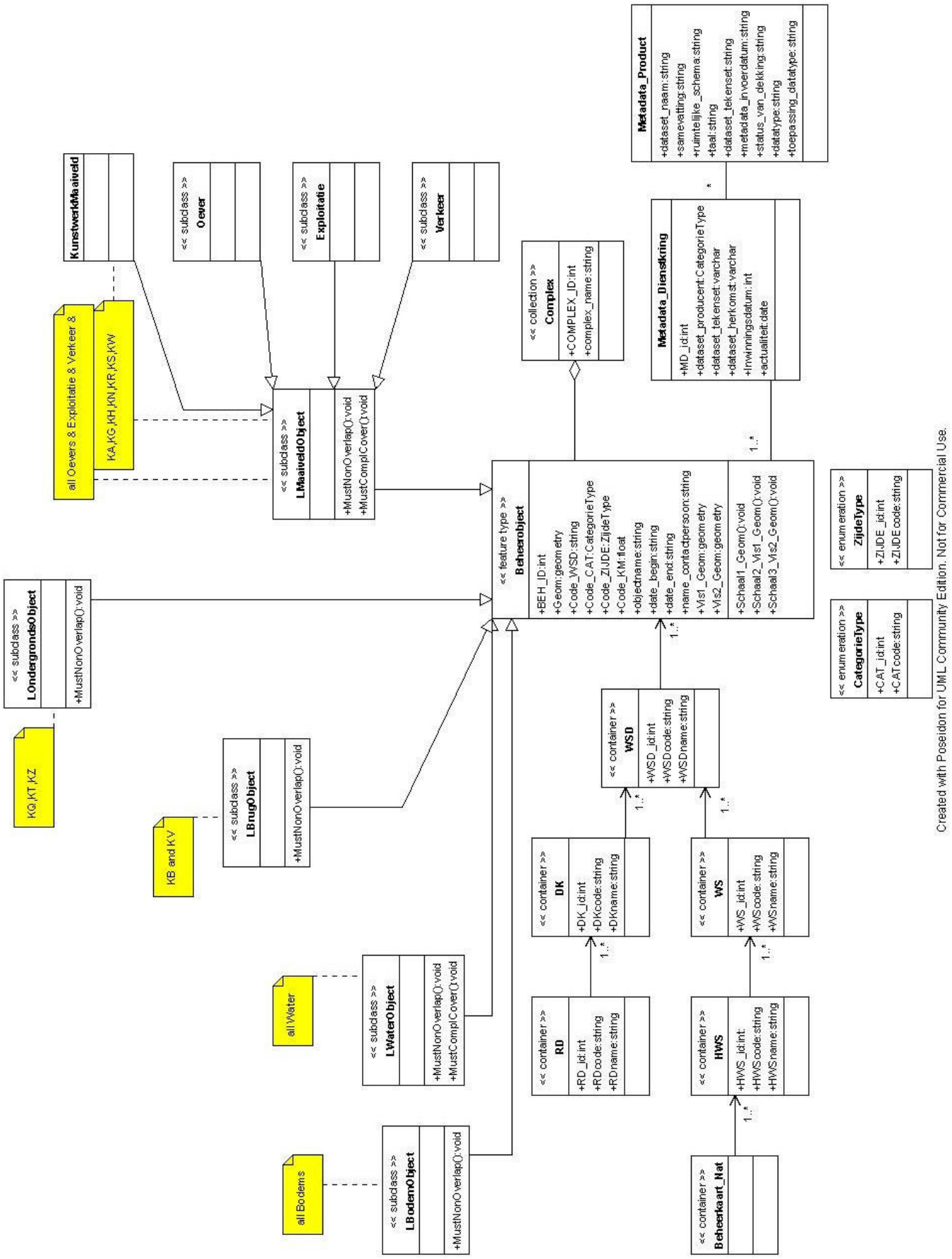


Figure 12: Logical model

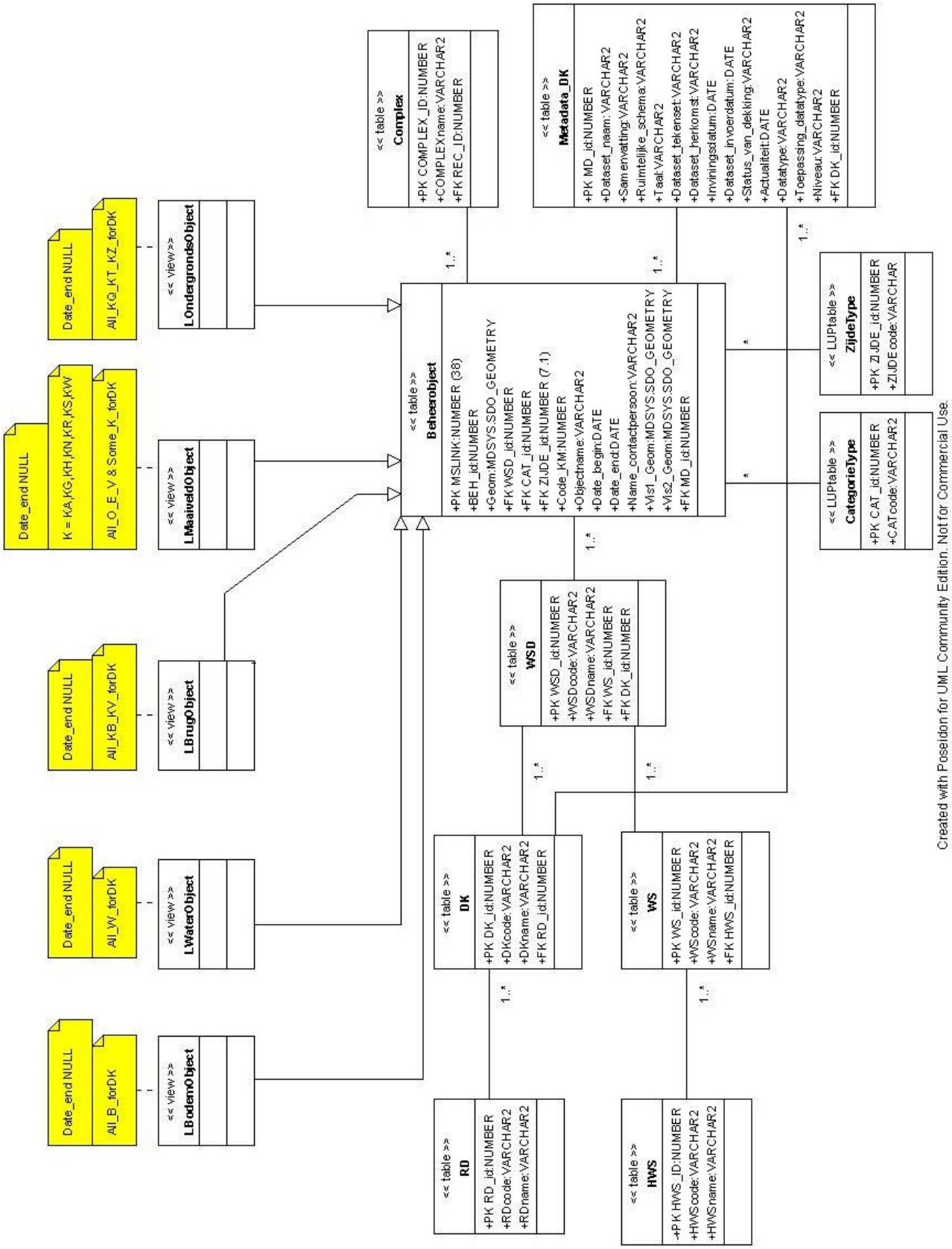


Figure 13: Physical model

References

Chithambaram, R. et al., 'Skeletonizing polygons for map generalization'. In: Technical papers, ACSM-ASPRS Convention, Cartography and GIS/LIS, Volume 2. Bethesda: ACSM, 1991, p.44-54.

DeLucia, A., en T. Black, 'A comprehensive approach to automatic feature generalization'. In: Proceedings of 13th International Cartographic Conference. International Cartographic Association, 1987, p.169-191.

IBM, 2002, Mapping object to data models with the UML, IBM white paper, available at: http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/tp185.pdf

Oosterom, P.J.M. van, 'Hartlijnen van wegennetwerken'. Informatieblad Kadaster, 7 (1999), p.10-11.

Uitermark, H. et al., 'Semantic and Geometric Aspects of Integrating Road networks'. In: A. Vckovski et al. (eds), Interoperating geographic information system: second international conference, INTEROP '99. Berlin: Springer-Verlag, 1999, p.177-188.

F. Penninga, 'Genereren van een 6-positie postcodebestand op basis van kadastrale registratie', afstudeerscriptie TU Delft, mei 2003, Delft

Berg, M. de, et al., 'Computational geometry, algorithms and applications'. Berlin: Springer-Verlag Berlin Heidelberg New York, 1997.

Steven A. Roberts, G. Brent Hall, Barry Boots, 'Street centerline generation with an approximated area Voronoi Diagram', proceedings Spatial Data Handling 2004, p. 435-446.

Tijssen T.P.M., Quak C.W., Oosterom P.J.M. van; 2001, Spatial DBMS testing with data from Cadastre and TNO-NITG. Uitgave: TU Delft/Geodesie, Delft, maart 2001, 125 p. Rapportnummer: GISt no 7.

Vries, M.E. de, and Stoter, J.E., 2003. Accessing a 3D geo-DBMS using Web technology. ISPRS Joint Workshop on Spatial, Temporal and Multi-Dimensional Data Modelling and Analysis. Québec, Canada.

Vries, M. de, and S. Zlatanova, 2004. Interoperability on the Web: the case of 3D geo-data, in: P. Isaias, P. Kommers and M. McPherson (eds.), Proceedings of the IADIS International Conference e-Society 2004, Avila, Spain.

Zlatanova, S., T.P.M. Tijssen, P.J.M. van Oosterom and W. C. Quak, 2003, Research on usability of Oracle Spatial within RWS Organisation, GISt No.21, ISSN: 1569-0245, ISBN: 90-77029-07-9, AGI-GAG-2003-21, Delft, The Netherlands, 75 p.

Appendix 1: FME mapping file (from shape file into Oracle Spatial table) for Bodem

```
# ==============================================================================
#
# This mapping file was generated by the FME
# on 07/13/04 10:23:14 for translation between SHAPE and ORACLE8I
#
# This mapping file was generated with FME build 1321
#
# You may edit this mapping file to customize its operation.  Comments are
# placed throughout to assist you.
#
# Modification History:
#
#     Name              Date     Description
#     ================= ======== ============================================
#
#
# ==============================================================================

# ==============================================================================
# The following line defines the title presented to the user when this
# mapping file is run through the FME GUI.  You may modify this
# if a more meaningful title would be appropriate.

GUI TITLE SHAPE to ORACLE8I Translation

# ==============================================================================
# The following line names the log file to which useful statistics about
# the translation will be written.  This line can be uncommented and
# updated if you do wish to keep these statistics.

# LOG_FILENAME translation.log
# LOG_APPEND NO

# ==============================================================================
# The following line instructs the FME to log any features that do not
# match any of the source feature patterns listed further down in
# this file.  If you are modifying this mapping file, this will be
# useful to describe to you exactly which features you are losing
# during translation, if the statistics indicate that features are
# not being correlated or grouped.  Uncorrelated features do not
# match any source specification, ungrouped features do not have
# any corresponding _DEF line.

# FME_DEBUG UNGROUPED UNCORRELATED

# ==============================================================================
# The following two lines define the type of reader and writer to be
# used for this translation.  If you want to translate your data
# back into its original format, you may make a copy of this file
# and switch the reader and writer types.  If you rerun the FME, you
# will get your original data back again (together with any modifications
# you made in the meantime).  Note that several formats are NOT
# bi-directional (for example, GIF can only be used as a WRITER)
# so a reverse translation may not always be possible.

READER_TYPE SHAPE

WRITER_TYPE ORACLE8I

# ==============================================================================
# The following GUI line prompts for the source shape files

# The dataset this mapping file was generated from was:
DEFAULT_MACRO SourceDataset E:\Beheerkaart\Beheerkaart\Laag4\bodem.shp

GUI FILENAME SourceDataset Shape_Files(*.shp)|*.shp|All_Files(*.*)|*.* Source ESRI Shape
File(s):

# ==============================================================================
# The following GUI lines will set the search envelope. If they are used, and
# if the shape files have index files (.sbnand.sbx), then the index will be
# used to do a fast lookup on this search envelope.
#
# DEFAULT_MACRO _SHAPE_MINX 0
# DEFAULT_MACRO _SHAPE_MINY 0
# DEFAULT_MACRO _SHAPE_MAXX 0
# DEFAULT_MACRO _SHAPE_MAXY 0
#
# GUI FLOAT _SHAPE_MINX Envelope Min X:
```

```
# GUI FLOAT _SHAPE_MINY Envelope Min Y:
# GUI FLOAT _SHAPE_MAXX Envelope Max X:
# GUI FLOAT _SHAPE_MAXY Envelope Max Y:
#
# SHAPE_SEARCH_ENVELOPE $(_SHAPE_MINX) $(_SHAPE_MINY) $(_SHAPE_MAXX) $(_SHAPE_MAXY)
#
# ==========================================================================
# The following GUI line sets whether measure values in the shape files should
# be treated as elevations.
#
# DEFAULT_MACRO _MEASURES_AS_Z no
#
# GUI CHOICE _MEASURES_AS_Z yes%no Treat Measures as Elevation:
#
# SHAPE_MEASURES_AS_Z $(_MEASURES_AS_Z)
#
SHAPE_DATASET "$(SourceDataset)"

# ==========================================================================
# The following GUI line prompts for the name of the Oracle Service
# to which data will be written.

DEFAULT_MACRO DestDataset gisbase
GUI TEXT DestDataset Destination Oracle Service:

DEFAULT_MACRO _ORACLE_Dimension 3
DEFAULT_MACRO _ORACLE_Minx 174600
DEFAULT_MACRO _ORACLE_Miny 306400
DEFAULT_MACRO _ORACLE_Minz 0
DEFAULT_MACRO _ORACLE_Maxx 200100
DEFAULT_MACRO _ORACLE_Maxy 362600
DEFAULT_MACRO _ORACLE_Maxz 0
DEFAULT_MACRO _ORACLE_Tolx 0.0000000005
DEFAULT_MACRO _ORACLE_Toly 0.0000000005
DEFAULT_MACRO _ORACLE_Tolz 0.0000000005
DEFAULT_MACRO _ORACLE_CreationParams
DEFAULT_MACRO _ORACLE_SRID
DEFAULT_MACRO _ORACLE_SpatialIndexName
DEFAULT_MACRO _ORACLE_ForceIndexCreation No
DEFAULT_MACRO _ORACLE_Levels 5
DEFAULT_MACRO _ORACLE_NumTiles 8

# ==========================================================================
# These next prompts ask for various layer creation parameters. Normally
# these will be hardcoded on a layer by layer basis
#

GUI OPTIONAL TEXT _ORACLE_CreationParams Additional table creation parameters:

GUI INTEGER _ORACLE_Dimension Geometric dimension:

GUI OPTIONAL INTEGER _ORACLE_SRID Spatial referencing (SRID) value:

GUI FLOAT _ORACLE_Minx Minimum X Coordinate:

GUI FLOAT _ORACLE_Miny Minimum Y Coordinate:

GUI FLOAT _ORACLE_Minz Minimum Z Coordinate:

GUI FLOAT _ORACLE_Maxx Maximum X Coordinate:

GUI FLOAT _ORACLE_Maxy Maximum Y Coordinate:

GUI FLOAT _ORACLE_Maxz Maximum Z Coordinate:

GUI INTEGER _ORACLE_Levels Tesselation Levels:

GUI OPTIONAL TEXT _ORACLE_SpatialIndexName Spatial index name:

GUI CHOICE _ORACLE_ForceIndexCreation Yes%No Force creation of new index:

GUI FLOAT _ORACLE_Tolx Comparison tolerance for X coordinate:

GUI FLOAT _ORACLE_Toly Comparison tolerance for Y coordinate:

GUI FLOAT _ORACLE_Tolz Comparison tolerance for Z coordinate:

GUI CHOICE _ORACLE_Indicies Yes%No Create Indices:

DEFAULT_MACRO _ORACLE_IndexCommitInterval 50
# GUI INTEGER _ORACLE_IndexCommitInterval Incremental index commit interval:
```

```
DEFAULT_MACRO _ORACLE_NumTiles 8
GUI INTEGER _ORACLE_NumTiles Hybrid index numtiles (0forfixedindex):

DEFAULT_MACRO _ORACLE_GeometryColumn GEOM

ORACLE8I_SERVER_TYPE ORACLE8i

# ==========================================================================
# The following GUI lines prompt for the user name and password to use for
# accessing Oracle Spatial

DEFAULT_MACRO _ORACLE_UserName beheer
GUI TEXT _ORACLE_UserName Oracle User ID:

ORACLE8I_USER_NAME "$(_ORACLE_UserName)"

DEFAULT_MACRO _ORACLE_Password beheer
GUI PASSWORD _ORACLE_Password Oracle Password:

ORACLE8I_PASSWORD "$(_ORACLE_Password)"


# ==========================================================================
# The following option specifies which feature number the writer should actually
# start writing features to the database on. This should normally be zero, but
# can be changed if a previous translation is interrupted.
#
# DEFAULT_MACRO _START_TRANSACTION 0
#
# GUI INTEGER _START_TRANSACTION Transaction to Start Writing At:
#
# ORACLE8I_START_TRANSACTION $(_START_TRANSACTION)
#
# ==========================================================================
# The following specifies how many features will be written to the database in
# on each bulk write.
#
# DEFAULT_MACRO _CHUNK_SIZE 1000
#
# GUI INTEGER _CHUNK_SIZE Features per Bulk Write:
#
# ORACLE8I_CHUNK_SIZE $(_CHUNK_SIZE)
#
# ==========================================================================
# The following specifies how many features will be written to the database in
# each transaction.
#
# DEFAULT_MACRO _TRANSACTION_INTERVAL 2000
#
# GUI INTEGER _TRANSACTION_INTERVAL Features to Write Per Transaction:
#
# ORACLE8I_TRANSACTION_INTERVAL $(_TRANSACTION_INTERVAL)
#
# ==========================================================================
# The following specifies an SQL command to execute before opening the first
# ORACLE table.
#
# DEFAULT_MACRO _BEGIN_SQL
#
# GUI OPTIONAL TEXT _BEGIN_SQL SQL Statement to Execute Before Translation:
#
# ORACLE8I_BEGIN_SQL{0} "$(_BEGIN_SQL)"
#
# ==========================================================================
# The following specifies an SQL command to execute after closing all the
# ORACLE tables.
#
# DEFAULT_MACRO _END_SQL
#
# GUI OPTIONAL TEXT _END_SQL SQL Statement to Execute After Translation:
#
# ORACLE8I_END_SQL{0} "$(_END_SQL)"
#
ORACLE8I_DATASET "$(DestDataset)"

# ==========================================================================
# The main body of the mapping file starts here.  Each of the
# _DEF lines describes the data model of the particular feature
# type, and the correlation lines describe how the feature is
# transformed from the source type to the destination type.
# You may edit the following lines to add or remove attributes, change
# attribute definitions, or invoke other FME functions as the
```

```
# features are translated.
# ==========================================================================

# ==========================================================================
# Source feature type definition section
# ==========================================================================

SHAPE_DEF bodem                                 \
    SHAPE_GEOMETRY       shape_polygon          \
    AREA                 number(20,6)           \
    PERIMETER            number(20,6)           \
    OBJECTCODE           char(16)               \
    COMPLEXCOD           char(16)               \
    OBJECT               char(16)               \
    OBJECTNAAM           char(18)               \
    BODEM_ID             number(9,0)

# ==========================================================================
# Transformation section
# ==========================================================================

SHAPE bodem                                     \
    AREA                 %AREA                  \
    PERIMETER            %PERIMETER             \
    OBJECTCODE           %OBJECTCODE            \
    COMPLEXCOD           %COMPLEXCOD            \
    OBJECT               %OBJECT                \
    OBJECTNAAM           %OBJECTNAAM            \
    BODEM_ID             %BODEM_ID

ORACLE8I BODEM                                  \
    oracle_type          oracle_area            \
    AREA                 %AREA                  \
    PERIMETER            %PERIMETER             \
    OBJECTCODE           %OBJECTCODE            \
    COMPLEXCOD           %COMPLEXCOD            \
    OBJECT_              %OBJECT                \
    OBJECTNAAM           %OBJECTNAAM            \
    BODEM_ID             %BODEM_ID

# ==========================================================================
# Destination feature type definition section
# ==========================================================================

ORACLE8I_DEF BODEM                              \
    oracle_model         object                 \
    oracle_create_indices $(_ORACLE_Indicies)   \
    oracle_params        "$(_ORACLE_CreationParams)" \
    oracle_srid          "$(_ORACLE_SRID)"      \
    oracle_levels        $(_ORACLE_Levels)      \
    oracle_numtiles      $(_ORACLE_NumTiles)    \
    oracle_min_x         $(_ORACLE_Minx)        \
    oracle_min_y         $(_ORACLE_Miny)        \
    oracle_min_z         $(_ORACLE_Minz)        \
    oracle_max_x         $(_ORACLE_Maxx)        \
    oracle_max_y         $(_ORACLE_Maxy)        \
    oracle_max_z         $(_ORACLE_Maxz)        \
    oracle_x_tol         $(_ORACLE_Tolx)        \
    oracle_y_tol         $(_ORACLE_Toly)        \
    oracle_z_tol         $(_ORACLE_Tolz)        \
    oracle_dim           $(_ORACLE_Dimension)   \
    oracle_default_geom_column $(_ORACLE_GeometryColumn) \
    oracle_force_index_creation $(_ORACLE_ForceIndexCreation) \
    oracle_index_name    "$(_ORACLE_SpatialIndexName)" \
    AREA                 number(20,6)           \
    PERIMETER            number(20,6)           \
    OBJECTCODE           varchar2(16)               \
    COMPLEXCOD           varchar2(16)               \
    OBJECT               varchar2(16)               \
    OBJECTNAAM           varchar2(18)               \
    BODEM_ID             number(38)
```

