GIMA

Geographical Information Management and Applications

Master of Science Thesis

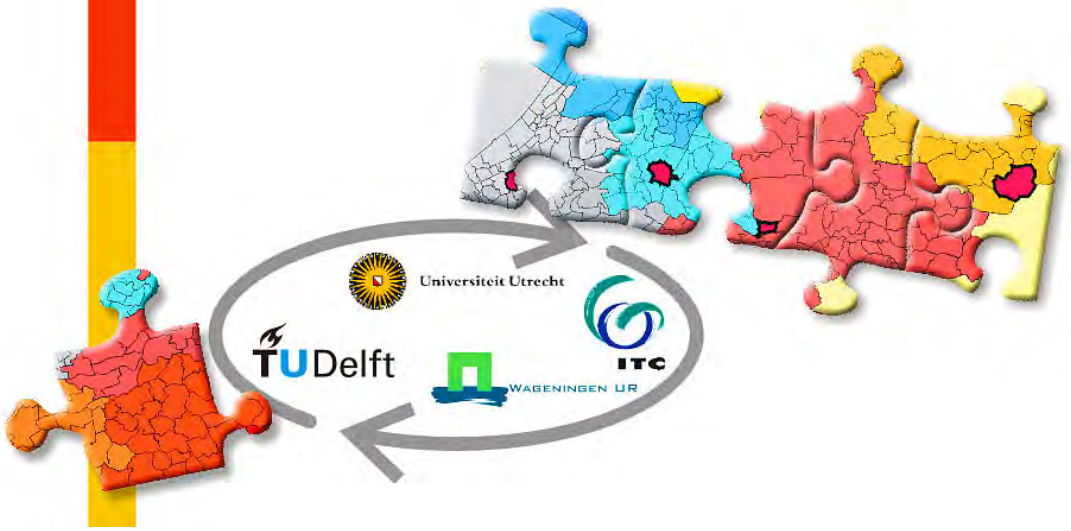
The Land Administration Domain Model 'Survey Package' and Model Driven Architecture

Jan van Bennekom-Minnema

May 2008

Professor: **prof. dr. ir. P.J.M. van Oosterom**
Delft University of Technology

Supervisor: **ir. C.H.J. Lemmen**
International Institute for Geo-Information Science and Earth Observation (ITC), and
The Netherlands' Cadastre, Land Registry and Mapping Agency (Kadaster)



Preface

My graduation research project (also referred to as the master thesis project) has been performed from September 2007 to May 2008, concluding the MSc programme **Geographical Information Management and Applications** (GIMA, URL 1).

During my GIMA MSc programme, since September 2005, I became father of a beautiful daughter Isabel, got married with my dear and beautiful wife Ilse, got hospitalised and recovered, moved to another village, worked in Croatia, Romania, Uzbekistan, Trinidad & Tobago, and Ghana, and became father of another beautiful daughter Phileine. I would not have achieved what I did, if Ilse wouldn't have supported and facilitated me like she did in combining and performing all these activities.

My graduation research project has been conducted under supervision of the Delft University of Technology, the International Institute for Geo-Information Science and Earth Observation (ITC), and the Netherlands' Cadastre, Land Registry and Mapping Agency (Kadaster). Peter van Oosterom has accommodated me greatly with his knowledge, support, reviews, and commitment, demonstrated during the course of my graduation research project.

I would like to express my gratitude to both Ilse and Peter, but also to Chrit Lemmen, and Joao da Fonseca Hespanha de Oliveira, co-authors of our article "*The Model Driven Architecture approach applied to the Land Administration Domain Model version 1.1 - with focus on constraints specified in the Object Constraint Language*", and the external supervisors and technical experts from Kadaster: Klaas van der Hoek, Joop van Buren, Hans Swarts en Tom Venhorst, for all their support.

Jan van Bennekom-Minnema

May 30, 2008

Table of Contents

Summary

Key words: Land Administration Domain Model, Survey Package, Model Driven Architecture, Object Constraint Language, spatial constraints, Enterprise Architect.

Introduction
The master of science thesis project called "**The Land Administration Domain Model 'Survey Package' and Model Driven Architecture**" will be described and concluded in this report. The main subjects for the research are the *Land Administration Domain Model* (LADM), specifically the Survey Package, dealing with survey measurements. Secondly, the *Model Driven Architecture* (MDA), a software design methodology to generate platform specific information systems based on platform independent models, specified in the Unified Modelling Language (UML) and the Object Constraint Language (OCL). A custom developed *MDA Prototype* has been developed, aiming at the implementation of the *Adapted LADM 'Survey Package'* in a PostgreSQL/PostGIS object-relational database. Thirdly, an analysis of the quality of the Dutch cadastral map is performed, based on data loaded into this the *Adapted LADM 'Survey Package'* PostGIS database.

The objective of the master thesis project was translated into the below mentioned main research question, which has been answered by literature research, case studies, and practical experiments, as described in this Master Thesis Report:

How can the Land Administration Domain Model 'Survey Package' be implemented and deployed based on Model Driven Architecture principles, and how can the Land Administration Domain Model 'Survey Package' be extended and improved?

The master thesis project will be summarised in the following sections by describing the main topics, highlighting the results and describing the recommendations for future development and research.

Land Administration Domain Model (LADM)
The Land Administration Domain Model (LADM), in the form of a UML class diagram, models the object classes of land registration and cadastre [Lemmen and Van Oosterom, 2006]. The LADM is described in an ISO TC211 standard 19152, currently "under development" [ISO/TC211, 2008]. The Land Administration Domain Model consists of a number of packages; the Survey Package contains classes, related to survey measurements, e.g. the class SurveyPoint and SurveyDocument. One of the goals of the LADM is to "serve as a basis for land administration system development executed on Model Driven Architecture principles".

Evaluation of extension of the LADM 'Survey Package'

As a basis for experimenting with Model Driven Architecture principles in a MDA prototype, classes of the LADM, as well as non-LADM classes have been selected and adapted, referred to as the *Adapted LADM 'Survey Package'*, which has been influenced by the availability of test data provided by Kadaster. In this process, some improvements have been recommended, for example the consideration of the class Survey Project. Various publications have been discussed [Ingvarsson, 2005, Lee, 2005, Open Geospatial Consortium, 2006b], which provide a basis for further improvements of the LADM 'Survey Package'.

Model Driven Architecture (MDA)
Model Driven Architecture (MDA) is a software design methodology to generate information systems on different target platforms, based on platform *independent* models and specifications. A platform independent model (PIM) contains platform independent details on application's data (data types) and functionality (operations). Based on MDA transformation rules, described in the *platform specific transformation specification*, the PIM will be preferably be converted automatically into a platform *specific* model (PSM), adding platform specific details to the model. For example, the transformation, from an object-oriented PIM to a PSM, targeting an object-relational database (investigated in this master thesis project), requires a mapping of object-oriented to relational data types and operations, described in MDA transformation rules. MDA is supported by the standards Meta Object Facility (MOF), Object Constraint Language (OCL), Unified Modelling Language (UML) as specified by the Object Management Group [OMG, 2003, OMG, 2006a, OMG, 2006b, OMG, 2007b].

Object Constraint Language (OCL)
One of the standards discussed is the Object Constraint Language (OCL), a formal language, which has been defined as an extension to UML, to define those constraints, which cannot be recorded in UML.

Constraints assessed and classified from an implementation viewpoint

From an implementation viewpoint, OCL invariants have been divided into: constraints applicable to one instance; constraints applicable to multiple instances for one class; or constraints applicable to multiple instances of multiple classes. Relational databases offer functionality to implement constraints with regard to mandatory columns, primary key, unique key, and foreign key constraints, and simple base table check constraints. For other types of constraints, examples of OCL

invariants have been defined on the Adapted LADM 'Survey Package' UML class diagram, and used in discussions and experiments on implementation.

Constraints implementation method specified for validation at transaction level

For this implementation of constraints in a relational database, the SQL *assertion* and the *base table check constraint* with sub queries could be useful, but this functionality is not offered by object-relational databases like PostgreSQL/PostGIS. An alternative implementation of OCL constraints is required with row and statement level triggers, and for some constraints, transaction level triggers are needed, which check the constraints only after executing a group of DML statements (Data Manipulation Language, i.e. insert, update, delete) for multiple tables. Transaction level triggers imply the implementation of a (custom developed) transaction management mechanism, which has been described.

MDA Prototype, Based on Enterprise Architect

Enterprise Architect (EA, URL 18, [SparxSystems, 2007]) offers standard support for relatively straightforward MDA transformation rules from object-oriented PIMs to relational database models (PSM), but more sophisticated transformations (e.g. the implementation of enumeration classes and attributes as base table check constraints) require a considerable custom development. With regard to OCL, Enterprise Architect offers validation of OCL constraints, but is not capable of transforming or implementing OCL constraints into a relational database, unless custom developed functionality is created, based on the EA Software Development Kit (EA SDK).

MDA Prototype created which automatically transforms PIM to PSM to PostGIS

A Model Driven Architecture (MDA) prototype has been built, based on the MDA processes and transformations, and with help of the possibilities offered by Enterprise Architect (EA) software and toolkit, to investigate the transformation of an object oriented platform independent model (PIM) to a platform specific model (PSM). The Adapted LADM 'Survey Package' functioned as the PIM (i.e. a UML class diagram), and the target PSM is an object-relational PostgreSQL database, with a PostGIS extension for spatial data and functions.

MDA prototype transforms and implements geometric data types and operations

The MDA prototype is capable of executing MDA transformation rules from PIM to PSM, handling and transforming a selection of geometric data types (e.g. GM_Point, GM_LineString, GM_Polygon). The MDA prototype has some limited functionality with regard to implementing spatial and non-spatial OCL invariants as based table check constraints, and with regard to transforming OCL defined on PIM elements to OCL based on PSM elements.

Solution for differences between O-O (PIM) and relational DBMS (PSM)

The MDA prototype is based on a selection of MDA Transformation Rules, applicable to specific PIM elements (in UML/OCL), resulting in a PSM implementation for each of the PIM elements. If the "gap" between object-oriented (PIM) and relational DBMS (PSM) is not too big, the transformation can be relatively simple and less arbitrary. When the difference between PIM and PSM elements is significant, a more complex implementation choice will have to be made (and custom developed).

A working Adapted LADM 'Survey Package' generated and implemented in object-relational database PostgreSQL/PostGIS by MDA Prototype

Based on the experiments with the MDA prototype, it is expected that the majority of MDA transformation rules, including the ones that have not been considered in the master thesis project, can be performed automatically, including handling and transforming a selection of geometric data types, provided that the PIM and PSM elements, and transformations between them are well defined and structured. The PIM of the Adapted LADM 'Survey Package' has been automatically generated by the MDA prototype to a PSM. The PSM has been used to generate DDL scripts for the creation of a PostGIS database, to serve as the basis for the analysis of the quality of the Dutch cadastral map.

Kadaster project "Registration Map Quality"

A Kadaster project called "Registration Map Quality" is dealing with differences between the measured coordinates of objects (i.e. parcels and buildings), and, the adjusted (NL: vereffende) coordinates of the representations of those objects on the digital map, respectively before and after the *2nd phase control point constrained network adjustment* (NL: tweede fase aansluitings-vereffening) [van Buren, 2006]. The 2nd phase adjustment transforms the (accurate) measurements, to fit them into the (less accurate) cadastral map. These differences provide an indication of the quality of the (digital) cadastral map.

Performed analysis of the quality of the Dutch cadastral map at different levels

Kadaster has provided data to populate the implementation of the Adapted LADM 'Survey Package' PSM in PostGIS (generated by the MDA Prototype). Several comments to the provided data have been made, and specifically the survey measurements from April 2006 to December 2007, loaded in 2 steps into PostGIS, have been subject to an analysis. The conclusion was drawn that in general, the required "graphical precision" of maximum 20 and 40 cm difference (between measured and transferred coordinate) in respectively urban and rural areas is obtained. The lowest difference (best quality) is seen in cadastral office Flevoland and Roermond, the highest is seen in Zoetermeer. However, individual cases (of cadastral sections) exist where these maximum differences were exceeded, even if the norm is applied that 95% of the measurements should compy with the maximum 20/40 cm differences. Further analysis is recommended into these exeptions, as well as into the large outliers that where identified.

Open source tools used for Adapted LADM 'Survey Package'

Open source tools have been used extensively for the activities of the master thesis project, like object-relational database PostgreSQL/PostGIS, uDig for visualisation of geographical data (analysis) in PostGIS, and FWTools for converting spatial data to and from PostGIS, and have proven to be suitable and stable.

Conclusion

The original scope and priorities of the master thesis project have been changed, the focus and priority were set on this part of the objective: "*to investigate the possibilities and limitations of the Model Driven Architecture (MDA) approach by performing a literature study, and by creating a prototype of the (adapted) LADM Survey Package, based on MDA principles"*. The significance of this master thesis project is determined by the work leading to:

- The conclusion that a transformation from a PIM to a PSM, based on MDA principles (i.e. platform specific transformation specification) can be performed fully automatically for most MDA transformation rules, also for spatial data types.

- The recommendation to design and build a MDA tool, fully compliant with MOF, UML, OCL; expanding the current amount and variety of MDA Transformation Rules; using XMI as model exchange format; capable of implementing UML elements and OCL constraints (PIM) in object-relational databases (PSM), in relation to the recommendation to extend OCL with spatial definitions of data types and operations.

- The preliminary analysis based on survey measurement project from April 2006 - December 2007, indicating that the quality of the Dutch Cadastral Map is compliant with the requested "graphical precision", in combination with the recommendation to perform additional research in some of the exceptions.

Other recommendations for future research have been provided, summarised as:

Extend and implement the LADM 'Survey Package'

The observed errors and established improvements can be used to extend the LADM 'Survey Package', also based on the mentioned relevant publications [Lee, 2005, Open Geospatial Consortium, 2006b]. The MDA tool will be used to implement the LADM 'Survey Package' again into PostGIS. The MDA tool should be extended to operate with more geometric and topological data types, structures and operation, as well as 'spatial' OCL. The OCL invariants will be (semi-)automatically implemented based on a database transaction management mechanism.

Implement improvements with regard to survey measurement handling

Further analysis of the processes and data with regard to survey measurement handling will be performed. One of the objectives is to be able to perform a reverse "fitting" process where accurate measurements are used to improve the quality of the cadastral map, as opposed to adjusting the accurate measurements to the (less accurate) map, as it is currently conducted.

List of Figures

List of Terms and Abbreviations

Term/Abbreviation	Description
(NL: text)	Dutch translation of preceding English term, e.g. *word (NL: woord).*
API	Application Programming Interface.
ASCII	American Standard Code for Information Interchange.
CASE	Computer Aided Software Engineering.
CCDM	Core Cadastral Domain Model, currently referred to as LADM.
CIM	Computation Independent Model in MDA.
CIV	Computation Independent Viewpoint in MDA.
DDL	Data Definition Language, defining database elements, e.g. create table scripts.
DML	Data Manipulation Language, SQL commands for manipulation of data in relational databases, e.g. SELECT, INSERT, DELETE, and UPDATE.
DRA	Digital Reconstruction Archive, with the 1st phase free network adjustment results based on survey measurements.
EA	Enterprise Architect (URL 18).
EA SDK	Enterprise Architect Software Development Kit.
ETRS89	European Terrestrial Reference System.
FIG	International Federation of Surveyors (URL 7).
GIMA	Master of Science in Geographical Information Management and Applications (URL 1).
GNSS	Global Navigation Satellite System.
IDE	Integrated Development Environment.
INSPIRE	Infrastructure for Spatial Information in Europe (URL 10).
ISO/TC211	International Organization for Standardization, Technical Committee 211 on standardization in the field of digital geographic information (URL 8).
Kadaster	The Netherlands' Cadastre, Land Registry and Mapping Agency (URL 2).
LADM	Land Administration Domain Model, a.k.a. CCDM.
LADM SP	Land Administration Domain Model 'Survey Package'.
LKI	Surveying Cartographic Information (NL: Landmeetkundig Kartografische Informatie).
MDA	Model Driven Architecture.
MDA prototype	The MDA prototype of the master thesis project, based on Enterprise Architect software, as described in Chapter 6 and 7.
MDG	Model Driven Generation.
MOF	Meta-Object Facility.
MOVE3	MOVE3 (URL 21), software for the design, adjustment, and quality control of 3D, 2D and 1D geodetic networks, the processing of inbound and outbound measurements.
O&M	OGCs Observations and Measurements model, as part of the Web Enablement activities (SWE).

Term/Abbreviation	Description
OCL	Object Constraint Language.
OGC	Open Geospatial Consortium (URL 5).
OMG	Object Management Group (URL 6).
Oracle CDM	Oracle's Custom Development Method.
PIM	Platform Independent Model in MDA.
PIV	Platform Independent Viewpoint in MDA.
PSM	Platform Specific Model in MDA.
PSV	Platform Specific Viewpoint in MDA.
RD	"Rijksdriehoek" 2D spatial reference system, used in the Netherlands, spatial reference id 28992.
RDBMS	Relational DataBase Management System.
SDK	Software Development Kit.
SFA-SQL	OpenGIS Simple Features Specification for SQL [Open Geospatial Consortium, 1999].
SQL	Structured Query Language.
SRID	Spatial Reference IDentifier of a spatial reference system.
STDM	Social Tenure Domain Model, a specialisation of the LADM.
SWE	OGCs Web Enablement activities.
the master thesis project	The master thesis project "The Land Administration Domain Model 'Survey Package' and Model Driven Architecture", described in this document.
TIR	Terrestrial Collection and Reconstruction (NL: Terrestrische Inwinning & Reconstructie).
UML	Unified Modelling Language.
UN-Habitat	The United Nations Human Settlements Programme (URL 9).
URL	Uniform Resource Locator, a string of characters used to represent and identify a page of information on the Internet.
XML	eXtended Mark-up Language.

1 Introduction

In this document, the approach, execution and results of the master of science thesis project "**The Land Administration Domain Model 'Survey Package' and Model Driven Architecture**" will be described. The main subjects and concepts for the project will be introduced in the following sections.

Land administration is executed in many different ways all over the world, with respect to *legal and organizational characteristics, levels of planning and control, aspects of multipurpose cadastres, and responsibilities of the public and the private sectors* [Kaufmann and Steudler, 2001]. Within continents, and even between neighbouring countries, major and many variations can be witnessed [Larsson, 1991]. Kaufmann and Steudler concluded that the most obvious trend in the land administration domain is the automation of the systems and the digitization of data.

Land Administration Domain Model
Driven and inspired by those variations and trends in land administration, an initiative was taken at the FIG congress in Washington (2002, URL 7), to develop *a standardized Land Administration Domain Model* (LADM), to model the object classes of land registration and cadastre [Lemmen and Van Oosterom, 2006]. The LADM, formerly known as the Core Cadastre Domain Model (CCDM), has evolved since 2002, resulting in a number of versions, enjoying the involvement of many individual participants, as well as the involvement of renowned organisations like OGC (URL 5), ISO/TC211 (URL 8), UN Habitat (URL 9), and INSPIRE (URL 10). The LADM is described in UML class diagrams [OMG, 2007a, OMG, 2007b]. One of the packages of the LADM is the "Survey Package", which deals with the measurements of immovable objects, and how these measurements are related to the representations of these objects in the information system in general, and on the cadastral map in particular (see Chapter 2).

Model Driven Architecture
One of the goals of the LADM is to provide an extensible basis for efficient and effective cadastral system development based on a *model driven architecture*. Model Driven Architecture (MDA) is a software design methodology to create model based specifications and model based generation of information systems [OMG, 2003]. One of the basic elements of MDA is a platform-independent model (PIM), for example the LADM 'Survey Package'. A PIM describes an application's functionality and data, independent of the intended implementation technology, which makes the PIM relatively stable in environments where technology is continuously updated and improved. The PIM will be the basis for a transformation to one or more platform-

specific models (PSM). The PSM will be used to create the actual implementation of the model in the chosen platform/environment, for example Data Definition Language (DDL) statements for a PostgreSQL (URL 14) database, or XML/GML schemas for data exchange. The MDA works together with other OMG modelling specifications, such as UML, MOF, OCL and XMI (section 3.3). A specific interest exists within this master thesis project with regard to the (im-) possibilities of the combination MDA and geographic data and constraints (see theory and practise in Chapter 3 and 6).

Object Constraint Language

One of the standards, related to the modelling of constraints in data models is the Object Constraint Language (OCL). OCL has been defined as an extension of the UML, because UML is not capable of modelling every kind of constraint to the classes, attributes and associations. These additional constraints can be defined and implemented in many different ways, which could lead to situations where constraints are not an integral part of the system, where ambiguities in communication on these constraints occur, and where maintenance of constraints is cumbersome. OCL is a formal language that enables the unambiguous specification of those constraints, related to elements in a UML model, for example a UML class diagram (see theory and practise in Chapter 4 and 6).

Kadaster Project "Registration Map Quality"

At The Netherlands' Cadastre, Land Registry and Mapping Agency (Kadaster), a project called "Registration Map Quality" (in Dutch: Registratie Kaart Kwaliteit) is being executed [van Buren, 2006]. The project deals with differences between the cadastral measurements of objects (i.e. measured coordinates of for example parcels and buildings), and the adjusted (transformed) coordinates of the representations of those objects on the map. These differences provide an indication of the quality of the cadastral map, which is expressed in a quality indication, referred to as "graphical precision" of ±20 cm in urban and ±40 cm in rural areas. The project "Registration Map Quality" will serve as the basis for a case study with regard to the LADM 'Survey Package', and will provide required (test) data for populating the database, generated as a result of MDA activities (see Chapter 5 and 7 for case study and analysis of the quality of the Dutch cadastral map).

1.1 Objective and Research Question

The goals of the Land Administration Domain Model (LADM), as cited in the first section of Chapter 2, are highly appreciated and considered to be very interesting by the author. Model Driven Architecture (MDA) concepts, as well as the opportunity to gain experience with MDA tools, including the application and implementation of the Object Constraint Language (OCL) are considered to be very interesting as well, and relevant for the international land administration domain and the LADM, especially with regard to geographic data. At the start of the master thesis project, its objective was defined in two parts:

- *On the one hand, the objective is to investigate the possibilities and limitations of the Model Driven Architecture (MDA) approach by performing a literature study, and by creating a prototype of the (adapted) LADM Survey Package, based on MDA principles.*

- *On the other hand, the objective is to establish an extension of the Land Administration Domain Model (LADM) with regard to its Survey Package by performing a literature study, and by investigating a case from Kadaster ("Registration Map Quality" project).*

This objective is translated into the main research question for the master thesis project:

How can the Land Administration Domain Model 'Survey Package' be implemented and deployed based on Model Driven Architecture principles, and how can the Land Administration Domain Model 'Survey Package' be extended and improved?

1.2 Approach

The approach obtaining the objective and answering the research question is explained by breaking the research question up into sub-topics and related sub-questions, and defining the specific activities for each of those. These sub-topics are:

- Evaluation of LADM 'Survey Package'
- Evaluation of Model Driven Architecture
- Evaluation of Constraints in Data Modelling
- Performing the Case Study: Survey Package Kadaster and LADM
- Create MDA Prototype to Implement Adapted LADM 'Survey Package'

1.2.1 Evaluation of LADM 'Survey Package'

The LADM 'Survey Package', modelled in UML class diagrams, will be reviewed by addressing the questions: *What is the current definition of the data collection part (Survey Package) of the LADM? Which current versions exist and are subject to research? Which other standards apply to this part of the model and to the modelling itself? What language and tool(s) will be used for data modelling?*

These questions will be answered based on literature research and interviews/meetings with participants, knowledgeable on the subject of LADM 'Survey Package'. Section 2.2 will address a number of variants of the LADM, section 2.3 will address the Survey Package, and section 2.4 will refer to standards and publications with regard to an extension of LADM. Section 5.4 will propose an *Adapted LADM 'Survey Package'* as platform independent input to the MDA prototype.

1.2.2 Evaluation of Model Driven Architecture

The topic of Model Driven Architecture (MDA) will be introduced by answering the questions: *What is Model Driven Architecture? What are its main elements? How is it related to object constraint language? What tool(s) will be used for Model Driven Architecture?*

This topic will be based on a study of available literature on MDA and the related standards like UML, MOF, OCL, and XMI, as a basis for the experimental phases of the master thesis project. Section 3.2 will address the MDA conceptual model, and the processes to transform platform independent (object-oriented) models to platform specific models (in the prototype an object-relational database management system), as a basis for prototyping in Chapter 6 (i.e. the MDA prototype). The MDA related standards will be discussed in section 3.3.

1.2.3 Evaluation of Constraints in Data Modelling

Since UML is not capable of modelling every type of constraint, the following questions will be addressed: *What is the role of constraints in data modelling, and how can the constraints to the data elements in the LADM 'Survey Package' be specified? What language and tool(s) will be used for object constraint modelling?*

This topic will be addressed by literature research on the one hand; Chapter 4 will describe Object Constraint Language (OCL), and discuss a possible implementation method, based on practises with regard to constraints. On the other hand, initial experiments will be conducted, which will be presented in Chapter 6, specifically section 6.7, as well as details in the "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)"

1.2.4 Performing the Case Study: Survey Package Kadaster and LADM

A case study will be performed to answer various questions: *What data with regard to the measurements of spatial objects is being handled by Kadaster? What rules/constraints apply to these data? What are the differences between Kadaster data and the current LADM 'Survey Package'? How can quality of the map be assessed based on the differences between measurements and representations?*

The case study on the Kadaster's "Registration Map Quality" project will provide the answers to these questions, specifically the investigation of the Kadaster's applications TIR and MOVE3, used for handling survey measurements. Chapter 5 will describe the processes and data involved with survey measurements, where as section 7.4 and 7.5 will provide more details and analysis on the survey data (and the quality of the cadastral map) of Kadaster, based on an implemented *Adapted LADM 'Survey Package'*.

1.2.5 Create MDA Prototype to Implement Adapted LADM 'Survey Package'

An MDA prototype will be designed and developed, for automatic execution of the MDA transformations from platform independent to platform specific environments. The final goal of the MDA prototype is the implementation of the adapted LADM 'Survey Package' into PostGIS, which will be populated with survey measurement related data provided by The Netherlands' Cadastre, Land Registry and Mapping Agency (Kadaster). The following sub-questions will be addressed: *How can the Adapted Survey Package of LADM be implemented, with help of MDA processes, demonstrated with the MDA prototype? Is the MDA prototype suitable for geographic elements of the LADM Survey Package? How can database object generation language statements (DDL) and exchange formats (e.g. XML/GML schemas) be generated, based on the Adapted Survey Package of LADM? Which standards apply? Which (open source) tools are suitable?*

A MDA prototype will be built for the MDA processes and viewpoints (section 6.2 to 6.4), with the proposed Adapted LADM 'Survey Package' (section 5.4) as input. The transformed Platform Specific Model is presented in section 6.8. The evaluation criteria for the MDA prototype that will be assessed, are the compliance with MDA concepts and processes (Chapter 3), the degree of automatic transformations (manual, semi-automatic or automatic), the standard support by MDA tools (i.e. Enterprise Architect) and required custom development for the MDA prototype. These criteria will be reported on in section 6.9 and Chapter 8.

1.2.6 Report Structure

The Chapters 2 to 7 in this master thesis report are in line with the topics defined in this section. Chapters 2 to 4 are mainly related to literature research, chapters 5 to 7 are mainly related to case studies, prototyping and analysis. Chapter 8 will contain the overall reflection, and conclusions of the master thesis project, leading to a number of recommendations. Additional details will be provided in the appendices A to K (page 122 to 191). In explaining the concepts, subject to this report, various articles, papers and chapters will be referred to. In discussing and putting the concepts into the perspective of this report, there is no escape from repetition of the relevant sections from these articles. Therefore, a choice has been made to repeat some of the information found in these articles, with proper referencing to source articles, in order to make this thesis report comprehensive and readable.

2 The Land Administration Domain Model 'Survey Package'

2.1 Introduction

Driven and inspired by many variations and trends in land administration, an initiative was taken at the FIG congress in Washington (2002, URL 7) to develop a standardized Land Administration Domain Model (LADM), to model the object classes of land registration and cadastre [Van Oosterom et al., 2006]. The LADM, at that point in time referred to as the Core Cadastral Domain Model (CCDM), has two important goals:

- "*Avoid reinventing and re-implementing the same functionality over and over again, and provide a extensible basis for efficient and effective cadastral system development based on a model driven architecture (MDA)*", and

- "*Enable involved parties, both within one country and between different countries, to communicate based on the shared ontology implied by the model*".

The LADM/CCDM has evolved since 2002, resulting in a number of versions, enjoying the involvement of many individual participants, as well as the involvement of renowned organisations like OGC (URL 5), ISO/TC211 (URL 8), UN Habitat (URL 9), and INSPIRE (URL 10). Currently the LADM is documented in ISO/TC211 standard 19152 [ISO/TC211, 2008].

2.2 Land Administration Domain Models

Different variants and specialisations of the LADM will be discussed in the following sections.

2.2.1 Core Cadastral Domain Model (Sixth version)

In the first few years of its existence the LADM has been referred to as the Core Cadastre Domain Model (CCDM). The last version of the CCDM is described by Lemmen and Van Oosterom [Lemmen and Van Oosterom, 2006, Van Oosterom et al., 2006]. The LADM/CCDM is modelled in class diagrams of the Unified Modelling Language [Jacobson et al., 1999, OMG, 2007a, OMG, 2007b], and is built up around core elements with regard to real estate objects and their geographical description, and persons holding rights and responsibilities to these objects (Figure 1

and the figures in "Appendix A: LADM UML Class Diagrams"). The LADM has been proposed to ISO/TC211 as a basis for a standard for a Land Administration Domain Model, currently documented in ISO19152 [ISO/TC211, 2008].

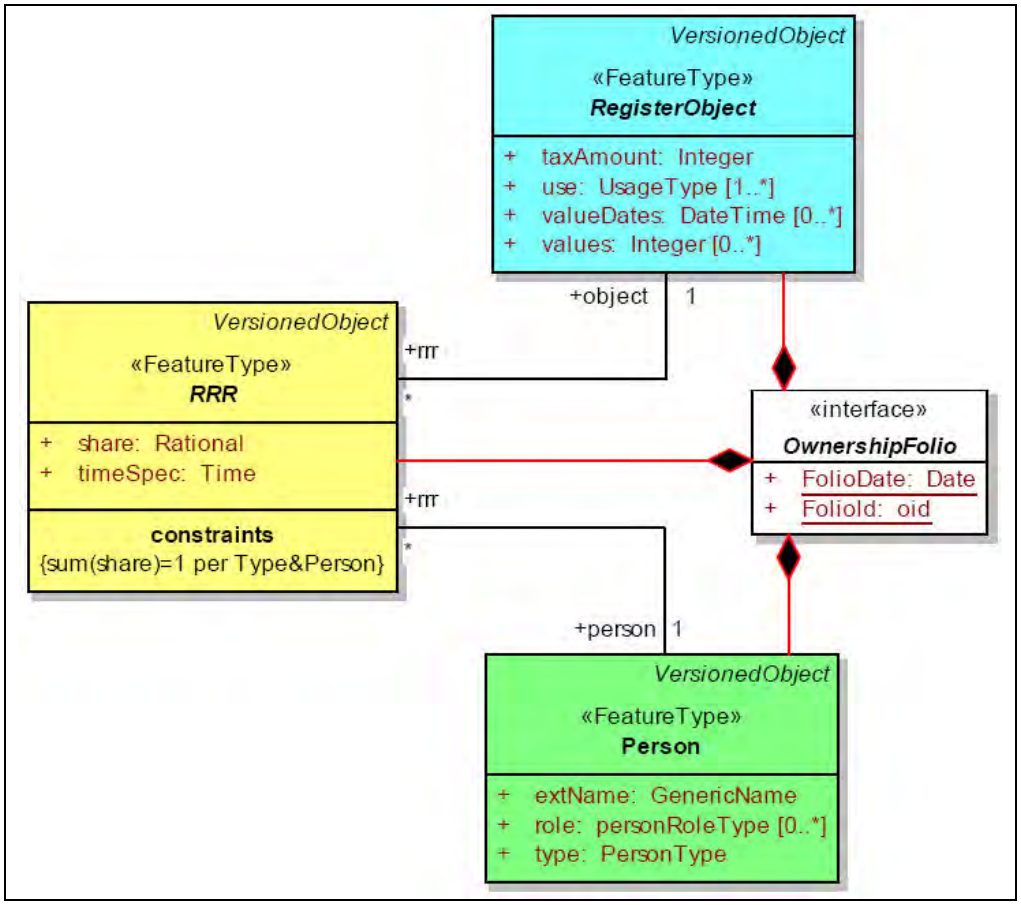


Figure 1 - The Core LADM Classes (taken from [ISO/TC211, 2008], fig.1)

The core of the LADM/CCDM (Figure 1) consists of the classes RegisterObject (e.g. parcels, buildings) and Person (natural and non-natural), which are combined through the class RRR (rights, restrictions and responsibilities). This core is the foundation of every land administration [Van Oosterom et al., 2006], and together with the other LADM/CCDM classes they are grouped into a number of packages (see Figure 74):

- Yellow: Legal and administrative related classes
- Green: Person related classes
- Blue: Immovable related classes
- Pink: Surveying related classes
- Purple: Geometric and Topological classes

The yellow package describes the rights that are applicable to a registered object, as well as the legal documents that establish and describe the right. The green package describes the various types of person related classes (with specialisation natural and non-natural persons), that play a role in registering the rights to immovable's. The classes Person and Surveyor are also related to the class SourceDocument and its

specialisation SurveyDocument. The blue package describes the registered object and all its specialisations and groupings.

The pink package (also referred to as the 'Survey Package') models the measurements of immovable objects, and how these measurements are related to the representations of these objects in the information system in general and on the cadastral map in particular [Lemmen and Van Oosterom, 2006] [section 7]. The Survey Package is subject to the prototype of the master thesis project, described in Chapter 6 and 7.

The purple package (see also Figure 71) describes the geometrical and topological related classes in the model, which are based on the standards of ISO and OGC [ISO/TC211, 2003b, Open GIS Consortium, 1999]. The class Parcel has a spatial description in class GeomTopoRepresentation, which is decomposed into a topology of TP_Volume, TP_Face, TP_Edge, TP_Node, following the standard ISO19107 [ISO/TC211, 2003b]. The class SurveyPoint has an association with TP_Node_2D, TP_Edge_2D, and TP_Node_3D, TP_Edge_3D, and TP_Face_3D. The class SurveyPoint is associated with SurveyDocument, as a specialisation of SourceDocument.

2.2.2 Land Administration Domain Model

Recently, the title Land Administration Domain Model (LADM) has been used instead of the Core Cadastral Domain Model [Groothedde et al., 2008] [Chapter 9], and [ISO/TC211, 2008]. The word "Cadastre" raises some semantic issues, because its meaning is perceived differently by (international) professionals in the field of land administration and registration, hence the name Land Administration Domain Model. Differences between the CCDM and the LADM (see "Appendix B: Overview LADM/CCDM/STDM Class", Figure 74) can be found in the green package on Persons, and in the purple package. The LADM purple package on geometric and topological classes is more in line with the standard ISO19107 "Geographic information — Spatial schema" [ISO/TC211, 2003b] [Chapter 6 & Figure 39].

In the LADM, the Parcel possesses an attribute spatialDescription, which is of type SpatialRepresentation. The class SpatialRepresentation serves as a list of possibilities, with two options, SpatialRep3D (type TP_Solid) and SpatialRep2D (type TP_Face). Similar to the CCDM, but not identical, the class SurveyPoint has a relation to TP_Solid, TP_Face, TP_Edge, TP_Node, through their generalisation TP_Primitive.

2.2.3 Social Tenure Domain Model

A draft version of the Social Tenure Domain Model (STDM) has been described by Lemmen and Augustinus [Augustinus et al., 2006, Lemmen et al., 2007]. The STDM is a specialisation of the LADM, to describe and specify to what extend the CCDM/LADM is suitable for customary tenure and informal settlement tenure. The necessary changes are described in the STDM, differences can be found with regard to the naming of classes and the absence of certain classes (see "Appendix B: Overview LADM/CCDM/STDM Class", Figure 74).

2.3 Survey Package

Figure 1 and the figures in "Appendix A: LADM UML Class Diagrams" are using the colours for LADM packages, as discussed in section 2.2.1. Additional description of all elements can be found as described by Ingvarsson [Ingvarsson, 2005, ISO/TC211, 2008]. The following classes from the LADM 'Survey Package' are relevant to the master thesis project, as well as LADM classes, related to the Survey Package:

- Parcel
- SurveyPoint
- SourceDocument and SurveyDocument
- LegalSpaceBuilding

Besides theses classes, Figure 20 will show that non-LADM classes are being considered in the prototype (e.g. CadastralOffice, CadastralMunicipality, CadastralSection, and SurveyProject). See the remarks in section 5.4 on how the selected classes have been used in the prototype.

2.3.1 Parcel

One of the core classes of the LADM is the class **Parcel** which currently has the attributes **computedSize**, **dimension** (derived attribute), **spatialDescription**, and **urban** (Figure 2). Attribute **computedSize** involves classes Measure, MeasureType, UnitOfMeasure, allowing the storage of many different types of measurement. **Parcel** is a specialisation of *VersionedObject*, with attributes beginValidityVersion and endValidityVersion.

The class Parcel contains a number of constraints related to attributes **dimension** and **spatialDescription**. One of them being that the dimension of spatialDescription must provide the value of the derived attribute **dimension** (2 or 3). The class **spatialDescription** refers to the topology classes as presented in Figure 71.

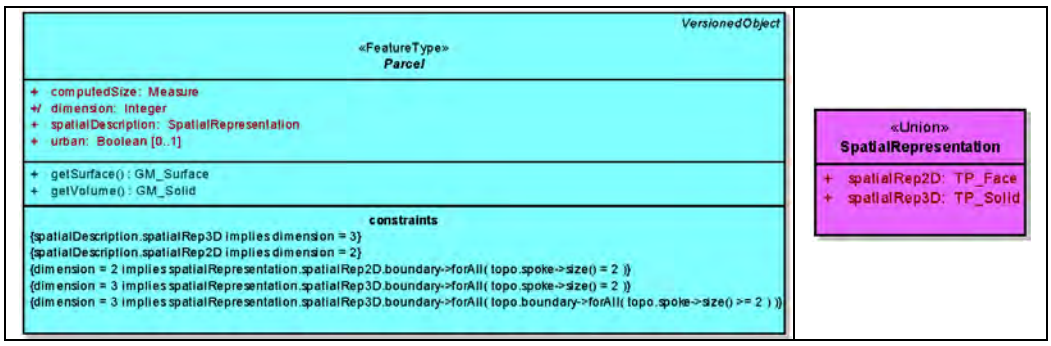


Figure 2 – Parcel & SpatialRepresentation (adapted from [ISO/TC211, 2008], fig.4)

2.3.2 SurveyPoint

An important class of the LADM 'Survey Package' is the **SurveyPoint**, with attributes **dimension** (derived), **locationOrig**, **locationTransf**, **pointType**, **quality**, and **transformation** (Figure 3).

The attribute transformationParams (CharacterString) in CCDM has been replaced in LADM by attribute **transformation**, of type *CC_Operation*, which is based on ISO19111 on spatial referencing by coordinates [ISO/TC211, 2007]. This **transformation** will hold information on original coordinates in a local spatial reference system to transformed coordinates in the target system. CC_Operation is capable of describing all the elements related to the point transformation from one coordinate reference system, another in a structured manner, as opposed to storing the parameters as text (see section 5.2.1 and 5.2.2 on processing and network adjustment of measurements).

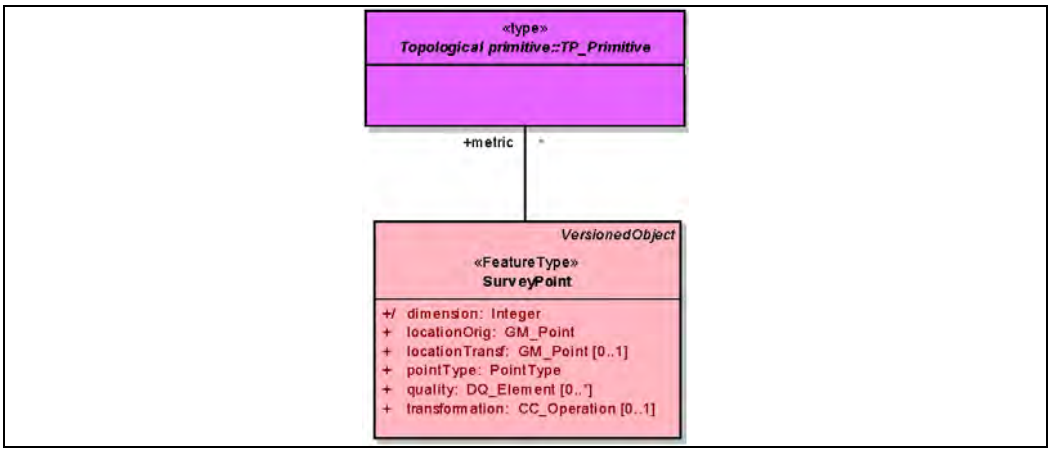


Figure 3 – SurveyPoint & TP_Primitive (adapted from [ISO/TC211, 2008], fig.4)

The type of the attribute **quality** (CodeList) in CCDM has changed in LADM into *DQ_Element*, which is based on ISO19115 on Metadata [ISO/TC211, 2003a]. *DQ_Element* is a generalisation of DQ_Completeness, DQ_LogicalConsistency, DQ_ThematicAccuracy, DQ_TemporalAccuracy, and DQ_PositionalAccuracy. For each **SurveyPoint**, zero up to multiple instances of **quality** can be recorded ([0..*]). The attribute PointCode (CodeList) in CCDM has in LADM been changed into attribute **pointType**, which refers through its type to a *PointType* (CodeList with attributes/values like endPointArc, midPointArc, pointStraightLine). Attributes **locationOrig** and **locationTransf** (optional through lower/upper bound specification [0..1]) represent the coordinate of the **SurveyPoint**, before and after **transformation**.

An association exists between **SurveyPoint** and **TP_Primitive**, which is specialised by TP_Solid, TP_Face, TP_Edge, or TP_Node, building up the **Parcel** topology.

2.3.3 SourceDocument and SurveyDocument

The abstract class **SourceDocument** has the attributes **acceptance**, **submission**, and **registration** of type *DateTime*, as well as an **electrSignature** of type *Binary* (Figure 4). The attributes tmin and tmax from **SourceDocument** in CDDM have not been maintained in LADM; however the dependent class SurveyPoint is specialisation of VersionedObject, as discussed earlier. **SourceDocument** is specialised by **SurveyDocument**, described by the attributes **measurements**, **number**, **quality**, **surveyDate**, and **type**.

The attribute **measurements** of type *Record*, contains "*files with terrestrial observations - distances, bearings, and referred geodetic control - on points*" [Van Oosterom et al., 2006]. The attribute **quality** is, unlike quality in SurveyPoint, referring to the type *CodeList*. It is presumed that the attribute **quality** should refer to a class with stereotype *CodeList*, similar to the attribute **type**, which is referring to class *SurveyDocumentType* (CodeList with values/attributes: fieldSketch, gnssSurvey, relativeMeasurement). Note that LegalDocument is not included in the Survey Package, but is included in the prototype for demonstration of handling super classes and their specialisations.

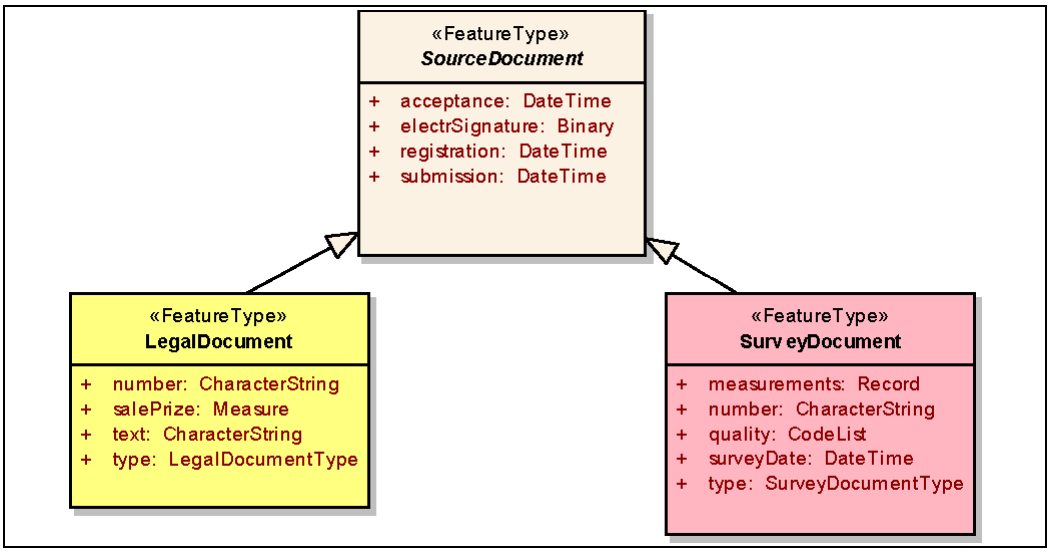


Figure 4 – SourceDocument & SurveyDocument (adapted from [ISO/TC211, 2008], fig.4)

2.3.4 LegalSpaceBuilding

The class **LegalSpaceBuilding** (Figure 5) is involved in the prototype, and contains the attributes **complNum**, **dimension**, and **extAddressId**, similar to *Building* in CCDM. Based on the attribute *Parcel.dimension*, the attribute *LegalSpaceBuilding.dimension* is assumed to be a derived value as well; *attribute_3* appears to be a mistake. **LegalSpaceBuilding** has no attribute *spatialDescription*, but is spatially described by an association to **SurveyPoint**.

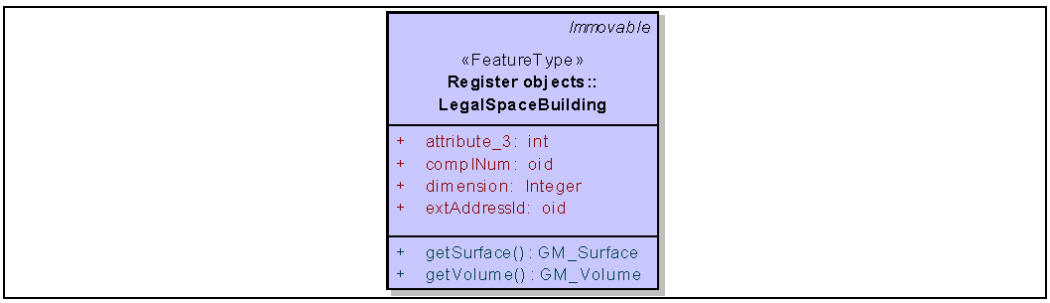


Figure 5 – LegalSpaceBuilding (adapted from [ISO/TC211, 2008], fig.3)

2.4 Extension of LADM 'Survey Package'

One of the original objectives of the master thesis was to extend and improve the Land Administration Domain Model 'Survey Package' based on literature research. Some relevant publications will be discussed here.

In his thesis, Lee describes a survey record management system (SRMS), a cadastral survey system to support a Land Information System [Lee, 2005]. As part of the SRMS, the subsystems Measurements, Computations, Survey Record Set, GIS Coordinates, Survey Points and Delivering System are identified, in recognition of the importance of an implementation of a flexible system, capable of data exchange between different organisations (with different data models). The need for managing and storing the survey measurements in well suited structures, as well as the manner in which they are used in the (digital) map is deemed important in relation to establishing the quality of the cadastral map, defined as the differences between the cadastral records and real situation (see Chapter 5 and 7 for a continuation on the subject of quality of the digital cadastral map). Lee describes a survey observation model, listing a number of survey observation types such as Angle, Direction, Length, Coordinate, Distance, Curve, and Height, see Figure 6.

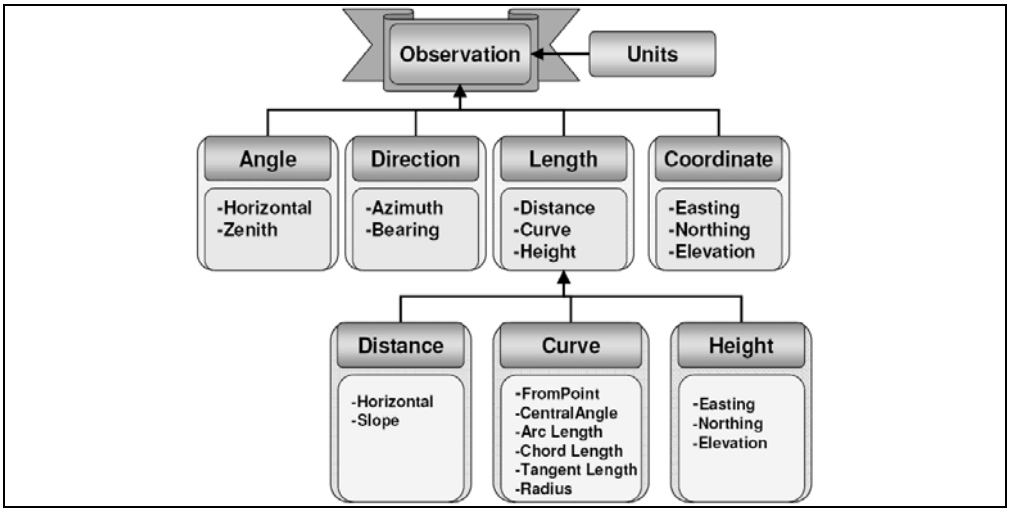


Figure 6 - Survey Observation Types (taken from [Lee, 2005], Figure 5.4)

In one of the "best-practises" documents of OGC, called "Observations and Measurements model (O&M)", a model for observations and associated components is described [Open Geospatial Consortium, 2006b]. As part of OGC's Web Enablement activities (SWE), a number of components are being defined, one of them being the Observations and Measurements model (O&M). The O&M is a conceptual model for observations and measurements (in UML), with the goal of providing a common ontology for sensor and observation systems. *"The key idea is that the observation result is an estimate of the value of some property of the feature of interest, and the other observation properties provide context or metadata to support evaluation, interpretation and use of the result"* [Open Geospatial Consortium, 2006b]. The **feature of interest** specifies the object upon which the **observation** was made, resulting in an estimate of the *value* of a *property* of the **feature of interest**. The class **measurement** refers to an **observation** whose result is a measure (a specialisation of observation, see Figure 7). Many of the classes and

UML types in the model are in line and based on ISO standards (see Figure 7 and Figure 8), for example:

- Persons/CI_Responsible; party responsible for the observation (ISO 19115),
- FeatureOfInterest/Feature; a representation of the real-world object regarding which the observation is made (ISO 19109, ISO 19101),
- Coverage/CV_Coverage; the spatio-temporal extent of the feature (ISO 19123), and
- Record; the result of the observation (ISO19103), the latter can be found in the LADM as well in class SurveyDocument

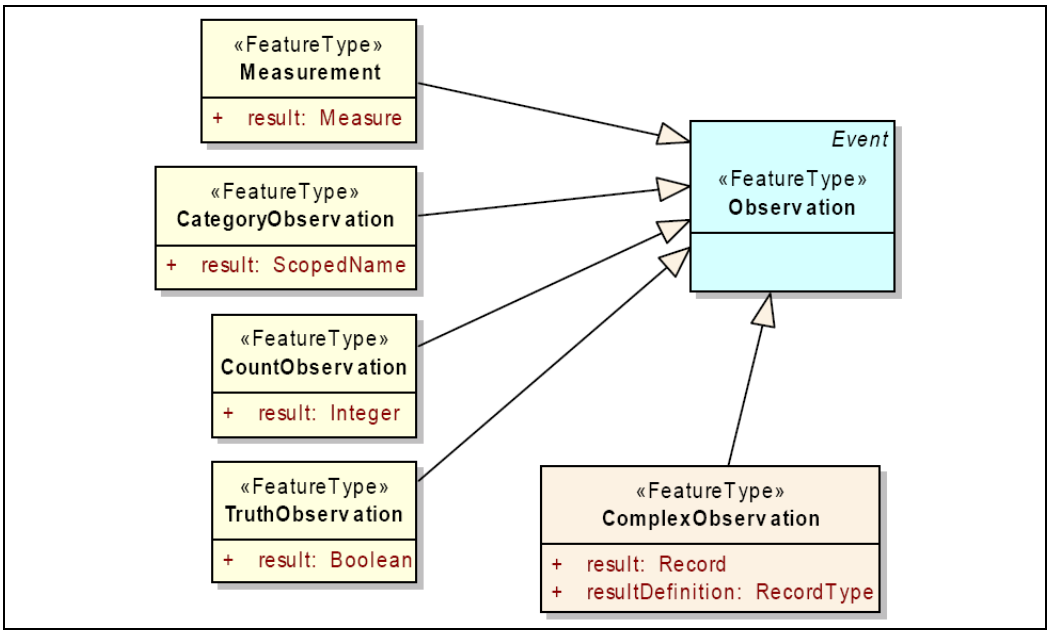


Figure 7 - Specialisations of Observation (taken from [Open Geospatial Consortium, 2006b], Figure 2)

Three viewpoints are described (Observation, Coverage, and Feature), which could be associated with the different phases of the data collection and processing cycles. The *Observation* viewpoint focuses on data collection, leading to a description of the feature of interest. The *Coverage* viewpoint focuses on the distribution and variation of a property within the spatio-temporal domain of interest, and the *Feature* viewpoint has an object-centric approach, in trying to identify discrete objects, based on the observations. These viewpoints together describe the complete O&M model, and dependent on the chosen viewpoint, certain selections of classes and UML types, simplifications of specialised classes, or extensions of associations and attributes in the classes provided, can be made. An XML/GML implementation of the model is provided in SFA-SQL [Open Geospatial Consortium, 2006b] [ANNEX D].

As expressed in OGCs "Observations and Measurements model (O&M)", such a specialisation of the Observation and Measurements model for a specific application domain, such as the LADM 'Survey Package', would involve careful consideration of the correct identification of the **feature of interest**, and of the compatibility between the properties of the feature of interest and the result of observations (see section 5.2.1). Modelling the **feature of interest** is left to the reader of "Observations and Measurements model (O&M)", the closest approach to the concept of **feature of interest** is provided in Figure 8, by **AnyIdentifyableFeature**.

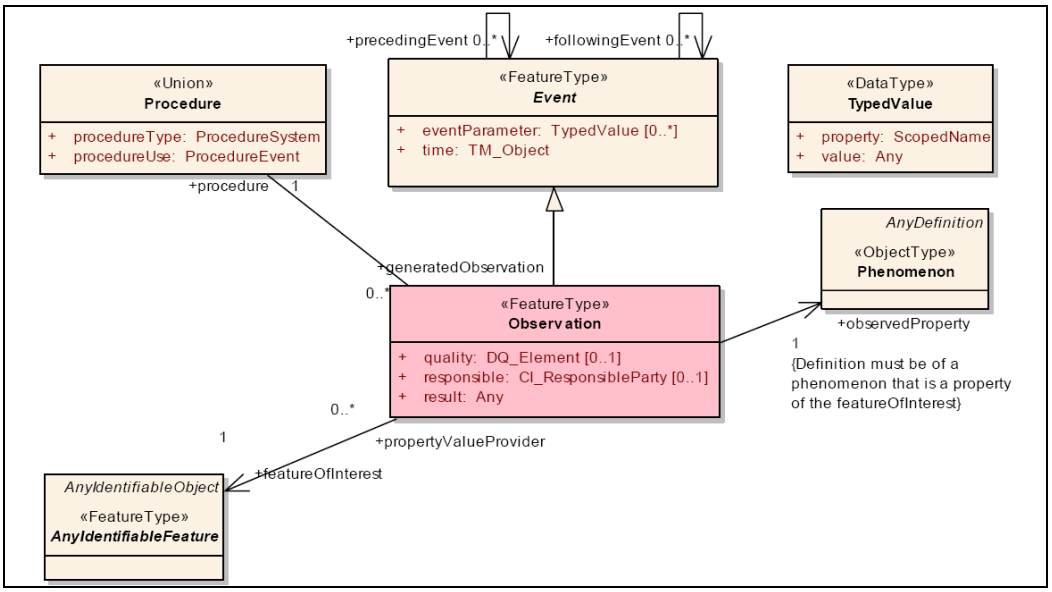


Figure 8 - Event and Observation types (taken from [Open Geospatial Consortium, 2006b], Figure 1)

During the first stages of the master thesis project, it was concluded that the objective of extending and improving the Land Administration Domain Model 'Survey Package' could probably not be reached within the duration and available resources for the master thesis project, and has been assigned a lower priority than some other activities, like experimenting with the MDA principles (see section 8.1). Initial research as described in this section, has shown that quite a few publications can be found, that have a similar goal of improving the modelling of the survey process and data, which can be used in future research.

2.5 Conclusion

The Land Administration Domain Model (LADM) is described with a UML class diagram. The core elements of the Land Administration Domain Model are Registered Objects, Persons, and the RRR (Right, Responsibility or Restriction) to this Registered Object, which these Persons are involved in. The Land Administration Domain Model has been under development for a few years by many participants, and quite a few variants and specialisations have been produced, even during the duration of the master thesis project. This has lead to an ISO TC211 standard 19152, which is currently under development [ISO/TC211, 2008].

The Land Administration Domain Model consists of a number of packages concerned with legal, administrative, person, immovable, survey, geometry and topology related information. Some minor improvements in the LADM 'Survey Package' classes have been reported, and quite a few publications are available with the goal of improving the modelling of the survey process and data, which could be the basis for further research on the LADM (for example: [Ingvarsson, 2005, Lee, 2005, Open Geospatial Consortium, 2006b]).

One of the elements mentioned in the goals for the Land Administration Domain Model, is its function as a basis for land administration system development executed on Model Driven Architecture principles. As a basis for experimenting with Model Driven Architecture in the master thesis project, classes of the LADM have been discussed: Parcel, SurveyPoint, SourceDocument, SurveyDocument, LegalSpaceBuilding. Section 5.4 will address the final composition of classes in the Adapted LADM 'Survey Package', as input to prototyping activities in the master thesis project.

3 Model Driven Architecture

3.1 Introduction

The Object Management Group (OMG, URL 6) has established standards for Model Driven Architecture (MDA), a software design methodology to create model based specifications and model based generation of information systems [OMG, 2003]. The basic idea of MDA is that the specification of an information system (a model) is model driven, and separated from the way in which the information system uses the specific possibilities and characteristics of the platform on which it is implemented. The definitions of model and platform are provided by the OMG as:

"A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text."

"A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented."

MDA aims at platform independent specification, resulting in a (semi) automatic (re-) generation of platform specific implementation code. In the professional field of software and database development projects it is commonly known that performing changes later in the development process is more costly than implementing changes in the beginning [Van Bennekom-Minnema, 2007]. However, quite often projects do not start with a detailed specification which will be fixed over the duration of the project, and are inclined to put focus on the platform specific part of the "development street". In many cases the initially available system specification needs to be further detailed, and is susceptible to changing requirements during the execution of software and database development projects. The concept of MDA addresses this need and destiny for change, by putting the focus on (platform independent) business and user needs, while the (not less crucial platform specific) technical aspects are handled by (semi-) automated MDA transformation tools, sometimes also referred to as Model Driven Generation tools.

3.2 MDA Viewpoints and Models

The MDA standard [OMG, 2003] describes three viewpoints on the representation of the information system. *A viewpoint on a system is defined as a level of abstraction (or level of suppressing details) when specifying that system, in order to focus on particular concerns within that system* [OMG, 2003]. Three viewpoints and accompanying models are defined:

- Computation Independent Viewpoint (CIV) and Model (CIM)
- Platform Independent Viewpoint (PIV) and Model (PIM)
- Platform Specific Viewpoint (PSV) and Model (PSM)

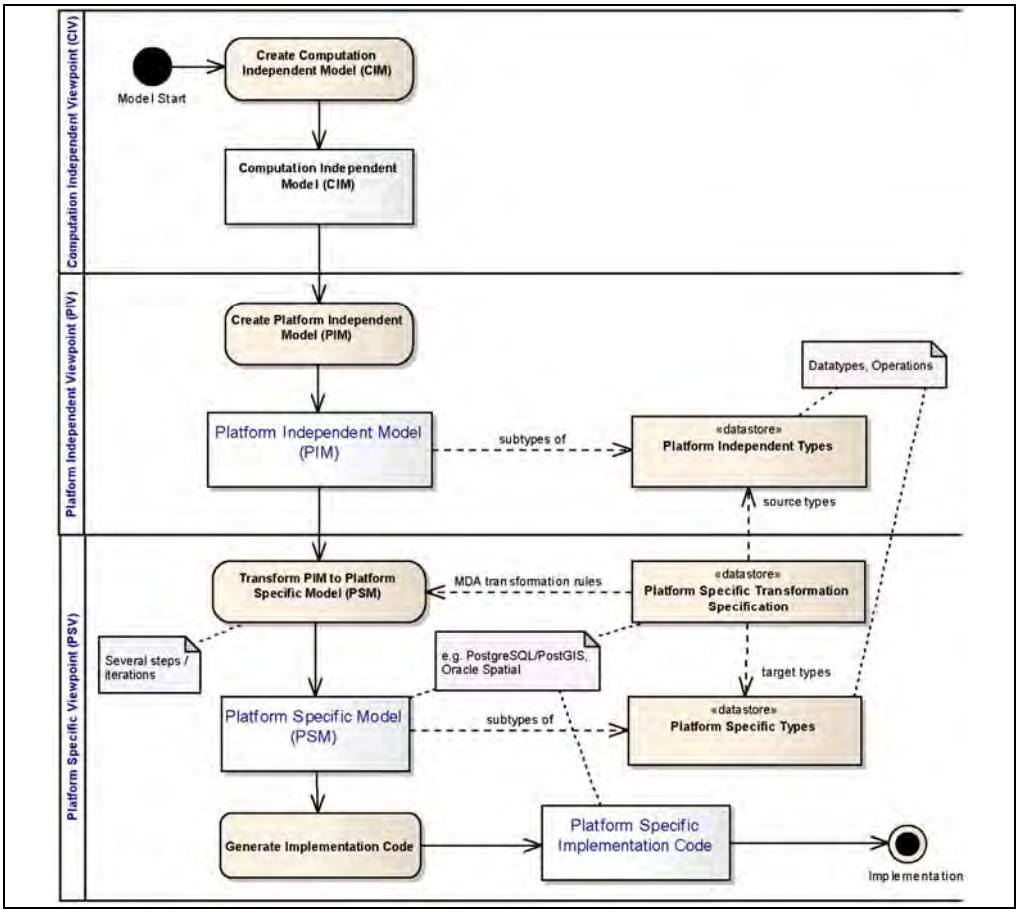


Figure 9 - MDA Elements and Processes, drawn up from the MDA Guide [OMG, 2003]

The specification of an information system will go through these viewpoints, which in as sense can be seen as information system development phases, from a Computation Independent to a Platform Independent to a Platform Specific viewpoint. In each phase (viewpoint), different models are used, adding more details to the previous model. This level of abstraction and the models, used in each of these viewpoints, will be described in the next section.

The Computation Independent Viewpoint (CIV) focuses on the system requirements, where system structure and environment are hidden. The details of the structure and

processing of the system are hidden or as yet undetermined. The Platform Independent Viewpoint (PIV) focuses on details on the operation of the system, where platform-dependent details are undetermined, as in the LADM. A PIV may use platform-independent modelling language such as UML [OMG, 2007a, OMG, 2007b]. The Platform Specific Viewpoint (PSV) focuses on details with regard to the implementation of the model on a certain platform. The MDA viewpoints are depicted in Figure 9 as "swimming lanes" in the context of which, models and transformation activities are visualised.

Each of the viewpoints addresses a specific model type: respectively, the Computation Independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM), see the OMG definition of a model in the first section of this chapter.

The CIM shows the initial model of the system, and consists of text and drawings, describing the requirements of the system. The CIM does not go into the details of the structure or the platform specifics of the information system, but serves as a source for shared vocabulary [OMG, 2003]. In the master thesis project the CIM will not be considered.

The CIM will be the basis for another basic element of MDA, the platform-independent model (PIM). A PIM, such as the LADM, described in section 2.2.2, specifies an application's data and functionality based on platform independent types for data and operations. The PIM is independent of the intended implementation technology, which makes the PIM relatively stable in environments where technology is continuously updated and improved. With the PIM as input, for each platform/environment, one or more platform-specific models (PSM) are defined, based on platform specific types of data and operations. The platform specific transformation specification (a set of MDA transformation rules), maps platform independent to specific types of data and operations. The platform specific transformation specification is used in the transformation process to transform marked PIM elements to PSM elements (Figure 9). The process from the PIM to the final PSM can be done in several steps, e.g. the prototype will demonstrate a transformation from the PIM to a first version of the PSM and then another step to the final version of PIM, see section 6.4.

The PSM will be used to create the actual implementation of the model in the chosen platform/environment, for example an Oracle (URL 13), or a PostgreSQL (URL 14) database, an XML schema for data transfer (URL 17), a Java (URL 16) or a .NET platform (URL 15) for the application's user interface. The transformation of a PIM to a PSM can be done manual, but preferably semi-automatic or automatic. Ideally, the model driven generation process can be repeated (automatically), after changes in the PIM, to result in an updated PSM. For example the changes in the LADM that could be witnessed in the past years could be input to MDA processes.

3.2.1 Object - Relational Contrast

The transformation from an object-oriented UML model (PIM) to an object-relational database model in PostgreSQL/PostGIS (PSM), as envisioned in the master thesis project, requires careful consideration and mapping of elements in both types of

models. This transformation is not straightforward because of the differences of object-oriented and relational models.

On the one hand, the object-oriented UML class diagram (PIM) contains Classes (sometimes stereotyped as <<enumeration>>), defining data (Attributes) of platform independent data types and behaviour (Operations). Objects are instances of classes and they are connected through Relationships, in various types such as: Association, Aggregation, and Composition. A special type of Relationship is Generalisation which supports inheritance and re-use of data and behaviour.

On the other hand, the object-relational database model in PostgreSQL/PostGIS (PSM) contains Tables, defining data (Columns) of platform specific data types. Tables are connected through foreign key relationships, ensuring referential integrity of the data(base). Other constraints are primary and unique key constraints, and the base table check constraints.

In performing the transformation, the platform specific transformation specification, with its MDA Transformation Rules, defines how the elements in the PIM should be converted to one or more elements in the PSM (i.e. a set of MDA transformation rules).

3.3 Standards Relevant to MDA

A number of standards are related to the objectives of the master thesis project. Some of the standards are related to Model Driven Architecture (MDA), which all in their respective manner enable portability and interoperability of the models and data. Other standards are dealing specifically with the definition of geographic information.

3.3.1 ISO19107 Standard: Spatial schema

The ISO19107 Standard "*Geographic Information - Spatial schema*" specifies conceptual schemas for describing the spatial characteristics of geographic features. For example GM_Point, GM_LineString, GM_Polygon, GM_MultiSurface to define geometric objects, used in the master thesis (Figure 10), at a *platform independent* level (i.e. PIM). ISO19107 defines a set of spatial operations consistent with these schemas, as well as the Topology packages and elements. A topological model describes the relation between the topological elements like node, edge, and face (see TP_Node, TP_Edge, and TP_Face in ISO19107 [ISO/TC211, 2003b],[Chapter 7]), based on their unique identifiers. Figure 71 shows the use that LADM makes of these topological elements.

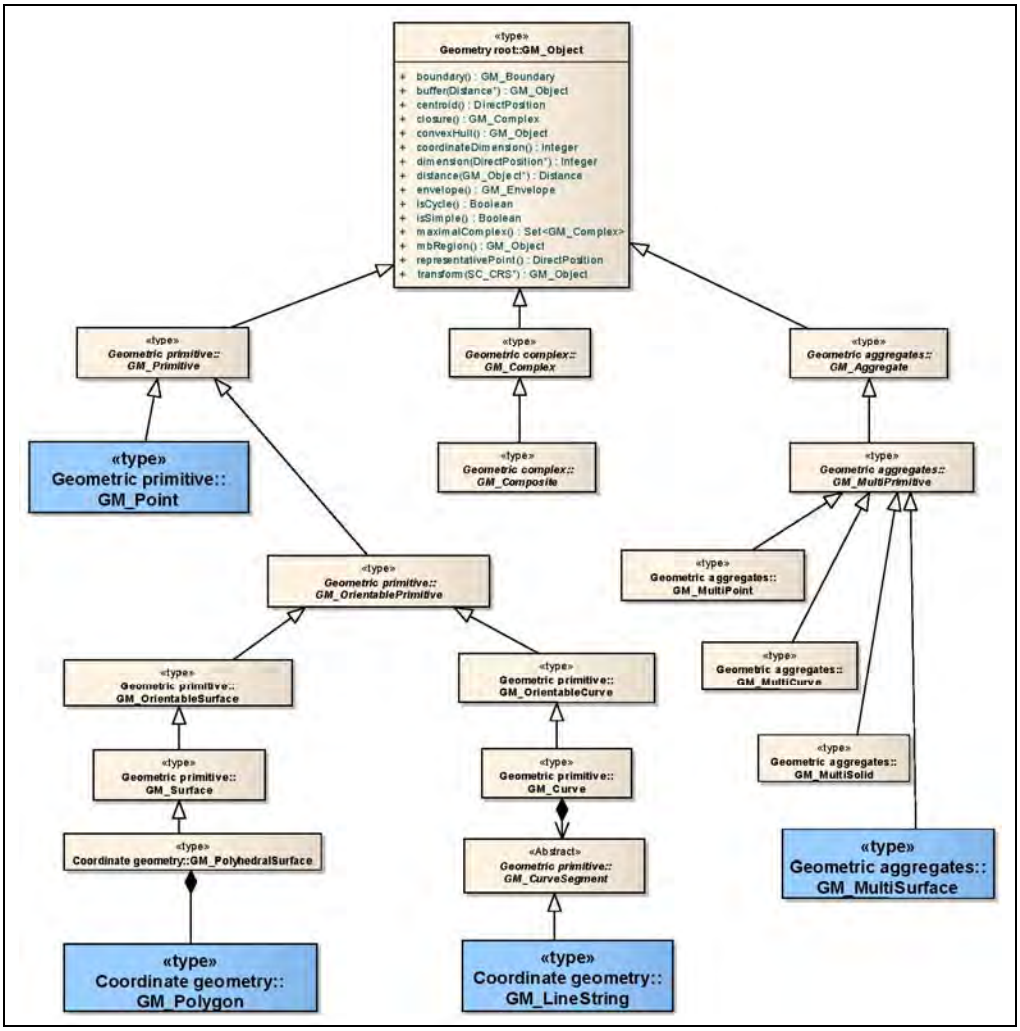


Figure 10 - ISO19107 geometry basic classes (adapted from [ISO/TC211, 2003b], Figure 5)

3.3.2 ISO/IEC 13249-3 SQL/MM - Part 3: Spatial

The ISO/IEC 13249-3 Standard "*Information technology - Database languages - SQL multimedia and application packages - Part 3: Spatial*" standard defines spatial user-defined types and their associated routines, at an implementation or *platform specific* level (i.e. PSM). It addresses the need to store, manage and retrieve information based on aspects of spatial data such as geometry, location and topology [ISO/IEC, 2006], as an extension of the SQL language [ISO/IEC, 2003]. The standard uses the prefix ST (Spatial Temporal) for all its elements and Part 3 intents to standardize extensions for multi-media and application-specific packages in SQL with regard to spatial data. The spatial methods, some of which have been used in the prototype, are addressing data exchange, retrieve properties of geometric data elements and their geometric relations. For example the formats Well Known Text (WKT), Well Known Binary (WKB) and Geography Mark-up Language (GML, section 3.3.8), the functions ST_IsEmpty, ST_Area, ST_Length for spatial element properties, and ST_Intersects, ST_Within for geometric relations. The prototype target platform (relational database PostgreSQL/PostGIS) provides these spatial *operations*, see example in Figure 11.

PostgreSQL/PostGIS statement:	`select code, ST_AsText(polygon) from parcel where oid = 368475`
Result:	`HTN04K 742G0000, POLYGON((143782.51306 443333.15,143782.08802 443332.812975,143784.92806 443329.221075,143787.77408 443325.313925,143789.41099 443323.243025,143799.97006 443331.589025,143792.35499 443340.928875,143782.75893 443333.344025,143782.51306 443333.15))`
PostgreSQL/PostGIS statement:	`select code, ST_AsGML(polygon) from parcel where oid = 368475`

Result:	`HTN04K 742G0000,`
	`<gml:Polygon srsName="EPSG:28992">`
	`<gml:outerBoundaryIs>`
	`<gml:LinearRing>`
	`<gml:coordinates>`
	`143782.51306,443333.15`
	`143782.08802,443332.812975`
	`143784.92806,443329.221075`
	`143787.77408,443325.313925`
	`143789.41099,443323.243025`
	`143799.97006,443331.589025`
	`143792.35499,443340.928875`
	`143782.75893,443333.344025`
	`143782.51306,443333.15`
	`</gml:coordinates>`
	`</gml:LinearRing>`
	`</gml:outerBoundaryIs>`
	`</gml:Polygon>`

Figure 11 - Examples of ISO/IEC 13249 SQL/MM - Part 3 methods

3.3.3 Unified Modelling Language (UML)

The Unified Modelling Language (UML, URL 11) is a standardized specification language for object modelling that includes a graphical notation used to create an conceptual model of a system [OMG, 2007a, OMG, 2007b]. In UML, different types of diagrams exist, such as Use Case, Class, Activity, Component, and State Chart diagrams. In the master thesis project only the class diagrams are used to describe the LADM SP elements, such as classes, attributes, associations, operations and constraints ("Appendix A: LADM UML Class Diagrams").

Tagged Values

The UML model (e.g. classes, attributes, associations) can be extended with stereotypes, tagged values and other string-based extensions [OMG, 2007b] [section 18.1.2] and [OMG, 2006a] [section 11]. A combination of those extensions can be stored in a so-called UML Profile. In the prototype, tagged values are used to support various transformation functions, which have a relation to the concepts of *marks* mentioned in the MDA Guide. A *mark* is applied to an element of the PIM, to indicate how that element is to be transformed. [OMG, 2003], and will also be referred to as a *MDA Transformation Rule*. Examples of tagged values, used in the MDA prototype can be found in 6.5.1.

3.3.4 Extensible Mark-up Language (XML)

The Extensible Mark-up Language (XML) is a general-purpose mark-up language defined by the World Wide Web Consortium for creating special-purpose mark-up

languages. With XML many different kinds of data can be described, and XML facilitates interoperability and exchange of structured data. XML Schema is used to describe the structure of the XML documents, as well as constraining its contents in XML Schema Definition Language (XSD, URL 17).