Appendix 2: Script for spatial indexing and registering in the Oracle Spatial user_metadata.

```
spool create_spatial_index.log
set echo on
set trimspool on
set serveroutput on

delete from user_sdo_geom_metadata where table_name = 'DTBMERGE';
delete from user_sdo_geom_metadata where table_name = 'BRUGGEN';
delete from user_sdo_geom_metadata where table_name = 'WATER';
delete from user_sdo_geom_metadata where table_name = 'LAAG';
delete from user_sdo_geom_metadata where table_name = 'ONDERGRONDS';
delete from user_sdo_geom_metadata where table_name = 'BODEM';
delete from user_sdo_geom_metadata where table_name = 'VAARWEGVAKKEN';
delete from user_sdo_geom_metadata where table_name = 'DNSN';

insert into user_sdo_geom_metadata values
('DTBMERGE','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

insert into user_sdo_geom_metadata values
('BRUGGEN','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

insert into user_sdo_geom_metadata values
('WATER','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

insert into user_sdo_geom_metadata values
('LAAG','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

insert into user_sdo_geom_metadata values
('BODEM','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

insert into user_sdo_geom_metadata values
('ONDERGRONDS','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

insert into user_sdo_geom_metadata values
('VAARWEGVAKKEN','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',0, 277700, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',310000, 630000, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

insert into user_sdo_geom_metadata values
('DNSN','GEOM', mdsys.SDO_DIM_ARRAY
```

```
(
mdsys.SDO_DIM_ELEMENT ('X',-46100, 278000, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',299400, 868000, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);
commit;

drop index idtbmerge force;
drop index ibruggen force;
drop index iwater force;
drop index ilaag force;
drop index iondergronds force;
drop index ibodem force;
drop index ivaarwegvakken force;
drop index idnsn force;

create index idtbmerge ON dtbmerge (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');

create index ibruggen ON bruggen (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');

create index iwater ON water (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');

create index ilaag ON laag (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');

create index iondergronds ON ondergronds (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');

create index ibodem ON bodem (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');

create index ivaarwegvakken ON vaarwegvakken (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');

create index idnsn ON dnsn (geom)
indextype is mdsys.spatial_index
parameters ('sdo_fanout=32 sdo_indx_dims=2');


call analyze_rtree ('IDTBMERGE');
call analyze_rtree ('IBRUGGEN');
call analyze_rtree ('IBRUGGEN');
call analyze_rtree ('ILAAG');
call analyze_rtree ('IONDERGRONDS');
call analyze_rtree ('IBODEM');
call analyze_rtree ('IVAARWEGVAKKEN');
call analyze_rtree ('IDNSN');

spool off
```

Appendix 3: Validation of geometries

```
spool validate_geom.log
set lines 100
set pages 500
set trimspool on
column status format a80
set echo on

select count(*),g.GEOM.sdo_gtype from DTBMERGE g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='DTBMERGE' and
column_name='GEOM')) status
from DTBMERGE) where status <> 'TRUE';

select count(*),g.GEOM.sdo_gtype from BRUGGEN g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='BRUGGEN' and
column_name='GEOM')) status
from BRUGGEN) where status <> 'TRUE';

select count(*),g.GEOM.sdo_gtype from WATER g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='WATER' and
column_name='GEOM')) status
from WATER) where status <> 'TRUE';

select count(*),g.GEOM.sdo_gtype from LAAG g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='LAAG' and
column_name='GEOM')) status
from LAAG) where status <> 'TRUE';

select count(*),g.GEOM.sdo_gtype from ONDERGRONDS g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='ONDERGRONDS' and
column_name='GEOM')) status
from ONDERGRONDS) where status <> 'TRUE';

select count(*),g.GEOM.sdo_gtype from BODEM g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='BODEM' and
column_name='GEOM')) status
from BODEM) where status <> 'TRUE';

select count(*),g.GEOM.sdo_gtype from VAARWEGVAKKEN g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='VAARWEGVAKKEN' and
column_name='GEOM')) status
from VAARWEGVAKKEN) where status <> 'TRUE';

select count(*),g.GEOM.sdo_gtype from DNSN g group by g.GEOM.sdo_gtype;
select MSLINK,status from (
  select MSLINK, sdo_geom.validate_geometry_with_context (GEOM,
    (select diminfo from user_sdo_geom_metadata where table_name='DNSN' and
column_name='GEOM')) status
from DNSN) where status <> 'TRUE';

spool off
exit;
```

Appendix 4: Overlap of objects in one ‘Layer’

```
#!/bin/sh
# 09      selfovrlap.sh          24-07-2004  TT
#
# Script to test if geometries in a table or layer overlap

BASETABLE=${1:-'water'}
  ORA_USR=${2:-'beheer'}

  ORA_PWD=${ORA_USR}
GEOM_ATTR='geom'
 OID_ATTR='mslink'
TOLERANCE='0.0001'

cat > selfovrlap.sql << EOB
--=========================================================================
set echo on
set lines 120
set pages 500
set trimspool on
set serveroutput on
col area_1  format 99999990.99999
col num_1   format 9999
col area_2  format 99999990.99999
col num_2   format 9999
col area_12 format 99999990.99999
execute dbms_output.enable (2097152);
spool ovrlap_${BASETABLE}.log

drop table ${BASETABLE}_ovrlap purge;
create table ${BASETABLE}_ovrlap (
    oid_1   number(12),
    oid_2   number(12),
    num_1   number(12),
    num_2   number(12),
    area_1  number,
    area_2  number,
    area_12 number);
select count(*) from ${BASETABLE};
set timing on
set echo off

DECLARE

  tmp_geom             ${BASETABLE}.${GEOM_ATTR}%TYPE;
  ovr_geom             ${BASETABLE}.${GEOM_ATTR}%TYPE;
  int_geom             ${BASETABLE}.${GEOM_ATTR}%TYPE;
  ${OID_ATTR}          ${BASETABLE}.${OID_ATTR}%TYPE;
  c_layer              ${BASETABLE}.layer_%TYPE;
--cursor base_table is select * from ${BASETABLE} order by ${OID_ATTR};
  cursor base_table is select * from ${BASETABLE};
  type object_list is  table of ${BASETABLE}.${OID_ATTR}%TYPE;
  tmp_objects          object_list;
  nbase                number(12);
  ndistinct            number(12);
  nchk                 number(12) := 0;
  nn                   number(12) := 0;
  interact             number(12);
  int_area             number;
  tot_interact         number(12) := 0;
  logstr               varchar2(255);
  logfile              UTL_FILE.FILE_TYPE;
  oid_not_unique       EXCEPTION;

BEGIN

  logfile := utl_file.fopen('/var/tmp','selfovrlap.log','a');

utl_file.put_line(logfile,'##############################################################');
  select count(*) into nbase from ${BASETABLE};
  logstr := 'Base table (${ORA_USR}.${BASETABLE}) rows: '||nbase;
  utl_file.put_line(logfile,logstr);
  select count(distinct ${OID_ATTR}) into ndistinct from ${BASETABLE};
  if nbase <> ndistinct then
      dbms_output.put_line('  ');
      dbms_output.put_line('ERROR: Base table (${BASETABLE}) ${OID_ATTR} not unique.');
      raise oid_not_unique;
  end if;
  logstr := 'Potential checks: '||nbase*nbase;
  utl_file.put_line(logfile,logstr);
```

```
  utl_file.put_line(logfile,' ');
  utl_file.fflush(logfile);

  for base_object in base_table loop
      nn := nn + 1;
      logstr := '   check object '||nn||' of '||nbase||' (oid='||base_object.${OID_ATTR}||')';
      utl_file.put_line(logfile,logstr);
      utl_file.fflush(logfile);

      tmp_geom := base_object.${GEOM_ATTR};

      select count(*) into interact from ${BASETABLE} where mdsys.sdo_relate
          (${GEOM_ATTR},tmp_geom,'mask=anyinteract querytype=window') = 'TRUE';
      nchk := nchk + nbase;

      if interact > 1 then

          select ${OID_ATTR} bulk collect into tmp_objects from ${BASETABLE} where
mdsys.sdo_relate
              (${GEOM_ATTR},tmp_geom,'mask=anyinteract querytype=window') = 'TRUE';

          for n in tmp_objects.first..tmp_objects.last loop

              select layer_ into c_layer from ${BASETABLE} where ${OID_ATTR} = tmp_objects(n);
              if tmp_objects(n) <> base_object.${OID_ATTR} and c_layer = base_object.layer_
then

--            if tmp_objects(n) <> base_object.${OID_ATTR} then

                  select ${GEOM_ATTR} into ovr_geom from ${BASETABLE} where ${OID_ATTR} =
tmp_objects(n);
                  int_geom := sdo_geom.sdo_intersection(tmp_geom,ovr_geom,${TOLERANCE});

                  if int_geom is NULL then
                      int_area := 0;
                  else
                      int_area := sdo_geom.sdo_area(int_geom,${TOLERANCE});
                  end if;

                  if int_area > 0 then
                      if base_object.${OID_ATTR} <= tmp_objects(n) then
                          insert into ${BASETABLE}_ovrlap values (base_object.${OID_ATTR},
tmp_objects(n), 0, 0,
                              sdo_geom.sdo_area(tmp_geom,${TOLERANCE}),
sdo_geom.sdo_area(ovr_geom,${TOLERANCE}), int_area);
                      else
                          insert into ${BASETABLE}_ovrlap values (tmp_objects(n),
base_object.${OID_ATTR}, 0, 0,
                              sdo_geom.sdo_area(ovr_geom,${TOLERANCE}),
sdo_geom.sdo_area(tmp_geom,${TOLERANCE}), int_area);
                      end if;

                      tot_interact := tot_interact + 1;
                      logstr := '${BASETABLE} objects '||base_object.${OID_ATTR}||' and
'||tmp_objects(n)||' intersect or overlap';
                      utl_file.put_line(logfile,logstr);
                      utl_file.fflush(logfile);
                  end if;
              end if;
          end loop;
      end if;

--        logstr := '   skipping object '||nn||' of '||nbase||'
(oid='||base_object.${OID_ATTR}||')';
--        utl_file.put_line(logfile,logstr);
--        utl_file.fflush(logfile);

  end loop;

  if tot_interact > 0 then

      select distinct oid_1 bulk collect into tmp_objects from ${BASETABLE}_ovrlap;
      for n in tmp_objects.first..tmp_objects.last loop
          update ${BASETABLE}_ovrlap set num_1 =
              (select count(oid_1) from ${BASETABLE}_ovrlap where oid_1 = tmp_objects(n))
              where oid_1 = tmp_objects(n);
      end loop;

      select distinct oid_2 bulk collect into tmp_objects from ${BASETABLE}_ovrlap;
      for n in tmp_objects.first..tmp_objects.last loop
          update ${BASETABLE}_ovrlap set num_2 =
              (select count(oid_2) from ${BASETABLE}_ovrlap where oid_2 = tmp_objects(n))
```

```
              where oid_2 = tmp_objects(n);
      end loop;
  end if;

  utl_file.put_line(logfile,' ');
  logstr := 'Testing ready, '||tot_interact||' interactions of potential of '||nchk||'.';
  utl_file.put_line(logfile,logstr);
  utl_file.fclose(logfile);
  dbms_output.put_line('      ');
  dbms_output.put_line(logstr);

EXCEPTION
  when oid_not_unique then
      return;
END;
/

set timing off
commit;
set echo on
select distinct * from ${BASETABLE}_ovrlap order by oid_1,oid_2;
spool off
quit
--=====================================================================
EOB

sqlplus ${ORA_USR}/${ORA_PWD} @selfovrlap.sql
rm -f selfovrlap.sql
exit
```

Appendix 5: Computing areas

```
spool calculate_area.log
column total_area format 999999990.99
column original_area format 999999990.99
set trimspool on
set echo on

alter table bodem       add area_tmp number;
alter table water       add area_tmp number;
alter table bruggen     add area_tmp number;
alter table laag        add area_tmp number;
alter table ondergronds add area_tmp number;

update bodem       set area_tmp = sdo_geom.sdo_area(geom,0.0001);
update water       set area_tmp = sdo_geom.sdo_area(geom,0.0001);
update bruggen     set area_tmp = sdo_geom.sdo_area(geom,0.0001);
update laag        set area_tmp = sdo_geom.sdo_area(geom,0.0001);
update ondergronds set area_tmp = sdo_geom.sdo_area(geom,0.0001);
commit;

select sum(area_tmp) total_area from bodem;
select sum(area_tmp) total_area from water;
select sum(area_tmp) total_area from bruggen;
select sum(area_tmp) total_area from laag;
select sum(area_tmp) total_area from ondergronds;

select sum(area) original_area from bodem;
select sum(area) original_area from water;

alter table bodem       drop column area_tmp;
alter table water       drop column area_tmp;
alter table bruggen     drop column area_tmp;
alter table laag        drop column area_tmp;
alter table ondergronds drop column area_tmp;

commit;
spool off
exit;
```

Appendix 6: Union of tables

```
DECLARE

 geom3D mdsys.sdo_geometry;

 vobjectcode varchar2(30);
 vobjectnaam varchar2(60);
 vcomplexcod varchar2(16);
 vwd varchar2(3);
 voever varchar2(1);
 vkmr number (14,3);


 ii number;
 ll number;
 kk number:=0;
 idn number:=0;



BEGIN

-----------------------------------------------------bruggen-------------------------------------
------------
select count(*) into ll from bruggen;
select count(*) into kk from beheer;

FOR ii in 1..ll LOOP

  idn:=kk+ii;
  select objectcode, complexcod, naam, geom into vobjectcode, vcomplexcod, vobjectnaam, geom3D
from bruggen where mslink=ii;
  insert into beheer values (idn, idn, vobjectcode, vcomplexcod, vobjectnaam, geom3D);

END LOOP;

--------------------------------------------------water----------------------------------------
-------------
select count(*) into ll from water;
select count(*) into kk from beheer;

FOR ii in 1..ll LOOP

  idn:=kk+ii;
  select objectcode, complexcod, objectnaam, geom into vobjectcode, vcomplexcod, vobjectnaam,
geom3D from water where mslink=ii;
  insert into beheer values (idn, idn, vobjectcode, vcomplexcod, vobjectnaam, geom3D);

END LOOP;

----------------------------------------------------ondergronds------------------------------
---------------
select count(*) into ll from ondergronds;
select count(*) into kk from beheer;

vobjectnaam:='';
vcomplexcod:='';
vobjectnaam:='';

FOR ii in 1..ll LOOP

  idn:=kk+ii;
  select objectcode, geom into vobjectcode, geom3D from ondergronds where mslink=ii;
  insert into beheer values (idn, idn, vobjectcode, vcomplexcod, vobjectnaam, geom3D);

END LOOP;

-------------------------------------------------------laag-------------------------------------
----------------
---------------------------------------------id is created with sequence-----------------------
----------------
select count(*) into ll from laag;
select count(*) into kk from beheer;

vobjectnaam:='';

FOR ii in 1..ll LOOP

  idn:=kk+ii;
```

```
  select objectcode,complexcod,geom into vobjectcode, vcomplexcod, geom3D from laag where
id=ii;
  insert into beheer values (idn, idn, vobjectcode, vcomplexcod, vobjectnaam, geom3D);

END LOOP;

-------------------------------------------------bodem-----------------------------------------
----------------
-----------------------------------------------------id is created with sequence---------------
----------------
select count(*) into ll from bodem;
select count(*) into kk from beheer;

FOR ii in 1..ll LOOP

  idn:=kk+ii;

  select objectcode,complexcod,objectnaam, geom into vobjectcode, vcomplexcod, vobjectnaam,
geom3D from bodem where id=ii;
  insert into beheer values (idn, idn, vobjectcode, vcomplexcod, vobjectnaam, geom3D);

END LOOP;


-------------------------------------------------------EV---------------------------------------
--------------------

select count(*) into ll from eg;
select count(*) into kk from beheer;
vcomplexcod:='';

FOR ii in 1..ll LOOP

  idn:=kk+ii;

  select substr (WD, 1, 3), substr (OEVER,1,1),KMR,omschrijvi, geom into vwd,voever, vkmr,
vobjectnaam, geom3D
  from eg
  where mslink=ii;

  if voever='K' then voever:='-';

  end if;

  vobjectcode:=vwd||'.EV.'||voever||'.'||vkmr;

  insert into beheer values (idn, idn, vobjectcode, vcomplexcod, vobjectnaam, geom3D);

END LOOP;

END;


/
```

Appendix 7: Creating a point from a line for a line from NWB.

```
drop view test_line;

create view test_line as select mslink, geom from vaarwegvakken where vaar_id=2872;

delete from user_sdo_geom_metadata where table_name = 'TEST_LINE';

insert into user_sdo_geom_metadata values
('TEST_LINE','GEOM', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

---------------------------------buffer around line-----------------------------

drop view test_polygon;

create view test_polygon as
select c.mslink as mslink, sdo_geom.sdo_buffer (c.geom, m.diminfo, 1) as buffer
from test_line c, user_sdo_geom_metadata m
where m.table_name='TEST_LINE' and m.column_name='GEOM' and
c.mslink=2872;

delete from user_sdo_geom_metadata where table_name='TEST_POLYGON';

insert into user_sdo_geom_metadata values
('TEST_POLYGON','BUFFER', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
);

--------------------------------centroid of buffer---------------------------------
drop view centroid_buffer;

create view centroid_buffer as
select c.mslink as mslink, sdo_geom.sdo_centroid (c.buffer, m.diminfo) as point
from test_polygon c, user_sdo_geom_metadata m
where m.table_name='TEST_POLYGON' and m.column_name='BUFFER' and
c.mslink=2872;


delete from user_sdo_geom_metadata where table_name = 'CENTROID_BUFFER';

insert into user_sdo_geom_metadata values
('CENTROID_BUFFER','POINT', mdsys.SDO_DIM_ARRAY
(
mdsys.SDO_DIM_ELEMENT ('X',174600, 200100, .00001),
mdsys.SDO_DIM_ELEMENT ('Y',306400, 362600, .00001),
mdsys.SDO_DIM_ELEMENT ('Z',-20, 800, .00001)
), NULL
)



/
```

Appendix 8: Point-in-Polygon match.

```
#!/bin/sh
# 06     LineInPoly_match.sh  29-08-2004  TT
#
# Script to match line geometries with areas

      ORA_USR=${1:-'beheer'}
      ORA_PWD=${2:-'beheer'}
    LINETABLE=${3:-'vaarwegen'}
    AREATABLE=${4:-'bodemvaargeul_polyg'}
LINEGEOM_ATTR=${5:-'geom'}
AREAGEOM_ATTR=${6:-'geom'}
  LINEID_ATTR=${7:-'vaar_id'}
  AREAID_ATTR=${8:-'beh_id'}
      DEF_TOL=${9:-'0.00001'}

#-------------------------------------------------------------------------------
cat > tmp_match_line.sql << EOB

set lines 120
set pages 500
set trim on
set trimspool on
set serveroutput on
column pct_inside format 990.9990

drop view base_line_table;
drop view base_area_table;
drop table ${LINETABLE}_match purge;

spool LiP_match_${LINETABLE}.log
set echo on

select count(*) from ${ORA_USR}.${LINETABLE};
select count(*) from ${ORA_USR}.${AREATABLE};

create view base_line_table as select * from ${LINETABLE} ll
  where ll.${LINEGEOM_ATTR}.sdo_gtype in (2002,3002);
create view base_area_table as select * from ${AREATABLE} aa
  where aa.${AREAGEOM_ATTR}.sdo_gtype in (2003,3003);

create table ${LINETABLE}_match (
  oid_line   number(12),
  oid_area   number(12),
  relation   varchar2(32),
  pct_inside number(9,4)
);
set echo off
set timing on

DECLARE

  cursor line_table is select ${LINEID_ATTR},${LINEGEOM_ATTR} from base_line_table order by
${LINEID_ATTR};
  type object_list is table of ${AREATABLE}.${AREAID_ATTR}%TYPE;
  tmp_objects         object_list;
  dim_info            mdsys.sdo_dim_array;
  tmp_line            mdsys.sdo_geometry;
  tmp_area            mdsys.sdo_geometry;
  n                   number(12) := 0;
  nn                  number(12);
  nlin                number(12);
  npoly               number(12);
  nlines              number(12);
  nareas              number(12);
  interact            number(12);
  tot_interact        number(12) := 0;
  nchk                number(12) := 0;
  tolerance           number;
  total_len           number;
  inside_len          number;
  pct_inside          number;
  geomatch            varchar2(32);
  logstr              varchar2(255);
  logfile             UTL_FILE.FILE_TYPE;
  oid_not_unique      EXCEPTION;

BEGIN
```

```
  logfile := utl_file.fopen('/var/tmp','LineInPoly_match.log','a');

utl_file.put_line(logfile,'#################################################################
###############');
  utl_file.fflush(logfile);
  select count(*) into nlin from ${LINETABLE};
  select count(*) into npoly from ${AREATABLE};
  logstr := 'Starting match run for ${ORA_USR}: ${LINETABLE} ('||nlin||' records),
${AREATABLE} ('||npoly||' records)';
  utl_file.put_line(logfile,logstr);
  select count(*) into nlines from base_line_table;
  logstr := 'Number of lines in ${LINETABLE}: '||nlines;
  utl_file.put_line(logfile,logstr);
  select count(*) into nareas from base_area_table;
  logstr := 'Number of areas in ${AREATABLE}: '||nareas;
  utl_file.put_line(logfile,logstr);
  dbms_output.put_line('      ');

  select count(distinct ${LINEID_ATTR}) into nn from ${LINETABLE};
  if nlin <> nn then
    logstr := 'ERROR: Line table ${ORA_USR}.${LINETABLE} ${LINEID_ATTR} attribute not
unique.';
    utl_file.put_line(logfile,logstr);
    utl_file.fclose(logfile);
    dbms_output.put_line(logstr);
    raise oid_not_unique;
  end if;
  select count(distinct ${AREAID_ATTR}) into nn from ${AREATABLE};
  if npoly <> nn then
    logstr := 'ERROR: Polygon table ${ORA_USR}.${AREATABLE} ${AREAID_ATTR} attribute not
unique.';
    utl_file.put_line(logfile,logstr);
    utl_file.fclose(logfile);
    dbms_output.put_line(logstr);
    raise oid_not_unique;
  end if;
  logstr := 'Potential checks for line in polygon matching: '||nlines*nareas;
  utl_file.put_line(logfile,logstr);

  select count(*) into nn from user_sdo_geom_metadata
    where table_name = upper('${AREATABLE}') and column_name = upper('${AREAGEOM_ATTR}');
  if nn = 1 then
    select diminfo into dim_info from user_sdo_geom_metadata
      where table_name = upper('${AREATABLE}') and column_name = upper('${AREAGEOM_ATTR}');
    tolerance := dim_info(1).sdo_tolerance;
    if not tolerance is NULL then
      logstr    := 'Area metadata tolerance is '||tolerance||', will be used as matching
tolerance';
    else
      tolerance := ${DEF_TOL};
      logstr    := 'WARNING: Cannot calculate proper tolerance, default of '||tolerance||'
will be used';
    end if;
  else
     tolerance := ${DEF_TOL};
     logstr    := 'WARNING: Cannot retrieve proper tolerance, default of '||tolerance||' will
be used';
  end if;
  dbms_output.put_line(logstr);
  utl_file.put_line(logfile,logstr);
  utl_file.put_line(logfile,' ');
  utl_file.fflush(logfile);

  for line in line_table loop

    n := n + 1;
    logstr := 'Checking line '||n||' of '||nlines||' (ID='||line.${LINEID_ATTR}||')';
    utl_file.put_line(logfile,logstr);
    utl_file.fflush(logfile);
    tmp_line := line.${LINEGEOM_ATTR};

    select count(*) into interact from base_area_table where mdsys.sdo_relate
      (${AREAGEOM_ATTR},tmp_line,'mask=anyinteract querytype=window') = 'TRUE';
    nchk := nchk + nareas;
    if interact > 0 then
      tot_interact := tot_interact + interact;
      select ${AREAID_ATTR} bulk collect into tmp_objects from base_area_table where
mdsys.sdo_relate
        (${AREAGEOM_ATTR},tmp_line,'mask=anyinteract querytype=window') = 'TRUE';

      for n in tmp_objects.first..tmp_objects.last loop
```

```
        select ${AREAGEOM_ATTR} into tmp_area from base_area_table where ${AREAID_ATTR} =
tmp_objects(n);
        geomatch := sdo_geom.relate(tmp_line,'determine',tmp_area,tolerance);

        if geomatch not in ('TOUCH','DISJOINT') then
          total_len  := sdo_geom.sdo_length(tmp_line,tolerance);
          inside_len :=
sdo_geom.sdo_length(sdo_geom.sdo_intersection(tmp_area,tmp_line,tolerance),tolerance);
          pct_inside := (inside_len / total_len) * 100.0;
        else
          pct_inside := 0.0;
        end if;
        insert into ${LINETABLE}_match values
(line.${LINEID_ATTR},tmp_objects(n),geomatch,pct_inside);
        logstr := 'Relation between line '||line.${LINEID_ATTR}||' and area
'||tmp_objects(n)||' is: '||geomatch||' (pct_inside='||to_char(pct_inside,'990.9990')||')';
        utl_file.put_line(logfile,logstr);
        utl_file.fflush(logfile);
      end loop;

--  else
--    insert into ${LINETABLE}_match values (line.${LINEID_ATTR},NULL,'not matched',0.0);
    end if;

  end loop;
  commit;

  utl_file.put_line(logfile,' ');
  logstr := 'Line matching ready, '||tot_interact||' interactions of potential of '||nchk||'
(tol='||tolerance||').';
  dbms_output.put_line(logstr);
  utl_file.put_line(logfile,logstr);
  if nlines > 0 then
    select count(distinct oid_line) into nn from ${LINETABLE}_match where relation in
('INSIDE','OVERLAPBDYDISJOINT');
    logstr := 'LiP matches '||nn||' of '||nlines||' ('||round(nn/nlines*100,2)||' %)';
    dbms_output.put_line(logstr);
    utl_file.put_line(logfile,logstr);
  end if;
  utl_file.fclose(logfile);

EXCEPTION
  when oid_not_unique then
    return;
END;
/

set timing off
drop view base_line_table;
drop view base_area_table;
set echo on
select * from ${ORA_USR}.${LINETABLE}_match order by oid_line,oid_area;

spool off
quit
EOB

sqlplus ${ORA_USR}/${ORA_PWD} @tmp_match_line.sql
/usr/bin/rm -f tmp_match_line.sql
exit
```

Appendix 9: Line-in-Polygon

```
#!/bin/sh
# 06     LineInPoly_match.sh  29-08-2004  TT
#
# Script to match line geometries with areas

      ORA_USR=${1:-'beheer'}
      ORA_PWD=${2:-'beheer'}
    LINETABLE=${3:-'vaarwegen'}
    AREATABLE=${4:-'bodemvaargeul_polyg'}
LINEGEOM_ATTR=${5:-'geom'}
AREAGEOM_ATTR=${6:-'geom'}
  LINEID_ATTR=${7:-'vaar_id'}
  AREAID_ATTR=${8:-'beh_id'}
      DEF_TOL=${9:-'0.00001'}

#-------------------------------------------------------------------------------
cat > tmp_match_line.sql << EOB

set lines 120
set pages 500
set trim on
set trimspool on
set serveroutput on
column pct_inside format 990.9990

drop view base_line_table;
drop view base_area_table;
drop table ${LINETABLE}_match purge;

spool LiP_match_${LINETABLE}.log
set echo on

select count(*) from ${ORA_USR}.${LINETABLE};
select count(*) from ${ORA_USR}.${AREATABLE};

create view base_line_table as select * from ${LINETABLE} ll
  where ll.${LINEGEOM_ATTR}.sdo_gtype in (2002,3002);
create view base_area_table as select * from ${AREATABLE} aa
  where aa.${AREAGEOM_ATTR}.sdo_gtype in (2003,3003);

create table ${LINETABLE}_match (
  oid_line   number(12),
  oid_area   number(12),
  relation   varchar2(32),
  pct_inside number(9,4)
);
set echo off
set timing on

DECLARE

  cursor line_table is select ${LINEID_ATTR},${LINEGEOM_ATTR} from base_line_table order by
${LINEID_ATTR};
  type object_list is table of ${AREATABLE}.${AREAID_ATTR}%TYPE;
  tmp_objects         object_list;
  dim_info            mdsys.sdo_dim_array;
  tmp_line            mdsys.sdo_geometry;
  tmp_area            mdsys.sdo_geometry;
  n                   number(12) := 0;
  nn                  number(12);
  nlin                number(12);
  npoly               number(12);
  nlines              number(12);
  nareas              number(12);
  interact            number(12);
  tot_interact        number(12) := 0;
  nchk                number(12) := 0;
  tolerance           number;
  total_len           number;
  inside_len          number;
  pct_inside          number;
  geomatch            varchar2(32);
  logstr              varchar2(255);
  logfile             UTL_FILE.FILE_TYPE;
  oid_not_unique      EXCEPTION;

BEGIN
```

```
  logfile := utl_file.fopen('/var/tmp','LineInPoly_match.log','a');

utl_file.put_line(logfile,'#################################################################
###############');
  utl_file.fflush(logfile);
  select count(*) into nlin from ${LINETABLE};
  select count(*) into npoly from ${AREATABLE};
  logstr := 'Starting match run for ${ORA_USR}: ${LINETABLE} ('||nlin||' records),
${AREATABLE} ('||npoly||' records)';
  utl_file.put_line(logfile,logstr);
  select count(*) into nlines from base_line_table;
  logstr := 'Number of lines in ${LINETABLE}: '||nlines;
  utl_file.put_line(logfile,logstr);
  select count(*) into nareas from base_area_table;
  logstr := 'Number of areas in ${AREATABLE}: '||nareas;
  utl_file.put_line(logfile,logstr);
  dbms_output.put_line('      ');

  select count(distinct ${LINEID_ATTR}) into nn from ${LINETABLE};
  if nlin <> nn then
    logstr := 'ERROR: Line table ${ORA_USR}.${LINETABLE} ${LINEID_ATTR} attribute not
unique.';
    utl_file.put_line(logfile,logstr);
    utl_file.fclose(logfile);
    dbms_output.put_line(logstr);
    raise oid_not_unique;
  end if;
  select count(distinct ${AREAID_ATTR}) into nn from ${AREATABLE};
  if npoly <> nn then
    logstr := 'ERROR: Polygon table ${ORA_USR}.${AREATABLE} ${AREAID_ATTR} attribute not
unique.';
    utl_file.put_line(logfile,logstr);
    utl_file.fclose(logfile);
    dbms_output.put_line(logstr);
    raise oid_not_unique;
  end if;
  logstr := 'Potential checks for line in polygon matching: '||nlines*nareas;
  utl_file.put_line(logfile,logstr);

  select count(*) into nn from user_sdo_geom_metadata
    where table_name = upper('${AREATABLE}') and column_name = upper('${AREAGEOM_ATTR}');
  if nn = 1 then
    select diminfo into dim_info from user_sdo_geom_metadata
      where table_name = upper('${AREATABLE}') and column_name = upper('${AREAGEOM_ATTR}');
    tolerance := dim_info(1).sdo_tolerance;
    if not tolerance is NULL then
      logstr    := 'Area metadata tolerance is '||tolerance||', will be used as matching
tolerance';
    else
      tolerance := ${DEF_TOL};
      logstr    := 'WARNING: Cannot calculate proper tolerance, default of '||tolerance||'
will be used';
    end if;
  else
     tolerance := ${DEF_TOL};
     logstr    := 'WARNING: Cannot retrieve proper tolerance, default of '||tolerance||' will
be used';
  end if;
  dbms_output.put_line(logstr);
  utl_file.put_line(logfile,logstr);
  utl_file.put_line(logfile,' ');
  utl_file.fflush(logfile);

  for line in line_table loop

    n := n + 1;
    logstr := 'Checking line '||n||' of '||nlines||' (ID='||line.${LINEID_ATTR}||')';
    utl_file.put_line(logfile,logstr);
    utl_file.fflush(logfile);
    tmp_line := line.${LINEGEOM_ATTR};

    select count(*) into interact from base_area_table where mdsys.sdo_relate
      (${AREAGEOM_ATTR},tmp_line,'mask=anyinteract querytype=window') = 'TRUE';
    nchk := nchk + nareas;
    if interact > 0 then
      tot_interact := tot_interact + interact;
      select ${AREAID_ATTR} bulk collect into tmp_objects from base_area_table where
mdsys.sdo_relate
        (${AREAGEOM_ATTR},tmp_line,'mask=anyinteract querytype=window') = 'TRUE';

      for n in tmp_objects.first..tmp_objects.last loop
```

```
        select ${AREAGEOM_ATTR} into tmp_area from base_area_table where ${AREAID_ATTR} =
tmp_objects(n);
        geomatch := sdo_geom.relate(tmp_line,'determine',tmp_area,tolerance);

        if geomatch not in ('TOUCH','DISJOINT') then
          total_len  := sdo_geom.sdo_length(tmp_line,tolerance);
          inside_len :=
sdo_geom.sdo_length(sdo_geom.sdo_intersection(tmp_area,tmp_line,tolerance),tolerance);
          pct_inside := (inside_len / total_len) * 100.0;
        else
          pct_inside := 0.0;
        end if;
        insert into ${LINETABLE}_match values
(line.${LINEID_ATTR},tmp_objects(n),geomatch,pct_inside);
        logstr := 'Relation between line '||line.${LINEID_ATTR}||' and area
'||tmp_objects(n)||' is: '||geomatch||' (pct_inside='||to_char(pct_inside,'990.9990')||')';
        utl_file.put_line(logfile,logstr);
        utl_file.fflush(logfile);
      end loop;

--  else
--    insert into ${LINETABLE}_match values (line.${LINEID_ATTR},NULL,'not matched',0.0);
    end if;

  end loop;
  commit;

  utl_file.put_line(logfile,' ');
  logstr := 'Line matching ready, '||tot_interact||' interactions of potential of '||nchk||'
(tol='||tolerance||').';
  dbms_output.put_line(logstr);
  utl_file.put_line(logfile,logstr);
  if nlines > 0 then
    select count(distinct oid_line) into nn from ${LINETABLE}_match where relation in
('INSIDE','OVERLAPBDYDISJOINT');
    logstr := 'LiP matches '||nn||' of '||nlines||' ('||round(nn/nlines*100,2)||' %)';
    dbms_output.put_line(logstr);
    utl_file.put_line(logfile,logstr);
  end if;
  utl_file.fclose(logfile);

EXCEPTION
  when oid_not_unique then
    return;
END;
/

set timing off
drop view base_line_table;
drop view base_area_table;
set echo on
select * from ${ORA_USR}.${LINETABLE}_match order by oid_line,oid_area;

spool off
quit
EOB

sqlplus ${ORA_USR}/${ORA_PWD} @tmp_match_line.sql
/usr/bin/rm -f tmp_match_line.sql
exit
```