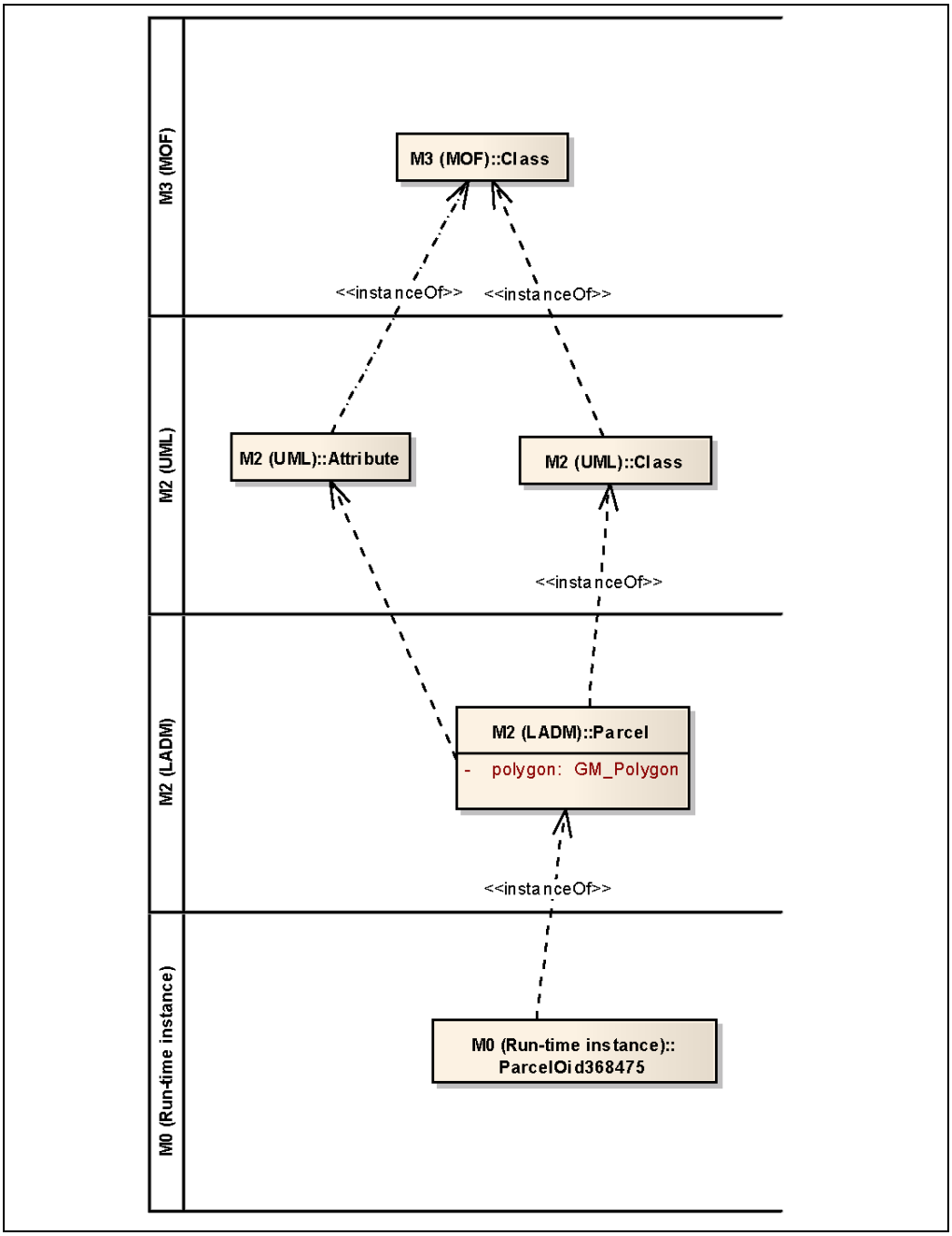


Figure 12 - Example of MOF levels (adapted from [OMG, 2007a], Figure 7.8)

3.3.5 Meta Object Facility (MOF)

Meta Object Facility (MOF) has been created to "*enable development and interoperability of model and metadata driven systems*" [OMG, 2006a] [pp. 5], to address the ability to exchange models between various modelling tools. An example of the MOF layers has been provided in Figure 12. The MOF architecture is described

in a meta-meta model (M3 layer), which describe meta models (M2 layer), such as for example the UML and OCL meta model. The actual models written in UML, e.g. the LADM class diagram, are on the M1 layer, and the real world is described in the M0 layer or data layer. A related standard, also defined by the OMG, is the Queries/Views/Transformations (QVT). QVT is a standard for model transformation, in conformance to MOF version 2.0 metamodel definitions. To "lower the barrier to entry for model driven tool development and tool integration" [OMG, 2005], the Essential MOF has been defined, representing a subset of Complete MOF (CMOF), because many meta models do not need the all of the extensive CMOF elements.

3.3.6 XML Metadata Interchange (XMI)

XML Metadata Interchange (XMI) is an interchange format defined by OMG for Meta-Object Facility (MOF) models on the M3-, M2-, or M1-Layer (section 3.3.5). XMI is based on Extensible Mark-up Language (XML) and the current version of XMI is version 2.1 [OMG, 2005]. Figure 13 shows a part of an XMI file, generated by Enterprise Architect, showing the packagedElement of type uml:Package "Survey Package", the packagedElement of type uml:Class "Building", with ownedAttribute of type uml:Property "polygon". The latter refers to id "EAID_8FF24017_E126_4fcb_9086_20E60069B524", which is an uml:Class "GM_Polygon", which defines the data type for attribute "polygon", and is specified elsewhere in the XMI file.

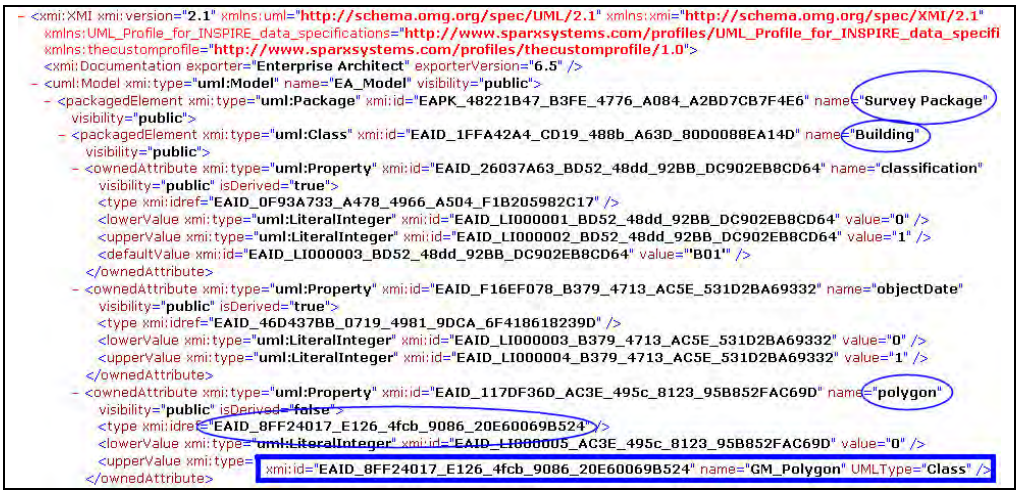


Figure 13 - Example of XMI file generated byEnterprise Architect

3.3.7 Object Constraint Language (OCL)

The Object Constraint Language (OCL), addresses the "*need to describe additional constraints about the objects in the model*" [OMG, 2006b]. OCL is a formal language for platform independently describing constraints and object query expressions, for example in UML models. OCL is based on MOF meta models, and can therefore be part of the model transformations discussed in 3.2. The current OCL version 2.0 is compliant with UML version 2 and MOF version 2 [OMG, 2006a, OMG, 2005, OMG, 2007a, OMG, 2007b], see also Chapter 4, for a continuation of the topic of "Constraints in Data Modelling". Essential OCL is the "*minimal OCL required to work with EMOF*" [OMG, 2006b], [Figure 2.16]. Essential OCL is motivated by the

same considerations as EMOF, namely, providing a simple query and constraint language for simple meta models [Bräuer, 2007] [Chapter 2].

3.3.8 Geography Mark-up Language (GML)

The Geography Mark-up Language (GML) is an XML application defined by the Open Geospatial Consortium (OGC, URL 5). GML describes and models geographical information platform independently, and is also used as an open interchange format for geographic information on the Internet [Open Geospatial Consortium, 2002]. The GML has been adopted in the standard ISO 19136 [ISO/TC211, 2006], see Figure 11, for an example of a GML fragment, generated by a PostgreSQL/PostGIS function.

3.3.9 Simple Features Profile for GML

Similar to the arguments for creating EMOF and Essential OCL, the "Geography Mark-up Language (GML) Simple Features Profile" identifies a *restricted but useful* subset of the GML/XML schema to facilitate a more easy implementation of the GML standard [Open Geospatial Consortium, 2006a]. The Simple Features Profile for GML have been aligned with the Simple Feature Access for SQL (see next section). The elements Point, Curve (LineString), Surface (Polygon), Geometry, MultiPoint, MultiCurve, MultiSurface, and MultiGeometry are part of this subset.

3.3.10 Simple Feature Access for SQL (SFA-SQL)

The Simple Feature Access for SQL (SFA-SQL), as described in the "OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option", specifies a platform specific standard for manipulation of geographic features with simple geometry [Open Geospatial Consortium, 2006c], also described in ISO19125, based on ISO's SQL/MM multimedia and application packages, part 3: Spatial [ISO/IEC, 2006].

SFA-SQL refers to the geometry types Point, Curve, Linestring, Surface, Polygon, PolyhedralSurface, GeomCollection, Multipoint, Multicurve, Multilinestring, Multisurface, and Multipolygon. Geometry type related functions are X(), Y(), Z(), M() for geometry type Point. Length(), StartPoint(), EndPoint(), IsClosed(), IsRing() for geometry type Curve, NumPoints(), and PointN() for geometry type Linestring, and Centroid, PointOnSurface and Area for geometry type Surface.

For testing spatial relationships, SFA-SQL supports the routines: Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, Overlaps and Relate. Other routines, applicable to all geometry types are WKTToSQL, WKBToSQL, Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, and Envelope, as well as Distance. Also applicable for all geometry types, SQL-SFA defines for constructive operations on geometry types: Intersection, Difference, Union, SymDifference, Buffer, and ConvexHull.

3.4 Conclusion

Model Driven Architecture (MDA) is a software design methodology to create model based specifications and model based generation of information systems to different platforms. Especially in situations where specifications may change during or after the information system development project, model driven based development and generation offers advantages. The MDA methodology addresses models, used in different phases, increasingly gaining in level of detail. A platform independent model (PIM), such as the LADM 'Survey Package' class diagram, contains platform independent details on application's data (classes, attributes, data types, associations) and functionality (operations). Based on a transformation pattern of MDA transformation rules (also referred to as the platform specific transformation specification), the PIM will be converted semi-automatically, but preferably automatically into a platform specific model (PSM), adding platform specific detail to the model. MDA principles allow for repetition of these activities, enabling changes in the PIM to be propagated to the PSM. The prototype (Chapter 6 and 7) is focussed at the target platform: the object-relational DBMS PostgreSQL (with extension PostGIS).

The transformation, from an object-oriented PIM to a relational database model in the PSM, requires a mapping of object-oriented to relational data types and operations, used in these types of models. The classes, attributes and operations on the PIM are defined by data types and operations at an abstract level. For example, the ISO 19107 standard provides spatial data types and operations for the geometry and topology of spatial class diagram elements. The spatial elements of the PSM can be defined by platform specific standards or (not-preferable) by platform supplier proprietary element specifications. The ISO/IEC 13249 SQL/MM standard part 3 defines spatial user-defined types and their associated routines, at an platform specific level, i.e. SQL [ISO/IEC, 2003, ISO/IEC, 2006]. Other standards, relevant for MDA are Meta Object Facility (MOF); facilitating model exchange, Geography Mark-up Language (GML), facilitating geographic information exchange, and Object Constraint Language (OCL) for specification of constraints. Many of these standards now have been 'extended' with simplifications, to lower the threshold for using these standards, thus increasing the common acceptance and use.

4 Constraints in Data Modelling

4.1 Introduction

Constraints in information systems can be defined and implemented in many different ways, where constraints are not always an integral part of the system. This will allow the constraint validation to be avoided and bypassed, which negatively affects data integrity. The constraints are often defined at the implementation phase, making communication about constraints quite complex and cumbersome, as well as the maintenance of constraints in these situations. Constraints described in a natural language, although seemingly understandable for layman, have the disadvantage that they can very easily lead to ambiguities in semantics, and in implementation. The resolution of the problems, arising from these situations, requires a formal description of constraints at the beginning of the development lifecycle.

UML, an Object Management Group (OMG) standard for modelling information systems [OMG, 2007a, OMG, 2007b], although powerful in describing and visualizing systems from a Platform Independent Viewpoint, is not capable of visually modelling every kind of constraints through its elements (e.g. classes, attributes, and associations) and their properties. Note that UML does provide the possibility to model constraints in natural language, involving navigation through multiple classes and associations, and using a variety of expressions and conditional statements, which could potentially lead to afore mentioned ambiguities.

The Object Constraint Language [OMG, 2006b] is a formal language, which has been defined as an extension to UML. OCL enables the specification of those constraints, which cannot be recorded in UML, in an unambiguous manner. In the definition of OCL, one of the goals was to keep OCL easy to read and write, resembling the English natural language. OCL is a descriptive language, which does not change anything to the model, nor does it specify the action to be taken when the constraint is violated.

The formal nature of the OCL, enables the automised parsing, processing and implementation of OCL constraints, referring to classes, attributes, associations, and operations [OMG, 2006b]. OCL can also be used as a query language, and it that sense it has commonalities with concepts of relational queries, where data collection can be constructed, starting from a certain context (e.g. a class), based on navigation along the elements of a class diagram (other classes and associations).

Two simple examples of an OCL invariant, applicable to one or two attributes, are provided below. A constraint with regard to the format of the attribute value (e.g. the name should be in uppercase):

```
context CadastralMunicipality
inv nameUppercase: self.name = self.name.toUpper()
```

Or a so-called tuple rule, for example requiring the startDate to be before the endDate:

```
context SurveyProject
inv startDateBeforeEndDate: self.startDate < self.endDate
```

See section 6.7 and "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)" for examples of OCL constraints.

4.2 Implementation of Constraints

The Object Constraint Language is well defined in a standard, but how could an OCL constraint be implemented? The implementation of a constraint in the database will be discussed by using one example of the OCL constraints in this "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)", in section "Constraints Applicable to Multiple Instances of Multiple Classes", page 177. Consider the example for a "Relationship Cardinality", stating that *the amount of survey points per survey document must be 0 or larger than 2* (Figure 14).

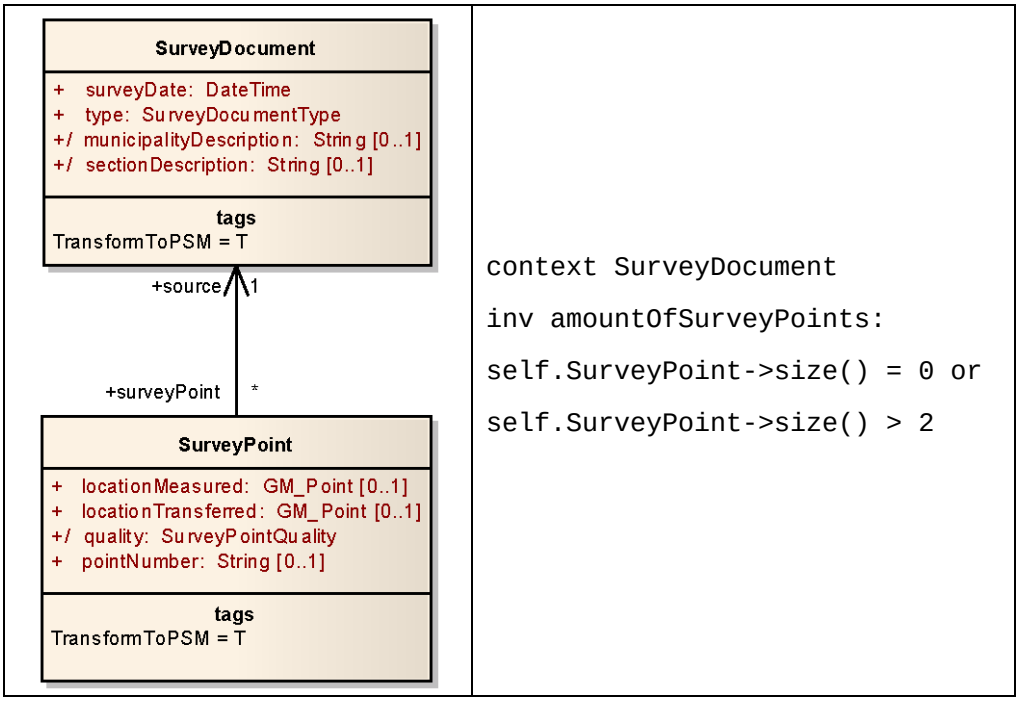


Figure 14 - Example of OCL Constraint

Note that in the UML class diagram, the multiplicity of the SurveyPoint end of the association (between SurveyDocument and SurveyPoint) end is set to '*' (equal to '0..*'). The constraint could be defined in UML like '0,3..*' or '0 or 3..*', but a separate MDA transformation rule for this UML notation must be defined, to implement it in the target platform. In the next section, the constraint is considered in OCL, based on PIM elements, as an example of OCL constraint implementation. After transformation to the PSM, an OCL view (see section 4.3 on 'Constraint Views') with the name: *v_ocl_amount_of_survey_points* could be created, returning the (constraint violating) records:

```
create   view v_ocl_amount_of_survey_points as
select   self.oid, count(spt.source_oid)
from     survey_document self
,        survey_point spt
where    self.oid = spt.source_oid
group by self.oid
having   not (count(spt.source_oid) = 0 or count(spt.source_oid) > 2);
```

The standard for SQL [ISO/IEC, 2003], provides for *general constraints*, a.k.a. **assertions** for implementation of this constraint, and the create statement could be:

```
create assertion amount_of_survey_points check
(not exists (select * from v_ocl_amount_of_survey_points));
```

However, the current RDBMS's do not support assertions (yet), see error message in PostgreSQL when attempting to create the assertion:

```
ERROR: CREATE ASSERTION is not yet implemented
```

The alternative could be to implement **check constraints** for the tables involved [Louwsma et al., 2006, Van Oosterom, 2006], in the example: survey_document and survey_point in PostgreSQL:

```
ALTER TABLE survey_document
ADD CONSTRAINT amount_of_survey_points CHECK
(not exists (select count(spt.source_oid)
             from survey_point spt
             where oid = spt.source_oid
             having   not (count(spt.source_oid) = 0
                      or count(spt.source_oid) > 2)
             )
);
```

These base table check constraints, based on OCL views are also not possible, because no sub select statements may be made in these check constraints, see error message in PostgreSQL when attempting to create the table check constraint:

```
ERROR: cannot use subquery in check constraint
```

In many papers, triggers are mentioned as a method to implement constraints like this, so that the consistency of the database with the specified constraints is guaranteed [Van Oosterom, 2006, Heidenreich et al., 2007, Cockcroft, 1997, Louwsma et al., 2006]. So in our example, a trigger needs to be created that is fired for EACH ROW (of one table), or for EACH STATEMENT (consisting of a number of row DML actions (i.e. insert, update, delete, for one table). For example the PostgreSQL trigger function *amount_of_survey_points()* below, raises an error if the view *v_ocl_amount_of_survey_points* returns any (constraint violating) row.

```
CREATE OR REPLACE FUNCTION amount_of_survey_points()
  RETURNS "trigger" AS
$BODY$
   DECLARE
      number_of_violations integer default 0;
   BEGIN
      select count(*) into number_of_violations
      from v_ocl_amount_of_survey_points;

      IF number_of_violations > 0 THEN
         RAISE EXCEPTION 'constraint "amount_of_survey_points" violated';
      END IF;
      RETURN NEW;
   END;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;
```

Both for the table *survey_document* and *survey_point*, the following triggers can be created, based on the trigger function **amount_of_survey_points():**

```
CREATE TRIGGER amount_of_survey_points
AFTER INSERT OR UPDATE ON survey_point
FOR EACH STATEMENT EXECUTE PROCEDURE amount_of_survey_points();

CREATE TRIGGER amount_of_survey_points
AFTER INSERT OR UPDATE ON survey_document
FOR EACH STATEMENT EXECUTE PROCEDURE amount_of_survey_points();
```

The constraint check will be performed in this case after the statement, which is defined as one DML statement (i.e. insert, update, delete), involving one or more records on one table. Consider the following transaction with 6 DML statements on **survey_project** (oid=80430), **survey_document** (oid=51122) and **survey_point** (oid=1234, 1235, 1236). Each of the DML statements will involve one (or more) row level DML actions, for example, one update statement number 6 below will involve three records, statement triggers will fire once (before and after). Row level triggers will fire three times (before and after) each row (Figure 15)

Insert Statement 1 on survey_project will be executed without any firing of triggers. *Insert Statement 2* on survey_document will start the FOR EACH STATEMENT trigger "amount_of_survey_points". The trigger function

"amount_of_survey_points()" will not find any violations (in OCL view v_ocl_amount_of_survey_points), since zero or more than 2 survey points per survey_document are allowed, which is true at this point in time. After *Insert Statement 3* on survey_project the trigger function "amount_of_survey_points()" will find violations of the OCL constraint "amount_of_survey_points" and will raise an error/exception, causing a rollback of all changes since the beginning of the transaction.

```
1) INSERT INTO survey_project
   (oid, survey_project_type, surveyor, start_date) VALUES
   (80430, 'collection', 'Van de Wetering', now() );

2) INSERT INTO survey_document
   (oid, survey_project_oid, survey_date) VALUES (51122, 80430, now() );

3) INSERT INTO survey_point (oid, source_oid, location_measured) VALUES
   (1234, 51122, ST_PointFromText(POINT(100874.278 428888.75), 28992) ) );
4) INSERT INTO survey_point (oid, source_oid, location_measured) VALUES
   (1235, 51122, ST_PointFromText(POINT(100881.217 428902.95), 28992) ) );
5) INSERT INTO survey_point (oid, source_oid, location_measured) VALUES
   (1236, 51122, ST_PointFromText(POINT(100897.223 428888.322), 28992) ) );

6) UPDATE survey_point set point_number = (oid-1233)
   where source_oid = 51122;
```

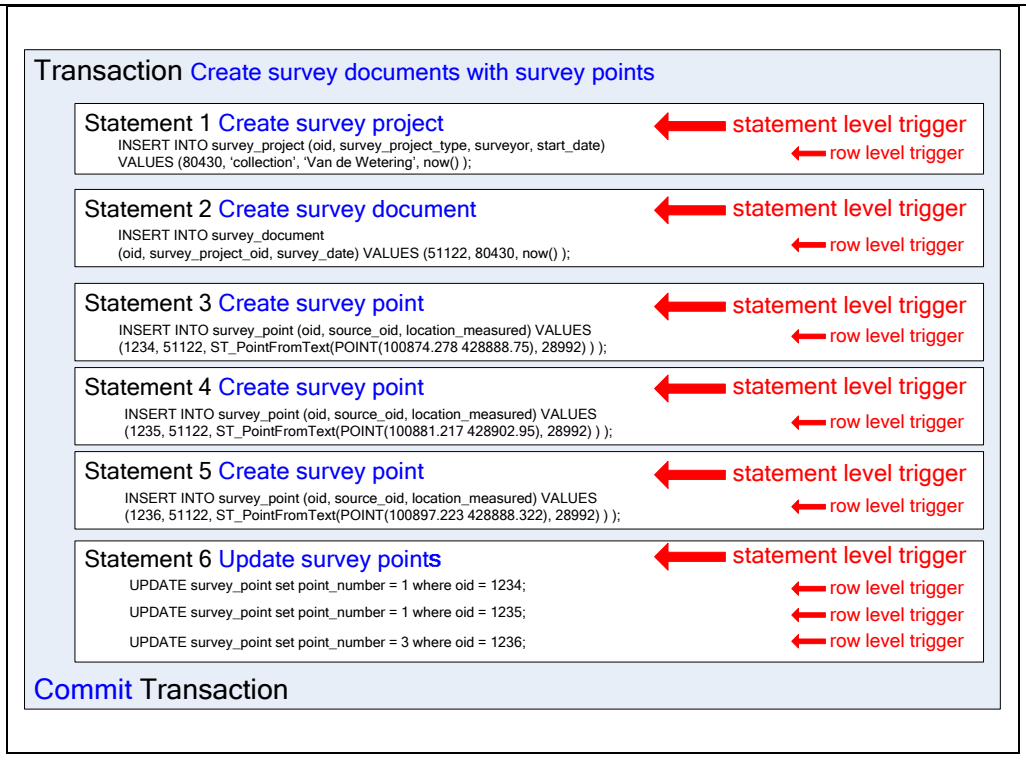


Figure 15 - Example of Transaction, Statement and Row level DML on survey_document and survey_point

The conclusion is that certain types of constraints can only be checked after a transaction and not after a statement, requiring a "transaction level trigger". An example of a constraint, often use in the context of LADM is "the sum of shares of all (ownership) rights related to one register object must be exactly one". This would also be a constraint that needs to be checked at transaction level, first the individual shares are inserted within the transaction, and (only) then the sum of shares is checked. A transaction management mechanism is required to implement such a transaction level trigger, of which an example is discussed in section 4.3.1.

Note that with this implementation based on "transaction based triggers" in a transaction management mechanism, the same result as assertions can be achieved (in terms of database consistency with constraints). During the statement situations may occur which temporarily violate constraints, but after the transaction, at the moment of committing all changes caused by all statements within the transaction, the integrity of the RDBMS in terms of compliance with the specified constraints is guaranteed.

Although performance has not been an issue in the master thesis, a comment could be made with regard to the efficiency of the OCL views, because these are potentially querying all rows in a table (or group of tables), instead of only the records affected in the transaction. The performance of the discussed check (based on OCL view *v_ocl_amount_of_survey_points* which queries the tables *survey_document* and *survey_point* with in total respectively 16268 and 147815 records, see section 7.4) was perceived as acceptable (with proper indexing of primary and foreign key columns). When an *oid* (unique record identifier) is used, for example the *oid* for the survey_document (see select statement below), which is available in each record of both survey_point (source_oid) and survey_document (oid), the performance will be acceptable, but then the integrity of the database might be jeopardised, because only a few (and not all) records are checked.

```
select count(*) into number_of_violations
from v_ocl_amount_of_survey_points
where oid = 51122;
```

4.2.1 Classification of Constraints from Platform Specific Viewpoint

The implementation of constraints at row, statement or transaction level depends on the involved instances of classes, a classification from PSM viewpoint needs to be made. One of the possible ways to classify spatial constraints, was described by Cockcroft [Cockcroft, 1997]. One of the goals of this *taxonomy of integrity constraints* was to identify appropriate implementation strategies of constraints into spatial database systems. Cockcroft designed a 2 dimensional classification of spatial integrity constraints, as well as an indication of a possible implementation of the constraints. On one axis, constraints are classified based on their static or transitional nature. On the other axis, constraints are classified based on their design level of abstraction: user defined, semantic, and topological integrity constraints.

The static part of Cockcroft's classification was refined by Van Oosterom [Van Oosterom, 2006], based on number of case studies, leading to the following criteria based on:

- the involved classes and instances
- the type of attributes
- the type of spatial relationship
- the dimension
- manner of constraint expression (e.g. 'never may' as opposed to 'always must')
- the nature of constraints

With regard to implementation of OCL constraints, a different classification may be needed from implementation viewpoint (i.e. Platform Specific Viewpoint). A classification has been described in section 6.7. The constraints are classified into categories ranging from constraints involving attribute values of a single instance, to a constraint, valid for multiple instances of different classes. The argument for this classification is that a certain category of constraints will likely be implemented in the same way. Once one implementation method for a certain category is defined, all other constraints in these categories can be implemented according to this method, which will save design and development time with regard to the platforms specific environment. In the prototype section, a number of OCL constraints have been investigated in different categories:

- Constraints Applicable to One Instance of One Class
- Constraints Applicable to Multiple Instances of One Class
- Constraints Applicable to Multiple Instances of Multiple Classes

4.3 Practices with Regard to Constraints

In the field of information system development an increasing interest can be witnessed for documenting constraints in a formal manner, a few examples of these developments and research will be described in the following paragraphs:

- *Oracle CDM Ruleframe*; constraint repository and transaction management mechanism
- *Dresden OCL22SQL* tool; OCL views for constraints

First, Oracle's Custom Development Method 'CDM Ruleframe' is described (CDM, Gylseth et al., 2000), in the context of the CASE tool Oracle Designer, as an example of constraint repositories. Although CDM Ruleframe does not use OCL for specifying the constraints, are therefore may seem less relevant, relevance for the master thesis can be found in two observations. CDM Ruleframe categorizes the constraints (business rules) into the categories related to their implementation in an Oracle database (PSM), which will also be addressed in section 6.7. A constraint repository can also help detecting conflicting constraints, which may be more cumbersome when constraints are stored in various locations and levels (e.g. invariants stored with the class, association or attribute, which determines the context of the constraint).

Furthermore, CDM Ruleframe generates a transaction management mechanism based on a structure of tables, views, stored packages, procedures, functions and triggers. This transaction management mechanism, required because of the lack of support for the SQL assertion [ISO/IEC, 2003] deals with enforcing rules at the moment of committing the changes of one transaction to the Oracle database, and ensures the

integrity of the database, see previous section 4.2 on the need for a transaction based implementation of constraints.

Secondly, The Dresden OCL22SQL tool (OCL version 2 to SQL) is described, which can convert UML models and OCL constraints to SQL statements (constraint / OCL views) for the Oracle and PostGIS database. The OCL views can be used in different manners, depending on the constraint evaluation strategy, one of the options is the transaction management mechanism, similar to CDM Ruleframe (see the example based on triggers in section 4.2).

4.3.1 Constraints Repository

As part of Oracle's Custom Development Method (CDM), business rule modelling is described [Gylseth et al., 2000a], [Chapter 3], in the context of the CASE tool Oracle Designer. CDM recognises 5 main classes of business rules:

- Static constraint rules
- Dynamic constraint rules
- Change event rules with data manipulation
- Change event rules without data manipulation
- Authorisation rules

Static constraint rules describe the state of the data, and are always true/valid in the RDBMS, dynamic constraints are related to the state of the data manipulation operation (create, update, delete or combinations thereof). Change event rules define derived actions, with or without subsequent data manipulation. Authorisation rules describe conditions for users to be allowed to perform a certain data manipulation or function. In the master thesis, only static rules will be addressed.

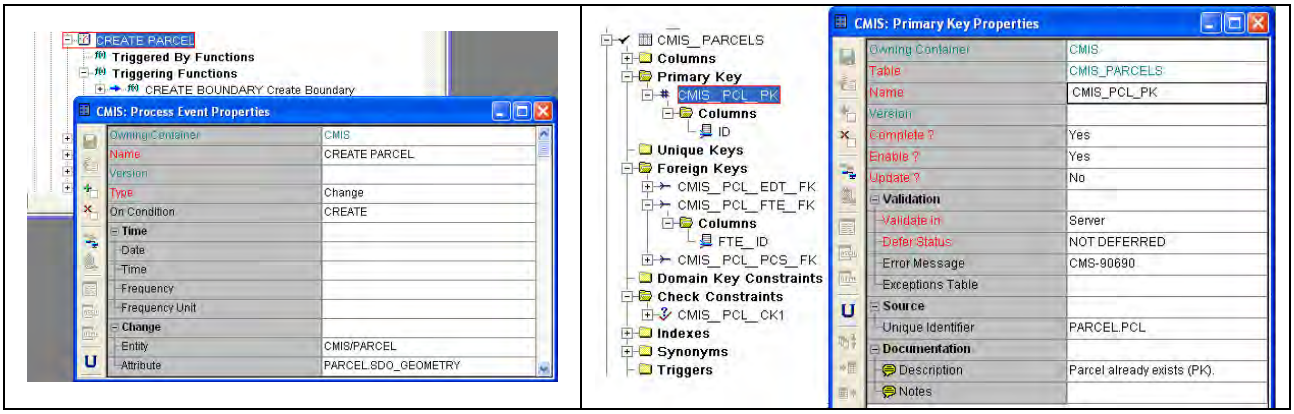


Figure 16 - Example Rule Notation Oracle Designer (process event, and primary key)

The business rules are registered through the Oracle Designer user interface in the Oracle Designer repository. Depending on the classification of the rule, it will be registered in different ways in the Oracle Repository. For example, a rule can be stored as a property of an attribute (describing entities), as a property of a primary key (unique identifier, Figure 16) or foreign key (relationship), or as a check constraint (defined for one row). But also as an Oracle Designer specific object "business function", in this case the equivalent of a rule, involving entities and attributes.

Another Oracle Designer specific object "Event" is used, describing what type of DML statement (e.g. create/insert) invokes the business function / rule).

The event driven specification of constraints is what Wang and Reinhardt describe in their constraint decision table, exclusively for spatial constraints [Wang and Reinhardt, 2007]. Spatial constraints are stored in a separate repository describing the triggering event, the condition and the spatial methods to be used (e.g. intersects), and the action as a result of the constraint check.

The architecture of CDM Ruleframe is explained by Gylseth et al. [Gylseth et al., 2000b], and is based on the elements:

- Table Application Programming Interface (TAPI)
- Custom Application Programming Interface (CAPI)
- View Application Programming Interface (VAPI)
- Transaction Management
- Message Handler

The TAPI is based on the table definitions in Oracle Designer, and on the related rules, stored as part of these table definitions. Oracle Designer generates a TAPI structure of stored packages and triggers for the DML actions (i.e. select, insert, update, delete, and lock), and for the rule validation. The custom developed rules (registered as business functions and events) are used to generate the CAPI, which are called by the TAPI. On behalf of the front-end VAPI's are created, based on table definitions, which call the TAPI elements. The Transaction Management Mechanism deals with all applicable rules, which are 'stacked' during the transaction, and checked and enforced at the moment of committing the changes of one transaction (as opposed to the level of statement or row). Any errors will be communicated trough the Message Handler.

A transaction management mechanism, ensures the integrity of the database, and could make use of OCL Views, as described in the next section.

4.3.2 Constraint Views

The Dresden OCL2 toolkit (URL 20), maintained by the Software Technology Group at the Technical University of Dresden, contains the OCL22SQL tool (Dresden OCL, 2008) for converting UML models and OCL (version 2) constraints to SQL statements for the Oracle and PostGIS database. Only invariants are considered, because pre- and post conditions of operations are not supported. The OCL22SQL tool is currently still based on the meta models of (older) MOF version 1.4 and UML version 1.5. Input to the tool are UML class diagrams in XMI format, and the OCL invariants, described a separate file (with *.ocl extension). These class diagrams are translated to SQL views, based on super classes and their specialised sub-classes.

One of the functionalities of the tool, is that UML classes (PIM) are transformed into tables (PSM), via vertical transformation (1:1 mapping of classes to tables) or typed transformation (n:1 mapping of classes to one resulting table), see section 6.6.1 for more on transforming super and sub classes. Regardless of the type of transformation, SQL views with the same set of columns are generated for each super class.

The OCL invariants are translated to so-called "OCL views", which are based on the abovementioned "super class SQL views". These so-called OCL views can be used in different manners, depending on the constraint evaluation strategy (section 4.2). Currently, the OCL invariant cannot contain spatial data types nor spatial operations and the influence on the MDA transformation is limited in the OCL22SQL tool.

A new infrastructure for the Dresden OCL2 toolkit has been developed on a so-called pivot model, serving as an exchange format for UML/OCL models [Bräuer, 2007], with a first implementation in the Eclipse Modelling Framework (EMF, URL 28). One of the elements in the research of Bräuer, was to examine how OCL and can be applied and mapped to arbitrary domain specific languages (DSL), such as (SQL, EMOF). Demuth et al. are performing additional research into parser techniques, in order to upgrade the OCL parser as part of the OCL Toolkit [Demuth et al., 2005].

Making use of the Dresden OCL22SQL tool based on XMI diagrams, generated by Enterprise Architect (used in the prototype), has proven to be difficult, often caused by unknown data types or incompatible XMI diagrams generated by the Enterprise Architect tool. Although building an OCL parser is out of scope for the master thesis project, some experiments with simple OCL constraints have been performed, see section 6.7.

4.3.3 OCL Spatial

When reviewing the data types and operations that can be used in OCL, the conclusion can be made that OCL does not directly support the creation of spatial constraints, nor does it support spatial data types and operations. An extension of OCL (OCL Spatial) was proposed by Pinet, et al. [Pinet et al., 2007]. Pinet proposes the (only) integration of the 8 topological Egenhofer binary relationships into OCL (i.e. disjoint, contains, inside, equal, meet, covers, coveredBy, overlap), which were used in the environmental information systems, subject to his research (Figure 17).

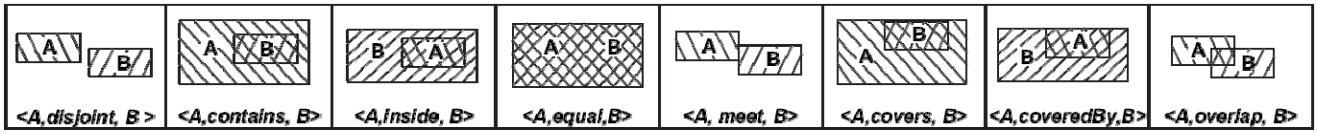


Figure 17 - Egenhofer Operations, to be used in OCL (taken from [Pinet et al., 2005])

The syntax of an OCL function would be "*A.Egenhofer_topological_relation(B)*" and an OCL invariant, describing a Building that must be within a Parcel, could be presented like:

```
context Building inv:
self.geometry.inside(self.Parcel.geometry))
```

Part of this proposal was also the definition of a new OCLBasicType: BasicGeoType, a generalisation of spatial types Point, Polyline and Polygon, which are used in UML to specify the geometry attributes of classes (Figure 18). For code generation based on these OCL and UML definitions, the abovementioned OCL22SQL tool (section 4.3.2) was proposed to be extended.