Bijlage B

OpenGIS compliance check of the beheerkaart-Nat data model

Title:	OGC check of the beheerkaart-Nat model
Creator:	IONIC Software S.A. (Alain Kabamba)
Date:	2004-10-21
Subject:	
Publisher	IONIC Software S.A.
Type	Text
Description	
Contributor	
Report number	IONIC-2005-GISRP-001

Table Of Contents

1 TERMINOLOGY 5

2 INTRODUCTION 9

3 CONCEPTS 11

3.1 Interoperability 11

3.2 Web Services 11

3.3 GI Standards Organizations 11

3.3.1 ISO 11

3.3.2 OpenGIS 12

3.4 UML (Universal Modelling Language) – OMG 12

3.5 The OpenGIS Web Feature Server (WFS) 13

3.6 Features/Feature Types 14

3.7 GML application schema 15

4 THE METHODOLOGY 17

5 UNDERSTANDING THE BEHEERKAART-NAT LOGICAL MODEL 19

6 THE GML MODEL 23

6.1 The Object/Property rule 23

7 THE BEHEERKAART-NAT GML APPLICATION SCHEMA 27

7.1 Introduction 27

7.2 The name space 27

7.3 Features 28

7.4 Global element declaration 28

7.5 Feature types 28

7.6 Spatial type 31

7.7 Metadata 33

7.8 Association/properties 34

8 WFS SCHEMA AND MAPPING CONFIGURATION 37

8.1 What is a mapping **37**

8.2 Beheerkaart-Nat GML application schema mapping to Oracle **38**

8.3 Configuring a transactional WFS **41**

8.3.1 Transactional features 41

8.3.2 Unique identifier 41

8.3.3 Locking mechanism 41

8.3.4 Update the schema and mapping files 42

9 SAMPLE REQUESTS ON THE BEHEERKAART-NAT WFS **43**

9.1 GetCapabilities **43**

9.2 DescribeFeatureType **44**

9.3 GetFeature **45**

9.4 LockFeature (POST request) **47**

9.5 Transactions **48**

10 BIBLIOGRAPHY **49**

1 Terminology

API

(Application Programming Interface) Defines how to access a software-based service. An API is a published specification that describes how other software programs can access the functions of an automated service.

Application Schema.

An application schema defines the content and structure of data; Specifications of operations for manipulating and processing data by an application.

Gazetteer

A gazetteer is a list of geographic names, together with references to their location and other descriptive information. A geographic name is a place name or a feature name.

GI

Geospatial Information

GML

Geographic Markup Language.

HTTP

(HyperText Transfer Protocol) The Web's communication standard, referenced in the `http://` that appears at the beginning of every web page address (an extra 's' on the end, as in `https://` denotes the secure, encrypted form). Stabilized at HTTP/1.1, it defines the universal mechanism for exchanging application-level messages between Web devices. All web services run over HTTP.

Internet

The shared global computing network. A network based on standards including Internet Protocol (IP), Simple Mail Transfer Protocol (SMTP) and the Domain Name System (DNS), which enables global communications between all connected computing devices. It provides the

platform for web services and the WorldWide Web.

ISO

International Organization for Standardization

Java

Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces a completely object-oriented view of programming. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build small application modules or applets for use as part of a Web page. Applets make it possible for a Web page user to interact with the page.

JSP

JavaServer Pages. A server-side technology, JavaServer pages are an extension to the Java servlet technology that was developed by Sun. JSPs have dynamic scripting capability that works in tandem with HTML code, separating the page logic from the static elements -- the actual design and display of the page. Embedded in the HTML page, the Java source code and its extensions help make the HTML more functional, being used in dynamic database queries, for example. JSPs are not restricted to any specific platform or server.

Servlet Engine

A Java based Web Application container

Servlet

A Java program that runs as part of a network service, typically a Web server and responds to requests from clients. Servlets extend a Web server by generating content dynamically.

Struts

Struts is an open framework for designing Web applications. It helps users to develop application structures.

UML

Universal Modeling Language
UML is a standard notation and modeling technique for analyzing real-world objects, developing systems, designing software modules in object-oriented approach.

URI(Universal Resource Identifier)

The address of an Internet resource. A URI is the unique name used to access the resource. It is not necessarily a specific file location (it may be a call to an application or a database, for example), which is why it is preferred over the similar acronym URL.

URL (Universal Resource Locator);

Uniform Resource Locator, the global address of documents and other resources on the World Wide Web. The first part of the address indicates what protocol to use, and the second part specifies the IP address or the domain name where the resource is located.

OpenGIS

The Open GeoSpatial Consortium is an international organization that focuses on specifying interfaces to access and discover geospatial data over the Internet.

W3C

(World Wide Web Consortium) The standards body of the Web. W3C defines and develops the protocols that ensure the universality of the Web, including HTML, HTTP, XML and many others. It is heavily involved in the development of web services standards, most notably SOAP and WSDL. A major project of the W3C is to use XML as the foundation for creating the Semantic Web.

XML

eXtensible Mark-up Language, a specification developed by the W3C. XML is a pared-down version of Standard Generalized Mark-Up Language, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

XML Schema

(eXtensible Markup Language) The data tagging language of web services. XML is not so much a language as a standardized set of rules for adding structure to any form of data using a system of markup tags. Anyone can create their own markup vocabulary (called an XML Schema), and XML ensures that the structure will be intelligible to anyone else who consults the XML Schema document. More importantly, referring to an XML Schema enables XML-aware

software to automatically manipulate the data without needing advance knowledge of the structure.

2 Introduction

This report provides an analysis of the implementation of the AGI - Beheerkaart-Nat data model as a GML application schema. After introducing some basic OGC standards concepts, we will explain the methodology we have used and then provide an overview of the steps undertaken to set up an OGC WFS on top of the Beheerkaat-Nat data using the Beheerkaart-Nat GML application schema.

The main goal of this report are to provide recommendations for OpenGIS complaince improvements in the model, shed some light on some specific topic like the mapping of the application schema to a database structure (in this case the Beheerkaart-Nat Oracle Spatial database), the multi-scale management and the transactions.

The Figure 1 gives a high level view of the different modules we will be dealing with in this document.

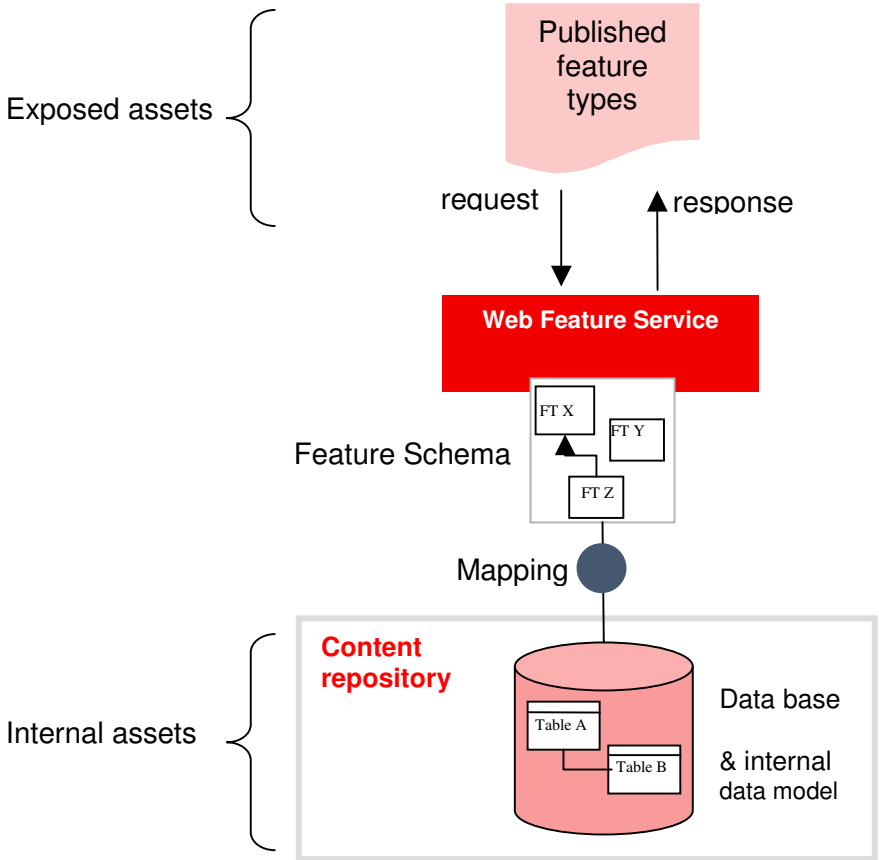


Figure 1 - Internal Data Model versus Exposed Feature Types

3 Concepts

3.1 Interoperability

The interoperability can be defined as the ability for one application, whether server or client, to discover another application's capabilities, to understand them and to dynamically use them from the application side. It is also possible to bind several online services together in a dynamic "service chain".

At a very high level, interoperability can be seen as defining a shared semantic for all process to understand each other. At a technical level, interoperability means building applications upon robust industry standards providing the application a set of well known interfaces with a robust semantic such as those defined by ISO and OpenGIS.

3.2 Web Services

As defined by the World Wide Web Consortium (W3C) [W3C04], web services are "*software applications identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via internet-based protocols* »."

More clearly, a Web Service is application or business logic that is accessible using standard Internet protocols. Web Services combine the best aspects of component-based development and the World Wide Web. Like components, Web Services represent black-box functionality that can be used and reused without regard of how the service is implemented

3.3 GI Standards Organizations

3.3.1 ISO

The International Organization for Standardization (ISO) is a worldwide federation of national standards bodies from more than 140 countries, one from each country. The mission of ISO is to promote the development of standardization and related activities in the world with a view to facilitating the international exchange of goods and services, and to developing cooperation in the spheres of intellectual, scientific, technological and economic activity. ISO's work results in international agreements, which are published as International Standards.

The ISO Geomatics technical committee (TC211) aims to establish a structured set of standards for information concerning objects or phenomena that are directly or indirectly associated with a location relative to the Earth. We will be mentioning some of the ISO TC211 specifications in this document like the ISO19115 Geographic Information – Metadata or the ISO19136 Geographic Information – Geography Markup Language.

3.3.2 OpenGIS

The Open Geospatial Consortium is an international organization that focuses on specifying interfaces to access and discover geospatial data. Consortium members include many GI vendors who are involved in the specification process and provide implementations. Consisting of more than 200 members, the OpenGIS represents the biggest organization active into the geospatial field.

ISO and OpenGIS work together to integrate their views and to avoid writing two different specifications on the same subject to harmonize the geo-spatial standards. For example the OpenGIS Web Map Server specification has also been adopted by ISO under the number ISO 19128 and the OpenGIS specification for metadata is ISO 19115.

The benefits of OpenGIS specifications are as follows:

1. Specifications based on the need to separate the interface of a system from its internal working. We are thus talking about an interface model and not about a specific database system or engine.

2. Usage of ISO standards: the ISO19115 is the OpenGIS standard for metadata and is used by the OpenGIS Web services specificationsv(WMS, WFS, WCS, etc.). Web Services also use ISO19139 for the XML encoding of the metadata.

3. The vision that those who maintain data should make them accessible online through web services. Since its inception, OpenGIS has understood that the most important issue in highly distributed systems is the self description of the service through its metadata.

The OpenGIS standards that are involved in this report:

- WMS - Web Map Service
- WFS - Web Feature Service
- GML - Geographic Mark-up Language
- Filter - Filter Encoding
- Map Context

3.4 UML (Universal Modelling Language) – OMG

UML is a standard notation and modeling technique for analyzing real-world objects, developing systems, designing software modules in an object-oriented approach. UML has been fostered and now is accepted as a standard for creating standard architecture for object technology. UML is managed by the Object Management Group (OMG) and is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for enterprise applications

3.5 The OpenGIS Web Feature Server (WFS)

A Web Feature Server (WFS) is a Web interface that allows one to publish and deploy geographic feature data locally or across the Internet. The WFS is an official specification from the OpenGIS which *"provides an interface for describing data manipulation operations (create, delete, update, and get features) on geographic data stored in databases that are accessible over Internet infrastructure (World Wide Web, WANs, LANs, etc.). The WFS interface exposes the data in these repositories as Geography Mark-up Language (GML)*". A WFS based on OpenGIS specifications can be implemented in either its "Basic" version, or a "Transactional" version. Basic WFSs are READ-ONLY Web Feature Servers while a transactional Web Feature Server (WFS-T) can support all the interfaces of a basic Web Feature Server as well as implement a type of file locking (LockFeature) and transactional interfaces. This allows multiple users to do a variety of manual operations, such as to create/update/delete data or make informatory requests simultaneously on the same geographic feature.

The OpenGIS Web Feature Service (WFS 1.0) became an adopted specification in January 2003. When a client sends a request for information to the OGC WFS, the service sends a response message that includes geographic feature data in GML.

Data requests may be filtered using the OpenGIS Common Query Language (CQL). Two possible examples of CQL expressions are: "Find all the bridges inside this " and "Find all bridges of the river X" GML is used to express parts of the request—such as GML feature type names, GML geometries and GML properties—in combination with the OpenGIS CQL filter grammar. The response from the WFS is encoded in GML.

A client can determine the types of features supported by a given WFS through the capabilities interface. This can enable a WFS client to determine, for example, that a given WFS can serve geographic information on bridges, rivers...

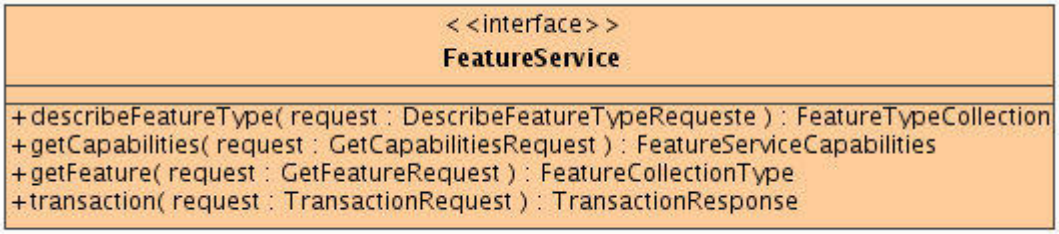


Figure 2 - WFS Operations

The WFS Operations are:

- **GetCapabilities** -You can send a request to a WFS asking what capabilities the server holds. Specifically, the WFS should indicate which feature types it can service, what operations are supported on each feature type, and what metadata is associated with each data source;

- **DescribeFeatureType** - The server should be able, upon request, to describe the structure of any feature type it can service in an XML schema;

- **GetFeature** - The server should be able to service a request to retrieve feature instances. In addition, the client should be able to specify which feature properties to fetch and constrain the query spatially and non-spatially.

- **LockFeature** - A transactional WFS may be able to process a lock request on one or more instances of a feature type for the duration of a transaction. This ensures that transactions in series are supported.

- **Spatial Operators and Filters** - There are several different spatialfilters and operators that can be issued to the WFS.

- **Transaction** - Transactional WFS may be able to service transaction requests. A transaction request is composed of operations that modify features; that is create, update, and delete operations on geographic features.

The last voted specification of the WFS is the version 1.0. That version is natively GML2, optionally GML3. The next version 1.1 will be natively GML3 and optionally GML2.

3.6 Features/Feature Types

According to the Open Geospatial Consortium (OpenGIS): "A feature is an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth."
These features are a standard representation of vector datasets as they deal with geographic information as a set of vector information. These features are also aggregated in classes of features, called feature type.
Each of these feature types represents a class of real world objects. For example in Figure 3, all the road features are held in a feature type called `RoadType`.

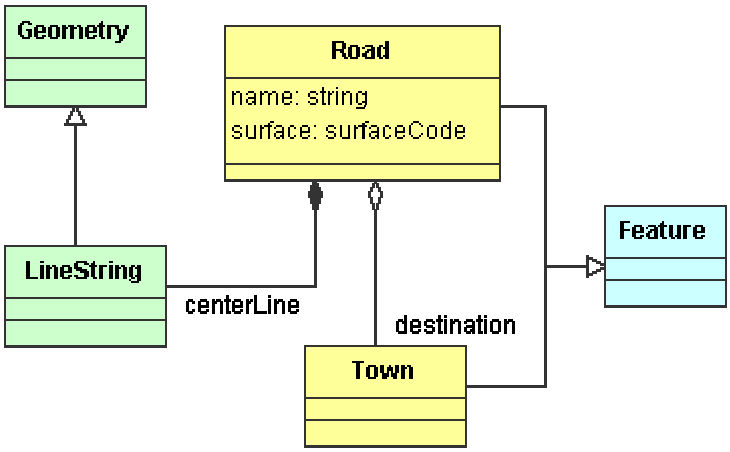


Figure 3 – Features class diagram

3.7 GML application schema

A schema can be viewed as a collection (vocabulary) of type definitions and element declarations whose names belong to a particular namespace called a target namespace.
The namespace in XML schema is used to provide “context” and avoid name collisions. The use of namespaces also promotes the re-use of elements from other domains and allows a reader to visually identify the origin of each element/attribute in an instance document.

4 The methodology

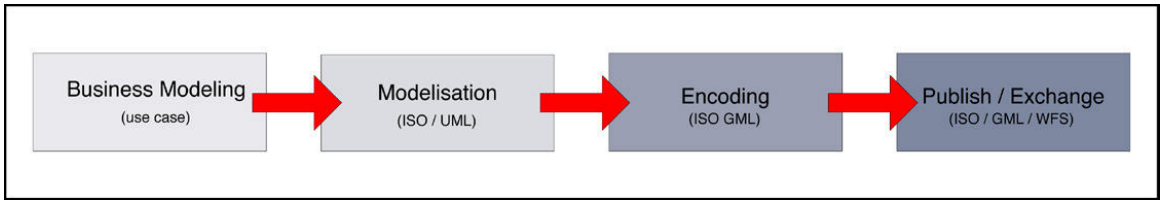


Figure 4 – Methodology

To specify an interoperable data infrastructure, we usually use a four steps process like in Figure4:

- Firstly, we conceptualise the types of problems the infrastructure intends to solve. This step captures use cases and defines scenarios in which the infrastructure could participate. This ensures that the system is able to help resolve a specific set of problems.

- Secondly, we undertake the modelling of the concept by defining a UML model of the Application Schema that the service will expose to the world. This modelling step formally describes how to structure information. This step uses as much standard or existing domain specific application schema information as possible.

- The third step translates the UML model into a GML application schema, an XML based description of the model. This translation is done according to a set of translation rules between UML and GML. The GML application schema provides a ready-to-use description of the model using a standard description mechanism and it allows usage of all related specifications issued by OpenGIS.

- The fourth and last step is to set up services that publish the data in an interoperable way by using the OpenGIS Web Feature Server (WFS) specification. The WFS provides GML information using a self-describing interface. These services are therefore discoverable and usable at run-time.

In the scope of the "Beheerkaart – Nat data model design", AGI and TU Delft made the first and the second steps. In this document we will be doing the third and the fourth steps: translating the Beheerkaart – Nat UML model into a GML application schema and set up a WFS upon the Beheerkaart – Nat database.

5 Understanding the Beheerkaart-Nat logical model

The Figure 5 below shows a simplified view of the beheerkaart-Nat logical model. The logical model describe the objects hierarchy like this:

- At the top level we have a Feature object
- A beheerObject is a feature: it extends the feature object
- Bodems, Waters,Oevers, Expoatatie, Verkeer and Kunstwerk are BeheerObjects: they extend the BeheerObject
- Kunstwerk1, Kunstwerk2 and Kunstwerk3 are Kunstwerk : they extend the Kunstwerk object.
- Lbodems, Lwater, Lbruggen, Lmaaiveld and Londergronds are containers or aggregation of Bodems, Waters…

The issue here is that according to the model, the object Lmaaiveld is an aggregation of Oevers, Expotatie, Kunstwerk2 and Verkeer. If we look at the Figure 5 below, we can see that they are different objects and so should play different roles in the aggregation. As a consequence Lmaaiveld would have four different collections each containing Oevers, Expotatie, Kunstwerk2 or Verkeer, but I guess this was not the expected result. To have the expected result we should have something like what we have in Figure 6 where we see that Lmaaiveld is an aggregation of an abstract *Maaiveld* object and the Oevers, Expoatatie, Kunstwerk2 and Verkeer objects extending the *Maaiveld* abstract object. As a result a Lmaaiveld will one collection of *Maaiveld* objects which could be either Oevers, Expotatie, Kunstwerk2 or Verkeer, but of course this will imply a complete redesign of the model.

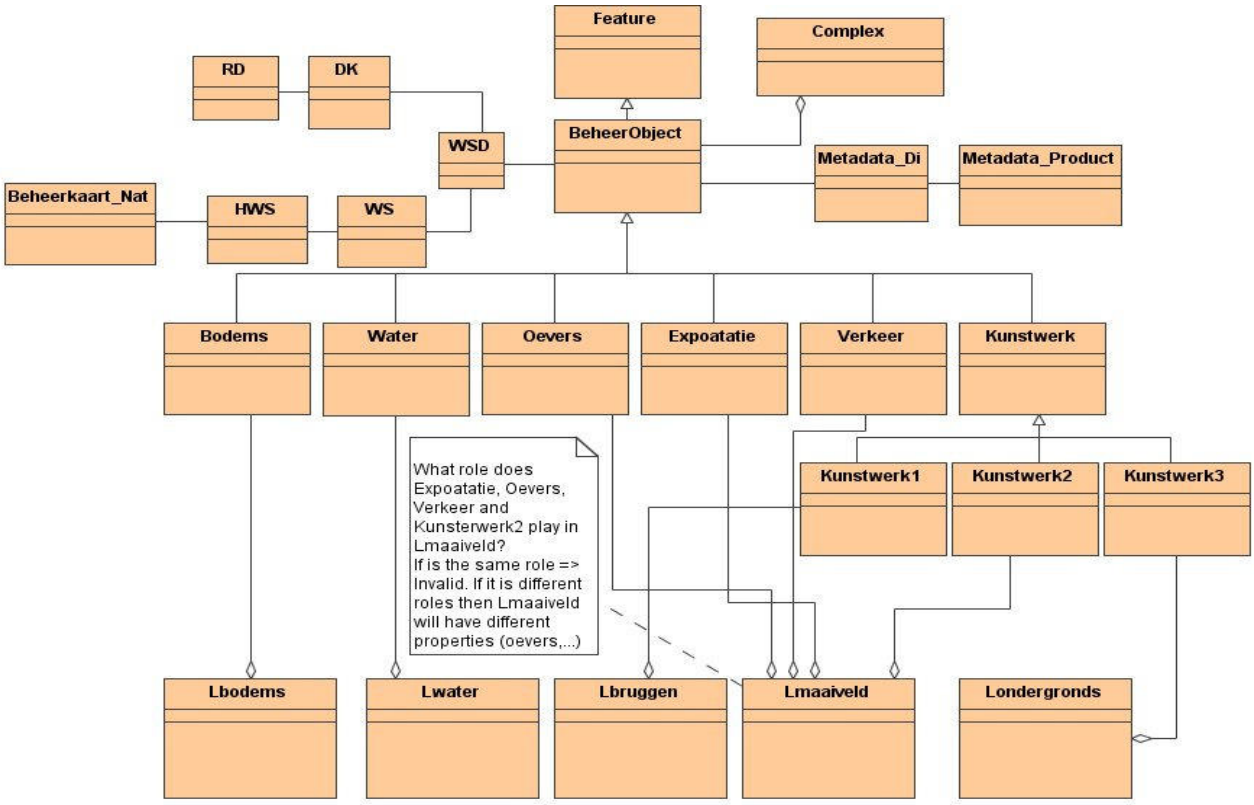


Figure 5 – Beheerkart-Nat logical model (simplified view)

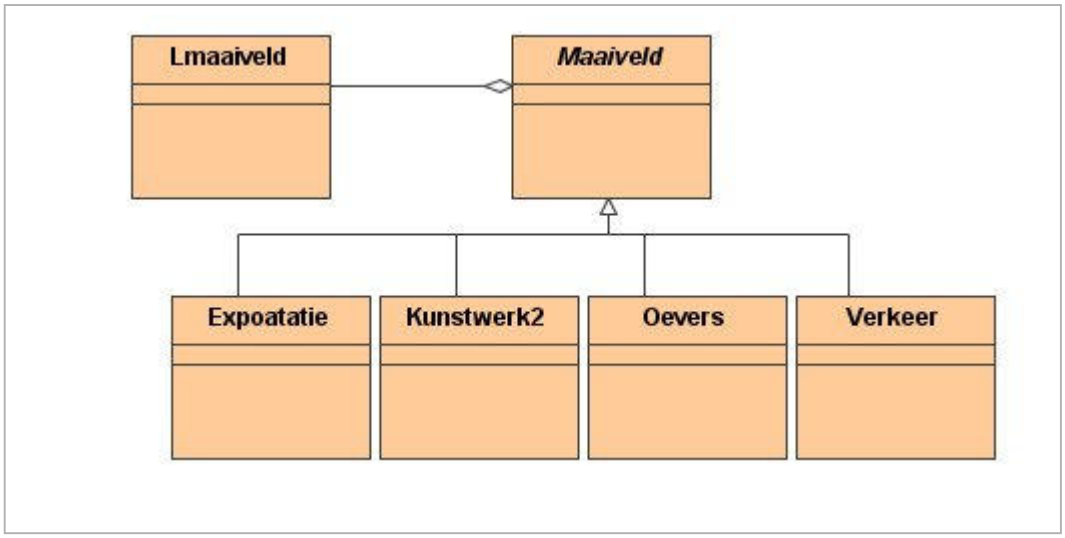


Figure 6 – Lmaaiveld aggregation

From our point of view the ideal modelization would be to have the feature types of the model represent classes of real world objects like Water, Bodems, Oevers (Figure 7)…The layer notion should be kept out of the data model and managed at a higher level. Based on OpenGIS standard it is of course possible to combine the features in different way to produce maps with the Maaiveld "layer" for instance… The constraint "all the object of a layer must not overlap" can be managed in the database (by triggers) and this will be completely transparent at a higher layer. Of course in order to implement that model, we would need to reorganize the database differently (create new views for Water, Bodems…) .

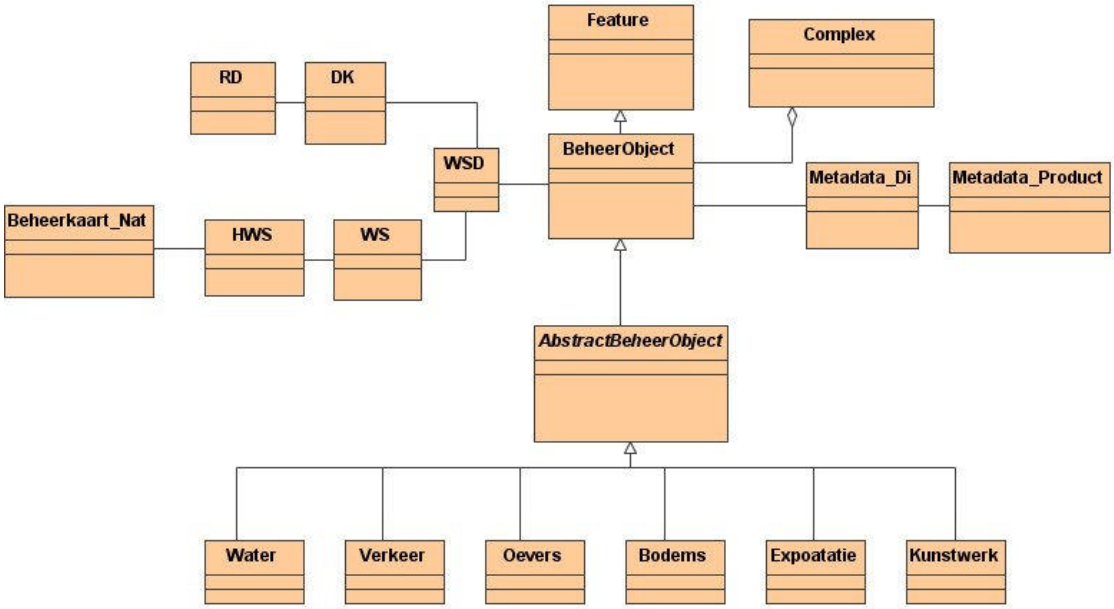


Figure 7 – Logical model proposition

The Figure 8 represents the logical model we have used to create the Beheerkaart-Nat GML application schema. As you can see the feature types are the “layers” and not the real world objects, but of course it will be very easy to switch from one to another since the structure is the same.

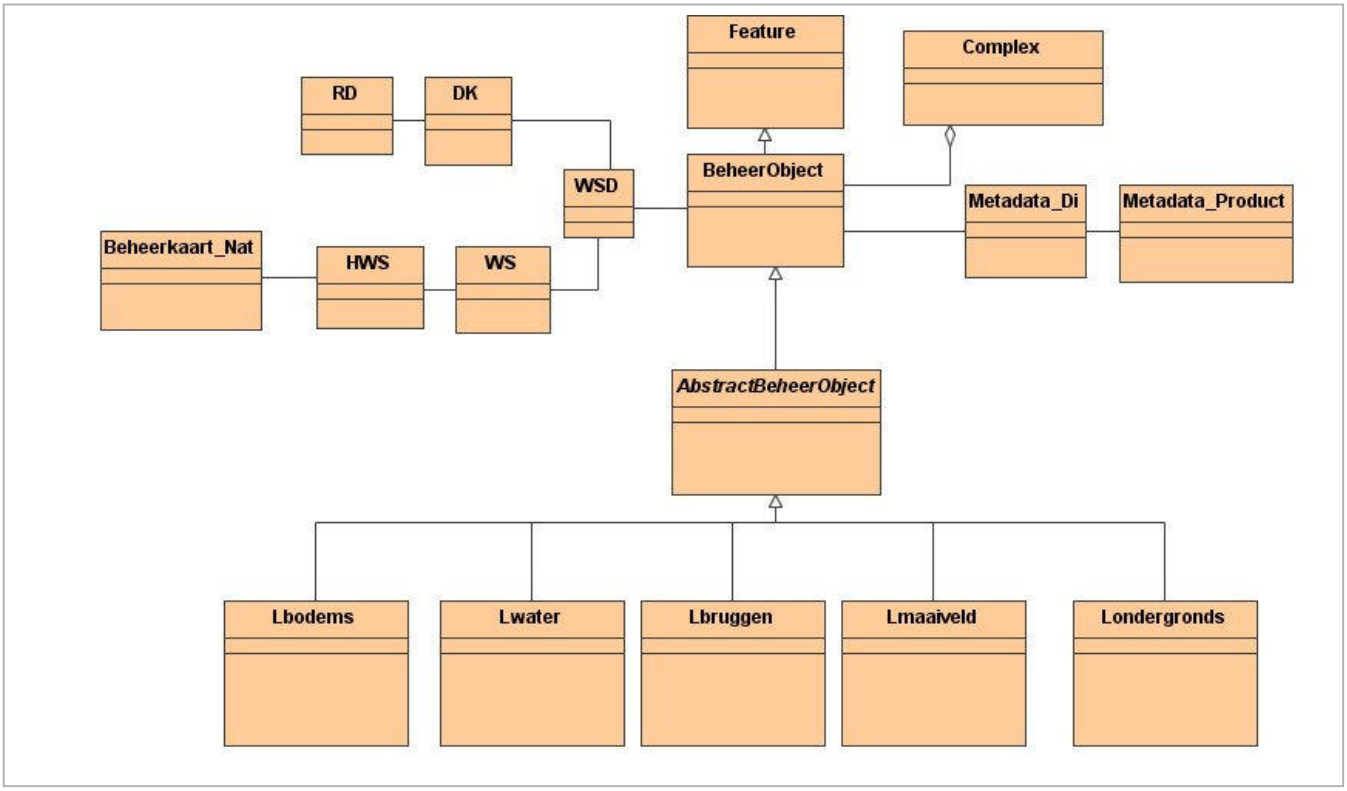


Figure 8 – Logical used to implement the GML application schema

6 The GML model

One of the primary objectives of GML is to provide a language for expressing geographic objects in a manner that is shareable over the Internet. GML provides a set of core schema components (e.g. features, geometry, topology, temporal, etc) together with a simple semantic model between objects and properties that is similar to Entity-Relationship diagrams or the class/property model of RDF (Resource Description Framework).

6.1 The Object/Property rule

An understanding of the object-property model is necessary to understand GML. The object/property model is encoded in GML by declaring a type and then assigning properties to that type. These XML property elements are called child elements of the type.

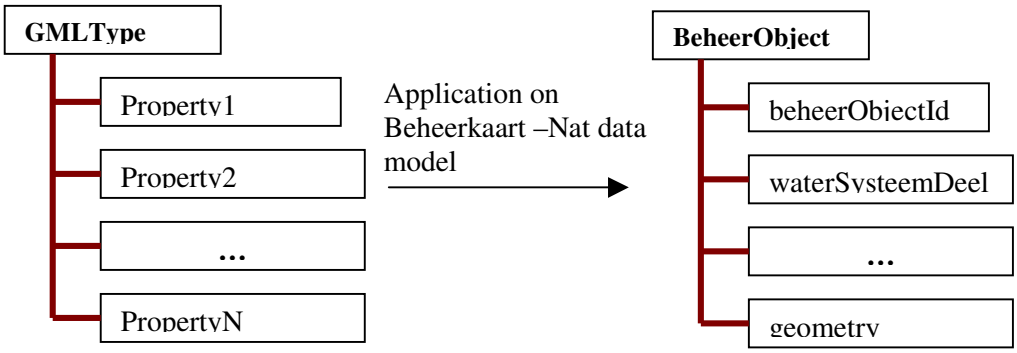


Figure 9 – A GML type is described by its property children

Note that in Figure 9 the GML Types are written in **UpperCamelCase** and the properties are in **lowerCamelCase**. This convention is used in GML to distinguish between properties and types. A Feature is not defined primarily as a geometric object, but as a meaningful object that might have some properties that are geometric and other properties that are not.

Properties can be used to describe relationships (associations) between two GML objects, where the property name should provide the name of the relationship or possibly the name of the role that the target (or source) value plays in the relationship(Figure 10).