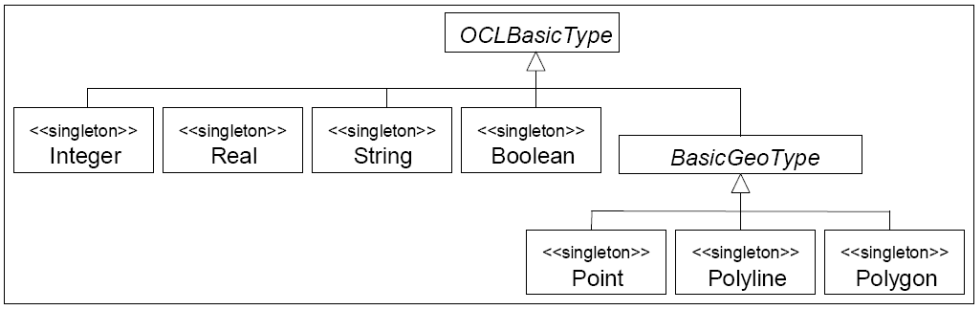


Figure 18 - New OCL Basic Types (taken from [Pinet et al., 2005])

Pinet focuses only at a few spatial data types (Point, Polyline, and Polygon) and spatial operations (topological operations from Egenhofer). Because platform independent and platform specific standards with regard to spatial data types and operations are emerging, maturing, and being used more often, a case can be made for expanding the OCL, for example based on ISO19107, with spatial data types for geometry (prefixed in ISO19107 with 'GM_') and topology (prefixed with 'TP_'), as well as spatial operations. For example the topological operations, as suggested by Pinet, but also operations like distance(), dimension(), centroid(), envelope(), buffer(), equals(), area(), volume(), length(), etc..

4.4 Conclusion

The Object Constraint Language (OCL) is a formal language, which has been defined as an extension to UML, because UML is not capable of visually modelling every kind of constraint. A part of the OCL constraints can be implemented by base table check constraints, but some of the more complex constraints, involving multiple classes, would require implementation through sub-selects in base table check constraints, or even with the use of SQL assertions, defined in the SQL standard. Both SQL assertions and the sub-selects in base table check constraints are not possible in the current relational databases.

An alternative implementation of OCL constraints is required with row and statement level triggers based on OCL views. In some cases, a handling of constraints at transaction level is required, checking the constraint only after executing a group of DML statements (i.e. insert, update, delete) for multiple tables. This is referred to as a transaction management mechanism.

A transaction management mechanism can be based on a constraint repository where constraints are categorised from a platform specific viewpoint. Such a transaction management mechanism is provided by the Oracle CDM Ruleframe, which has been described as an example. CDM Ruleframe registers (non-spatial and non-OCL) constraints in an Oracle Designer repository. Another example is the Dresden OCL2 toolkit, which contains the OCL22SQL tool, capable of transforming PIM elements to a PSM, for example in target platforms Oracle and PostgreSQL. The OCL22SQL tool also transforms (non-spatial) OCL invariants into OCL views based on PSM elements, which could be used in a constraint implementation approach, for example based on a transaction management mechanism.

A recommendation is made to expand the OCL with spatial data types and operations for geometry and topology, for example based on the ISO19107 standard "*Geographic Information - Spatial schema*", as well as based on other (platform specific) spatial standards that are now reaching sufficient stability and maturity levels.

5 Kadaster Survey Measurements and LADM SP

5.1 Introduction

At The Netherlands' Cadastre, Land Registry and Mapping Agency (Kadaster), a project called "Registration Map Quality" (NL: Registratie Kaart Kwaliteit [RKK]) is being executed [van Buren, 2006]. The project deals with differences between the actual measurements of objects (i.e. measured coordinates of parcels and buildings), and, the adjusted (NL: vereffende) coordinates of those objects on the digital cadastral map. This chapter will address the following subjects:

- Kadaster and Survey Measurements (section 5.2)
- Project "Registration Map Quality" (section 5.3)
- Adjustment of LADM 'Survey Package' (PIM) (section 5.4)

First, the Kadaster's systems, processes and information involved in survey measurements will be discussed, followed by a description of the project "Registration Map Quality". This project will produce data, that on the one hand will be used to populate the Adapted LADM 'Survey Package' PostGIS database (section 7.4), and on the other hand will be analysed (section 7.5). Finally, the Adapted LADM 'Survey Package' (PIM) will be introduced, suitable as input for prototyping with MDA processes, and suitable to contain the data as provided by Kadaster.

5.2 Kadaster and Survey Measurements

On behalf of the survey projects, that are being executed to collect measurements of parcels and buildings in the Netherlands, the Kadaster has created a manual for the activities with regard to collection and handling of geographic information, the Manual for Technical Operations of the Kadaster (NL: Handleiding voor de Technische Werkzaamheden van het Kadaster, abbreviated HTW, [Polman and Salzmann, 1996]). The HTW manual (i.e. "How To Work") is an extensive description of the surveying processes and products and the relevant quality control, which must also be used by other (external) parties, conducting part of the surveying work for the Kadaster. In the following section, parts of these processes and products will be described. It is not the intention to describe all details of the survey process; only those details will be described, which are relevant to the objectives of the master thesis project.

Within Kadaster a number of systems are used in the processes related to survey measurement handling:

- **LKI**; Surveying Cartographic Information (NL: Landmeetkundig Kartografische Informatie).
- **TIR**; Terrestrial Collection and Reconstruction (NL: Terrestrische Inwinning & Reconstructie).
- **MOVE3**; processing and quality control of GPS and terrestrial observations (URL 21).

With LKI the (final) digital cadastral map with parcels and buildings in the Netherlands can be accessed. TIR provides a management environment with regard to survey projects. With TIR, information can be imported from and exported to LKI and MOVE3. MOVE3 provides functionality for the design, adjustment, and quality control of 3D, 2D and 1D geodetic networks, and the processing of inbound and outbound measurements. MOVE3 provides the functionality with regard to the *1st phase* and the *2nd phase* measurement adjustments, as depicted in Figure 19:

- 1st phase free network adjustment (NL: eerste fase vereffening in het vrije net [Polman and Salzmann, 1996], [Chapter 4])
- 2nd phase control point constrained network adjustment (NL: tweede fase aansluitingsvereffening)

The ***1st phase free network adjustment*** calculates, based on multiple measurements for one surveyed/measured point, the **measured coordinate**, related to the attribute *locationOrig* of class SurveyPoint in LADM [ISO/TC211, 2008]).

The ***2nd phase control point constrained network adjustment*** converts the calculated *measured coordinate* to a **transformed coordinate**, related to the attribute *locationTransf* of class SurveyPoint in LADM [ISO/TC211, 2008]).

5.2.1 1st Phase Free Network Adjustment

In the *1st phase free network adjustment*, the survey measurements will be processed and used to calculate the (consolidated) *measured coordinate* of the surveyed points. The *survey measurements files* may consist of GPS measurements, tacheometer measurements, tape measurements, or a survey fieldwork sketch (NL: veldwerk), and others. The measured coordinates, dependent on the original measurements, can be stored in:

- A *Local spatial reference system*; originating from survey projects in a local so-called '2000-2000 meter' spatial reference system.
- *The Local "Rijksdriehoek" (RD) spatial reference system*, originating from survey projects with GNSS observations, without certified coordinates for reference stations.
- *The RDNAP-TRANS spatial reference system* (URL 29), originating from survey projects with GNSS observations, based on certified reference station coordinates.

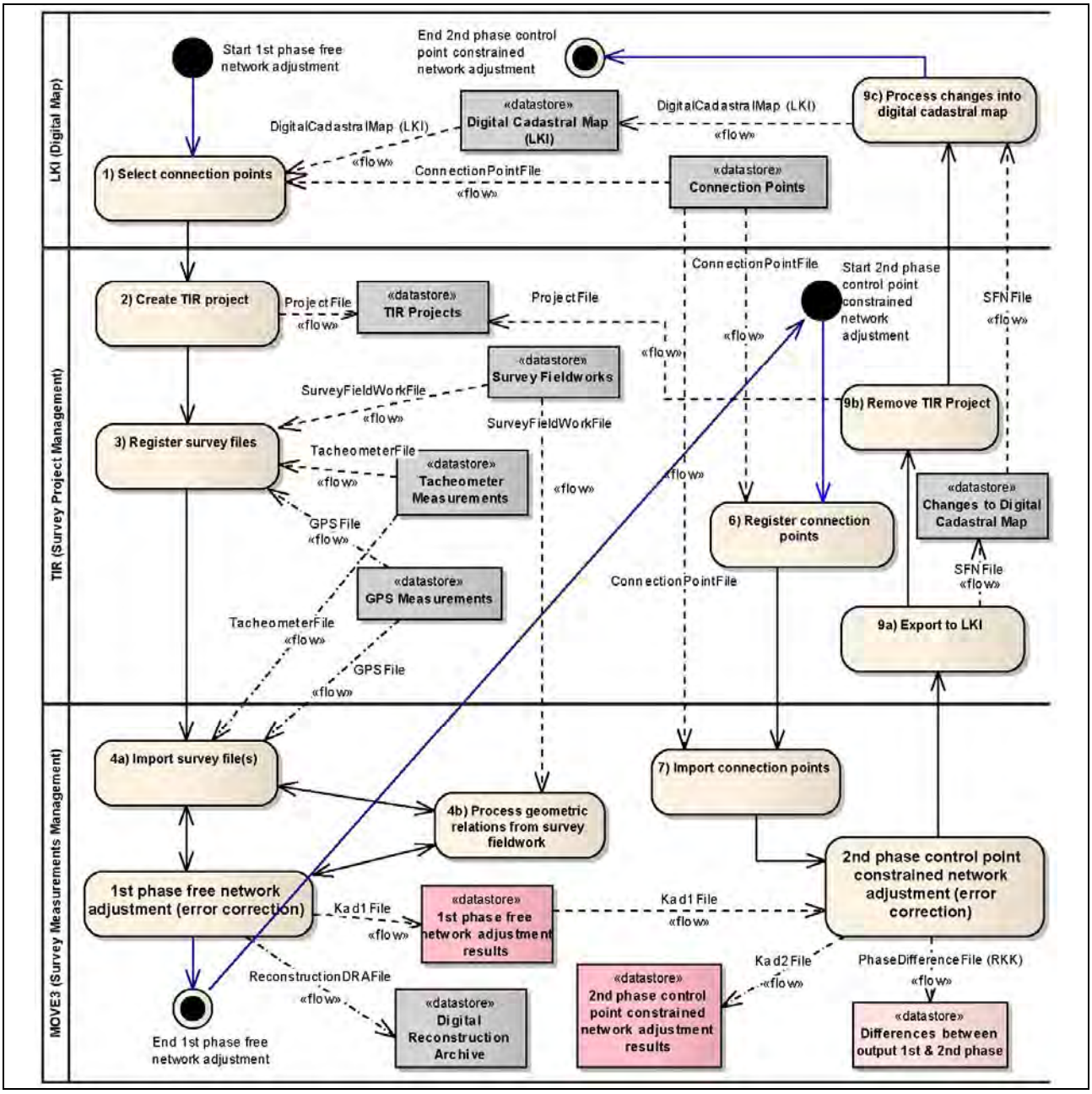


Figure 19 - Kadaster Process for Handling Survey Measurements (LKI, TIR, MOVE3)

The following steps are performed in the *1st phase free network adjustment*, see corresponding step numbers in Figure 19, drawn up from the TIR/MOVE3 training, provided by Kadaster:

1. A number of *existing connection points* (NL: aansluitpunten) must be selected from the *digital cadastral map*, which will be re-measured during the survey project (to be used in the *2nd Phase Control Point Constrained Network Adjustment*).

2. A *TIR project* is defined, which will exist during the handling of the survey project and measurements.

3. The available survey measurement files (e.g. survey fieldworks, tacheometer, GPS, and tape measurements) will be *registered* in TIR.

4. These files will be *imported in the MOVE3* environment where they will be processed, connected, and balanced to each other.

5. After error control and error resolution, the *1st phase free network adjustment results will be generated* (measured coordinates) as input to the 2nd phase control point constrained network adjustment process. These first phase free network adjustment results are stored in the *digital reconstruction archive* (DRA).

5.2.2 2nd Phase Control Point Constrained Network Adjustment

The calculated measured coordinates from the *1st phase free network adjustment* will be transformed (adjusted) to fit them in the digital cadastral map (in *RDNAP-TRANS spatial reference system*). The 2nd phase adjustment to fit measured coordinates into the map is required because the original measurements are in local spatial reference system, and need to be transferred to the *RDNAP-TRANS spatial reference system*, used in the cadastral map. But another reason for this adjustment is the difference in accuracy of the cadastral map and the accuracy of the measurements. When the measurements are more accurate (i.e. closer to the value of the feature of interest in the real world) than the cadastral map, the measurements need to be adjusted to fit in.

This adjustment is performed based on *connection points* (a.k.a. control points), which are already present on the cadastral map, and also have been (re-)measured in the survey project. Based on the calculated *measured coordinate* of the connection points, and their known *transferred coordinate* on the cadastral map, the required adjustment can be determined. This required adjustment of measurements, can be used in the *2nd phase control point constrained network adjustment*, to transform the calculated measured coordinate of the *non-connection points* (in various spatial reference systems) to their transferred coordinate on the cadastral map (in *RDNAP-TRANS spatial reference system*), eventually leading to new objects (parcels and buildings) on the map. The 2nd phase control point constrained network adjustment (NL: tweede fase aansluitingsvereffening) will result in the final proposed *changes to the digital cadastral map*.

6. Essential in this process are the connection points which earlier have been *registered* (and re-measured) for the survey project.

7. The connection points will be *imported* into MOVE3.

8. The 2nd phase control point constrained network adjustment can be performed for the non-connection points based on the *original* and *transferred* location of the connection points. This will enable the new measured objects to be positioned and adjusted in the digital cadastral map.

9. The transformed measurements are exported to LKI, for processing into the digital cadastral map. Eventually the TIR project will be removed.

5.2.3 Information Required for Survey Measurement Handling

In Figure 19 the information flows are depicted by dotted arrows, marked by the label <<flow>> and the name of the file. The files that play a role within the process for survey measurement handling are:

- Digital Cadastral Map (LKI)
- Connection Point File (*.COO)
- Measurements (e.g. fieldwork, GPS, tacheometer)
- Kad1 File (Move3.kad1.xml; 1st phase free network adjustment results)
- Reconstruction DRA File (NL: Reconstructiebestand.dra.xml)
- Kad2 File (Move3.kad2.xml; 2nd phase control point constrained network adjustment results)
- SFN File (NEN1878, SUFNEN format)

These files have different structures, as depicted in Figure 75 in the "Appendix C: Examples of Survey Files (Kadaster)". The variety of the files is caused by the evolution of the cadastral system, based on several applications like LKI, TIR, and MOVE3. The process of handling survey measurements is dependent on these applications, and on the export and import of the abovementioned files between them.

There is no integrated database to hold all survey related information, used or created during the survey process, described in section 5.2.1 and 5.2.2. The survey fieldwork, which is the basis for the measurements in a survey project, is stored and kept, however, information on the actual transformation, error checking and result (Kad1 and Kad2 file) will be deleted after a period of time. The relation between a measured coordinate of a point, calculated from the original measurements (observations), and the resulting coordinate of the same point on the cadastral map (after 2nd phase adjustments) is usually not stored after finalisation of a survey project. [Polman and Salzmann, 1996] [Section 2.5.3].

As part of future development and improvement of the cadastral system, a recommendation can be made *to store all survey related the data in one integrated system/database*. This has also been proposed and described by Lee, who addressed the quality of the cadastral map, in terms of the need for managing and storing the survey measurements in well suited structures, as well as the manner in which they are used in the cadastral map [Lee, 2005] [figure 5.9]. This development could result in an extended/improved LADM 'Survey Package', as envisioned in section 1.1 or a variation or specialisation of the current LADM 'Survey Package'.

One of the advantages of an integrated database is that the potential security issue with digital file based processing can be mitigated. Another advantage is the possibility of performing various kinds of automatic improvement of the digital cadastral for large areas. Consider the 'degeneration' of the more accurate measurements that currently occurs in the *2nd phase control point constrained network adjustment*, when dealing with a less accurate cadastral map. After transforming the coordinates (making them less accurate) to be able to fit them in the cadastral map, all information about the 2nd phase adjustment process (e.g. the link between measured and transferred coordinates, the error reports) is eliminated, so no automatic upgrading of the map, based on more accurate measurements is possible.

Storing all information about measured and transferred points as described, would enable a reverse "fitting" process, of adjusting the (less accurate) cadastral map to the measured coordinates.

The project "Registration Map Quality" (section 5.3) is a first approach to store, analyse, and use survey information, taking into account original and transformed coordinates of measured survey points.

5.3 Project "Registration Map Quality"

The *2nd phase control point constrained network adjustment* transforms measurements of new objects (parcel and buildings), in order to fit them into the (less accurate) cadastral map. This transformation is done based on connection points (control points). For a connection point, the measured coordinate is known (in *local*, *local RD*, or *RDNAP-TRANS spatial reference system*), as well as the (transformed) coordinate on the cadastral map (a.k.a. the transferred coordinate in *RDNAP-TRANS spatial reference system*).

The difference between the calculated measured coordinate (attribute *location_measured* for class *survey_point* in Figure 40) of the connection point and its coordinate on the digital map (*location_transferred*) is subject to the analysis with regard to the quality (accuracy) of the digital cadastral map. *In other words, the difference between the coordinate of a connection point, before and after the 2nd phase control point constrained network adjustment.*

The "Registration Map Quality" project aims at providing files with these differences for connection points, which now are being collected since April 2006. The differences provide an indication of the quality (accuracy) of the cadastral map, an initial analysis and visualisation of this data, provided by Kadaster, will be presented in section 7.4 and 7.5. With regard to the quality, Kadaster's products are defined in terms of a "graphical precision"; the differences between the measured and transferred coordinate of a connection point before and after the 2nd phase control point constrained network adjustment (NL: tweede fase aansluitingsvereffening, see section 5.2.2), should be smaller or equal to ± 20 cm and ± 40 cm respectively in urban (built-up and town areas) and rural areas [Polman and Salzmann, 1996]. These values for graphical precision have been used in the analysis of survey (connection) points in section 7.5, at the level of cadastral office, municipality and section.

As described in section 5.2.1, the connection points have originally been measured in Local, Local "Rijksdriehoek" (RD), or the RDNAP-TRANS spatial reference system. The connection points, measured in the *RDNAP-TRANS spatial reference system*, will be labelled survey_point.quality = *'gnss'*. The measurements in *Local* or *Local RD spatial reference system* will be labelled with survey_point.quality = *'local'*.

Note that, as part of the "Registration Map Quality" project, the measurements in Local and Local RD spatial reference system have been transformed to the RDNAPTRANS spatial reference system, to be included in the analysis of connection points and the differences between their *measured* and their *transferred* coordinate, see section 7.5. This transformation has been done, as part of the "Registration Map

Quality" project by executing a *similarity transformation* with a rotation around the origin (to align the local spatial reference system with RDNAP-TRANS), and a shift in X and Y direction [Polman and Salzmann, 1996] [section 4.4.3]. The data on connection points, used in the master thesis project, were provided by Kadaster in RDNAPTRANS spatial reference system (section 7.4.3).

5.4 Adjustment of LADM 'Survey Package' (PIM)

The LADM 'Survey Package' will be adjusted to Dutch local circumstances, with the knowledge gained from the Kadaster case study, described in the previous sections, with the primary goal of creating a variant of LADM 'Survey Package',

- suitable as *input for prototyping* with MDA processes, and
- suitable *to contain the data*, provided by Kadaster, on connection points, parcels and buildings, cadastral offices, municipalities, and sections.

See "***Figure 20 - Adapted LADM 'Survey Package', Input to the MDA Prototype***", and see section 7.4 and 7.5 for more on the Kadaster data, and the analysis performed. The extension and improvement of the LADM 'Survey Package' has not been a primary consideration, see section 8.1 for an explanation of this approach.

Some specific remarks with regard to this adjustment will be listed here; the **Adapted LADM 'Survey Package'** has been created based on the LADM Survey Package, extended and adapted with a number of classes, present in the Kadaster data:

CadastralOffice, **CadastralMunicipality**, and **CadastralSection**, describing the administrative structure, used by Kadaster in the Netherlands. Note that the unique identification of these tables is based on the primary "oid" columns, generated by the MDA prototype, which is different from the actual identification based on cadastral municipality (e.g. code "HTN04" for municipality Houten) and cadastral section (e.g. code "K"). The administrative structure and unique identification in other countries may consist of other and more aggregation levels. The class AdminParcelSet in ISO 19152 [ISO/TC211, 2008] is replaced by these classes.

SurveyProject, identifying the fieldwork, the survey documents and the measured survey points. The class SurveyProject is not known in ISO 19152, and can be recommended for future improvement of the LADM. Note that SurveyProject has an attribute *ProjectMessage* to contain the logging provided as part of the "Registration Map Quality" project on connection points used, which would not be part of such a recommendation.

LegalDocument is part of the yellow package with classes related to administrative and legal LADM matters, has been added to the prototype Survey Package, to demonstrate how PIM to PSM conversion would work for transformation of super and sub classes in a PIM (section 6.6.1).

The Adapted LADM 'Survey Package' also contains a class **Parcel** and **Building**. Note that the provided Kadaster data (see section 7.4) only contains parcels with a geometric primitive polygon to describe these spatially (and not topology data in for

example a winged-edge data structure as in LKI). This type of data can be qualified as "spaghetti data". This term indicates that spatial features are described and stored without any relationship between them, as explained by Ingvarsson [Ingvarsson, 2005]. Describing parcels with only geometric primitives, will ensure the internal topological consistency (e.g. PostgreSQL will not allow geometries of type POLYGON which are not 'closed'), but will allow for overlapping of instances of individual parcels within the same data set. One of the disadvantages is that the boundaries and points of adjacent parcels are stored more than once. Another disadvantage is that relational queries, for example with regard to adjacency, overlapping or gaps are calculated at the moment of demand, which can be complex, and time consuming (see "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)", the example on OCL view v_ocl_no_overlapping_parcels)

A topological model would describe the relation between the topological elements like node, edge, and face (see TP_Node, TP_Edge, and TP_Face in ISO19107 [ISO/TC211, 2003b], [Chapter 7], based on their unique identifiers. Unlike geometric models, topological models are not influenced by rotation, scaling and translation operations, because they do not contain metric information. The relationships, described in the model, are based on the basic topological elements, e.g. the relationships node-edge, edge-face, face-solid, sometimes with a description of direction/orientation of the elements. Queries based on a topological model would use relationships between topological elements which are explicitly stored in a topological model.

The class **Parcel** in the Adapted LADM 'Survey Package' is in fact the **SpaghettiParcel** in Figure 2 and 3 of the standard ISO19152 [ISO/TC211, 2008], with an added attribute polygon, to eventually be able to contain the Kadaster data. The operations getSurface() and getVolume() have not been considered; the prototype focused at MDA processes with regard to transformation of classes, attributes and associations. As described in section 2.3.1, Parcel is a specialisation of VersionedObject, with attributes beginValidityVersion and endValidityVersion, which could be inherited by the table parcel to indicate current and archived parcels, however in the Adapted LADM 'Survey Package' these attributes have not been used.

Class **Building** has been added to represent LADM class LegalSpaceBuilding. Class **ParcelBoundary** and **BuildingBoundary**, reflect part of the topology of parcels and buildings, the main reason for adding them, was to be able to investigate the association between SurveyPoint and Parcel or Building (note the association between class **TP_Primitive** and **SurveyPoint**).

With regard to the class diagram as the basis for the prototype, a choice has been made to experiment with MDA processes for classes and attributes, for which Kadaster data was available. A number of attributes in the LADM 'Survey Package' classes with data types such as *CC_Operation, Binary, DQ_Element, PointType,* and *Record* have not been used.

Both class **SurveyDocument** and class **LegalDocument** have an attribute number (in LADM representing a name) which has been placed up in source document named "Code" with data type String, see the attribute number in Figure 72.

The associations between **SurveyPoint** and **SurveyDocument**, and between SurveyDocument and **SurveyProject** have been used with association target multiplicity 1, while in reality (in The Netherlands, or elsewhere) situations can be found, where this could also be [0..1], i.e. survey points without survey documents.

The attributes code in **CadastralOffice** and **CadastralMunicipality** have a property IsStatic = "True", indicating that in the PSM this should be converted to a Unique Key. This property is presented in the PIM as an underlined attribute, see section 6.6, section "Create Uniqueness Constraint". Note that Enterprise Architect wrongly underlines the attribute name as well.

The attribute *type* of class **SurveyDocument** has the type *SurveyDocumentType*, referring to an enumeration class. The attribute *quality* of class **SurveyPoint** has been (mis-)used to contain the values 'gnss' and 'local', indicating survey type as present in the Kadaster data, which is overlapping with the mentioned SurveyDocumentType, see section 5.3 for more on classifications 'gnss' and 'local'. Note that the data, provided by Kadaster, only contains connection points, for which measurements are available (measured coordinates in *location_measured*), as well as the (transferred) coordinates on the map (in *location_transferred*). The Adapted LADM 'Survey Package' has not been changed on behalf of various measurements (observations) for one point, for example based on OGS's "Observations and Measurements [Open Geospatial Consortium, 2006b].

The attributes in the enumeration and CodeList classes have been limited to the ones that were encountered in the Kadaster data (Figure 21 - Adapted LADM 'Survey Package'; <<enumeration>>, <<CodeList).

Note that for some classes, for example SurveyDocument, Cadastral Section, and Parcel, *derived attributes* have been included, to experiment with derivation of attributes, and to be able to store the data as provided by Kadaster.

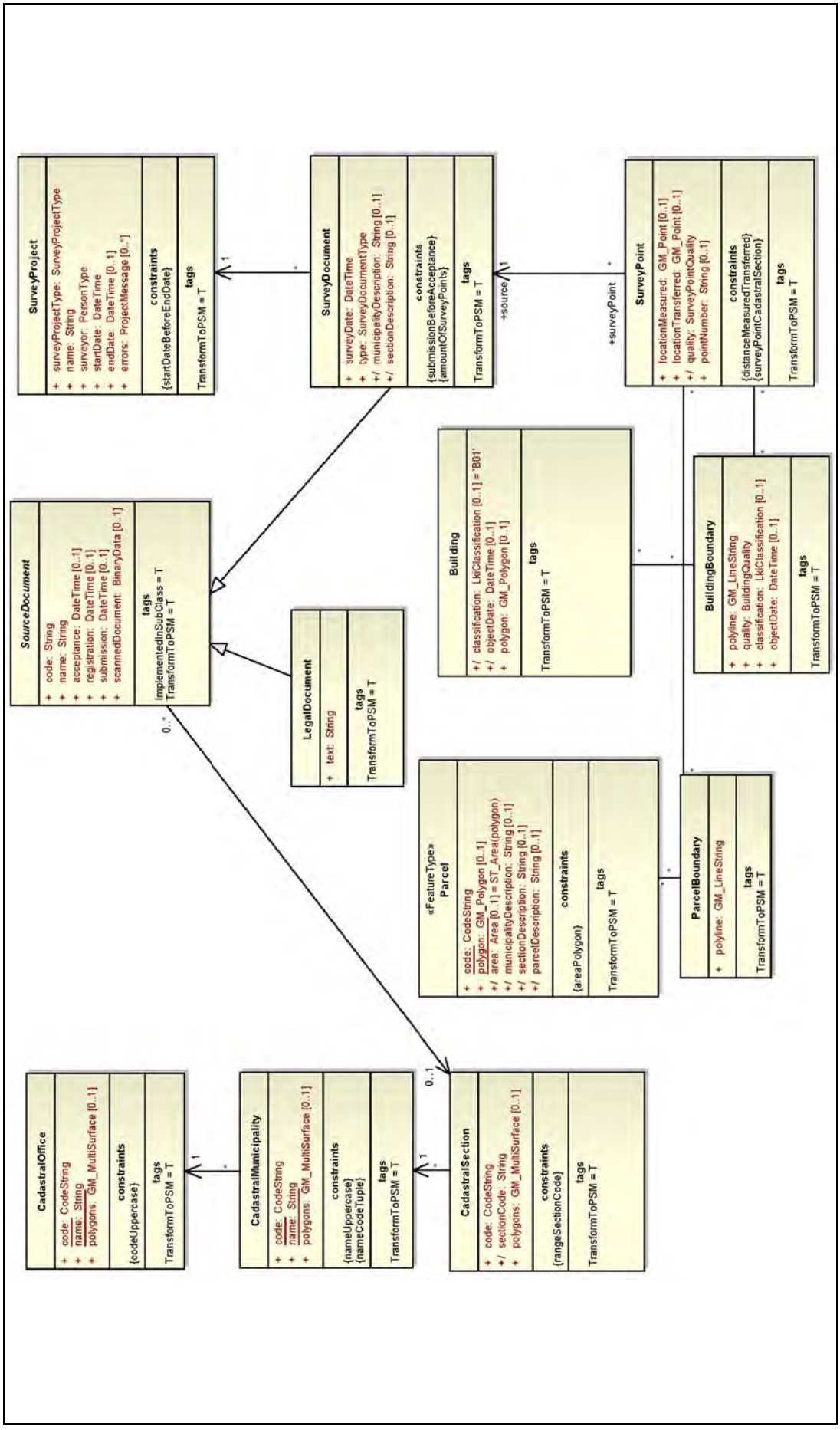


Figure 20 - Adapted LADM 'Survey Package', Input to the MDA Prototype

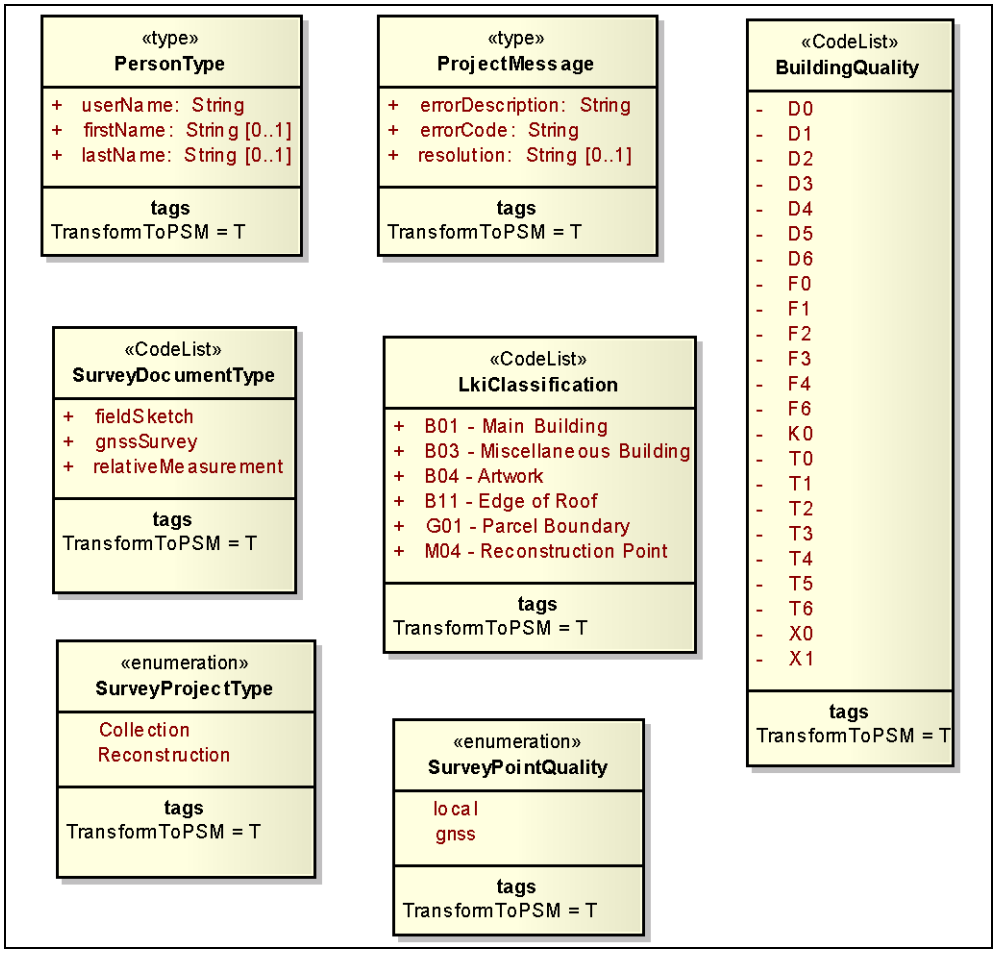


Figure 21 - Adapted LADM 'Survey Package'; <<enumeration>>, <<CodeList>>, and <<type>> classes

5.5 Conclusion

A Kadaster project called "Registration Map Quality" is being executed, dealing with with differences between the measured coordinates of parcels and buildings, and the adjusted (NL: vereffende) coordinates of the representations of those objects on the digital map, respectively before and after the 2nd phase control point constrained network adjustment (NL: tweede fase aansluitings-vereffening). This 2nd phase adjustment of coordinates is necessary to handle the difference between the (lower) accuracy of the cadastral map, and the accuracy of the measurements. These differences provide an indication of the quality (accuracy) of the digital map, which should be within the "graphical precision"; ± 20 cm and ± 40 cm respectively in urban and rural areas.

The handling of survey measurements is performed, based on three applications (i.e. LKI, TIR, MOVE3) and a digital file based exchange of survey project information between these applications. The case study has lead to a recommendation with regard to future information system developments of Kadaster. The (permanent) storage of all survey project related data (original and transferred measurements, error checking results, etc.) is recommended, as well as storage of meta data on their use, all in an integrated database, with a constraint validating mechanism in place. One of the advantages of storing the relation between originally measured coordinates and the transferred coordinates of the cadastral map, is that a reverse "fitting" process would be possible; adjusting the (less accurate) cadastral map to the (more accurate) measured coordinates.

As a basis for experimenting with Model Driven Architecture in the master thesis project, classes of the LADM, as well as non-LADM classes have been selected: Parcel, SurveyPoint, SurveyDocument, LegalSpaceBuilding, CadastralOffice, CadastralMunicipality, CadastralSection, and SurveyProject. The expansion and improvement of the LADM 'Survey Package' has not been the primary goal in composing the Adjusted LADM 'Survey Package', although comments have been made with regard to improvements. Future research based on the comments and based publications mentioned (e.g. [Ingvarsson, 2005, Lee, 2005, Open Geospatial Consortium, 2006b]) is recommended.

6 MDA Prototype

6.1 Introduction

This chapter addresses the MDA Prototype that has been designed and developed to experiment with MDA processes and principles, as described in Chapter 3. For the choice of the supporting tools the following consideration was made.

Enterprise Architect (EA, URL 18) offers a considerable amount of functionality with regard to UML modelling, as well as MDA related capabilities. Enterprise Architect is a commercially available software tool that has been used for modelling the Land Administration Domain Model (LADM). Current research efforts at Delft University of Technology aim to develop a tool, involving MDA and OCL transformations, based on the open source Eclipse development environment [Hespanha et al., 2008]. For reasons of timing and time, experience with Eclipse, and available resources in the master thesis project, the Eclipse environment has not been chosen for the MDA Prototype.

A choice has been made for experimenting with the MDA related functionalities that EA (Corporate Edition, version 7.0.817) offers. This chapter will address the following subjects:

- The description of the model *transformation possibilities in EA* (section 6.2)
- The *set-up of the MDA prototype*, based on the EA possibilities (section 6.3)
- The model *transformations* that the MDA prototype will offer (section 6.4)
- The *final generated PSM* for the Adapted LADM 'Survey Package' (section 6.8)

Transformation Possibilities in EA
Section 6.2 "Transformation Possibilities in EA" will describe and reflect on the transformation possibilities in Enterprise Architect. EA facilitates transformations from platform independent to platform specific models in two ways:

- EA Transformation Definition
- EA Software Developers Kit

The *EA Transformation Definitions* are capable of transforming a PIM to a PSM, based on a proprietary template language (described in section 6.2.1). Standard "built-in" EA transformation definitions are provided with EA (e.g. "DLL" for relational databases, Figure 22). In the MDA prototype, the EA Transformation Definitions are used in the "First Transformation from PIM to PSM-1", section 6.5.

The second way of transforming models is offered by the *EA Software Developers Kit* (EA SDK, described in section 6.2.2), providing direct access to the model elements, through "Add-in" program units, for example in programming language C#. In the MDA prototype the EA SDK is used in the "Second Transformation from PSM-1 to PSM-2" (section 6.6) and the "Third Transformation from PIM OCL to PSM-2" (section 6.7). Section 6.2.3 addresses the standard OCL related capabilities of EA, which has lead to the "Third Transformation from PIM OCL to PSM-2".

MDA Prototype Set-up based on EA
Based on the transformation possibilities of Enterprise Architect, the set-up of the MDA Prototype will be described (section 6.3). For example the PIM and PSM environment in EA for the Adapted LADM 'Survey Package' (section 6.3.2); the constants for the MDA prototype, and the data type mapping between PIM and PSM, stored in xml files (section 6.3.1).