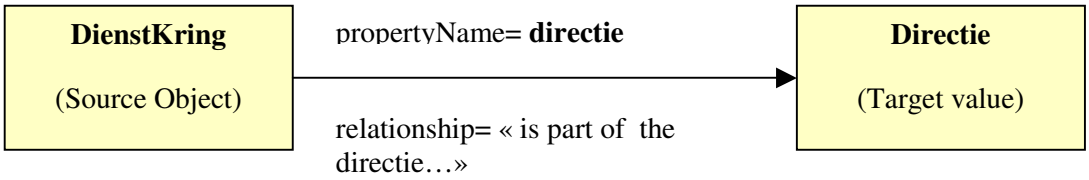


Figure 10 – A property name in GML should provide the name of the relationship or possibly the name of the role that the target (or source) value plays in the relationship

For example in the Beheerkaart-Nat data model, the relationship (association) between the **DienstKring** object and the **Directie** object is made by the property called **directie**. Below is a portion of GML – a feature collection – of **DienstKring** features:

```
<gml:featureMember>
   <agirws:dienstKring fid="DienstKring.31">
     <agirws:dienstKringId>1</agirws:dienstKringId>
        <agirws:directie fid="Directie.31">
            <agirws:directieId>1</agirws:directieId>
            <agirws:directieCode>li</agirws:directieCode>
            <agirws:directieName>Limburg  </agirws:directieName>
        </agirws:directie>
     <agirws:dienstKringCode>LIMM                                    </agirws:dienstKringCode>
     <agirws:dienstKringName>LIMM       | Dkr. Maastricht-Maas
</agirws:dienstKringName>
</agirws:dienstKring>
  </gml:featureMember>
  <gml:featureMember>
   <agirws:dienstKring fid="DienstKring.32">
     <agirws:dienstKringId>2</agirws:dienstKringId>
        <agirws:directie fid="Directie.31">
       <agirws:directieId>1</agirws:directieId>
       <agirws:directieCode>li</agirws:directieCode>
       <agirws:directieName>Limburg                 </agirws:directieName>
     </agirws:directie>
     <agirws:dienstKringCode>LINM                                    </agirws:dienstKringCode>
     <agirws:dienstKringName>LINM       | Dkr. Nijmegen-Maas
</agirws:dienstKringName>
</agirws:dienstKring>
  </gml:featureMember>
```

Remote Properties

The values of GML properties are defined either inline or remotely. In the previous **DienstKring** sample instance, for example, all of the property values are defined inline. Remote values, on the other hand, are not inline children of the property element, as shown in the following example:

```
<gml:featureMember>
   <agirws:dienstKring fid="DienstKring.31">
     <agirws:dienstKringId>1</agirws:dienstKringId>
        <agirws:directie gml:id="D1"/>
                <agirws:directieId>1</agirws:directieId>
                <agirws:directieCode>li</agirws:directieCode>
                <agirws:directieName>Limburg  </agirws:directieName>
            </agirws:directie
     <agirws:dienstKringCode>LIMM  </agirws:dienstKringCode>
     <agirws:dienstKringName>LIMM | Dkr. Maastricht-Maas  </agirws:dienstKringName>
</agirws:dienstKring>
  </gml:featureMember>
  <gml:featureMember>
   <agirws:dienstKring fid="DienstKring.32">
     <agirws:dienstKringId>2</agirws:dienstKringId>
        <agirws:directie xlink:href="#D1">
       <agirws:dienstKringCode>LINM </agirws:dienstKringCode>
     <agirws:dienstKringName>LINM | Dkr. Nijmegen-Maas  </agirws:dienstKringName>
</agirws:dienstKring>
  </gml:featureMember>
```

The `directie` property references a `Directie` object defined elsewhere in the instance document by using an xlink:href attribute. The xlink:href attribute, from the XLinks schema, refers to a GML type in the same document whose unique identifier is `gml:id=`**`"D1"`**.

In GML, remote property values can reference data instances that are located within the same instance document, as shown above, or in another document that may reside anywhere on the Internet. In the above example, the hash symbol "#" in front of **`D1`** indicates that the reference points to an instance that is located within a document. The absence of a filename preceding "#" indicates that the reference is to a location in the same document.

7 The Beheerkaart-Nat GML application schema

7.1 Introduction

The Beheerkaart-Nat GML application schema has been done with a syntax that conforms to the W3C XML Schema specification (see http://www.w3.org/XML/Schema) and respects the GML constraints. We have tried to follow the OpenGIS guidelines and best practices in developing and managing GML application schema.

7.2 The name space

A schema can be viewed as a collection (vocabulary) of type definitions and element declarations whose names belong to a particular namespace called a target namespace.
The namespace in XML schema is used to provide "context" and avoid name collisions. The use of namespaces also promotes the re-use of elements from other domains and allows a reader to visually identify the origin of each element/attribute in an instance document.
A target namespace is declared in the application schema using the targetNamespace attribute of the schema element from XML Schema, as shown below:

```
<schema targetNamespace=http://www.agi-rws.nl/xml/namespaces/gml
xmlns:agirws="http://www.agi-rws.nl/xml/namespaces/gml"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.ionicsoft.com/wfs"
elementFormDefault="qualified">
```

The value of the targetNamespace is any URI, which does not necessarily signify a physical target, meaning that there may be nothing at the end of the URI.

A namespace should be interpreted as a unique string that serves as an identifier only. The attribute **xmlns:agirws** is also anyURI valued and is used to declare the namespace of the application domain and associates the prefix "**agirws**" to the namespace.

7.3 Features

One important thing to do when designing a GML application schema is to make sure the features are directly or indirectly substitutable for "`gml:_Feature`" (rationale - participation within a WFS). All the Beheerkaart-Nat feature types are directly or indirectly substitutable for "gml:_Feature".

```
<xsd:element name="beheerObject" substitutionGroup="gml:_Feature" type="agirws:BeheerObjectType"></xsd:element>
<xsd:element name="complexObject" substitutionGroup="gml:_Feature" type="agirws:ComplexObjectType"></xsd:element>
<xsd:element name="dienstKring" substitutionGroup="gml:_Feature" type="agirws:DienstKringType"></xsd:element>
<xsd:element name="directie" substitutionGroup="gml:_Feature" type="agirws:DirectieType"></xsd:element>
<xsd:element name="hoofdWaterSysteem" substitutionGroup=" gml:_Feature" type="agirws:HoofdWaterSysteemType"></xsd:element>
```

7.4 Global element declaration

In GML features and feature collections, elements must be declared globally. This is what has been done in the Beheerkaart-Nat GML application schema. Other property elements should be declared locally unless they are reused extensively (for schema legibility). In the Beheerkaart-Nat GML application schema all the global elements are features.

```
<!-- features -->
<xsd:element name="beheerObject" substitutionGroup="gml:_Feature" type="agirws:BeheerObjectType"></xsd:element>
  <xsd:element name="complexObject" substitutionGroup="gml:_Feature" type="agirws:ComplexObjectType"></xsd:element>
  <xsd:element name="dienstKring" substitutionGroup="gml:_Feature" type="agirws:DienstKringType"></xsd:element>
  <xsd:element name="directie" substitutionGroup="gml:_Feature" type="agirws:DirectieType"></xsd:element>
  <xsd:element name="hoofdWaterSysteem" substitutionGroup="gml:_Feature" type="agirws:HoofdWaterSysteemType"></xsd:element>
  <xsd:element name="waterSysteem" substitutionGroup="gml:_Feature" type="agirws:WaterSysteemType"></xsd:element>
  <xsd:element name="waterSysteemDeel" substitutionGroup="gml:_Feature" type="agirws:WaterSysteemDeelType"></xsd:element>
  <xsd:element name="metadataDinsKring" substitutionGroup="gml:_Feature" type="agirws:MetadataDinsKringType"></xsd:element>
  <xsd:element name="lBodems" substitutionGroup="gml:_Feature" type="agirws:LBodemsType"></xsd:element>
  <xsd:element name="lBruggen" substitutionGroup="gml:_Feature" type="agirws:LBruggenType"></xsd:element>
  <xsd:element name="lmaaiveld" substitutionGroup="gml:_Feature" type="agirws:LMaaiveldType"></xsd:element>
  <xsd:element name="lWater" substitutionGroup="gml:_Feature" type="agirws:LWaterType"></xsd:element>
  <xsd:element name="lOnderGronds" substitutionGroup="gml:_Feature" type="agirws:LOnderGrondsType"></xsd:element>
```

7.5 Feature types

In GML, geographic features are represented exclusively by global XML element declarations in the GML application schema. Features cannot be represented as attributes. Note that as a consequence of this rule, GML data instances contain elements whose names specify a feature type. For example the global element declaration of beheerObject is:

```
<xsd:element name="beheerObject" substitutionGroup=" gml:_Feature" type="agirws:BeheerObjectType"></xsd:element>
```

The content models of GML feature types must derive, directly or indirectly from `gml:AbstractFeatureType`. It is recommended that the feature element should be also made substitutable for the GML abstract feature element `gml:_Feature`. This permits the GML feature to be substituted anywhere that a `gml:_Feature` can be used, such as in a `FeatureCollections` and as values of GML properties (i.e. as a participant in a feature relationship).

```
<!--features -->

<xsd:element name="beheerObject" substitutionGroup="gml:_Feature" type="agirws:BeheerObjectType"/>
<xsd:element name="complexObject" substitutionGroup="gml:_Feature" type="agirws:ComplexObjectType"/>
<xsd:element name="dienstKring" substitutionGroup="gml:_Feature" type="agirws:DienstKringType"/>
<xsd:element name="directie" substitutionGroup="gml:_Feature" type="agirws:DirectieType"/>
<xsd:element name="hoofdWaterSysteem" substitutionGroup="gml:_Feature" type="agirws:HoofdWaterSysteemType"/>
<xsd:element name="waterSysteem" substitutionGroup="gml:_Feature" type="agirws:WaterSysteemType"/>
<xsd:element name="waterSysteemDeel" substitutionGroup="gml:_Feature" type="agirws:WaterSysteemDeelType"/>
<xsd:element name="metadataDinsKring" substitutionGroup="gml:_Feature" type="agirws:MetadataDinsKringType"/>
<xsd:element name="lBodems" substitutionGroup="agirws:_bodems" type="agirws:LBodemsType"/>
<xsd:element name="lBruggen" substitutionGroup="agirws:_kunstwerk1" type="agirws:LBruggenType"/>
<xsd:element name="lmaaiveld" substitutionGroup="agirws:_lmaaiveldSubGroup" type="agirws:LMaaiveldType"/>
<xsd:element name="lWater" substitutionGroup="agirws:_water" type="agirws:LWaterType"/>
<xsd:element name="lOnderGronds" substitutionGroup="agirws:_kunstwerk3" type="agirws:LOnderGrondsType"/>
```

We have also created a `agirws:AbstractFeatureType` abstract feature type from which all the Beheerkaart-Nat feature types derive from. The `agirws:AbstractFeatureType` extends `gml:AbstractFeatureType` to ensure compliance with GML:

```
<xsd:group name="FeatureGroup">
        <xsd:annotation>
                <documentation xml:lang="en">
                        This defines the properties common to all AGI-RSW features.
                        As the schema will evolve all the new common elements will be added on that group
                </documentation>
        </xsd:annotation>
        <xsd:sequence>
                <xsd:element name="version" type="xsd:unsignedInt"/>
                <xsd:element name="versionDate" type="xsd:date"/>
        </xsd:sequence>
 </xsd:group>
<xsd:complexType name="AbstractFeatureType">
        <xsd:annotation>
                <xsd:documentation xml:lang="en">
                        Abstract base type for all  AGI-RSW features.  Extends the gml abstract
                        feature in order to ensure compliance with GML.
                </xsd:documentation>
        </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">

        <group ref="agirws:FeatureGroup"/>
      </xsd:extension>
    </xsd:complexContent>
   </xsd:complexType>
```

All new properties common to all Beheerkaart-Nat feature types should be added to the group named “FeatureGroup” and they will be automatically available since all the feature types extend `agirws: AbstractFeatureType`.

See below the declaration of BeheerObjectType extending the `agirws:AbstractFeatureType`:

```
<xsd:complexType name="BeheerObjectType">
        <annotation>
                <documentation xml:lang="en">
                        Main class
                </documentation>
        </annotation>
  <xsd:complexContent>
    <xsd:extension base="agirws:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="beheerObjectId" minOccurs="0" type="xsd:int"/>
        <xsd:element name="waterSysteemDeel" minOccurs="0" nillable="true" type="agirws:WaterSysteemDeelType"/>
        <xsd:element name="categorieType"  minOccurs="0" nillable="true" type="agirws:CategorieType"/>
         <xsd:element name="zijdeType"  minOccurs="0" nillable="true" type="agirws:ZijdeType"/>
        <xsd:element name="codeKm" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="nameContactPerson" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="geometry" minOccurs="0" nillable="true" type="gml:GeometryAssociationType"/>
        <xsd:element name="vis1Geometry" minOccurs="0" nillable="true" type="gml:GeometryAssociationType"/>
        <xsd:element name="vis2Geometry" minOccurs="0" nillable="true" type="gml:GeometryAssociationType"/>
        <xsd:element name="metadataDinsKring" minOccurs="0" nillable="true" type="agirws:MetadataDinsKringType"/>
        <xsd:element name="complexObject"  minOccurs="0" nillable="true" type="agirws:ComplexObjectType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Important note: in the Oracle database all the column of all the tables are nillable except the primary keys that's why we have nillable="true" for all feature type properties. AGI needs to determine which fields are nillable and which are not and update the schema in consequence.

7.6 Spatial type

As with features, spatial types (`Point`, `Node`, etc) are represented in GML exclusively by global XML element declarations in the GML Application Schema. It is possible to define new spatial or temporal types, for instance geometry types, by deriving them from base geometry types or by deriving them from `gml:AbstractGeometryType`. It is recommended to avoid deriving anything from the top-level `gml:AbstractGeometryType`, since this provides very limited information to GML aware software. It is recommended to always try to choose the most specific geometry type possible in constructing the derived geometry types. For example, if your geometry is some type of curve (in GML 3.0), then derive from `gml:AbstractCurveType` or, if possible, one of its concrete subclasses, such as `gml:LineString`.

This recommendation is broken in the Beheerkaart-Nat datamodel, as the geometry property of the `agirws:BeheerObject` can be of any type. We are obliged to use the generic `gml:GeometryAssociationType` geometry type which makes things more complicated for GML aware applications (including a WMS on top the WFS). A big question could be raised

here: isn't it possible to redesign the model in a way such that all the feature types have more specific geometry type (Polygon, Line, Point)?

We would then have something that would look like what we have in Figure 11:

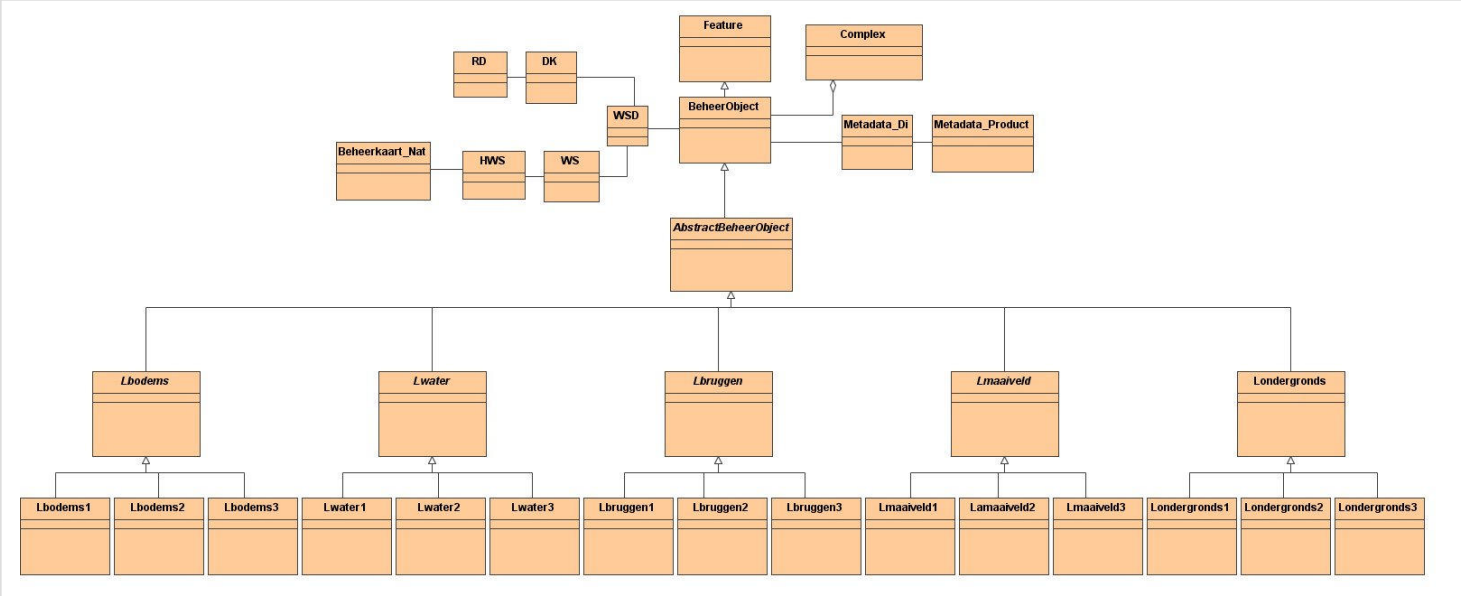


Figure 11 – Alternative model

With this approach it is still possible to have the expected result as specified in the ***Table 2: Visualisation requirements (P. v. Asperen)*** of the "Design of a logical and physical model for the beheerkaart-Nat" document by putting the logic at the WMS level (in the rendering rules). See below. And then use an OpenGIS context to specify which layer is visible at which scale.

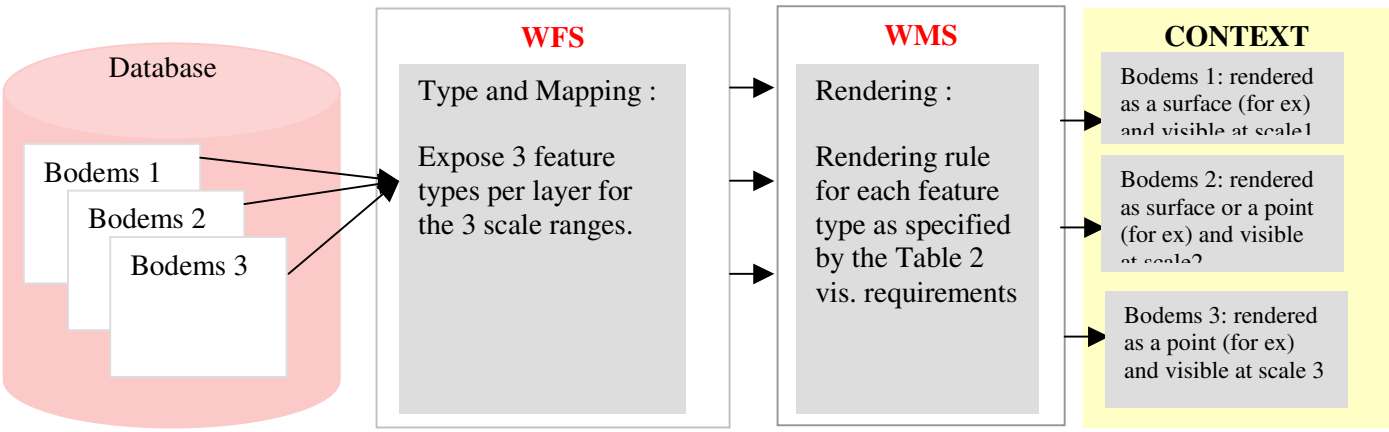


Figure 12 – Alternative view of the beheerkaart-Nat model

The OpenGIS Map Context would look to something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<ViewContext version="1.0.0" id="none" xmlns="http://www.opengis.net/context"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ogc="http://www.opengis.net/ogc" xsi:schemaLocation="http://www.opengis.net/context context.xsd">
   <General>
…
<LayerList>
<Layer queryable="false" hidden="0" minScale="250000" >
          <Server service="OGC:WMS" title="Bodems">
            <OnlineResource xlink:type="simple" xlink:href="http://localhost:8081/agibeheer/wfs/AGIBEH"/>
          </Server>
          <Name>Bodems1 </Name>
          <Title>Bodems Layer <1:250000</Title>
          ...
</Layer>
<Layer queryable="false" hidden="0" minScale="25000" maxScale="249999">
          <Server service="OGC:WMS" title="Bodems">
            <OnlineResource xlink:type="simple" xlink:href="http://localhost:8081/agibeheer/wfs/AGIBEH"/>
          </Server>
          <Name>Bodems2</Name>
          <Title>Bodems Layer 1:25000 - 1:250000</Title>
          ...
</Layer>
<Layer queryable="false" hidden="0" minScale="0.0" maxScale="24999.0">
          <Server service="OGC:WMS" title="Bodems">
            <OnlineResource xlink:type="simple" xlink:href="http://localhost:8081/agibeheer/wfs/AGIBEH"/>
          </Server>
          <Name>Bodems3</Name>
          <Title>Bodems Layer >1:25000 </Title>
          ...
</Layer>

</LayerList>
</ViewContext>
```

7.7 Metadata

Different categories of metadata can be distinguished:

- General - describes a data resource as a whole
- Lifecycle - pertains to the history and current state of the resource
- Meta-metadata - info about the metadata instance itself
- Technical - domain-specific info pertaining to data usage and quality.
- Rights – access rights or use restrictions
- Relation - links to related resources (e.g. services that provide the data)
- Annotation
- Classification

In the beheerkaart-Nat model, the metadata is more “General” or “Lifecycle” metadata. For General metadata a feature type called `agirws:metadataDinsKring` has been created to expose the metadata and the “Lifecycle”or the history management has been left to the RDBMS (Oracle in this case).