MDA Prototype Transformations
The MDA prototype will be capable of performing a number of model transformations and implementations (section 6.4). These transformations were defined, making use of the different transformation possibilities of Enterprise Architect:

- The "First Transformation from PIM to PSM-1" (section 6.5), based on the "PostgreSQL" Transformation Definition.
- The "Second Transformation from PSM-1 to PSM-2" (section 6.6), to fine-tune the 1st Transformation, based on custom developed program units, operating with the EA Software Development Kit.
- The "Third Transformation from PIM OCL to PSM-2" (section 6.7), aimed at the OCL part of the model.
- The "Transformation from PSM to DDL (PostgreSQL/PostGIS)" (section 7.3), resulting in Data Definition Language (DDL) and Data Manipulation Scripts (DML) scripts to create the PostgreSQL/PostGIS database.

Some aspects of the transformations are discussed separately: Tagged Values, transformation of Super and Sub Classes, dealing with spatial data types and indexes, <<enumeration>> and <<CodeList>> classes, and OCL implementation.

Final PSM for the Adjusted LADM 'Survey Package'
At the end of this chapter, the result of the model transformation by the MDA Prototype will be presented in section 6.8, describing the Transformed Adjusted LADM 'Survey Package' (PSM-2).

6.2 Transformation Possibilities in EA

The model transformation possibilities in EA are considerable, although many specifications for these transformations are left to the EA user (programmer). Only basic transformations are standard provided. The possibilities of EA are discussed in three sections:

- *EA Transformation Definitions*; The standard transformation from PIM to PSM is based on the Transformation Definitions (Conversion Templates and Intermediary Files, e.g. for 'DDL', 'Java', and 'C#' platform), which can be changed by the EA user (section 6.2.1).
- *EA Software Developers Kit*; Advanced possibilities are provided by the EA Software Developers Kit, providing access to EA model elements, outside the standard EA user interface, which has been used extensively in the prototype (section 6.2.2)
- *EA and OCL*; The EA transformation possibilities with regard to OCL are discussed in section 6.2.3.

6.2.1 EA Transformation Definition

Transformations from the platform independent (source) to the platform specific models (target) can be conducted with *transformation definitions.* For example, the standard EA transformation definition "DDL" (Figure 22) has been used a basis for the custom developed "PostgreSQL" transformation definition (used during the prototype). The Transformation Definition consists of a collection of specific *conversion templates* for UML elements. The result of executing a Transformation Definition is temporarily stored in a text file: the *Intermediary File*. See "Appendix D: Examples of EA Transformation Definition 'PostgreSQL'" for some additional details on EA Transformation Definitions.

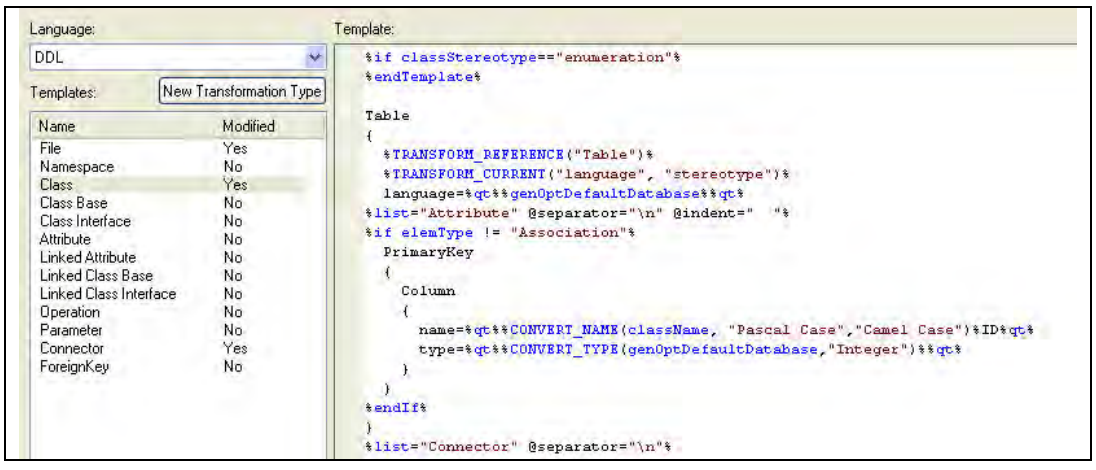


Figure 22 - EA Standard Transformation Definition "DDL", conversion template Class is selected

Conversion Template
Each conversion template (e.g. for a Class, an Attribute, and an Association (EA: Connector) is able to call/invoke another conversion template, leading to a certain hierarchy within the transformation definition. The hierarchy as shown in Figure 23 is the standard structure, drawn up from an analysis of the "DDL" Transformation Definition, as standard delivered by EA, adapted in the MDA Transformation Prototype (i.e. "PostgreSQL").

The conversion template "File" is the starting point of the transformation, invoking the "Namespace" conversion template, to ensure that the proper hierarchy of packages is maintained as presented (see the left side of Figure 34). An example of the contents of a conversion element for a package is provided in Figure 24, whereas a conversion template for a class (to table) is presented in Figure 28. Note that currently the constraints, which can be recorded at the level of class, attribute and connector (association), are not depicted in Figure 23, because they can not be transformed with the EA conversion templates.

The *conversion template for Class* in the standard EA Transformation Definition "*DDL*" (Figure 22) transforms each non-enumeration *Class* to tables, calls the conversion template for *Attribute*, creates a *Primary Key*, and call the conversion template for *Connector* (i.e. Associations). See Figure 77 and Figure 78 for the full details on the *conversion template* for *Class* and *Connector* (Association) in the custom developed Transformation Definition "*PostgreSQL*", used in the MDA prototype.

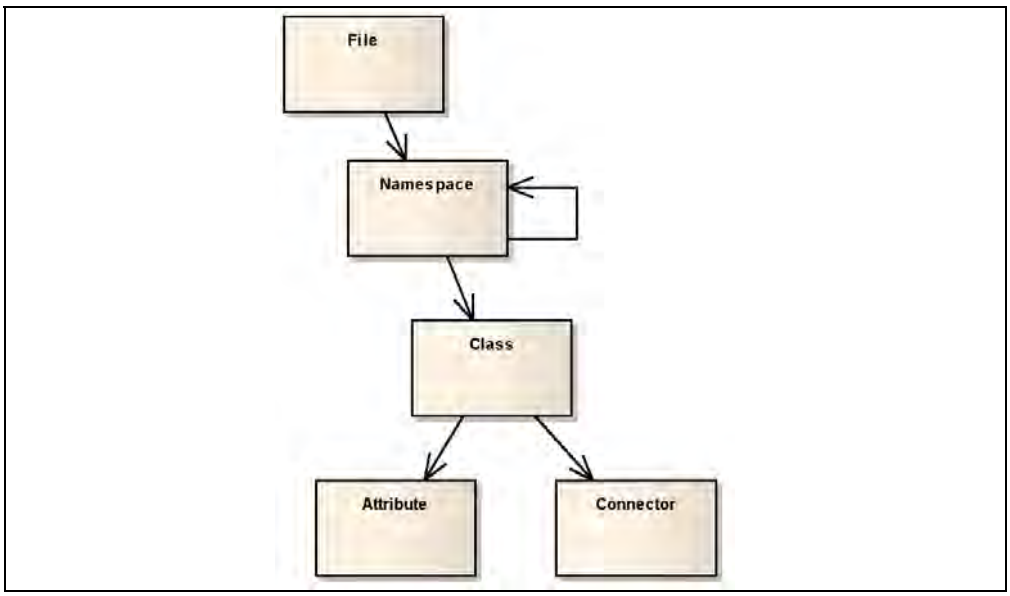


Figure 23 - Conversion Template Structure for the EA Transformation PIM to PSM-1

Conversion Template Examples
In the conversion template, a considerable amount of commands and variables can be used [SparxSystems, 2007] [Chapter 16], a few examples are provided here:

- the command "TRANSFORM_CURRENT" will copy everything from the source namespace/package to the target namespace/package, except for the explicitly mentioned list of excluded items (e.g. "scope", "abstract", "name", "notes");
- the command "REPLACE" can be used to replace text strings in the variable "packageName";
- the variables "eaDateTime" and "eaGUID" can be used to construct notes, indicating the date, time and identification of the transformation;
- the command "list" will call the other conversion template, e.g. Class and Namespace.

```
Package
{
  %TRANSFORM_CURRENT("scope", "abstract", "name", "notes")%
  name=%qt%%REPLACE(packageName,"PIM","PSM")%%qt%
  notes=%qt%PIM synchronised to PSM at %eaDateTime%, generation ID = %eaGUID%%qt%
%list="Namespace" @separator="\n" @indent="  "%
%list="Class" @separator="\n" @indent="  "%
}
```

Figure 24 - Conversion Template for Namespace (Package)

Intermediary File

The result of this transformation can be shown, for debugging purposes, in a so-called "*Intermediary File*", an ASCII file which will be used by EA to actually create or update the target model elements (in the first transformation). Figure 25 shows the first (simplified) part of an intermediary file.

```
---------- PostgreSQL ----------
Package
{
  name="PostgreSQL"
  Package
  {
    name="Land Administration Domain PSM"
    Package
    {
      name="Survey Package"
      notes="PIM synchronised to PSM at 09-mei-2008 15:23:38, generation ID = {DCFFB3EF-9E1F-42c8-A961-701C2C9919C3}"
      Table
      {
        XRef{namespace="PostgreSQL" name="Table" source="{CDF52BC5-935B-4fc3-A496-F56A3C613A7B}"}
        name = "survey_point"
        language="PostgreSQL"

        PrimaryKey
        {
          name = "pk_survey_point"
          Column
          {
            name="oid"
            type="integer"
```

Figure 25 - EA Transformation Intermediary File (first part)

The transformation relation between an element of the PIM and the PSM is established and maintained during the transformation by the command "TRANSFORM_REFERENCE". This leads to an "Xref" notation in the EA Transformation Intermediary File in Figure 25, where the transformed target table is linked to its source class. The *Xref* notation is used by EA to store and visualise the relation ("transformed to") between source class and target table. This link is used when a PIM to PSM transformation is executed again.

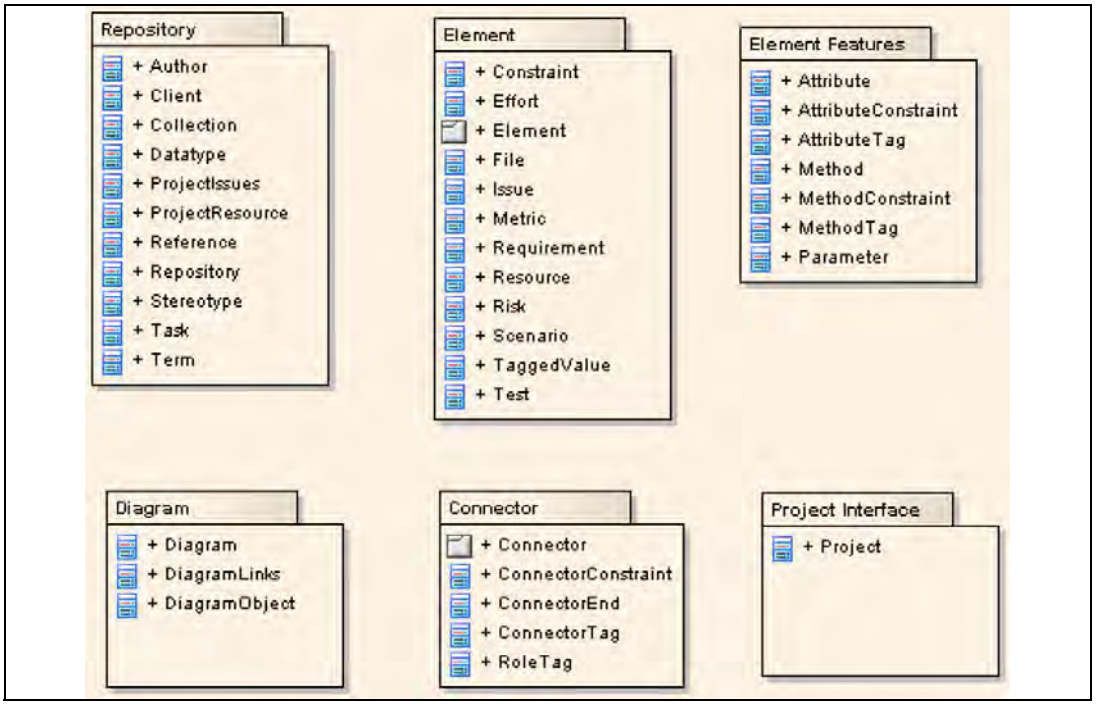


Figure 26 - EA SDK Interface Object Model (taken from [SparxSystems, 2007], section 16.6.2.1)

6.2.2 EA Software Developers Kit

In addition to the transformation definitions, EA offers a *Software Developers Kit* (EA SDK), which allows for using custom developed program units in the transformation, to fine-tune the transformation in subsequent steps (i.e. the 2nd and 3rd transformation), which cannot be done in the EA transformation definitions (i.e. the 1st transformation). An interface *Interop.EA.dll* has been provided by EA (Figure 26), which offers direct access to model elements.

The interface contains model elements like:

- Element (UML: *Class*)
  - Element *constraints* (e.g. OCL)
  - Element *Tagged Values*
- *Attribute*
  - Attribute *Tagged Values*
- Method (UML: *Operation*)
  - Method *Tagged Values*
- Connector (UML: *Association*, *Generalisation*, *Aggregation*)
  - Connector Tagged Values

Program Units for 2nd and 3rd Transformation

The program units can be called from a separate application (see the custom developed MDA prototype in Figure 31), which handles the 2nd and 3rd transformation. Figure 27 shows an example of a program unit, which adds a tag by the name of *myTagName* with tagged value *myTagValue* to a class *myClass*.

An overview of these program units (in Program Unit Package 'Transformation') is provided in "Appendix E: Example EA MDA Prototype Source Code", as well as the description of Program Unit: GetClassTagValue and Program Unit: ProcessEnumerationClass).

```
// set or overwrite class tag value
public void SetClassTagValue(EA.Element myClass, string myTagName, string myTagValue, bool
overwriteTags)
{
     if (overwriteTags)
     {
          // Delete old attribute tags with that name
          for( short iTag = 0; iTag < myClass.TaggedValues.Count; iTag++ )
          {
               EA.TaggedValue OldClassTag = (EA.TaggedValue) myClass.TaggedValues.GetAt(iTag);
               if (OldClassTag.Name == myTagName)
               {
                    myClass.TaggedValues.DeleteAt(iTag,true);
               }
          }
     }
     EA.TaggedValue NewClassTag = (EA.TaggedValue)
               myClass.TaggedValues.AddNew(myTagName,myTagValue);
     NewClassTag.Update();
     myClass.TaggedValues.Refresh();
}
```

Figure 27 - Example of Program Unit 'SetClassTagValue'

```
Table
{
  %TRANSFORM_REFERENCE("Table")%
  %TRANSFORM_CURRENT("language", "stereotype", "abstract", "name")%
  name = %qt%%CONVERT_NAME(className, "Pascal Case","Underscored")%%qt%
  language=%qt%PostgreSQL%qt%

  %if elemType != "Association"%
  PrimaryKey
  (
    name = %qt%pk_%CONVERT_NAME(className, "Pascal Case","Underscored")%%qt%
    Column
    (
      name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName", classGUID)%%qt%
      type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
    )
  )
  %endIf%

  %list="Attribute" @separator="\n" @indent="  "%
)
%list="Connector" @separator="\n"%
```

Figure 28 - Example of Program Units used by Transformation Definitions / Conversion Template for Class

Program Units for 1st Transformation

Although the majority of the custom developed program units, was used for the 2nd and 3rd transformation, some of the program units can also be used in the previously discussed Transformation Definitions. These program units produce (only) text strings for the Intermediary File. In the conversion template for Class (to table) in Figure 28, the program unit "GetPrimaryKeyColumnName" returns a name for the primary key, for example "*oid*", and "GetPrimaryKeyColumnDatatype" returns the data type, for example "*integer*".

An overview of these program units (in Program Unit Package 'Prototype') is provided in "Appendix E: Example EA MDA Prototype Source Code", as well as the description of Program Unit: transformtoPSM).

6.2.3 OCL in Enterprise Architect

Many UML tools provide some functionality with regard to the specification of OCL constraints on UML models, however, most of them don't offer validation and (type) checking, parsing and implementation of the constraints. This also holds for EA. EA offers the possibility to store OCL constraints at the level of classes (EA: Elements), attributes, and associations (EA: Connectors). The OCL constraints can be an invariant (must always be valid), and a pre- and post condition for an operation (must be valid respectively before or after the operation). The transformation of constraints with the transformation templates (first transformation) is not possible in EA, nor is an approach to the implementation of OCL constraints available, requiring the custom development of program units, based on the EA Software Development Kit.

To a certain extent, validation is offered with regard to the formation of the OCL constraint, for example: *Well-Formedness*, which checks whether an element, relationship is well-formed; *Composition*, which checks the element and its children; *Property*, which checks the element and relationship properties, and the *OCL Conformance*, to validate the defined constraints in OCL. Validation occurs at two instances:

- Validation when the OCL constraint is entered.
- Validation when the Model Validation function is executed.

Upon entering OCL constraints at the element level (for example a class), EA successfully validates certain OCL constraints, for example with (currently non-existing OCL spatial operations like ST_Distance, ST_Area).

EA also offers *a Model Validation function*, which can be executed separately from the EA user interface for a complete package or EA project. The OCL constraint that is successfully validated upon entry of the constraint, could now be rejected in the Model Validation function provided by EA (*checking any OCL constraint defined within the model* [SparxSystems, 2007] [page 526]). This could potentially lead to inconsistencies in the OCL part of the model. However, this gap, between *validation upon entry* of the constraint and EA project based *Model Validation*, is (mis-)used in section 6.7, to be able to stored OCL constraints with spatial data types and operations, which are currently not part of the OCL standard [OMG, 2006b], on behalf of OCL experiments.

6.3 MDA Prototype Set-up Based on EA

The MDA prototype will be based on the transformation possibilities of Enterprise Architect, as previously described. On the one hand, the EA Transformation Definition for "DLL" will be used and adapted, to create the "First Transformation from PIM to PSM-1" as described in section 6.5, also referred to as the *"PostgreSQL" Transformation Definition*. On the other hand, program units have been developed to provide the required functionality in the "Second Transformation from PSM-1 to

PSM-2" and in the "Third Transformation from PIM OCL to PSM-2" (respectively described in section 6.6 and 6.7). The "Transformation from PSM to DDL (PostgreSQL/PostGIS)", resulting in DDL and DML scripts, is also handled by the MDA prototype, as discussed in section 7.3.

These program units have been developed in Visual Studio .NET / C#, based on the EA Software Development Kit (section 6.2.2). A Windows Forms user interface (Figure 31) has been developed, which can run independently, or as an "Add-in" for the EA user interface.

The prototype consists of the following parts with about in total 5800 lines of code (10 % unchanged or adjusted code delivered by EA, 90% custom developed), including simple comments and empty lines, addressing the:

- "First Transformation from PIM to PSM-1".
  - Adapted EA Transformation Definition "PostgreSQL"; about 500 lines in the EA proprietary template language (25% unchanged, 50% adapted, and 25% new lines of code).
  - C# program unit package "*PrototypeAddin*" to provide extra functionality in the transformation definitions (e.g. program unit "GetPrimaryKeyColumnName", and "GetPrimaryKeyColumnDatatype"); about 500 lines of code (50% unchanged, 50% new).

- "Second Transformation from PSM-1 to PSM-2", the "Third Transformation from PIM OCL to PSM-2", and the "Transformation from PSM to DDL (PostgreSQL/PostGIS)".
  - C# program unit package "*Transformation*"; about 4800 lines of code (100% new).

Examples of the "PostgreSQL" Transformation Definition are discussed in "Appendix D: Examples of EA Transformation ", and "Appendix E: Example EA MDA Prototype Source Code" will show some examples in C# programming language. The complete "MDA Prototype", consisting of the EA transformation definition "PostgreSQL", the Enterprise Architect Project file, the C# source code, and the DDL and DML scripts to create a PostGIS database, is available at URL 30.

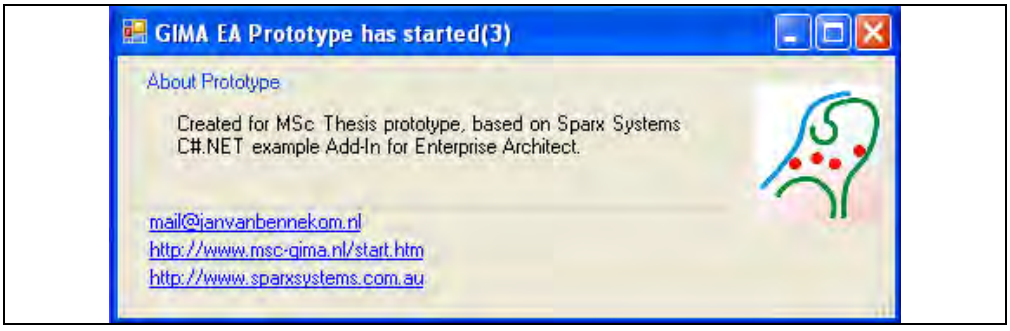


Figure 29 - GIMA EA Prototype Start Dialog Box

Part of the prototype entails a user interface "MDA Transformation prototype (GIMA MSc Thesis)" that can be called from the EA menu (Figure 29 and Figure 30).

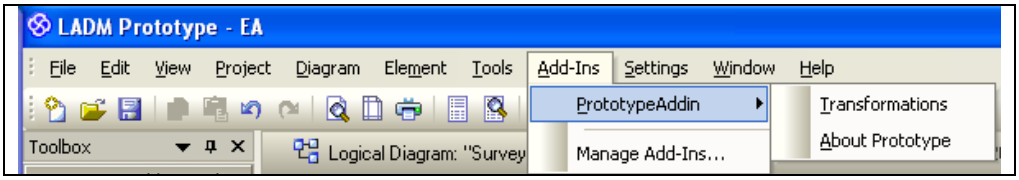


Figure 30 - The Prototype Add-in menu for EA

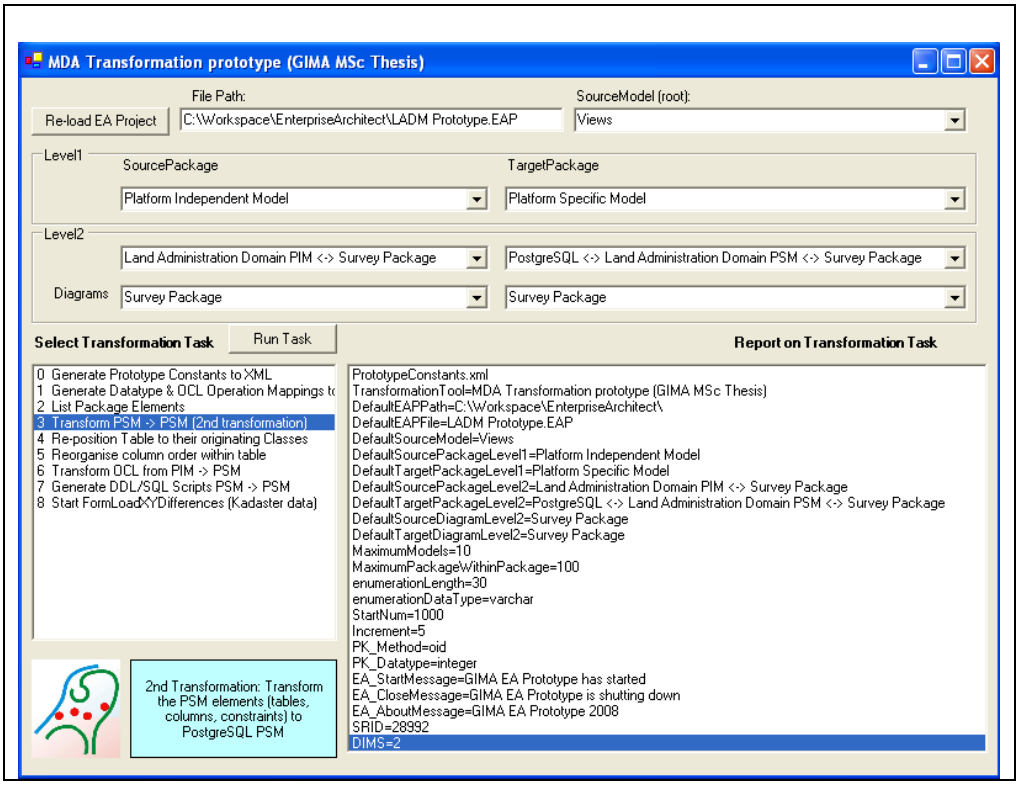


Figure 31 - Prototype User Interface for Transformations

When invoking the prototype's user interface for transformations, a form appears with the possibility of selecting source and target package for the transformation (see Figure 31 and Figure 34), respectively:

- "Land Administration Domain PIM <-> Survey Package" and
- "PostgreSQL <-> Land Administration Domain PSM <-> Survey Package"

A number of Transformation Tasks can be selected and executed, the result of which will be reported in the "Report on Transformation Task" lower-right part of the MDA prototype user interface (Figure 31).

6.3.1 Prototype Constants and Data Type Mapping

Upon start-up, a selection of source and target packages is made, based on Prototype Constants, which, together with other constants such as name and data type of the generated primary keys, are stored in a simple, custom made XML file PrototypeConstants.xml (Figure 32), in the folder "C:\GIMAPrototype". The sole purpose of these files is *to provide constants and prototype settings* (e.g. for Default

name of the Enterprise Architect Project file, i.e. the constant *DefaultEAPFile*), therefore no XML Schema has been defined for these files.

Another XML file, custom made for the prototype, is the file DatatypeMapping.xml (Figure 33), used for determining a mapping between PIM (source) data types (e.g. based on ISO standards), and the PostgreSQL and PostGIS PSM (target) data types (see section 6.6, section "Transform Attribute"). For example the PIM data type "*GM_Point*" (used in class SurveyPoint) is mapped to PSM data type "*POINT*". A similar XML file OclOperationMapping.xml is used for the mapping between OCL and PostgreSQL/PostGIS (spatial and non-spatial) operations.

The argument for using these xml files is that in this manner, the prototype constants, and mapping of data types and operations are not hard coded in the C# prototype source code, and could be changed on behalf of other information systems being modelled in EA. This has been successfully tested within a project for the development of a Land Information System in Ghana, see "Appendix F: Details on First Transformation in MDA Prototype (PIM to PSM-1)", the section "Alternative primary key column data type and name".

```
- <PrototypeAddinConstants version="16-03-2008 15:42:36" usedBy="Transformation/PrototypeAddin">
    <TransformationTool Value="MDA Transformation prototype (GIMA MSc Thesis)" />
    <DefaultEAPPath Value="C:\Workspace\EnterpriseArchitect\" />
    <DefaultEAPFile Value="LADM Prototype.EAP" />
    <DefaultSourceModel Value="Views" />
    <DefaultSourcePackageLevel1 Value="Platform Independent Model" />
    <DefaultTargetPackageLevel1 Value="Platform Specific Model" />
    <DefaultSourcePackageLevel2 Value="Land Administration Domain PIM <-> Survey Package" />
    <DefaultTargetPackageLevel2 Value="PostgreSQL <-> Land Administration Domain PSM <-> Survey Package" />
    <DefaultSourceDiagramLevel2 Value="Survey Package" />
    <DefaultTargetDiagramLevel2 Value="Survey Package" />
    <MaximumModels Value="10" />
    <MaximumPackageWithinPackage Value="100" />
    <enumerationLength Value="30" />
    <enumerationDataType Value="varchar" />
    <StartNum Value="1000" />
    <Increment Value="5" />
    <Sequence_cadastral_office Class="CadastralOffice" Column="oid" Start="16" Increment="1" />
    <Sequence_cadastral_municipality Class="CadastralMunicipality" Column="oid" Start="1300" Increment="1" />
    <Sequence_cadastral_section Class="CadastralSection" Column="oid" Start="7800" Increment="1" />
    <PK_Method Value="oid" />
    <PK_Datatype Value="integer" />
    <EA_StartMessage Value="GIMA EA Prototype has started" />
    <EA_CloseMessage Value="GIMA EA Prototype is shutting down" />
    <EA_AboutMessage Value="GIMA EA Prototype 2008" />
    <SRID Value="28992" />
    <DIMS Value="2" />
  </PrototypeAddinConstants>
```

Figure 32 - Prototype Constants (PrototypeConstants.xml)

```
<DatatypeMapping version="18-4-2008 11:07:11" source="PIM">
  <Boolean Datatype="boolean" />
  <Integer Datatype="integer" />
  <DateTime Datatype="timestamp" />
  <GM_Point Datatype="POINT" Index="INDEX USING GIST" />
  <GM_LineString Datatype="LINESTRING" Index="INDEX USING GIST" />
  <GM_Polygon Datatype="POLYGON" Index="INDEX USING GIST" />
  <GM_MultiSurface Datatype="MULTIPOLYGON" Index="INDEX USING GIST" />
  <Number Datatype="numeric" Precision="15" Scale="3" />
  <Area Datatype="numeric" Precision="14" Scale="2" />
  <CodeString Datatype="varchar" Length="17" />
  <String Datatype="varchar" Length="100" />
  <NotesString Datatype="varchar" Length="2000" />
  <BinaryData Datatype="bytea" />
</DatatypeMapping>
```

Figure 33 - PIM (Source) and PSM (Target) Data Type Mapping (DatatypeMapping.xml)

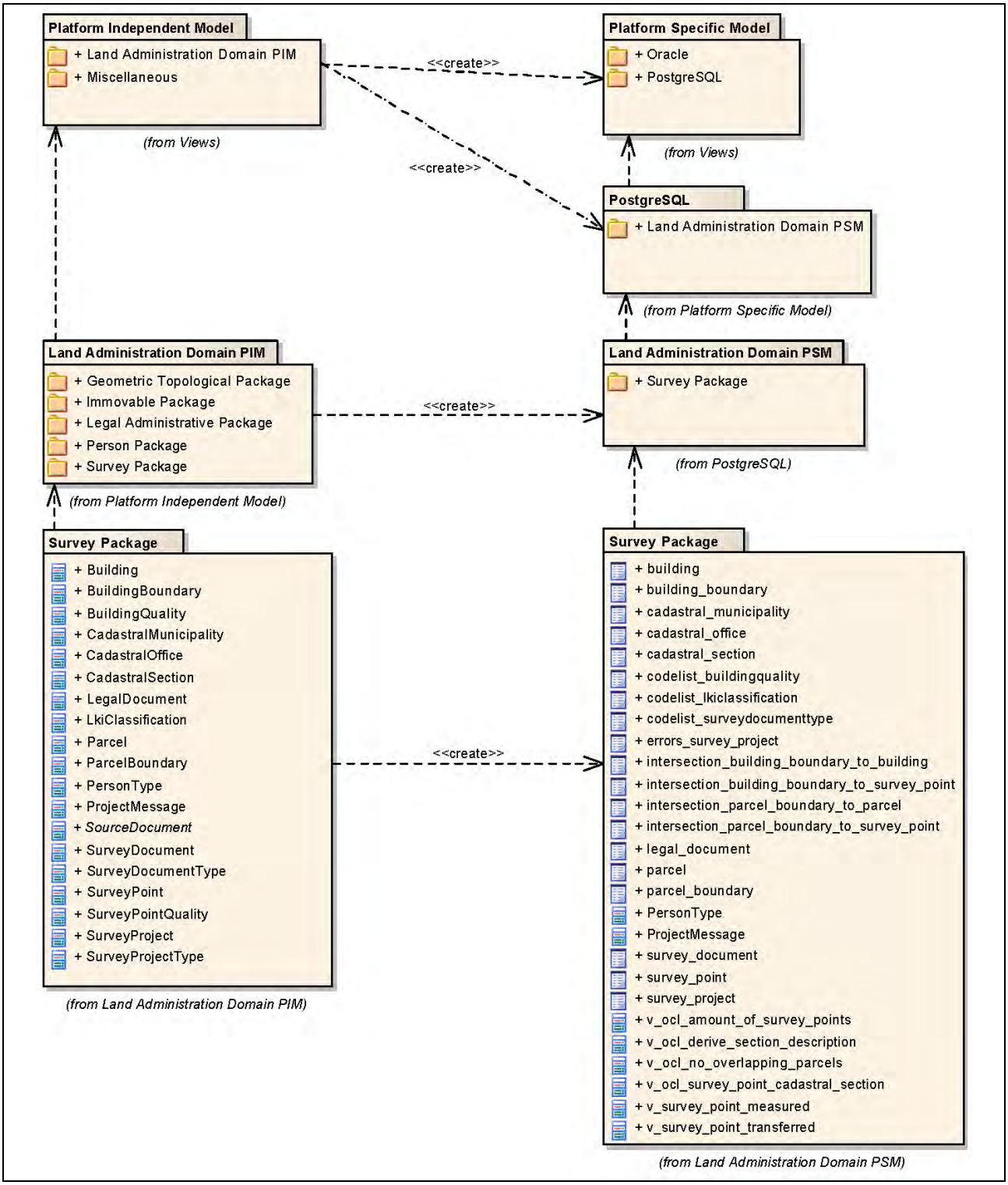


Figure 34 - Prototype Set-up (Package Dependency Diagram) in Enterprise Architect for the Adapted LADM 'Survey Package'

6.3.2 PIM and PSM Setup for Prototype

UML Modelling in EA occurs within the context of a project, and its so-called "Views" (Figure 34). Within these views, packages can be created, which (can) have a certain analogy with the platform independent and specific models, as defined in section 3.2.

In the prototype, a package "Platform Independent Model" and "Platform Specific Model" have been created. The "Platform Independent Model" contains the package "Land Administration Domain PIM" with its sub-packages. Note that the classes, used in the prototype (Figure 20), are grouped in a source package, called **Survey Package**, which diverges from the classes in the Survey Package in the ISO 19152 [ISO/TC211, 2008], as explained in section 5.4. The "Platform Specific Model" could in principle show multiple databases, however, the prototype is aiming at implementing the platform independent model in a PostgreSQL/PostGIS environment. The PostgreSQL package shows the database implementations of the Land Administration Domain PSM, similar to its platform independent counterpart.

6.4 MDA Prototype Transformations

In Chapter 3, the MDA processes from a Platform Independent Model (PIM) to a Platform Specific Model (PSM) are described, which are executed based on a platform specific transformation specification. In the prototype, experiments will be conducted with the transformation from an object-oriented UML model (PIM) to a (object-)relational database model in PostgreSQL/PostGIS (PSM). As discussed in section 3.2.1, this type of transformation is characterised by a certain incompatibility of object and relational models, which should be resolved by the definition of a MDA Transformation Rule for each of the PIM elements. The MDA transformations, using the MDA Transformation rules to implement PIM elements in a PSM, will be discussed in this section.

EA offers two types of transformation functionalities: The *Transformation Definitions* (section 6.2.1) and the *EA Software Developers Kit* (section 6.2.2). After initial experimenting, it became clear that the Transformation Definitions were not capable of executing all the MDA transformation rules that are discussed in the next sections. The EA SDK needs to be involved in the MDA prototype, which can (semi) automatically perform the transformation from an object oriented PIM (e.g. the Adapted LADM 'Survey Package' UML class diagram) to a PSM, targeting an object-relational PostgreSQL database, with a PostGIS extension for spatial data and functions.

Each transformation will satisfy a number of *MDA transformation rules*. Each MDA transformation rule requires a description of a specified action; based on input element (from the PIM or the PSM); resulting in output elements (in the PSM); and a description of the program unit/the tool in which the action is performed. The MDA Prototype is aimed at the creation of new elements in the PSM, and has not been developed nor tested for situations where changes in the PIM are propagated to the (existing) PSM. The following transformation steps have been defined to arrive at an implementation of the Adapted LADM 'Survey Package', in a PostgreSQL/PostGIS database (section 7.3), populated with data, provided by Kadaster (section 7.4):

- **First Transformation from PIM to PSM-1** (section 6.5), based on EA Transformation Definition.
- **Second Transformation from PSM-1 to PSM-2** (section 6.6), based on MDA Prototype in C# / .NET.
- **Third Transformation from PIM OCL to PSM-2** (section 6.7), based on MDA Prototype in C# / .NET.

Note that this definition of transformation steps is a choice of implementation for a MDA Prototype, inspired by the preference to experiment with different variants of EA transformation functionality. Another implementation (i.e. a combination of the 1st, 2nd, and 3rd transformation into one transformation), solely based on the EA Software Developers Kit would also have been possible, but has not been chosen for this MDA prototype. The transformations will be described in the next sections; the following topics are discussed separately:

- Tagged Values
- Transformation of Super and Sub Classes
- Geometry Data types, Indexes and Spatial Constraints
- Transformation of <<enumeration>> and <<CodeList>> Classes
- OCL Implementation

The details and examples of the MDA transformation of the Adapted LADM 'Survey Package' are provided in the appendices:

- Appendix F: Details on First Transformation in MDA Prototype (PIM to PSM-1)
- Appendix G: Details on Second Transformation in MDA Prototype (PSM-1 to PSM-2)
- Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)

The **"Transformation from PSM to** DDL (PostgreSQL/PostGIS)**"** also referred to as the 4th transformation, is also handled by the MDA prototype, but will be discussed in section 7.3. During this transformation the PSM-2 elements will be used to generate DDL and DML scripts, to create the PostgreSQL / PostGIS database.