In GML3, it is possible to derive a metadata complex type from `gml:AbstractMetaDataType`. It is then recommended that the metadata element should be made substitutable for the GML abstract metadata element `gml:_MetaData`. This permits the metadata to be substituted as the value of the built-in metadata property on all GML types that derive by extension from AbstractGMLType.

The definition of `agirws:metadataDinsKring` in GML3 could be then:

```
<xsd:element name="_metadataFeature" type="gml:AbstractMetaDataType" abstract="true"
substitutionGroup="gml:_MetaData"/>
<xsd:element name="metadataDinsKring" substitutionGroup="agirws:_metadataFeature"
type="agirws:MetadataDinsKring"/>

<xsd:complexType name="MetadataDinsKringType" mixed="true">
  <xsd:complexContent mixed="true">
    <xsd:extension base="gml:AbstractMetaDataType">
      <xsd:sequence>
        <xsd:element name="metadataDinsKringId" type="xsd:int"/>
        <xsd:element name="datasetNaam" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="samenvatting" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="datasetProducent" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="ruimtelijkSchema" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="datasetTekenset" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="datasetHerkomst" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="statusVanDekking" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="actualiteit" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="datatype" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="toepassingDatatype" minOccurs="0" nillable="true" type="xsd:string"/>
        <xsd:element name="directie" minOccurs="0" nillable="true" type="agirws:DirectieType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

GML 3 provides a simple mechanism (`gml:MetaDataPropertyType`) for including feature or feature collection metadata either by reference (using XLink attributes) or by value (in-line). Higher-level metadata are typically stored elsewhere and accessed through catalogue services; GML metadata properties may reference such catalogue entries (using the XLink attributes contained in `gml:AssociationAttributeGroup`).

7.8 Association/properties

It is recommended for each association to generate an XML element with a name that represents the association relationship/role name, following the Association Type Pattern (or derive the complex type of this XML element from `gml:AssociationType`). GML property names should be used to name roles, the property name does not need to indicate the content of the value.

In the beheerkaart-Nat model most of the association could be named “is part of …”. We could have properties like “isPartOfTheDirectie”, “isPartOfTheWaterSysteemDeel”… We have decided for simplification to keep the property name to the name of the target value like “directie”, “waterSysteemDeel”...

One important thing we need to mentione here, is that in GML 3 the value of a property can be either a feature or an object, NOT a collection of features: a property never has more than one value. In the beheerkaart-Nat data model (implemented in GML2), there is only one aggregation: the relation between the beheerObject and complex. Below is how that relation has been expressed in the GML application schema:

```
<xsd:complexType name="ComplexObjectType">
    <xsd:complexContent>
      <xsd:extension base="agirws:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="complexObjectId" minOccurs="0" nillable="true" type="xsd:int"/>
          <xsd:element name="complexObjectName" minOccurs="0" nillable="true" type="xsd:string"/>
          <xsd:element name="beheerObjectCollection" minOccurs="0" nillable="true"
type="agirws:BeheerObjectCollectionType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="BeheerObjectCollectionType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureCollectionType">
        <xsd:sequence>
          <xsd:element name="beheerObject" maxOccurs="unbounded" nillable="true"
type="agirws:BeheerObjectType"/>
        </xsd:sequence>
      </xsd:extension>
```

The ComplexObject has a property of type `agirws:BeheerObjectCollectionType` extending the gml:AbstractFeatureCollectionType.

8 WFS schema and mapping configuration

In order to check the GML application schema we have generated, we have set up a WFS using that schema on an Oracle database. The data used has been provided as Oracle dump file. This section will explain the different steps we went through to map the feature types defined in the GML application schema onto the database (Oracle) objects (tables or views). Before that we are going to define a mapping.

8.1 What is a mapping

The mapping is the configuration that describes the link between the FeatureType definition and the objects returned by the underlying engine. The mapping document associated with a WFS service presents the necessary information for the WFS to convert client requests to queries understood by the data server. It also converts the result into a compliant collection of features. Therefore, it makes the link between the internal data structure and the published information.

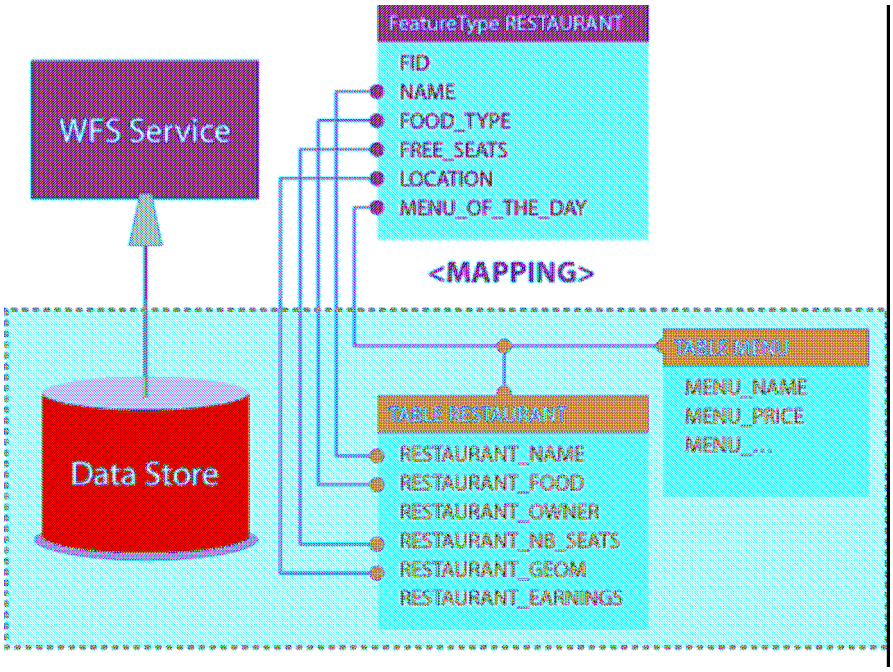


Figure 13 – WFS Mapping

The way the mapping is done is specific to a vendor. IONIC writes the mapping in an XML document that can be built either using one of the command-line tools coming along with RedSpider Web, or manually. In some cases, the mapping is implicit and does not need any document.

8.2 Beheerkaart-Nat GML application schema mapping to Oracle

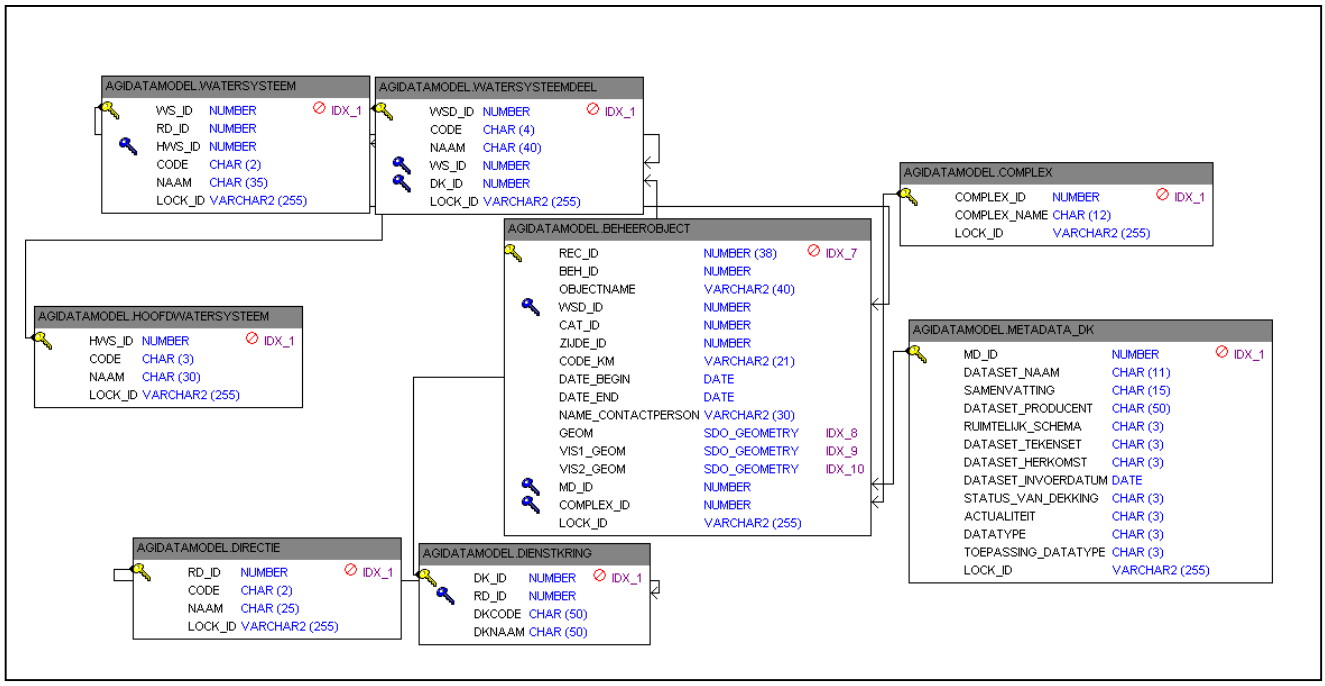


Figure 14 – The Beheerkaart database Entity Relation diagram (without the views)

The Oracle database is composed of 8 tables and 15 views that will be used to map the 13 features types of the beheerkaart-Nat GML application schema.

Those features types can be put in two groups:

1. The 8 core feature types: basic features types of the model (see the Figure 14 above).

For those features types, a one-to-one mapping with the Oracle tables has been done.

- `agirws:BeheerObjectType -> BEHEEROBJECT`
- `agirws:ComplexObjectType -> COMPLEX`
- `agirws:DienstKringType - > DIENSTKRING`
- `agirws:DirectieType -> DIRECTIE`
- `agirws:HoofdWaterSysteemType - > HOOFDWATERSYSTEEM`
- `agirws:WaterSysteemType -> WATERSYSTEEM`
- `agirws:WaterSysteemDeelType -> WATERSYSTEEMDEEL`
- `agirws:MetadataDinsKringType -> METADATA_DK`

Very simple rules are applied in the one-to-one mapping:
- A simple type element is mapped onto a column of a table.
- A complex type element is mapped onto a column of a table. The WFS use that column to retrieve the necessary information from the table associated to the complex type.
- An attribute is mapped onto a column of a table.
- A collection type element is mapped using a foreign key tag (see ComplexObjectType).

See below the sample one-to-one mapping of the BeheerObjectType to the Oracle Spatial table BEHEEROBJECT:

```
<?xml version='1.0' encoding='utf-8' ?>
 <xsd:schema xmlns:gml="http://www.opengis.net/gml"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 targetNamespace="http://www.agi-rws.nl/xml/namespaces/gml"
 xmlns:agirws="http://www.agi-rws.nl/xml/namespaces/gml"
 elementFormDefault="qualified">

  <Mapping>

  <!-- Mapping for type agirws:BEHEEROBJECT-->
   <SQL name="agirws:BeheerObjectType" generation="autogen">
    <Table nameSQL="BEHEEROBJECT"/>
   <NoPrimary/>
    <Lock nameSQL="LOCK_ID" />
    <Element name="agirws:beheerObjectId" nameSQL="BEH_ID"/>
    <Element name="gml:name" nameSQL="OBJECTNAME"/>
    <Element name="agirws:waterSysteemDeel" nameSQL="WSD_ID" >
        <Type name="agirws:WaterSysteemDeelType"/>
    </Element>
    <Element name="agirws:categorieType" nameSQL="CAT_ID"/>
    <Element name="agirws:zijdeType" nameSQL="ZIJDE_ID"/>
    <Element name="agirws:codeKm" nameSQL="CODE_KM"/>
    <Element name="agirws:nameContactPerson" nameSQL="NAME_CONTACTPERSON"/>
    <Element name="agirws:geometry" nameSQL="GEOM"/>
    <Element name="agirws:vis1Geometry" nameSQL="VIS1_GEOM"/>
    <Element name="agirws:vis2Geometry" nameSQL="VIS2_GEOM"/>
    <Element name="agirws:metadataDinsKring" nameSQL="MD_ID" >
        <Type name="agirws:MetadataDinsKringType"/>
    </Element>

   </SQL>
```

MAPPING THE ENUMERATION:

The enumeration lookup tables have been mapped using the “RelationalEnumeration” tag, see below:

```
    <RelationalEnumeration
        name="agirws:ZijdeType"
        nameSQL="ZIJDETYPE"
        pkey="ZIJDE_ID"
        column="ZIJDECODE"
        />

    <RelationalEnumeration
        name="agirws:CategorieType"
        nameSQL="CATEGORIETYPE"
        pkey="CAT_ID"
        column="CATCODE"
        />
```

MAPPING THE AGGREGATION

The “Relation” tag defines a relationship between two types; we have used it to map the aggregation relationship between agirws:ComplexObjectType and agirws:BeheerObjectType.

```
<Relation
        sourceType="agirws:ComplexObjectType"
        targetType="agirws:BeheerObjectType"
        source="agirws:complexObjectId"
        targetSQL="COMPLEX_ID"
        ownership="true"
        />
```

2. The 5 “Layers” feature types: read-only feature types that give a view per category of the core feature type beheerObject.

Those feature types represent the 5 “layers” as described by the beheerkaart-Nat data model. Each of those feature types has been mapped to 3 different views depending on the scale. The different views do have just one geometry field being “GEOM” (for XXX _VIS0 views), “GEOM_VIS1”(for XXX_VIS1 views) or “GEOM_VIS2”(for XXX_VIS2 views).

- `agirws:LbodemsType-> LBODEMS_VIS0, LBODEMS_VIS1 or LBODEMS_VIS2 depending on the scale`
- `agirws:LBruggenType-> LBRUGGEN_VIS0, LBRUGGEN_VIS1 or LBRUGGEN_VIS2 depending on the scale`
- `agirws:LMaaiveldType-> LMAAIVELD_VIS0, LMAAIVELD_VIS1 or LMAAIVELD_VIS2 depending on the scale`
- `agirws:LWaterType-> LONDERGRONDS_VIS0, LONDERGRONDS_VIS1 or LONDERGRONDS_VIS2 depending on the scale`
- `agirws:LOnderGrondsType-> LWATER_VIS0, LWATER_VIS1 or LWATER_VIS2 depending on the scale`

See below the sample one-to-one mapping of the LBodemsType feature type to the Oracle Spatial table LBODEMS_VIS0, LBODEMS_VIS1 or LBODEMS_VIS2 depending on the scale:

```
<!-- Mapping for type agirws:lBodems-->
  <SQL name="agirws:LBodemsType" generation="autogen">
   <Table nameSQL="LBODEMS_VIS0" >
        <TableScale scaleMin="0.0" scaleMax="24999.0" nameSQL="LBODEMS_VIS0" />
        <TableScale scaleMin="25000.0" scaleMax="249999.0" nameSQL="LBODEMS_VIS1" />
        <TableScale scaleMin="250000.0"  nameSQL="LBODEMS_VIS2" />
   </Table>
  <!-- Caution no primary defined ... -->
   <NoPrimary/>
   <Element name="agirws:beheerObjectId" nameSQL="BEH_ID"/>
   <Element name="gml:name" nameSQL="OBJECTNAME"/>
   <Element name="agirws:codeKm" nameSQL="CODE_KM"/>
   <Element name="agirws:nameContactPerson" nameSQL="NAME_CONTACTPERSON"/>
   <Element name="agirws:geometry" nameSQL="GEOM"/>
   <Element name="agirws:categorieCode" nameSQL="CATCODE"/>
   <Element name="agirws:zijdeCode" nameSQL="ZIJDECODE"/>
   <Element name="agirws:metadataDinsKring" nameSQL="MD_ID">
       <Type name="agirws:MetadataDinsKringType"/>
   </Element>
   <Element name="agirws:complexObjectName" nameSQL="COMPLEX_NAME"/>
   <Element name="agirws:waterSysteemCode" nameSQL="WS_CODE"/>
   <Element name="agirws:waterSysteemName" nameSQL="WS_NAAM"/>
   <Element name="agirws:hoofdWaterSysteemCode" nameSQL="HWS_CODE"/>
   <Element name="agirws:hoofdWaterSysteemName" nameSQL="HWS_NAAM"/>
   <Element name="agirws:dienstKringCode" nameSQL="DK_CODE"/>
   <Element name="agirws:dienstKringName" nameSQL="DK_NAAM"/>
   <Element name="agirws:directieCode" nameSQL="RD_CODE"/>
   <Element name="agirws:directieName" nameSQL="RD_NAAM"/>
  </SQL>
```

This type of mapping allows selecting a SQL table (or view) according to the current standard scale. This has been used to provide different definitions of geometry according to the scale (and so the requested precision). To do that, of course all the tables used need to have at least the same columns as the master table. The master table is defined using the "Table" tag and each scale range is defined using a "TableScale" tag. This feature is only available for a GetFeature operation, for all others operations (transactions:insert,delete,update) the master table is used and so it sounds reasonable to use it only for read-only features.

8.3 Configuring a transactional WFS

A transactional WFS (WFS-T) is used when insertion, update and deletion actions against features are needed. This is the case for the Beheerkaart-Nat model. In order to set up a transactional WFS, we had to ensure that:

- We have identified the features for which transaction will be allowed
- We have a unique identifier on all the feature types where transaction will be made
- We have a column for the locking flags
- We have a table to hold locking durations
- We have the schema and mapping files updated to support transactions

8.3.1 Transactional features

In this analysis we have made the choice to allow transactions only on the core feature types (see section 8.2). The "Layer" features types will be read-only (only query operation will be allowed).

8.3.2 Unique identifier

The 8 core Beheerkaart-Nat feature types do have a unique identifier defined as primary key. That primary key will be generated by the database on insert operations this mentioned by the attribut **fid="auto"**. It is also possible to have the WFS generate the primary key but this was not appropriate in the Beheerkaart-Nat model.