6.5 First Transformation from PIM to PSM-1

This first transformation step converts the PIM elements to the first version of PSM (PSM-1), preparing for a subsequent transformation to the final PSM (PSM-2).

The first transformation step makes use of the standard EA transformation definition called "DDL", specifically for Relational Databases (RDBMS). The DDL transformation template creates Tables, one-to-one mapped onto Class elements, Columns, one-to-one mapped onto attributes, Primary Keys for each table, and Foreign Keys. This template has been changed and extended for the transformation to a PostgreSQL/PostGIS database, requiring knowledge of the programming language in the EA standard transformation definition, referred to as "EA's simple code generation template language" [SparxSystems, 2007]. This transformation is performed with the Transformation Definition "PostgreSQL". From all the MDA

transformation rules, to arrive at a final PSM, the first transformation takes care of the MDA Transformation Rules, listed below. Note that MDA Transformation Rules are implementation choices, and could in some cases be performed with other methods of implementation.

- Create Target Package and Target Platform
  A target package to hold the PSM for the target platform (PostgreSQL) platform is created.

- Copy Source Structure to Target Package Structure
  The structure of packages and namespaces in the PIM will be copied/used in the PSM, see Figure 34.

- Transform Classes (Stereotyped <<enumeration>> or <<CodeList>>)
  Enumeration and CodeList classes will be temporarily transformed to classes (instead of tables) in the PSM, to be used and cleaned up (deleted) in the second transformation to create check constraints and look-up tables.

- Transform Class to Table
  The PIM classes, marked with tagged value "*TransformToPSM*" will be transformed to tables in the PSM, with lowercase table names (words separated by underscores). See "Appendix D: Examples of EA Transformation Definition 'PostgreSQL'" for the transformation template, used for classes.

- Transform Attribute to Column
  The attribute names are changed to underscored, lowercase column names, the (spatial) data types will be transformed in the second transformation.

- Generate Primary Key
  For each table, a primary key and column are being created. Also, sequences are defined, providing values for the primary key column at record creation.

- Transform Associations to Relationships or Tables
  Associations in the PIM will be transformed to RDBMS relationships or "intersection" tables to represent a many-to-many association, based on multiplicity and association type. See "Appendix D: Examples of EA Transformation Definition 'PostgreSQL'" for the transformation template, used for associations.

The details on the first transformation step are provided in "Appendix F: Details on First Transformation in MDA Prototype (PIM to PSM-1)", the use of tagged values in the first transformation is highlighted in the next section.

6.5.1 Tagged Values

The UML model can be extended with tagged values, which in the MDA prototype are used to support and influence various transformations. Some examples:

- **ImplementedInSubClass**; The super class is not implemented as table, all its attributes, operations and associations are inherited by subclasses.

- **IndexUsingGIST**; The file "DatatypeMapping.xml" (section 6.3.1) determines for the data types GM_Point, GM_LineString, GM_Polygon, and GM_MultiSurface the PostGIS data type together with an attribute Index="INDEX USING GIST", indicating the use of a GIST index (see section 6.6.2).
- **MarkForDeletePSM**; This element is marked for deletion in the PSM.
- **PartofUniqueKey**. This tag indicates if the attribute is part of a unique key, to workaround a specific transformation problem. The attribute property IsStatic (PIM) is used to indicate uniqueness (PSM), but cannot be transformed as property of a column.
- **TransformedFromAttribute**; Indicates the originating attribute for a column.
- **TransformedFromClass**; Indicates the originating class for a table, to establish a link between PIM and PSM elements on behalf of the MDA prototype.
- **TransformedFromRelationship**; Indicates the originating association for a relationship.
- **TransformToPSM**; Class is selected in the PIM for transformation to the PSM.

See Figure 27 for a program unit, capable of setting a tagged value for a class and Figure 81 in the "Appendix E: Example EA MDA Prototype Source Code" for a program unit, capable of retrieving tagged values for classes.

The EA software uses of tagged values to achieve functionality that can, apparently, not be offered by the standard EA (SDK) data model. One example is the PSM element *sequence*, for an attribute, which is implemented as a tagged value tagged value "**Property**" with a value like: "*AutoNum=1; StartNum=1; Increment=1; NotForRep=0;*" (see "Appendix F: Details on First Transformation in MDA Prototype (PIM to PSM-1)" for more details on sequences). Another example is the use of tagged values "**view def**" and "**proc def**", respectively to store the DDL create statements for creating these views and stored procedures in the target platform.

6.6 Second Transformation from PSM-1 to PSM-2

As indicated in section 6.4, the EA transformation definitions (used in the previous section 6.5) do not offer all the required transformation functionality for the model elements (classes, attributes, associations, OCL constraints). The second transformation addressed this requirement, and converts the first version of PSM (PSM-1) to the transformation to the final PSM (PSM-2), to deal with all MDA transformation rules, that the first transformation couldn't offer. The second transformation is fully based on functionality, provided by the EA Software Developers Kit (i.e. in .NET / C#). The second transformation takes care of the following MDA Transformation Rules:

- Set Column "Not Null" property
  The "Not Null" property of the columns are set based on Lower and Upper Bound properties of attributes in the PIM, or based on the multiplicity of related associations in the PIM

- Process columns defined by <<Type>> classes

Columns that have a data type, corresponding to a class, stereotyped as <<type>> will lead to different implementations based on the Lower bound and Upper bound properties, for example [0..*].

- Transform Attribute
  Based on a mapping of PIM to PSM data types, column data types and lengths are defined.

- Transform Classes, stereotyped <<enumeration>> and <<CodeList>>
  The temporary <<enumeration>> and <<CodeList>> classes will be transformed to check constraints and look-up tables. See Figure 82 in "Appendix E: Example EA MDA Prototype Source Code" for the program unit, used for enumeration and CodeList classes.

- Create Uniqueness Constraint
  Based on the tagged value "PartofUniqueKey" a unique key will be created.

- Re-organise Order of Columns within Table
  The position (order) of columns within a table will be changed based on a defined MDA rule (Primary Key, mandatory Foreign Key, mandatory, optional Foreign Key, and optional columns.

- Implement Super Class in Sub Class (Table)
  Based on tagged value "ImplementedInSubClass", columns and relationships will be inherited by sub classes (tables).

The details on the second transformation step are provided in "Appendix G: Details on Second Transformation in MDA Prototype (PSM-1 to PSM-2)", a number of MDA transformation issues are highlighted in the next sections:

- Transformation of Super and Sub Classes
- Geometry Data types, Indexes and Spatial Constraints
- Transformation of <<enumeration>> and <<CodeList>> Classes

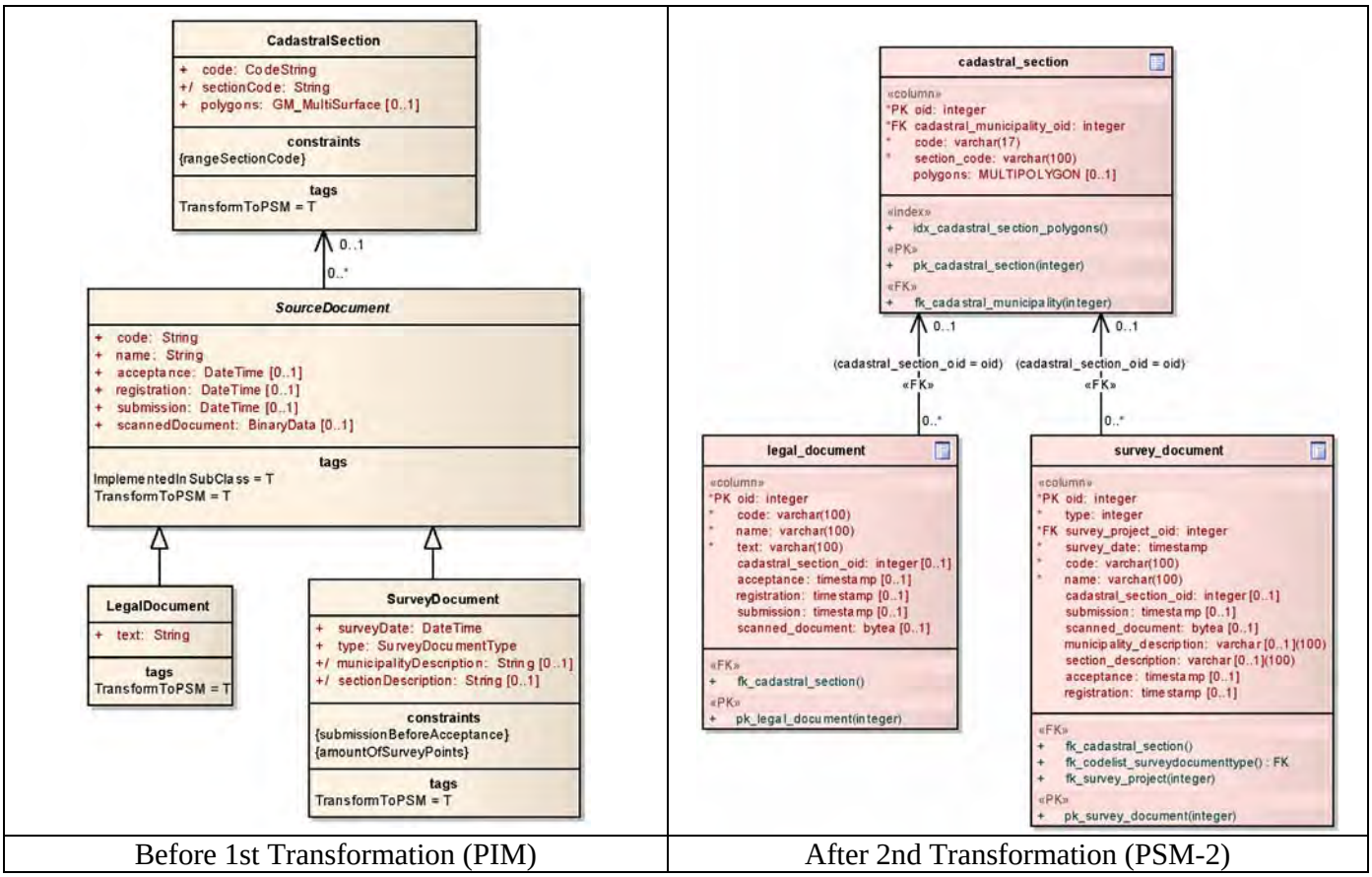
6.6.1 Transformation of Super and Sub Classes

Relational databases do not support the (object-oriented) concept of inheritance. For super and sub classes in a PIM, a decision (MDA Transformation Rule) has to be made on how to convert these to a PSM. The abstract Super class *SourceDocument* and sub classes *LegalDocument* and *SurveyDocument* as presented in Figure 35 (left side) can be transformed to tables in three different manners:

- One Table for One Class
- One Table for One Class Hierarchy Branch (flattening)
- One Table for One Class Hierarchy

One Table for One Class
Each class in the class hierarchy of Super and Sub classes is transformed to one table. For example, Super class *SourceDocument* and Subclasses *LegalDocument* and *SurveyDocument* will be mapped to the tables *source_document*, *survey_document*, and *legal_document*. This transformation is performed in the first transformation; see Figure 88 (PIM on the left, and PSM on the right side). For each generalisation, a foreign key and column will be generated, for example "*fk_source_document*", and column "*source_document_oid*".

One Table for One Class Hierarchy Branch (flattening)
For each branch in the class hierarchy, a table is created with all the attributes within that branch, also referred to as "flattening" of class hierarchies. For example, the transformation of one super class (*SourceDocument*) and two sub classes (*LegalDocument* and *SurveyDocument*) into two implemented tables (*legal_document* and *survey_document*), where all attributes (*code*, *name*, *acceptance*, *registration*, *submission*, *scannedDocument*) and outbound foreign keys (*fk_cadastral_section*) are inherited, in addition to their own (Figure 35). This transformation can be performed by the "Second Transformation from PSM-1 to PSM-2", based on a tagged value "ImplementedInSubClass", at super class level.



Before 1st Transformation (PIM)	After 2nd Transformation (PSM-2)

Figure 35 - 2nd Transformation (PSM-1 to PSM-2): Implement Super class in Sub class

The class *LegalDocument* is not part of the LADM 'Survey Package', but has been included to demonstrate the "One Table for One Class Hierarchy Branch" capability

of the prototype. Also note that the prototype can currently only deal with 2 level class hierarchies, but technically it would be possible to use hierarchies of more than two levels.

Another MDA transformation rule could have been designed based on the "abstract" property of classes, however a choice for implementation as described above was made, that enables an approach which offers, from the perspective of the MDA user of the prototype, more flexibility and influence based on tagged values. The level of influence is needed because otherwise the last two options may lead to undesirable results; consider for example class hierarchies with multiple levels.

One Table for One Class Hierarchy
One Table will be generated for all the classes within the hierarchy, holding all attributes, operations and associations of all classes together. The result will be a class with many optional columns (i.e. "Not Null" column property is false), requiring the creation of (dependent on the choice of implementation/MDA transformation rule), a new column indicating the type or inherited class of the object that is being instantiated, along with constraints enforcing certain (optional) columns to be "Not Null", dependent of the value of this new column. This option has not been implemented in the prototype, but would technically not be very complex.

6.6.2 Geometry Data types, Indexes and Spatial Constraints

The standard data type in PostGIS is the "geometry", and the ISO19107 data types GM_Point, GM_LineString, GM_Polygon, GM_MultiSurface [ISO/TC211, 2003b], used in the PIM, could be transformed into this PSM data type "geometry". In addition, constraints would have to be defined, to make sure that only geometries of those types will be stored, for example when the "geometry" column should only contain "Points". However, in PostGIS there is a possibility to use geometry specific data types in DDL scripts, for example the data types POINT, LINESTRING, POLYGON and MULTIPOLYGON. These data types, used in the PSM, are defined in the Simple Feature Access for SQL [Open Geospatial Consortium, 2006c], the use of which will generate and implement spatial constraints. Note that the *data types* from ISO/IEC SQL/MM, prefixed with "ST_" are not used by PostGIS [ISO/IEC, 2006], as opposed to the spatial *operations*.

For example, a POINT data type for a 2 dimensional column '*location_measured*' of table '*survey_point*', with a spatial reference 28992 (the Dutch RD spatial reference system), would be created (added to a table) with statement (based on settings in the prototype constants for SRID [spatial reference system identification] and DIMS [dimension] in Figure 32):

```
select addgeometrycolumn ('survey_point','location_measured',28992,'POINT',2);
```

Note that this type of statement should preferably be used to add a geometry column, because only then will the table *geometry_columns* (with the definition of all spatial columns, [Open Geospatial Consortium, 2006c], [Chapter 6]) be updated accordingly. This table *geometry_columns* is required for other applications such as uDig (section 7.2.2, URL 25). This aspect of PostGIS, is one of the reasons for making MDA prototype functionality to create DDL implementation scripts, instead of using the EA

functionality for that, see section 7.3, describing the "Transformation from PSM to DDL (PostgreSQL/PostGIS)".

The above mentioned select statement will automatically create a number of spatial constraints with the following alter table statements:

- Enforce the geometry to be stored in 2 dimensions.

```
ALTER TABLE survey_point
ADD CONSTRAINT enforce_dims_location_measured
CHECK (ndims(location_measured) = 2);
```

- Enforce the geometry to be empty or to contain only POINT elements.

```
ALTER TABLE survey_point
ADD CONSTRAINT nforce_geotype_location_measured
CHECK (geometrytype(location_measured) = 'POINT' OR location_measured IS
NULL);
```

- Enforce the spatial reference system 28992 to be used.

```
ALTER TABLE survey_point
ADD CONSTRAINT enforce_srid_location_measured CHECK (srid(location_measured) =
28992);
```

Spatial Indexes
The XML file DatatypeMapping.xml, as described in section 6.3.1 (Figure 33) also contains information about the index to be used for certain spatial columns, for example for PIM data type GM_point (PSM data type POINT):

```
<GM_Point Datatype="POINT" Index="INDEX USING GIST" />
```

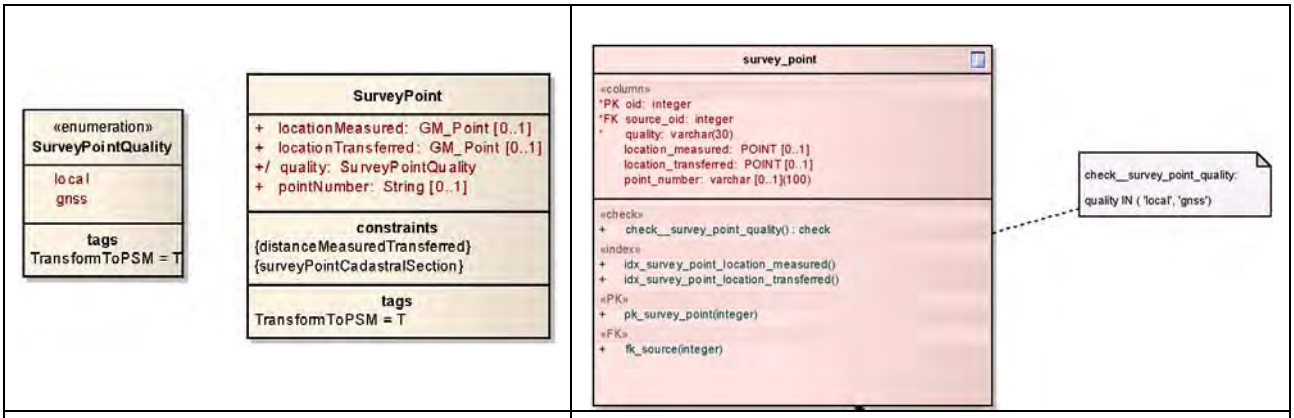
The prototype will use the setting "INDEX USING GIST" to generate PostGIS GIST indexes, such as indexes idx_survey_point_location_measured and idx_survey_point_location_transferred in Figure 36.

6.6.3 Transformation of <<enumeration>> and <<CodeList>> Classes

The MDA transformation rule for enumeration and CodeList classes has been preliminary reported through the article by Hespanha et al. [Hespanha et al., 2008]. In the first transformation the classes, stereotyped <<enumeration>> and <<CodeList>> are transformed, or actually copied. The stereotype classes <<enumeration>> and <<CodeList>> are both used in the LADM (PIM) for indicating a list of allowed values. As enumeration types are considered to be part of the model (fixed list of values) and code lists are considered to be part of the application (open, changeable list of values) they are treated differently by the MDA transformation rule. See also Figure 82 in "Appendix E: Example EA MDA Prototype Source Code".

Enumeration Classes
For a <<enumeration>> class a check constraint is added on the attribute and hard coded in the check are the allowed values. See Figure 36, where the check constraint

"check_survey_point_quality" is generated, based on enumeration class SurveyPointQuality.

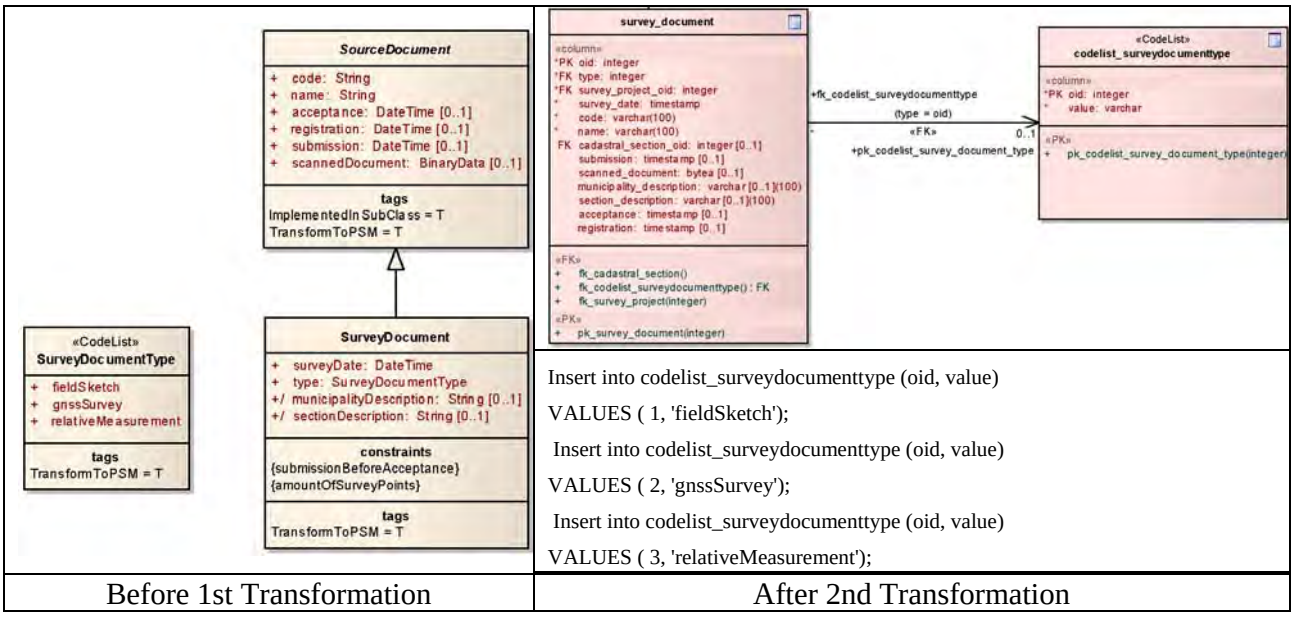


Before 1st Transformation	After 2nd Transformation

Figure 36 - 2nd Transformation (PSM-1 to PSM-2): <<enumeration>> Class

CodeList Classes

For a <<CodeList>> class, a (look-up) table with the allowed values is defined, as well as a foreign key constraint from the original table to the look-up table. For example the <<CodeList>> SurveyDocumentType (Figure 37) has been converted into a lookup table "codelist_surveydocumenttype", used to fill the column "type", and a DML script, to insert the initial values into the lookup table, has been generated.



Before 1st Transformation	After 2nd Transformation

Figure 37 - 2nd Transformation (PSM-1 to PSM-2): <<CodeList>> Class

6.7 Third Transformation from PIM OCL to PSM-2

As discussed in 4.2.1, an alternative classification of constraints will be investigated from the Platform Specific Viewpoint, in this case the PostgreSQL / PostGIS platform. For every type of constraint, considerations with regard to the notation in the PIM (UML/OCL), and implementation method in the PSM and PostgreSQL/PostGIS will be shown.

In the prototype the focus has been set on a limited set of examples of OCL *invariant* constraints on classes, making use of the gap between the EA validation upon entering OCL constraints on elements on the one hand, and the Model Validation that can be invoked for a complete EA project on the other hand. For example, certain SQL/MM operations like ST_Distance, ST_Intersects [ISO/IEC, 2006], can be used in OCL constraints, receiving a "OCL Validation successful" remark upon entry of the constraints, which would be reported as false by the EA Model Validation functionality. This allowed for testing of MDA Transformation Rules for spatial constraints, stored at the level of classes in a PIM.

In the next section, research into the application of OCL constraints has been done, based on examples of OCL invariants within a number of classes. Also, investigation of the constraint classifications from a platform specific viewpoint has been performed. Creating a complete OCL parser, as described by Demuth [Demuth et al., 2005], that allows for transforming OCL constraints, as well as implementing the OCL constraints in the target platform (i.e. PostgreSQL/PostGIS), would be a considerable task, which does not fit into the scope of this master thesis project. Therefore a limited set of OCL related transformation functionality has been investigated; the prototype offers the following:

- Automatic implementation of OCL invariants through generation of base table *check constraints*, for a limited set of OCL operations (e.g. toUpper, IsEmpty, toLower, but also spatial operations as ST_Area), which is described in the xml file OclOperationMapping.xml.
- The implementation of OCL constraints as *"OCL views"* or *"constraint views"*, based on the Dresden approach (URL 20), which will show the violating records and can be used in constraint implementation frameworks (see section 6.7.1).
- Conversion of OCL constraints from the PIM to the PSM, accounting for the changes in class and attribute names (to respectively table and column names), on behalf of *manual implementation*.

If constraints can be modelled in both UML and OCL, the UML variant of the constraint will be maintained, because OCL is considered to describe additional object constraints, which cannot be modelled in UML [OMG, 2006b] [section 7.1]. Special attention has been paid to geographic constraints; a few examples have been given, based on the Adapted LADM 'Survey Package' objects. The below-mentioned categories and sub-categories of constraints for spatial and non-spatial data types and operations will be described. The constraints labelled with "OCL" are examples of OCL, with the purpose of demonstrating MDA and OCL related functionality.

- Constraints Applicable to One Instance of One Class
  Constraints that can be checked by only assessing attributes of one instance (tuple) of one class. Note that these constraints should also hold for the constraints that apply to multiple (all) instances of one class, but they can be checked at record individual basis. Examples that are being discussed in "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)":

  - Mandatory Attribute (Not Null)
  - Maximum Attribute Length
  - Range (OCL, example alphanumeric range)
  - Domain (list of possible values)
  - Autonumber
  - Format (OCL, example: Upper)
  - Tuple (OCL, example: IsEmpty & ST_Area)

- Constraints Applicable to Multiple Instances of One Class
  Constraints that must be checked by addressing/knowing attributes of multiple instances of one class. Examples that are being discussed in "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)":

  - Primary Key Constraint
  - Unique Key Constraint
  - Other (OCL, example: overlapping parcels)

- Constraints Applicable to Multiple Instances of Multiple Classes
  Constraints that must be checked by querying the attributes of multiple instances of multiple classes. Most of these constraints, if defined in OCL, can be implemented with OCL views (see section 4.2), see the examples of OCL views that are being discussed in "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)":

  - Foreign Key Constraint
  - Relationship Cardinality (OCL)
  - Derivation (OCL)

The details on the third transformation step are provided in "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)", one of the examples is highlighted in the next section.

6.7.1 OCL Implementation

In the previous section, the following types of implementation of OCL constraints were mentioned:

- Table Check Constraint
- OCL View
- Manual Implementation

Table Check Constraint

Most of the constraints that apply to one instance of one class, if defined in OCL, can be implemented with base table check constraints. See the example for a "tuple" rule (*distanceMeasuredTransferred*) involving more than one attribute for the same record: the *locationMeasured* and *locationTransferred* of a *SurveyPoint* should be *within* 5 meters of each other (*distance* is smaller than 5), see section 7.5.1 for some background information on outliers in connection point data.

```
context SurveyPoint
inv distanceMeasuredTransferred:
ST_Distance(self.locationMeasured, self.locationTransferred) < 5
```

Figure 38 shows the tuple OCL constraint "*distanceMeasuredTransferred*" as it is entered in the EA user interface for maintaining the class SurveyPoint in the PIM.

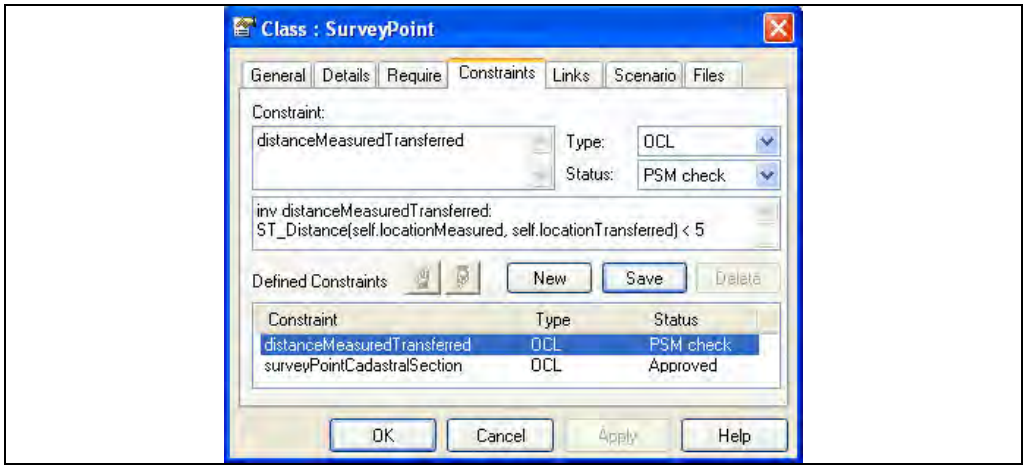


Figure 38 - Constraint Property "Status": "PSM check"

The "Status" property of the constraint, which is set to "PSM check", determines that the OCL constraint will be implemented as base table check constraint *check_distance_measured_transferred*. First, the OCL constraint has been transformed to a PSM invariant, to account for the changed class and attribute names to the transformed table and column names, i.e. *survey_point*, *location_measured* and *location_transferred*. Note the two different methods for the implementation of this constraint based on operations ST_Distance, and ST_DWithin. Then the check constraint is created which would be implemented in PostgreSQL as:

```
ALTER TABLE survey_point ADD CONSTRAINT check_distance_measured_transferred CHECK (
ST_Distance(location_measured, location_transferred) < 5 );
```

or as

```
ALTER TABLE survey_point ADD CONSTRAINT check_distance_measured_transferred CHECK (
ST_DWithin(location_measured, location_transferred, 5)
```

OCL View
For the OCL invariants, applicable to multiple instances of one or more classes, an approach based on OCL Views can be applied. The OCL view is one of the possibilities to implement an OCL invariant, as described in section 4.2 (for OCL view *v_ocl_amount_of_survey_points*). A number views have (manually) been defined to demonstrate the OCL view concept.

Manual Implementation
Within the scope of the master thesis project, transforming and implementing all constraints was not feasible. The MDA prototype does provide functionality to convert OCL based on PIM elements, to OCL based on PSM elements, to account for the changes in class and attribute names (to respectively underscored, lower case table and column names). These non-implemented constraints will be listed based on PSM classes and attributes, serving as the basis for further manual implementation, see section 7.3, section "Present OCL Constraints" (semi-automatic approach). Note that the MDA prototype cannot transform OCL constraints with regard to associations, implemented as "intersection" tables (section 6.5, section "Transform Associations to Relationships or Tables").

6.8 Transformed Adjusted LADM 'Survey Package' (PSM-2)

An overview of the Platform Specific Model for the Adapted LADM 'Survey Package' after the 2nd and 3rd transformation (PSM-2, as input to the deployment of the Adapted LADM 'Survey Package' in Chapter 7) is provided in the next figures, compare the PIM for the Adapter LADM 'Survey Package' in Figure 20. Some examples of the previously discussed MDA transformation rules have been indicated in the figures:

- Figure 39 - The LADM SP PSM-2 - part 1
  1. Type *ProjectMessage* implemented as table **errors_survey_project**.
  2. Type *PersonType* implemented as **survey_project.surveyor**, referring to PostgreSQL type **PersonType**.
  3. Generated primary key **pk_survey_project**.
  4. Optional column (no asterix, "Not Null" is false) for **survey_project.end_date** based on lower and upper bound setting [0..*].
  5. CodeList *SurveyDocumentType* implemented as look-up table **codelist_surveydocumenttype**.
  6. Ordering of **survey_document** columns based.
  7. Inheritance of class *SourceDocument* into table **legal_document** (e.g. column **acceptance** and association/relationship **fk_cadastral_section** to **cadastral_section**).

- Figure 40 - The LADM SP PSM-2 - part 2
  8. Implementation of data type *GM_Point* as spatial PostGIS data type **POINT** in **survey_point.location_measured**.
  9. Implementation of <<enumeration>> class *SurveyPointQuality* as check constraint **check_survey_point_quality** for table **survey_point**.
  10. Generation of unique key constraint **uk_parcel_code** on table **parcel**.
  11. Generation of spatial index **idx_survey_point_location_measured** based on spatial data type *GM_Point*.
  12. Propagation from PIM to PSM of non-implemented OCL constraint **surveyPointCadastralSection**.
  13. Implementation of spatial OCL constraint *areaPolygon* as check constraint **check_area_polygon** for table **parcel**.
  14. Implementation of (many-to-many) association between Parcel and ParcelBoundary as table **intersection_parcel_boundary_to_parcel**.

- Figure 41 - The LADM SP PSM-2 - part 3
  15. The views **v_survey_point_measured** and **v_survey_point_transferred** have been created for uDig to be able to present the spatial data (section 7.2.2)
  16. **OCL views** as part of an implementation of OCL constraints (see section).

.

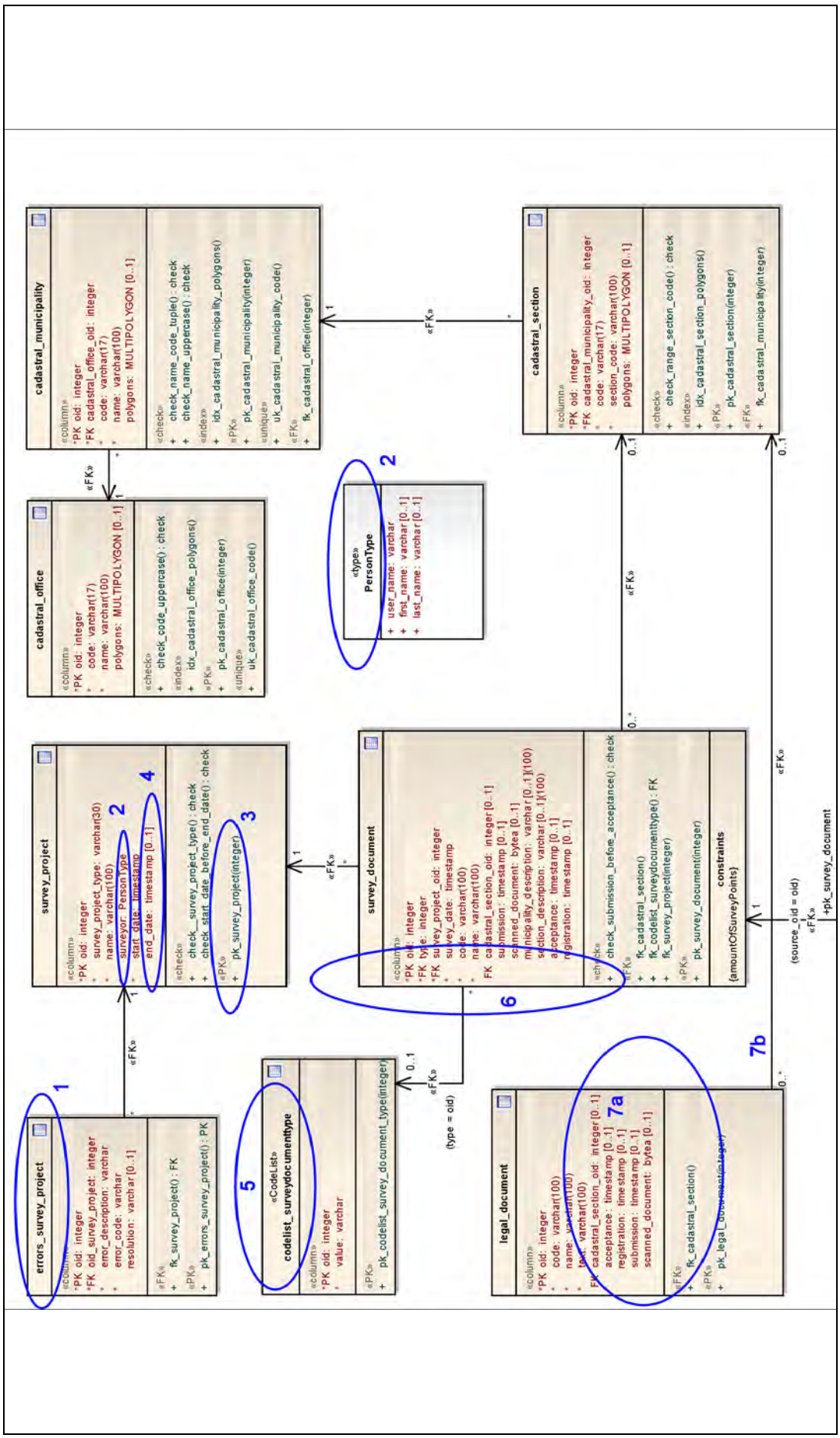


Figure 39 - The LADM SP PSM-2 - part 1

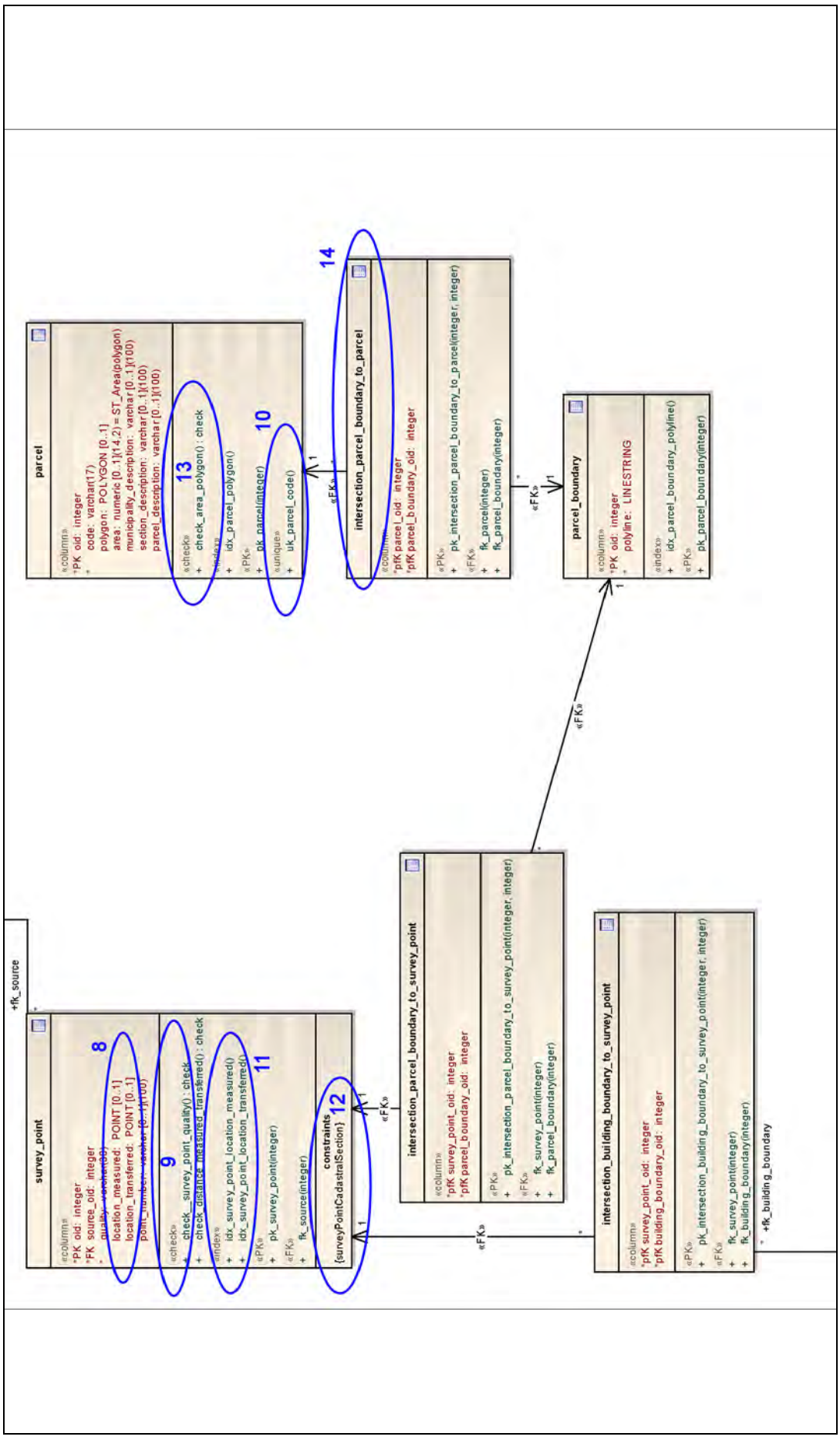


Figure 40 - The LADM SP PSM-2 - part 2

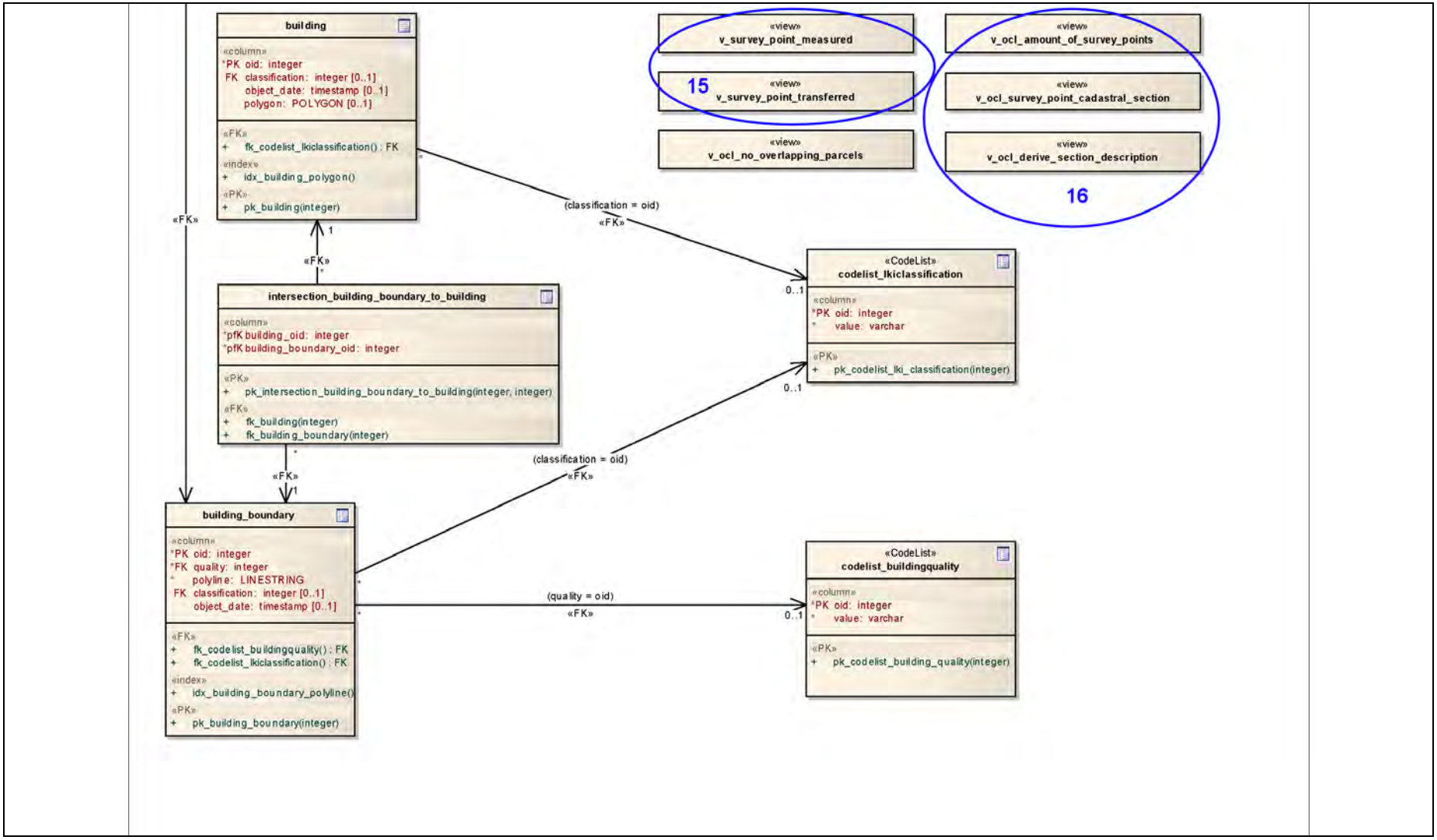


Figure 41 - The LADM SP PSM-2 - part 3

6.9 Conclusion

The standard transformation in Enterprise Architect (EA) from PIM to PSM is based on the Transformation Definitions, more advanced possibilities are provided based on the EA Software Developers Kit (SDK). A choice was made for building a MDA prototype (.NET / C#), to use and experiment with both types of MDA functionalities offered by EA. These MDA possibilities are comprehensive, but require a *considerable amount of custom development*. EA offers import and export of custom developed code templates, but limited support and availability of code templates for relational databases as target platform have been experienced. The MDA prototype can (semi) *automatically perform the transformations* from an object oriented PIM UML class diagram (i.e. the Adapted LADM 'Survey Package', Figure 20) to a PSM UML class diagram, targeting a object-relational PostgreSQL database, with a PostGIS extension for spatial data and functions (Figure 39 - Figure 41).

With regard to constraints, a classification has been made, related to the method of implementation in the target relational database. Relational databases offer non-OCL related functionality to implement constraints like mandatory column, default value for column, column maximum length and precision, primary key constraint, unique key constraint, foreign key constraint, and simple base table check constraint. For other types of constraints, examples of OCL invariants have been defined, for implementation experiments with the MDA prototype.

The *first transformation* offered by the MDA prototype (for about 50% based on *standard EA Transformation Definitions*) handles the relatively straightforward transformation to PSM implementations (tables, columns, primary and foreign keys). The *second transformation* (*fully custom developed*, based on the EA SDK) provides transformation of specific column properties, spatial data types and indices, enumeration and CodeList classes, unique constraints, and flattening of class hierarchies. The *third transformation* of the MDA prototype (*custom developed*, based on EA SDK) is based on initial experimenting with (spatial) OCL invariants in three ways; implement the OCL invariant as a (spatial) base table check constraint; or implement the invariant as an OCL view to be used in transaction management mechanisms (see section 4.2); or "translate" the invariant to an OCL constraint using the PSM element names, as documentation for manual implementation in a PSM. The MDA prototype is also capable of *transforming PSM elements to DDL and DML* scripts (URL 30), which are used to create the Adjusted LADM 'Survey Package' in the PostgreSQL/PostGIS database.

The MDA prototype is based on MDA concepts, for example the *platform specific transformation specification*, which consists of a set of MDA Transformation Rules, applicable to specific PIM elements, resulting in a PSM implementation for each of these PIM elements. If the "gap" between object-oriented (PIM) and relational DBMS (PSM) is not too big, the transformation can be relatively easy. When the difference between PIM and PSM elements is significant, a more complex implementation choice will have to be made. These (arbitrary) implementation decisions could potentially raise criticism, which would have to be settled with alternative MDA

Transformation Rules and implementations. Based on the executed custom development, the conclusion can be drawn, that most MDA Transformation Rules can be handled in a (semi) automatic way. Other MDA Transformation Rules for other PIM elements, offering other kinds of implementations, can be identified, but have not been assessed as part of the MDA prototype.

Fully automatic transformation/generation is justified by the argument of being cost-effective, but the objections of users to commit to such a generation, might be fed by the lack of influence they have on the implementation choices. The use of tagged values in this custom development (see the platform specific transformation specification in section 3.2), provides user control over the MDA process, implying more influence on the transformation to a PSM, and on the implementation in the target platform.

7 Deployment of the Adapted LADM 'Survey Package'

7.1 Introduction

With regard to the Adapted LADM 'Survey Package', transformed to a PSM, as described in the previous chapter the following issues will be addressed:

- Open Source Tools
- Transformation from PSM to DDL (PostgreSQL/PostGIS)
- Populating the PSM in PostgreSQL/PostGIS with Data
- Analysis Connection Points

First the open source tools, which have been used, will be mentioned, one of them being the target platform: PostgreSQL/PostGIS. The final PSM (PSM-2, section 6.8) of the Adapted LADM 'Survey Package' will be implemented by the MDA prototype in the PostgreSQL/PostGIS database. The PostGIS database will be filled with data, as provided by the Netherlands' Cadastre, Land Registry and Mapping Agency (Kadaster). Especially the connection points will be prepared and analysed, which will result in a statement on the quality and accuracy of the cadastral map.

7.2 Open Source Tools

Ingvarson discusses open-source and its geo applications [Ingvarsson, 2005], [Chapter 6]; he concludes that open source can more and more be considered as an alternative to commercial-off-the shelf GIS software and databases. Open source tools offer the source code of software under a certain type of license, which is free of charge and allows for modification of the software and distribution of the open source software. A characteristic of open-source software is that it addresses international standards, specifications and trends, often better and sooner than their commercial counterparts [Ingvarsson, 2005], for example OGCs Simple Feature Specification for SQL [Open Geospatial Consortium, 1999], GML [ISO/TC211, 2006, Open Geospatial Consortium, 2002]. The open source tools PostgreSQL and PostGIS, uDig, and FWTools have been used extensively in the master thesis project.

7.2.1 PostgreSQL and PostGIS

The implementation of the Adapted LADM 'Survey Package' will be done in the open source object-relational database PostgreSQL with extension PostGIS. PostGIS uses both the definitions in Simple Feature Access for SQL (SFA-SQL) [Open Geospatial Consortium, 2006c] and the SQL/MM *operations* [ISO/IEC, 2006]. In the prototype the *data types* Point, Linestring, Polygon, MultiPolygon are used from SFA-SQL, which are supported by SQL/MM routines, such as ST_Dimension, ST_GeometryType, ST_AsText, ST_IsSimple, ST_Envelope, ST_Intersects, ST_Overlaps, ST_Relate, and ST_Distance. Support for topology in PostGIS is under development, but not available in the version of PostGIS that was used (PostgreSQL version 8.2.5 and PostGIS version 1.3.2).

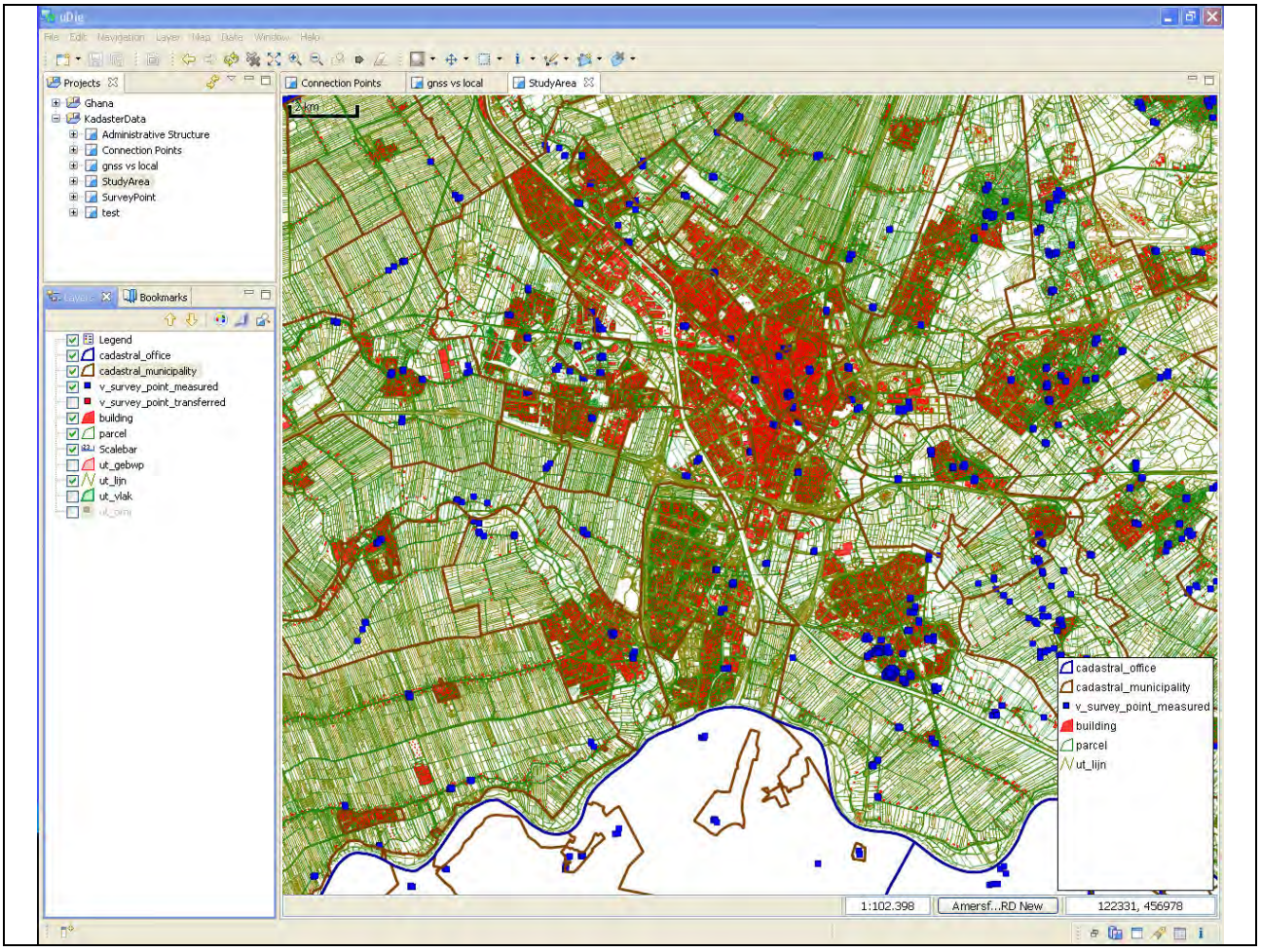


Figure 42 - uDig Screenshot (showing part of Province of Utrecht, with Measured Survey Points)

7.2.2 uDig

For visualisation of geographic data, desktop application uDig has been used in the project (URL 25). uDig is a rich client desktop GIS application, developed in Java on the Eclipse platform (URL 28, Figure 42), which can be used as a stand-alone application, as has been done in the prototype, or uDig can be extended and/or used in other Java applications. uDig can use and visualise several geographic formats, such

as ArcSDE, Oracle Spatial, PostGIS, and Web Map Server (WMS), and Web Feature Server (WFS).

Note: for tables with more than one spatial column, uDig cannot visualise all spatial columns. For table **survey_point**, containing two spatial columns **location_measured** and **location_transferred**, two views had to be created for uDig to be able to present both columns; One view **v_survey_point_measured** with column *location_measured* (and not location_transferred) and **v_survey_point_transferred**, with only spatial column *location_transferred*.

7.2.3 FWTools

FWTools is a set of Open Source GIS binaries produced by Frank Warmerdam (URL 22). Especially the GDAL/OGR library has been used to convert spatial datasets from one format to another (for example MapInfo *.TAB to PostGIS). GDAL is the abbreviation of Geospatial Data Abstraction Library and is a translator library for raster geospatial data formats (URL 23), currently under the responsibility of the Open Source Geospatial Foundation. Part of GDAL is the OGR Simple Features Library (URL 24), which provides access to a variety of vector file formats, such as "ESRI Shapefile", "MapInfo File", "DGN", "GML", "KML", "PostgreSQL", and "Oracle Spatial". Specifically the OGR2OGR tool as part of the OGR Simple Features Library has been used frequently, examples are provided in "Appendix J: Load Data into Adapted LADM 'Survey Package' PostGIS Database". FWTools has also been used to convert data from PostGIS to a GML format, which could be read by a GML viewer (URL 26).

7.3 Transformation from PSM to DDL (PostgreSQL/PostGIS)

The final PSM (PSM-2, section 6.8) of the Adapted LADM 'Survey Package' in Enterprise Architect is used to generate Data Definition Language (DDL) and Data Manipulation Language (DML) scripts to achieve the actual implementation in PostgreSQL/PostGIS. The standard EA functionality to generate DDL and DML has not been used in the prototype. One of the reasons initially was that geographic columns and indexes in PostGIS need to be created separately or differently, which EA does not offer for the PostgreSQL/PostGIS platform. For example with statements to add a spatial column or create a spatial index:

```
select addgeometrycolumn ('parcel','polygon',28992,'POLYGON',2);

CREATE INDEX  idx_parcel_polygon ON parcel USING GIST ( polygon );
```

Also the generation of PSM OCL constraints for further (manual) implementation, the population of CodeList tables, as well as the general need for flexibility in script generation were a reason for custom development of the MDA prototype script generation functionality. Therefore, program units as part of the MDA prototype have been created to generate DDL and DML scripts in the PostgreSQL/PostGIS database. The "Appendix I: Details on the Generation of DDL Scripts in MDA Prototype (PSM-2 to PostgreSQL/PostGIS)" provides the details on these scripts (URL 30) for

the actual implementation of the Adapted LADM 'Survey Package' in PostgreSQL, which must be executed in this order:

- Delete Objects
  Before creating types, sequences, tables, geometry columns, constraints, indexes and views, the database will be cleaned up, by deleting the objects that are about to be created (with drop cascade statements).

- Create Sequences
  Sequences, providing (sequential) values for the primary key columns are created before the creation of tables and primary key columns, which are referring to them. For example sequence *cadastral_office_oid_seq*.

- Create Types
  Types, which are being used by columns, are created before the tables and columns, referring to them. For example composed type *PersonType*.

- Create Tables
  Tables will be created, with the exception of geographic columns. For example table *survey_point*. All the column settings for "Not Null" properties, Default/Initial values referring to the next value of a sequence, data types, lengths, etc. are included in the DDL script.

- Create Geographic Columns
  On behalf of correctly updating the spatial meta data tables in PostGIS, the spatial columns need to be created separately, with an "alter table statement", after the creation of tables. For example table.column *survey_point.location_measured*.

- Create Primary Key Constraints
  Primary key constraints [pk] for the primary key columns, which are being used by foreign key constraints, are created before the constraints and columns using them. For example primary key constraint *pk_parcel* for table *parcel*.

- Create Constraints
  The non-primary key base table constraints are generated (Unique key [uk], Check [check], and Foreign key [fk] constraint). For example constraints *uk_cadastral_office_code, check_code_uppercase, fk_cadastral_municipality*.

- Create Indexes
  Besides the automatically created indexes for primary, unique and foreign key columns upon their creation, other (spatial) indexes, explicitly defined in the PSM on one or more columns, will be created. For example the spatial index *idx_survey_point_location_measured* on *survey_point.location_measured*. Note that the MDA prototype tool has automatically created spatial indexes for all spatial columns (see section 6.6.2)

- Create Views

Classes, stereotyped as <<view>> in the PSM, will be created in the database, for example the OCL view *v_ocl_survey_point_cadastral_section* possibly serving as a basis for implementation of an OCL constraint.

- Populate Look-up Tables
  <<CodeList>> tables will be populated with a DML insert script, for example insert script *Createcodelist_lkiclassification.sql* to create records for table *codelist_lkiclassification*.

- Present OCL Constraints
  When an OCL constraint has not automatically been implemented by the MDA tool, e.g. as a base table check constraint, it will be transformed to an OCL constraint based on PSM elements. This function creates a listing of OCL constraints defined on PSM-2 elements, serving as a basis for manual implementation, for example the non-implemented OCL constraint *surveyPointCadastralSection*.

The DDL and DML scripts described in this section have been used to generate the PostgreSQL/PostGIS database, which could then be populated with the data as provided by Kadaster.

7.4 Populating the PSM in PostgreSQL/PostGIS with Data

The prototype that has been created in PostgreSQL/PostGIS after executing the DDL scripts as a result of the transformation from the final PSM to DDL (section 7.3) will be filled with data coming from a relevant and representative situation. The Netherlands' Cadastre, Land Registry and Mapping Agency (Kadaster) has made a number of datasets available (in MapInfo *.tab [URL 27] or ASCII format, semicolon separated values), generated and consolidated from the Central Information DataBase (CIDB) of Kadaster (see "Appendix J: Load Data into Adapted LADM 'Survey Package' PostGIS Database"). The following data was provided for the Netherlands and for the province of Utrecht:

- Parcels and Buildings for the Province of Utrecht (February 2008)
  - Parcels
  - Buildings

- Administrative Structure for The Netherlands (January 2007)
  - Cadastral Offices
  - Cadastral Municipalities
  - Cadastral Sections

- Survey Measurements for the Netherlands (April 2006 - December 2007)
  - Survey Projects
  - Survey Connection Points
  - Survey Project Error Logging

These datasets are described in the following sections, with some analyses of the data with regard to population of the Adapted LADM 'Survey Package' in PostGIS.

7.4.1 Parcels and Buildings for the Province of Utrecht (February 2008)

Two main data components for the Province of Utrecht (~ Cadastral Office) have been delivered: Parcels and Buildings.

Parcels

About 430,000 Parcels are provided via MapInfo file *ut_vlak* (Figure 43) as "polygons". The parcels are uniquely identified by an object number, in a format such as "HTN04K 710G0000" where "*HTN04K*" identifies the cadastral municipality (i.e. HTN04) and "*K*" the section (i.e. K), eventually loaded into *parcel.code*, *parcel.section_description* in Figure 40. The rest of the number identifies the parcel number, unique within a section.

Figure 43 - Kadaster Data Provided: Parcels (ut_vlak, ut_prnr), Buildings (ut_gebw2nd)

The MapInfo file *ut_grns* contains 1,656,077 boundaries (data type "linestring") which are identical to parts of the polygons in Parcels (*ut_vlak*), presumably, representing part of the parcel topology, stored in CIDB. The MapInfo file *ut_prnr* contains 429,107 parcel identifiers for the parcels, also uniquely identified by an object number, and described by the area of the parcel, the page number (NL: bladnummer) in the format "HTN04I 3912" (municipality, section, page number). The MapInfo file *ut_text* contains 257,018 textual annotations on the parcels, which have not been used in the prototype.

Close to 5800 parcels exist with at least one interior ring (hole), up to parcels with over 100 interior rings, for example a parcel with 5 interior rings (Figure 44).

Buildings

About 680,000 Building linestrings are provided via MapInfo file *ut_gebw2nd* as "linestrings". The lines are classified with LKI classification (e.g. "B01" = Main Building, "B03" = Miscellaneous Building, "B04" = Artwork [NL: Kunstwerk], "B11" = Edge of Roof). A Quality classification ranging from D0 to X1 is used, consisting of 2 digits, the first one describing the codes for the method of collection (e.g. "T" = Terrestrial Measurement, "F" = Photogrammetric Collection, "K" = Cadastral Map Improvement), and the second digit describing the Precision Code

(e.g. "1" = 1 cm., "2" = 5 cm., "3" = 12 cm.). See Figure 45, for a visualisation of *Buildings* and *Parcels* for the Province (Cadastral Office, see section 7.4.2) of Utrecht, which were loaded into the prototype PostGIS database.

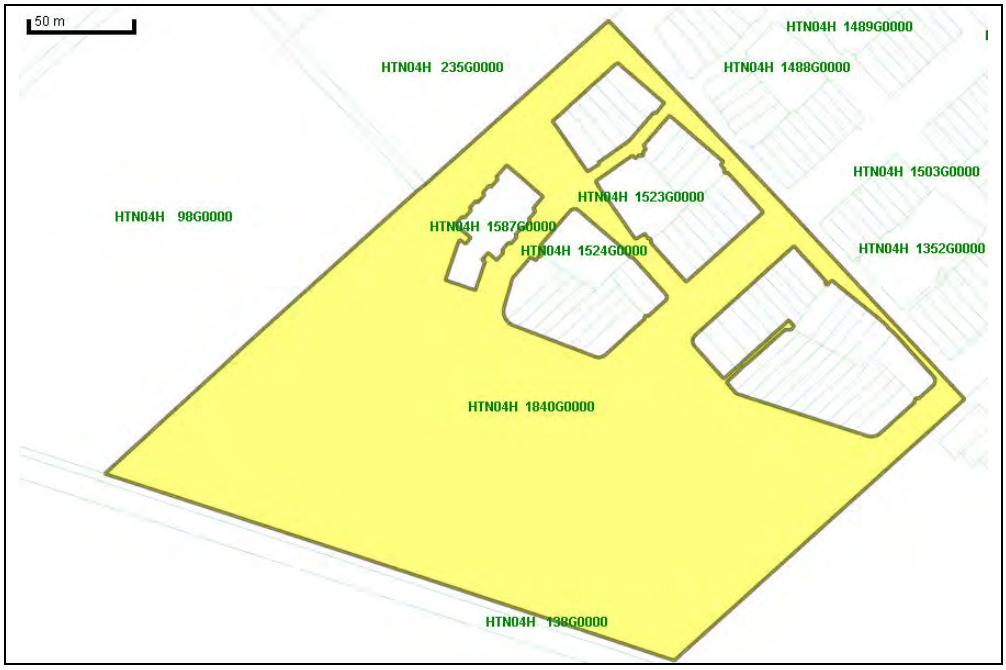


Figure 44 - Parcel with Interior Rings

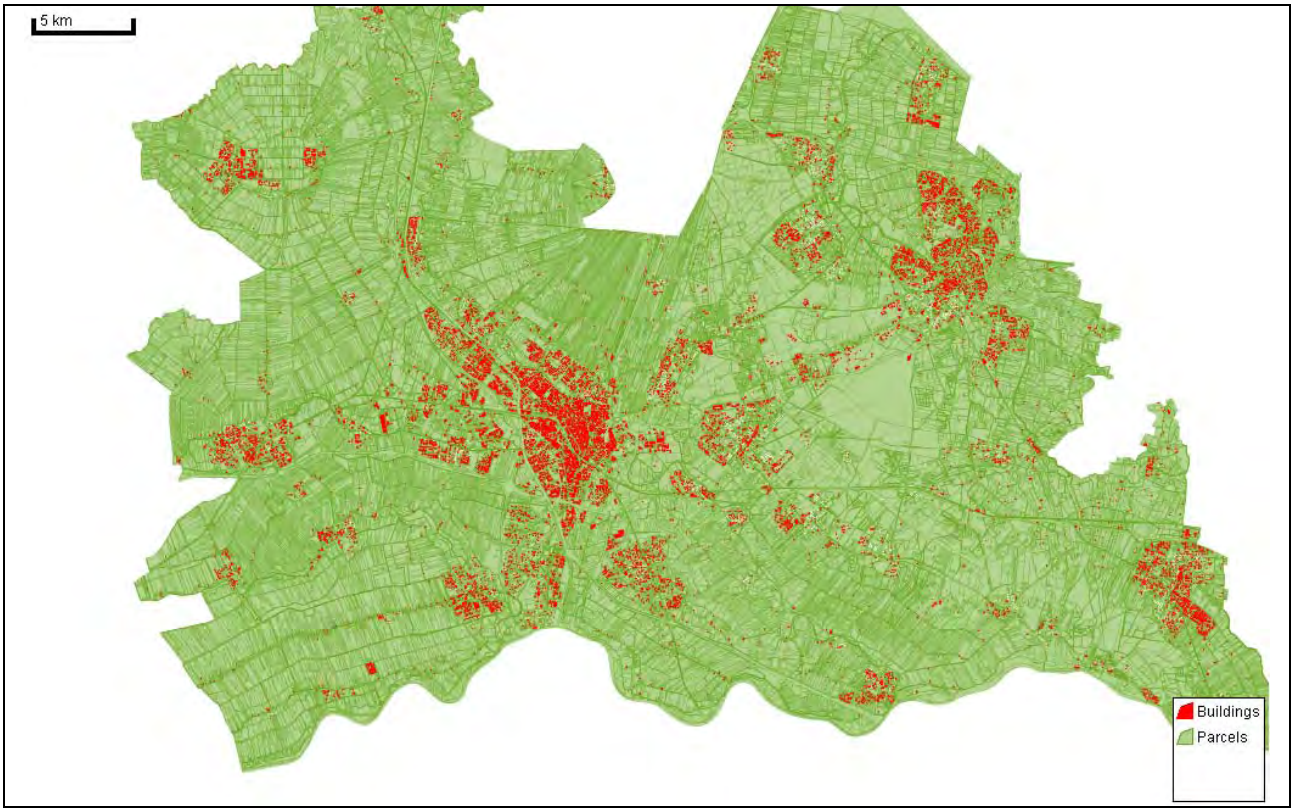


Figure 45 - Parcels and Buildings (Province of Utrecht, February 2008)

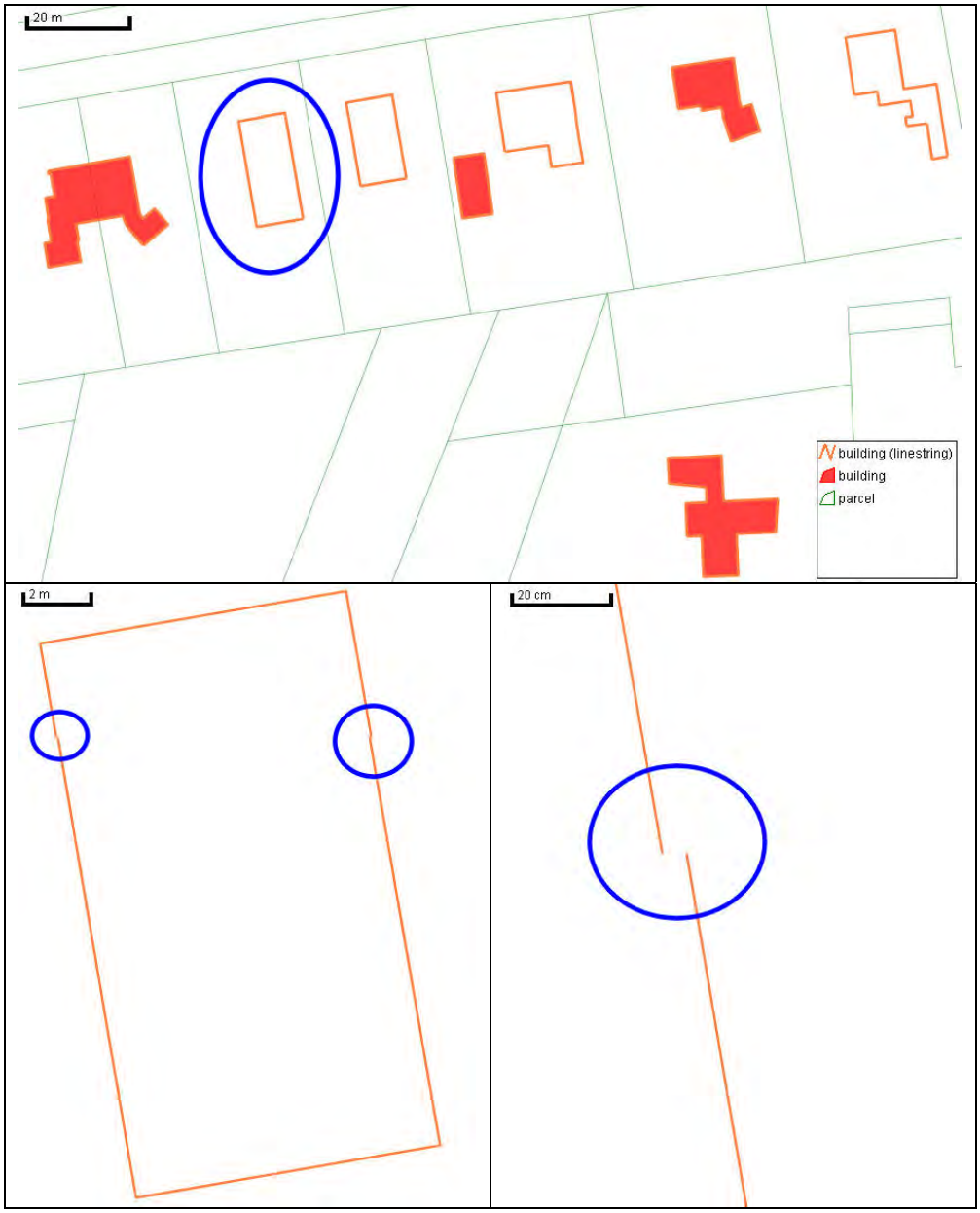


Figure 46 - Non-Closed Building Linestrings

About 20% (~ 138.000) of the building linestrings in *ut_gebw2nd* could form a closed polygon (begin-point is equal to end-point), the PostGIS function *ST_Polygonize* was able to create about 250,000 polygons of these 680000 building linestrings. See Figure 46 for an example of two building linestrings which together do not make up a closed polygon, and thus could not be "polygonised". Details of the second building from the left, with non-closing linestrings have been provided. Some limited amount of anomalies were found in the data; 4 records were found without spatial data, or not of the data type "linestring" (oid 635139, 635142, 635143, 635213). One linestring was found of a length of 292 kilometres (id 635140).

A MapInfo file with lines (*ut_lijn*) was provided with 3,685,521 records which provides information on topography and in some cases overlap with the building linestrings.

7.4.2 Administrative Structure for The Netherlands (January 2007)

The survey points, as described in section 7.4.3, will be aggregated based on the hierarchical level of Cadastral Offices, Municipalities, and Sections (Figure 47).

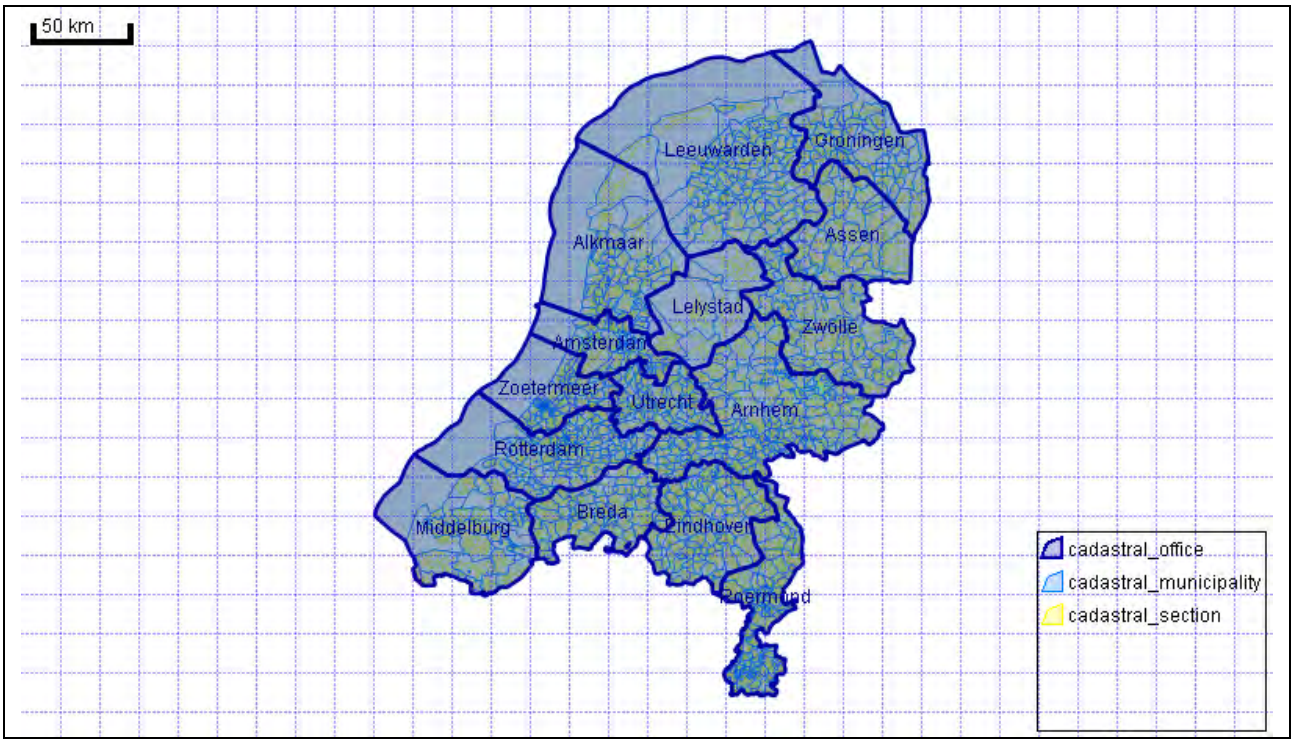


Figure 47 - Cadastral Office, Municipalities & Sections

Cadastral Offices

15 cadastral offices have been loaded into the Adapted LADM 'Survey Package' PostGIS database (Figure 47). The cadastral offices indicate how the Netherlands is divided in area for which a specific cadastral office is responsible. Two cadastral offices exist with multiple polygons: Breda and Amsterdam. The former has 22 interior rings (holes), which is related to the cadastral municipality Baarle-Nassau (Figure 48). The chosen spatial data type for cadastral offices in the PIM is GM_MultiSurface.