```
<Primary nameSQL="BEH_ID" type="int" fid="auto" />
```

8.3.3 Locking mechanism

The locking mechanism uses an extra field on each feature table and one extra global table used to manager lock timeout. The extra field will be a varchar(255) and will contain the lock status.
The extra global table called LOCKTIMEOUT will have the following structure

- PKEY : the primary key RAW(16) automatically generated
- LOCKID: the lockid associated to the specified feature
- NAME: the feature name
- FIELD: the lock field name
- EXPIRE: the time when the lock expires

We have added the lock column on all Beheerkart-Nat core tables using the following sql script:

```
ALTER TABLE AGIDATAMODEL.BEHEEROBJECT       ADD (LOCK_ID VARCHAR2(255 BYTE));
ALTER TABLE AGIDATAMODEL.COMPLEX            ADD (LOCK_ID VARCHAR2(255 BYTE));
ALTER TABLE AGIDATAMODEL.DIENSTKRING        ADD (LOCK_ID VARCHAR2(255 BYTE));
ALTER TABLE AGIDATAMODEL.DIRECTIE           ADD (LOCK_ID VARCHAR2(255 BYTE));
ALTER TABLE AGIDATAMODEL.HOOFDWATERSYSTEEM ADD (LOCK_ID VARCHAR2(255 BYTE));
ALTER TABLE AGIDATAMODEL.METADATA_DK        ADD (LOCK_ID VARCHAR2(255 BYTE));
ALTER TABLE AGIDATAMODEL.WATERSYSTEEM       ADD (LOCK_ID VARCHAR2(255 BYTE));
```

The "LOCKTIMEOUT" table has been created using the following SQL script.

```
CREATE TABLE
(
PKEY RAW(16) PRIMARY KEY
        ,LOCKID VARCHAR2(255)
        ,NAME VARCHAR2(255)
        ,FTLOCK VARCHAR2(255)
        ,EXPIRE DATE
);

 CREATE INDEX LOCKTIMEOUT_IDX1 ON LOCKTIMEOUT (EXPIRE);
 CREATE INDEX LOCKTIMEOUT_IDX2 ON LOCKTIMEOUT (LOCKID);
 CREATE INDEX LOCKTIMEOUT_IDX3 ON LOCKTIMEOUT (NAME);
```

8.3.4 Update the schema and mapping files

The types of operations allowed on a feature type are defined in the mapping file in the “Info” tag. The following sample shows for instance that only “query” operations are allowed on the agirws:LWaterType feature type and all operations “*”

```
<Info name="agirws:LWaterType">
    <Operations>Query</Operations>
    <SRS>EPSG:28992</SRS>
    <BoundingBox SRS="EPSG:28992" minx="55000" miny="425000" maxx="110000"maxy="450000"/>
  </Info>

<!--Info for type agirws:BeheerObject-->
  <Info name="agirws:BeheerObjectType">
    <Operations>*</Operations>
    <SRS>EPSG:28992</SRS>
    <BoundingBox SRS="EPSG:28992" minx="55000" miny="425000" maxx="110000" maxy="450000"/>
  </Info>
```

9 Sample requests on the beheerkaart-Nat WFS

We make the assumption that the WFS is deployed and responds to the URL **http://localhost:8081/agibeheer/wfs/AGIBEH**

9.1 GetCapabilities

http://localhost:8081/agibeheer/wfs/AGIBEH?request=GetCapabilities&service=WFS&version=1.0


```
<?xml version="1.0" encoding="UTF8" ?>
<WFS_Capabilities version="1.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
      xsi:schemaLocation="http://www.opengis.net/wfs http://
      localhost:8081/agibeheer/wfs/AGIBEH/REQUEST/get/DATA/LPR/wfs/capabilities.xs
      d" xmlns="http://www.opengis.net/wfs"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:ogcwfs="http://www.opengis.net/wfs"
      xmlns:ionic="http://www.ionicsoft.com/versions/2">
   <Service>
        <Name>WFS Feature Server</Name>
        <Title>Ionic WFS server over AGI Beheerkaart Database</Title>
        <Abstract>WFS Server maintained by Ionic Software
            (http://www.ionicsoft.com).</Abstract>
       <OnlineResource>http://localhost:8081/agibeheer/wfs/AGIBEH?</OnlineResource>
     </Service>
   <Capability>
      <Request>
         <GetCapabilities>
            <DCPType>
               <HTTP>
                   <GetonlineResource="http://localhost:8081/agibeheer/wfs/AGIBEH?" />
                   <PostonlineResource="http://localhost:8081/agibeheer/wfs/AGIBEH" />
                 </HTTP>
              </DCPType>
            </GetCapabilities>
        [...]
   <FeatureTypeList>
      <Operations>
            <Query />
            <Insert />
            <Delete />
            <Update />
            <Lock />
        </Operations>
      <FeatureType xmlns:agirws="http://www.agirws.nl/xml/namespaces/gml">
            <Name>agirws:beheerObject</Name>
            <SRS>EPSG:28992</SRS>
            <LatLongBoundingBox minx="3.9306958630298654" miny="51.80467309290902"
               maxx="4.735338050789477" maxy="52.03649047030586" />
     </FeatureType>
        [...]

  </WFS_Capabilities>
```

9.2 DescribeFeatureType

http://localhost:8081/agibeheer/wfs/AGIBEH?request=DescribeFeatureType&service=WFS&version=1.0


```
<?xml version="1.0" encoding="utf8" ?>
 <xsd:schema xmlns:gml="http://www.opengis.net/gml"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.agirws.nl/xml/namespaces/gml"
      xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.ionicsoft.com/wfs"
      xmlns:agirws="http://www.agirws.nl/xml/namespaces/gml" elementFormDefault="qualifiec
     <xsd:element name="beheerObject" substitutionGroup="gml:_Feature"
        type="agirws:BeheerObjectType" />
    <xsd:complexType name="WaterSysteemDeelType">
       <xsd:complexContent>
          <xsd:extension base="agirws:AbstractFeatureType">
             <xsd:sequence>
                 <xsd:element name="waterSysteemDeelId" type="xsd:int" />
                 <xsd:element name="waterSysteemDeelCode" minOccurs="0" nillable="true"
                    type="xsd:string" />
                 <xsd:element name="waterSysteemDeelName" minOccurs="0" nillable="true"
                    type="xsd:string" />
                 <xsd:element name="waterSysteem" minOccurs="0" nillable="true"
                    type="agirws:WaterSysteemType" />
                 <xsd:element name="directie" minOccurs="0" nillable="true"
                    type="agirws:DirectieType" />
              </xsd:sequence>
           </xsd:extension>
        </xsd:complexContent>
     </xsd:complexType>
    <xsd:complexType name="HoofdWaterSysteemType">
       <xsd:complexContent>
          <xsd:extension base="agirws:AbstractFeatureType">
             <xsd:sequence>
                 <xsd:element name="hoofdWaterSysteemId" type="xsd:int" />
                 <xsd:element name="hoofdWaterSysteemCode" minOccurs="0" nillable="true"
                    type="xsd:string" />
                 <xsd:element name="hoofdWaterSysteemName" minOccurs="0" nillable="true"
                    type="xsd:string" />
              </xsd:sequence>
           </xsd:extension>
        </xsd:complexContent>
     </xsd:complexType>
    <xsd:complexType name="DirectieType">
       <xsd:complexContent>
          <xsd:extension base="agirws:AbstractFeatureType">
             <xsd:sequence>
                 <xsd:element name="directieId" type="xsd:int" />
                 <xsd:element name="directieCode" minOccurs="0" nillable="true" type="xsd:string" /
                 <xsd:element name="directieName" minOccurs="0" nillable="true" type="xsd:string"
              </xsd:sequence>
           </xsd:extension>
        </xsd:complexContent>
     </xsd:complexType>
                    [...]
  </xsd:schema>
```

9.3 GetFeature

http://localhost:8081/agibeheer/wfs/AGIBEH?request=GetFeature&typename=DienstKringType&maxfeatures=5


```
<ogcwfs:FeatureCollection
    xsi:schemaLocation="http://www.agirws.nl/xml/namespaces/gml
    http://localhost:8081/agibeheer/wfs/AGIBEH?REQUEST=DescribeAllFeatureTy
    pe&SERVICE=WFS http://www.opengis.net/wfs
    http://localhost:8081/agibeheer/wfs/AGIBEH/REQUEST/get/DATA/LPR/wfs/
    basic.xsd" xmlns:gml="http://www.opengis.net/gml"
    xmlns:agirws="http://www.agirws.nl/xml/namespaces/gml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:ogcwfs="http://www.opengis.net/wfs">
  <gml:boundedBy>
      <gml:null>inapplicable</gml:null>
   </gml:boundedBy>
  <gml:featureMember>
     <agirws:dienstKring fid="DienstKringType.1">
         <agirws:dienstKringId>1</agirws:dienstKringId>
        <agirws:directie fid="DirectieType.1">
            <agirws:directieId>1</agirws:directieId>
            <agirws:directieCode>li</agirws:directieCode>
            <agirws:directieName>Limburg</agirws:directieName>
         </agirws:directie>
         <agirws:dienstKringCode>LIMM</agirws:dienstKringCode>
         <agirws:dienstKringName>LIMM | Dkr.
             MaastrichtMaas</agirws:dienstKringName>
      </agirws:dienstKring>
   </gml:featureMember>
  <gml:featureMember>
     <agirws:dienstKring fid="DienstKringType.2">
         <agirws:dienstKringId>2</agirws:dienstKringId>
        <agirws:directie fid="DirectieType.1">
            <agirws:directieId>1</agirws:directieId>
            <agirws:directieCode>li</agirws:directieCode>
            <agirws:directieName>Limburg</agirws:directieName>
         </agirws:directie>
         <agirws:dienstKringCode>LINM</agirws:dienstKringCode>
         <agirws:dienstKringName>LINM | Dkr.
             NijmegenMaas</agirws:dienstKringName>
      </agirws:dienstKring>
   </gml:featureMember>
[...]
 </ogcwfs:FeatureCollection>
```

http://localhost:8081/agibeheer/wfs/AGIBEH?request=Getfeature&typename=Lmaaiveld&maxfeatures=5


```
<?xml version="1.0" encoding="utf8" ?>
 <ogcwfs:FeatureCollection xsi:schemaLocation="http://www.agirws.nl/xml/namespaces/gml
      http://localhost:8081/agibeheer/wfs/AGIBEH?REQUEST=DescribeAllFeatureType&SER
      VICE=WFS http://www.opengis.net/wfs
      http://localhost:8081/agibeheer/wfs/AGIBEH/REQUEST/get/DATA/LPR/wfs/basic.xsd
      " xmlns:gml="http://www.opengis.net/gml"
      xmlns:agirws="http://www.agirws.nl/xml/namespaces/gml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:ogcwfs="http://www.opengis.net/wfs">
    <gml:boundedBy>
      <gml:Box srsName="EPSG:28992">
          <gml:coordinates>175307.44,307293.88 191493.48,353801.62</gml:coordinates>
        </gml:Box>
     </gml:boundedBy>
    <gml:featureMember>
      <agirws:lmaaiveld>
          <agirws:beheerObjectId>87</agirws:beheerObjectId>
          <agirws:codeKm>20.8</agirws:codeKm>
          <agirws:nameContactPerson>BEHEER</agirws:nameContactPerson>
         <agirws:geometry>
           <gml:Polygon srsName="EPSG:28992">
             <gml:outerBoundaryIs>
               <gml:LinearRing srsName="EPSG:28992">
                   <gml:coordinates>183902.503,338604.309,0.0 183884.085,338610.003,0.0
                       [...]
                       3902.503,338604.309,0.0</gml:coordinates>
                 </gml:LinearRing>
               </gml:outerBoundaryIs>
             </gml:Polygon>
          </agirws:geometry>
         <agirws:metadataDinsKring fid="MetadataDinsKringType.1">
             <agirws:metadataDinsKringId>1</agirws:metadataDinsKringId>
             <agirws:datasetNaam>Beheerkaart</agirws:datasetNaam>
             <agirws:samenvatting>Beheerkaart RWS</agirws:samenvatting>
             <agirws:datasetProducent>LIMM | Dkr. MaastrichtMaas</agirws:datasetProducent>
             <agirws:ruimtelijkSchema />
             <agirws:datasetTekenset />
             <agirws:datasetHerkomst />
             <agirws:statusVanDekking />
             <agirws:actualiteit />
             <agirws:datatype />
             <agirws:toepassingDatatype />
          </agirws:metadataDinsKring>
          <agirws:categorieCode>KS</agirws:categorieCode>
          <agirws:zijdeCode></agirws:zijdeCode>
          <agirws:waterSysteemCode>mk</agirws:waterSysteemCode>
          <agirws:waterSysteemName>Maas kanalen</agirws:waterSysteemName>
          <agirws:hoofdWaterSysteemCode>MAA</agirws:hoofdWaterSysteemCode>
          <agirws:hoofdWaterSysteemName>Maas</agirws:hoofdWaterSysteemName>
          <agirws:dienstKringCode>LIMM</agirws:dienstKringCode>
          <agirws:dienstKringName>LIMM | Dkr. MaastrichtMaas</agirws:dienstKringName>
          <agirws:directieCode>li</agirws:directieCode>
          <agirws:directieName>Limburg</agirws:directieName>
        </agirws:lmaaiveld>
     </gml:featureMember>
 <gml:featureMember>
[...]
  </ogcwfs:FeatureCollection>
```

9.4 LockFeature (POST request)

This is a sample LockFeature POST request, to lock the instances of agirws:BeheerObjectType that have beheerObjectId equal 31 for duration of a transaction:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:LockFeature
        xmlns:ogc="http://www.opengis.net/ogc"
        xmlns:ogcwfs="http://www.opengis.net/wfs"
        version="1.0.0" expiry="10" service="WFS">
        <ogcwfs:Lock handle="lock1" typeName="agirws:BeheerObjectType">
                <ogc:Filter>
                        <ogc:PropertyIsEqualTo>
                                <ogc:PropertyName>beheerObjectId</ogc:PropertyName>
                                <ogc:Literal>31</ogc:Literal>
                        </ogc:PropertyIsEqualTo>
                </ogc:Filter>
        </ogcwfs:Lock>
</ogcwfs:LockFeature>
```

If successful we get the result below:

```
<?xml version="1.0" encoding="utf-8" ?>
 <WFS_LockFeatureResponse xmlns="http://www.opengis.net/wfs">
     <LockId>BD259E07674144ADE034080020F056A4</LockId>
 </WFS_LockFeatureResponse>
```

9.5 Transactions

Below is a sample transaction request: this is an update transaction on BeheerObjectType. The name of the BeheerObject having the property beheerObjectId equals “31” is set to “NEWNAME”. Of course an Update transaction on a locked features will succeed if the LockId obtained on a previous LockFeature request is used.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:Transaction handle="transac1"
        xmlns:ogc="http://www.opengis.net/ogc"
        xmlns:ogcwfs="http://www.opengis.net/wfs"
        version="1.0.0">
        <ogcwfs:LockId>BD259E07674144ADE034080020F056A4</ogcwfs:LockId>
        <ogcwfs:Update handle="upd1">
                <agirws:BeheerObjectType >
                        <gml:name>NEWMANE</wfs:TYPE>
                </ agirws:BeheerObjectType >
                <ogc:Filter>
                        <ogc:PropertyIsEqualTo>
                                <ogc:PropertyName>beheerObjectId</ogc:PropertyName>
                                <ogc:Literal>31</ogc:Literal>
                        </ogc:PropertyIsEqualTo>
                </ogc:Filter>
        </ogcwfs:Update>
</ogcwfs:Transaction>
```

10 Bibliography

- *Geographic Markup Language (GML) 2.00* – OpenGIS® Implementation Specification, OGC 01-029, Open GIS Consortium, Inc. 20th of February 2001.

- *Web Feature Server – OpenGIS® Implementation Specification*, OGC 02-058, Open GIS Consortium, Inc. 19th of September 2002.

- *Geographic Markup Language (GML) 3.00 – OpenGIS® Implementation Specification*, OGC 03-023-r4, Open GIS Consortium, Inc. 29th of January 2003.

- *Unified Modeling Language* - www.uml.org.

- *XML Schema*: www.w3c.org/XML/Schema.

- IONIC Products Team, *RedSpider Concept Guide,* IONIC Software S.A., 2004

- IONIC Products Team, *RedSpider Administrotor's Guide,* IONIC Software S.A., 2004

- XML The Site - http://www.softwareag.com/xml/about/