Cadastral Municipalities

Furthermore, 1219 cadastral municipalities have been provided. A cadastral municipality is the highest level in uniquely identifying a parcel. The cadastral municipalities Abcoude, Amsterdam, Baarle-Nassau have respectively 2, 2, and 9 polygons, where the latter has 22 interior rings (Figure 48). The chosen spatial data type for cadastral municipalities in the PIM is GM_MultiSurface.

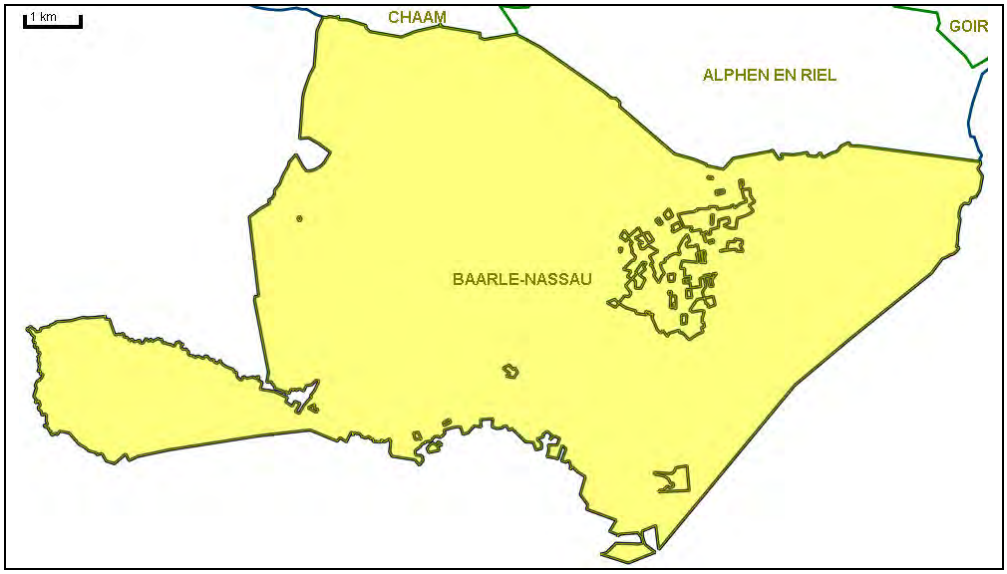
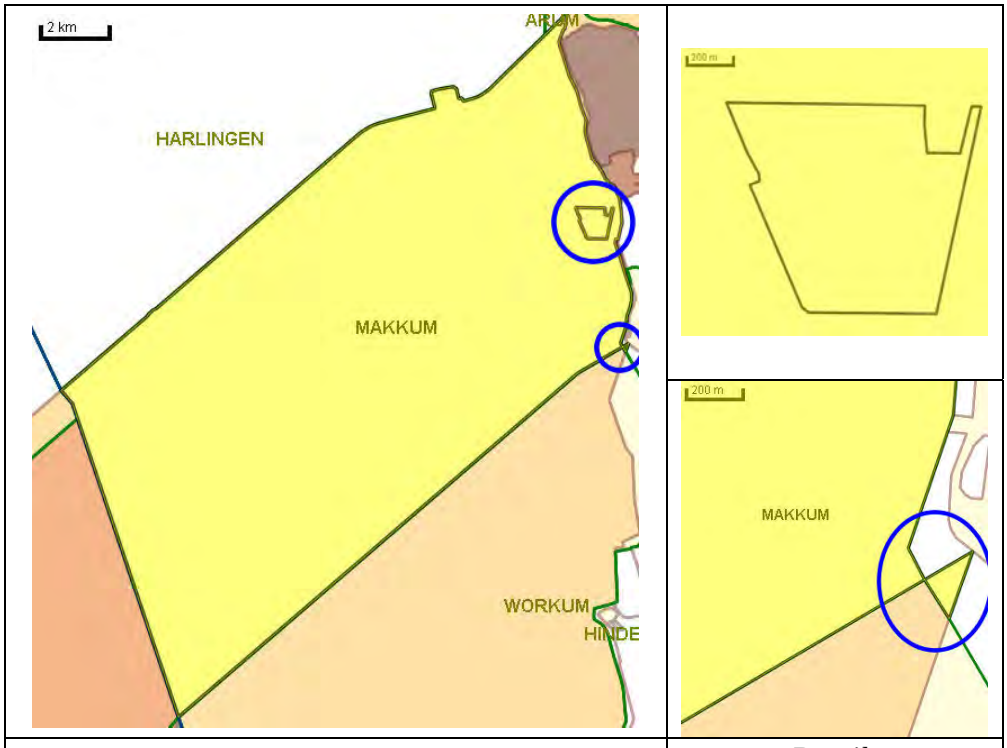


Figure 48 - Cadastral Municipality with Multiple Polygons and Interior Rings



Overview:
Cadastral Section

Detail:
interior ring (above)
second polygon (below)

Figure 49 - Cadastral Section with Multiple Polygons and Interior Rings

Cadastral Sections

In the test data 7,703 cadastral sections have been included. A cadastral section is the second level in uniquely identifying a parcel. About 800 cadastral sections with multiple polygons exist and almost 400 with interior rings. See for example cadastral section "D" within cadastral municipality "Makkum" in Figure 49, showing an interior ring and multiple polygons. The chosen spatial data type for cadastral section in the PIM is GM_MultiSurface.

Note that the cadastral offices, municipalities, and sections are from January 2007, and that new cadastral sections have been created, which can not be found in the survey point data (for about 150 survey projects). For example cadastral municipality Staphorst, which has new cadastral sections "AM", "AN" and "AP", created after a large land (re-) organisation project. New cadastral section can also be created after subdivision of an "old" cadastral section. In some cases cadastral sections expire, and will not be used anymore.

Some detailed examples of the Province of Utrecht are provided in the next section, all visualised with uDig (section 7.2.2, URL 25). For the Province of Utrecht, as visualised in Figure 50, data was provided for parcels and buildings (Figure 45).

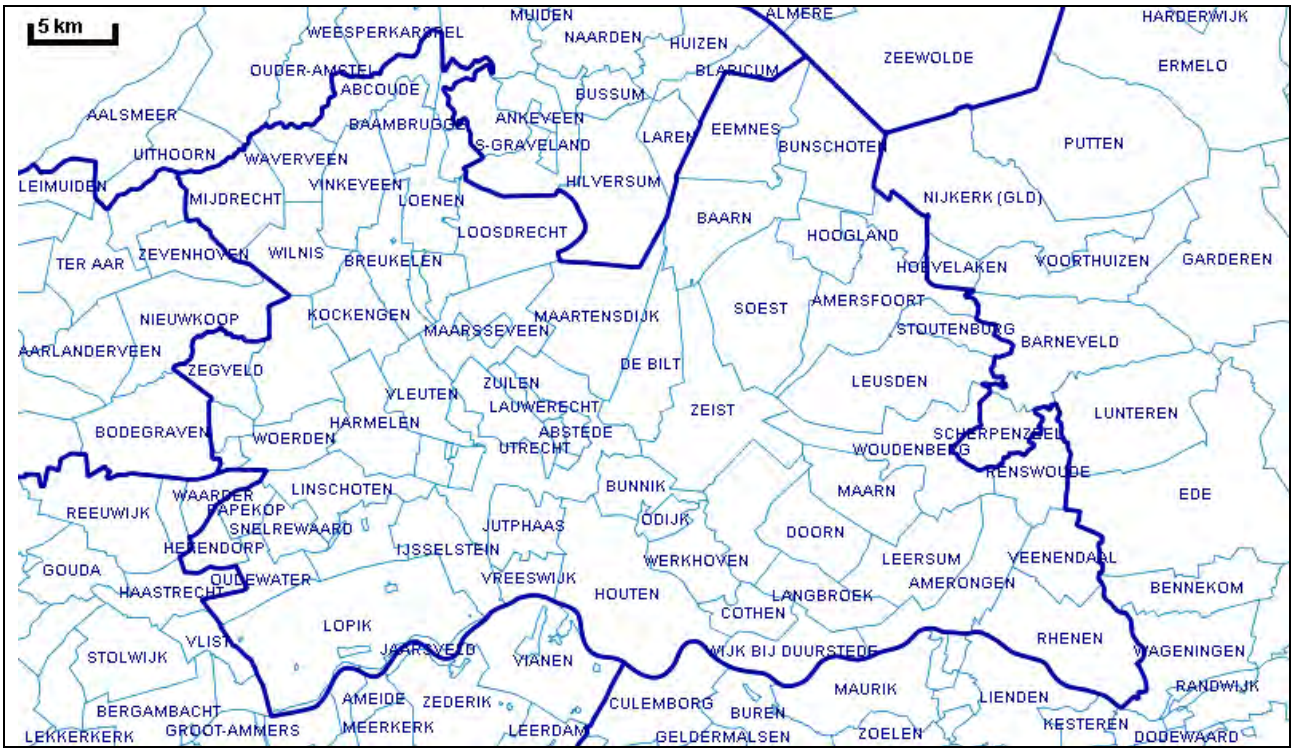


Figure 50 - Cadastral Office Utrecht (showing Cadastral Municipalities)

Figure 51 and Figure 52 show the cadastral municipality of Houten with the division into cadastral sections, presented with and without the abovementioned parcels and buildings.

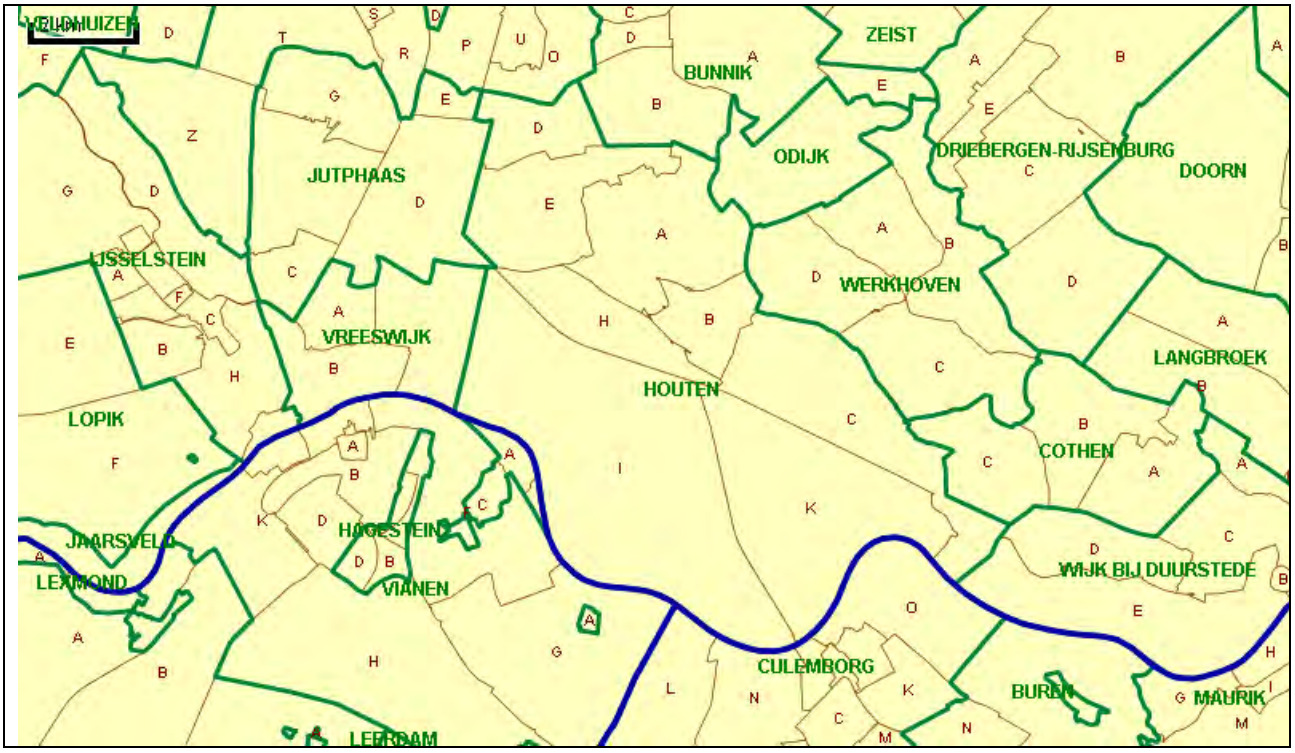


Figure 51 - Cadastral Municipality Houten

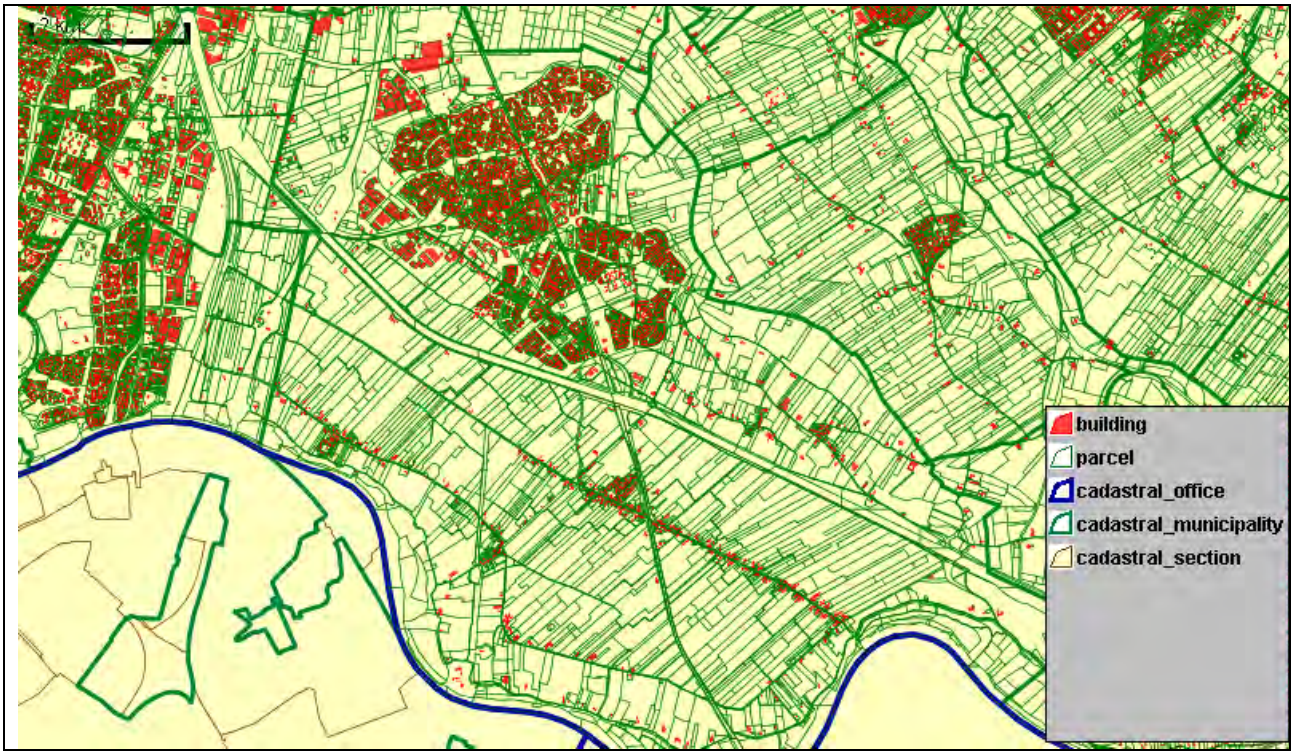


Figure 52 - Cadastral Municipality Houten (with Parcels and Buildings)

7.4.3 Survey Measurements for the Netherlands (April 2006 - December 2007)

The difference between measured coordinates of connection points and their transferred coordinates on the digital cadastral map has been discussed in section 5.2. The following data has been provided by Kadaster to facilitate analysis based on the populated PostgreSQL/PostGIS database. See Figure 101 in "Appendix J: Load Data into Adapted LADM 'Survey Package' PostGIS Database" for an impression of the tool used to convert the ASCII files to a definition of PSM tables. Figure 75 shows the relevant tables *ProjectOverview*, *PhaseDifferenceFile*, and *DifferenceLogFile*.

Survey Projects
Apart from MapInfo files describing the digital cadastral map, ASCII files were provided with the differences between the coordinates of connection points (NL: aansluitpunten) before and after the 2nd phase control point constrained network adjustment (see section 5.2.2). The connection points cover the whole area of the Netherlands, measured in a period from April 2006 - December 2007. These connection points have been measured and collected in about 16000 distinct survey projects, executed within the same period. Duplicates of projects have been excluded, by the use of a primary key on *survey_project.oid*.

Survey Connection Points
The differences between the connection points (see also survey_point.location_measured and survey_point.location_transferred in Figure 40) were provided in ASCII file *phase_difference_file* (with 147815 connection points), which have the attributes project_id (surveying project), x & y coordinate of connection point after 2nd phase control point constrained network adjustment (i.e. location_measured in *RDNAP-TRANS spatial reference system*), the difference with the coordinate from before the 2nd phase network adjustment, the classification code (always "M00"), and an indication if the point has been measured with GPS ("gnss" about 23%), see Figure 75. An overview of the measured connection points is provided in Figure 53. Some minor anomalies were identified, for example Project id 32540 was used by connection points, but could not be found in the project overview, Project id 26210 is (still) in local spatial reference system. Note that the data provided only concerns connection points, already present on the cadastral map. Measured coordinates of the new objects, or supporting measured coordinates are not part of the data.

The (manual) linkage of a connection point (measured coordinate) to a different connection point (transformed coordinate), has been witnessed to be possible and causing outliers in the data (section 7.5.1). The link from survey connection points to cadastral municipality and section has been provided in the data. This reference to cadastral municipality and section is created/maintained by the TIR user. The geometric correctness of this allocation is not checked by a defined constraint in the TIR application, although the TIR user will have to confirm the choice for cadastral municipality and section. This could have an influence on the correctness of the analysis in section 7.5 at cadastral section level.

An example of such a constraint is discussed in "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)" in the section "Constraints Applicable to Multiple Instances of Multiple Classes". The constraint

surveyPointCadastralSection leads to the OCL view *v_ocl_survey_point_cadastral_section*, which queries survey points, which are not within the cadastral section, they are assigned to.

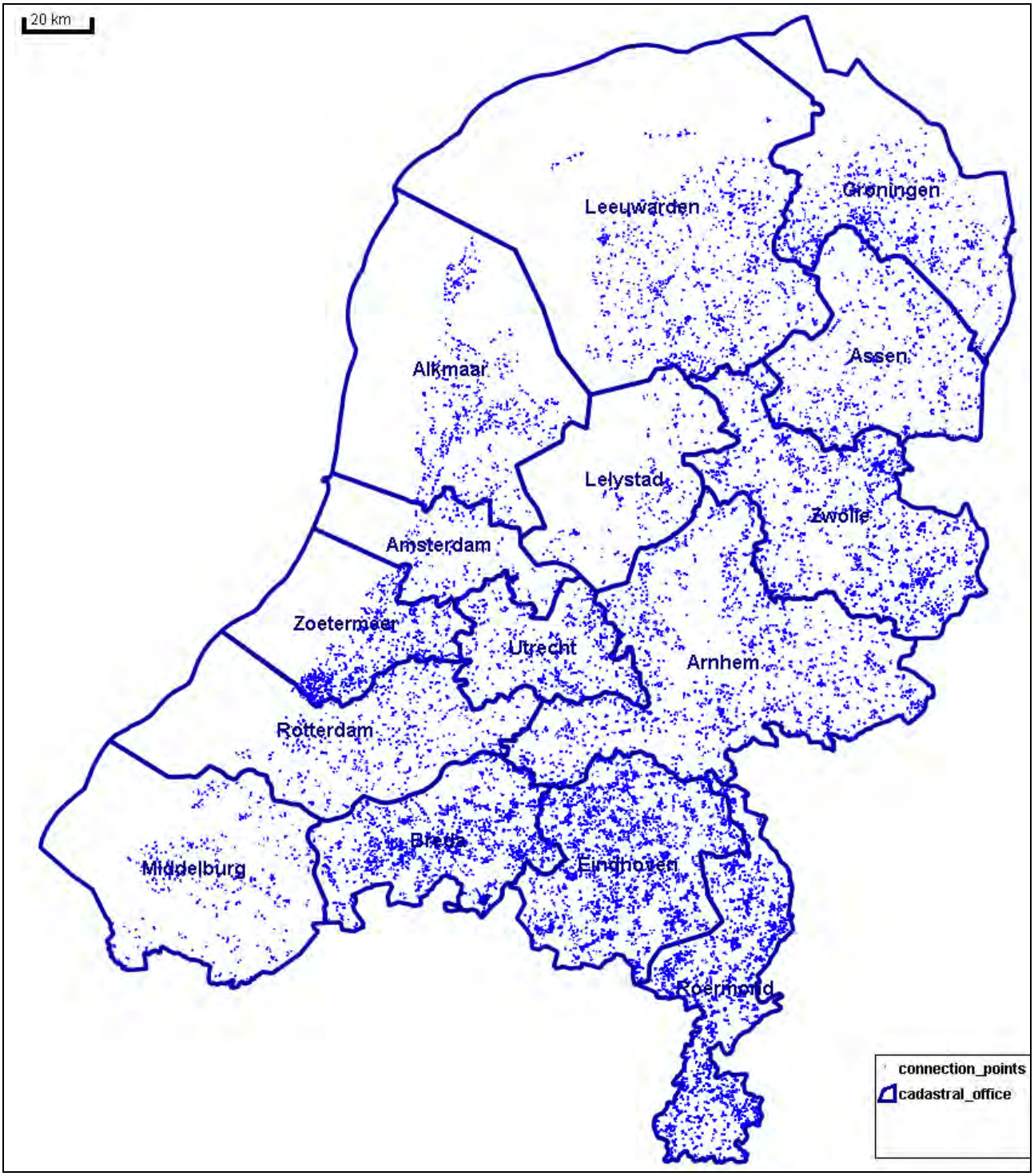


Figure 53 - Measured Connection Points (April 2006 - December 2007)

Survey Project Error Logging
As part of the Kadaster project "Registration Map Quality" (Chapter 5), the errors in preparing the difference between connection points, before and after 2nd phase control point constrained network adjustment, have been categorised in the difference_log_file, for 2% of the survey projects [van Buren, 2006]. The main errors that have been identified by the project "Registration Map Quality" are "identical map coordinates" (19%), "different map coordinates" (6%), "no transformation possible" (2%) and "deviation >1,32m" (73 %), see errors_survey_project in Figure 39.

7.4.4 Description of Data Load Process into PostGIS

The information in the MapInfo tables has been transformed into a PostGIS database, generated by the MDA prototype and specified in Figure 39, Figure 40, and Figure 41. The data loading approach has been based on two steps. The first steps focussed at converting the source data in various formats (e.g. MapInfo, ASCII files) into the target database (PostgreSQL/PostGIS) in *temporary* tables, with a structure similar, if not identical to the source data. The second step was to convert the data from the temporary table into the target tables of the Adapted LADM 'Survey Package', see details in "Appendix J: Load Data into Adapted LADM 'Survey Package' PostGIS Database":

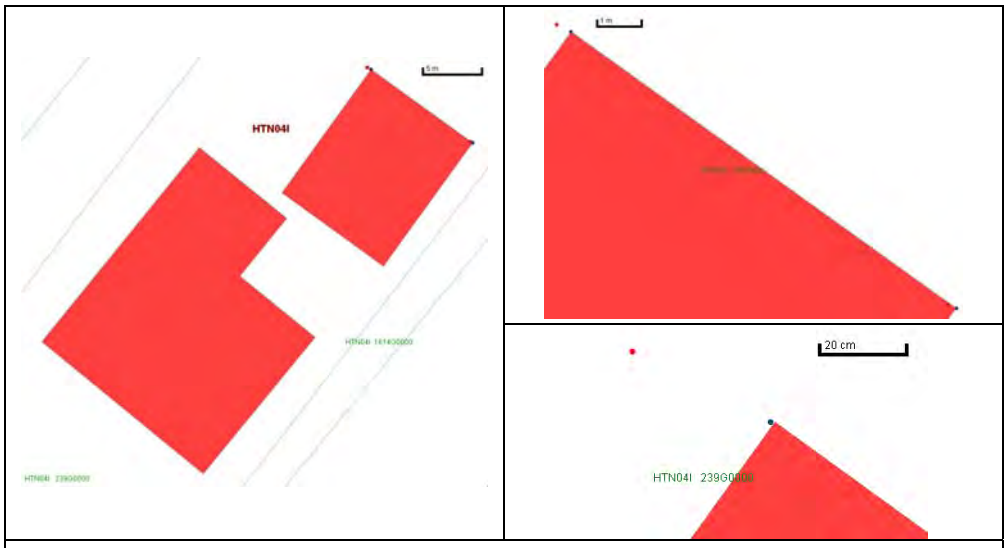
- Conversion MapInfo to PostGIS (temporary tables)
  Conversion of the spatial data (e.g. Parcels and Building) with FWtools (section 7.1.3, URL 19) from one format to another (i.e. MapInfo to PostGIS).

- Conversion ASCII Files to PostGIS (temporary tables)
  The 3 ASCII files with differences between coordinates before and after the 2nd phase control point constrained network adjustment, survey projects and transformation error logging (Figure 102 and Figure 101) were used as input to the MDA prototype, a form "Load XY Differences". This form generates an insert script to insert the data into survey_project, survey_document, survey_point, erros_survey_project.

- Conversion temporary tables into LADM SP (PostGIS)
  The data in temporary PostGIS tables will be loaded into the Adapted LADM 'Survey Package' tables. Note that the spatial data types of for example table survey_point, parcel, cadastral_section, respectively POINT, POLYGON, MULTIPOLYGON will prevent any other geometric type of data (e.g. LINESTRING or MULTILINESTRING) from being loaded in these tables.

The information loaded in PostGIS is presented in uDig (URL 25), some more examples of the data are provided below. Figure 54 shows an overview of Kadaster data in the cadastral municipality of Houten, in an area where survey measurements have been done. When focussing at one building in the lower left quadrant of this figure, the difference between connection points before and after 2nd phase control point constrained network adjustment can be visualised, in this individual case a difference of 35 cm.

Figure 54 - Kadaster Data, Detail of Cadastral Municipality Houten



Detail of Building in lower left quadrant of Figure 54, zooming in on one connection point before and after 2nd phase control point constrained network adjustment.

Figure 55 - Buildings and Connection Points (Measured and Transferred Coordinates)

7.5 Analysis Connection Points

When looking more closely at the connection points (also referred to as control points) in table survey_point a number of observations can be made:

- 147815 survey_points are loaded, 77% measured in "local" spatial reference systems (relative measurement), and 23% with GPS ("gnss" ~ Global Navigation Satellite System).
- Large differences (outliers) between location_measured (before 2nd phase control point constrained network adjustment) and location_transferred (after) can be witnessed, sometimes more than 30 kilometres. These differences are only seen in the "gnss" measurements, because "local" measurements over a 1.32 meter have not been provided.

7.5.1 Exclude Outliers in Connection Points

The outliers are presumably created by TIR operators, involving the wrong connection points to the data. The manual allocation of survey projects (and its connection points) to a cadastral section (section 5.2), enables mistakes to be made in that allocation. This has been examined with a query, similar to the view v_ocl_survey_point_cadastral_section (as described in "Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)". The query resulted in about 7% of the connection points (survey points) that are not geometrically within the cadastral section.

Errors have been found in linking a connection point (measured coordinate) to a different connection point (transformed coordinate), causing the outliers. The large outliers will be excluded, but the "smaller" errors might not be excluded. The "local" connection points have undergone a pre-selection, before being provided to the master thesis project, and the "gnss" show larger differences. The reliability of this analysis is influenced by these observations.

A way of excluding these outliers must be defined. Under the assumption that the distances/differences between the connection points (location_measured /location_transferred) are behaving like a "normal distribution" of statistics, an arithmetic mean value (μ ~ average) and a standard deviation (σ) can be calculated for the connection points in table survey_point.

μ = 11,55 meter

σ = 912 meter

After applying a confidence interval of 95.4 % ($\mu \pm 2\ \sigma$), differences of close to a 1000 meter still participate in the analysis, and these differences seam very unrealistic (outliers). The effect of the large outliers on the average difference, is very high, the measurements do not comply with a normal distribution of values. When investigating some of the survey projects a situation like project 9100 (Figure 56) can be witnessed quite often. The TIR application allows for selecting connection points into a survey_project, which belong to another project, i.e. these are outside the

surveyed area. One outlier (difference 24.37 meter) is part of a set of differences, all smaller than 88 cm.

project_id	quality	point_number	dx	dy	distance
9100	gnss	1	-0.214999999996508	-0.152999999932945	0.263882549589736
9100	gnss	2	-0.0439999999944121	0.351999999955297	0.354739340880085
9100	gnss	3	-0.0460000000020955	0.315999999991618	0.319330549736312
9100	gnss	4	-0.092000000004191	-0.512999999918975	0.521184228385357
9100	gnss	5	0.169999999983702	-0.407000000006519	0.441077090767323
9100	gnss	6	-0.216000000014901	-0.238999999943189	0.322144377537901
9100	gnss	7	-24.372000000003	-0.337999999988824	24.3743436424479
9100	gnss	8	0.0970000000088476	-0.148999999975786	0.177792013303468
9100	gnss	9	-0.182999999989988	-0.616000000038184	0.642607967615854
9100	gnss	10	-0.0319999999774154	-0.403999999980442	0.405265345153952
9100	gnss	11	-0.187000000005355	-0.305999999982305	0.358615392853086
9100	gnss	12	-0.277999999991152	-0.195999999996275	0.340147027024521
9100	gnss	13	-0.170000000012806	-0.324999999953434	0.366776498666539
9100	gnss	14	0.0380000000004657	0.879000000073574	0.879821004596605

Figure 56 - Outlier in Survey Project (with oid 9100)

An alternative approach to selecting the connection points as a basis for the analysis of the quality of the map (and ignoring outliers), has been provided, based on assessing the connection points in table survey_point per survey_project:

- All connection points with a difference in coordinates (before and after the 2nd phase control point constrained network adjustment), larger than 5 meters (parameter *max_distance*), are marked as outliers and excluded from the analysis.

- If the connection point differences within a project average $\mu > 1.32$ meter (parameter *max_arithmetic_mean*), then the number of connection points in projects is taken into account, to eliminate outliers as presented in Figure 56:

    - if the number of connection points per survey project is >5 and the difference is outside the range $\mu \pm 2\ \sigma$ (parameter *sigma_multiplier*), then the survey point is excluded from analysis.

    - if the number of connection points per survey project is 5 or less, and the difference is outside the range $\mu \pm \sigma$ (68% of the values), then the survey point is excluded from analysis.

A stored PostGIS function "load_survey_point_analysis" has been created to serve this purpose, the parameters that can be provided are: *max_distance*, e.g. 5 meter, *max_arithmetic_mean*, e.g. 1.32 meter, *sigma_multiplier*, e.g. 2. See "Appendix K: Stored Function to Select Survey Points for Analysis" for the specification of the function, that can be executed with different parameters.

```
select load_survey_point_analysis(5,1.32,2);
```

After executing this function (the command above) the result message below shows that in total 1097 (1018+79) records (about 0.7 %) are excluded for participating in the analysis:

max_distance=5, max_arithmetic_mean =1.32, sigma_multiplier =2, excluded for max_distance=1018, excluded for sigma_multiplier=79

This function can be run with other settings for the abovementioned max_distance, max_arithmetic_mean and sigma_multiplier, but with current settings 146,718 survey points are part of the following analysis. An additional 4 points have been excluded which were still in a local reference system.

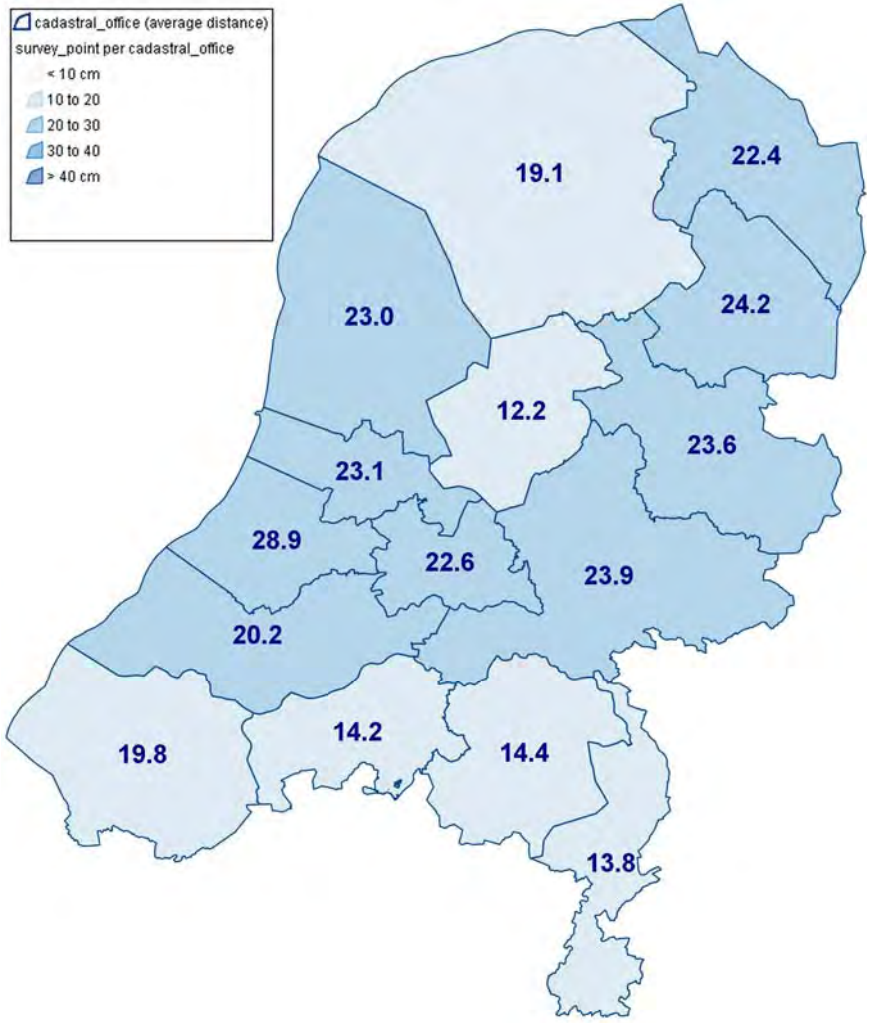
The function has also been executed to exclude outliers for a maximum difference (max_distance) of 3 meter, and for a maximum difference of 7 meters, and a range $\mu \pm 3\ \sigma$ (sigma_multiplier). These results for the 3 and 7 meter setting excludes respectively over 1600 and over 800 connection points, the result of which is presented in Figure 57. The average difference for the 3 and 7 meter setting is respectively 18.0 and 20.1 cm, compared to the 19.3 cm of the 5 meter setting of the function. The two cadastral offices showing the lowest and highest differences (highest and lowest quality of the cadastral map) are indicated with a purple and green colour.

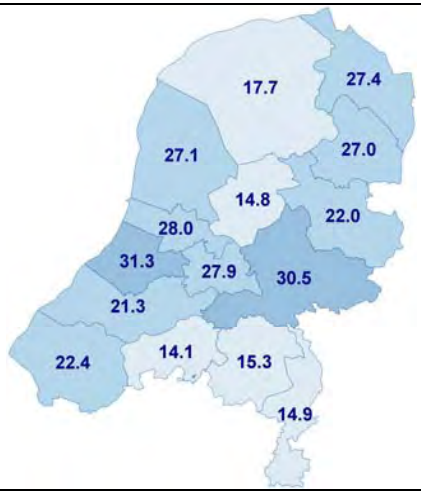
The current procedure of excluding outliers in the survey points, gives different result with different parameters (Figure 57), but similar trends are shown at cadastral office level. In the next section, the 5 meter setting will be used for further analysis.

Code	Name	% Total	Average Difference (cm)	Deviation with 5m parameter	Average Difference (cm)	Average Difference (cm) [with direction of difference vector]	Deviation with 5m parameter	Average Difference (cm)
			max. Difference = 3m		max. Difference = 5m		max. Difference = 7m	
AD	Amsterdam	1%	21.6	-6%	23.1	1.1	0%	23.1
AH	Arnhem	10%	22.9	-4%	23.9	1.2	2%	24.4
AM	Alkmaar	4%	21.8	-5%	23.0	1.0	7%	24.5
AS	Assen	4%	23.2	-4%	24.2	1.2	3%	24.9
BD	Breda	10%	12.6	-11%	14.2	1.5	4%	14.7
EH	Eindhoven	16%	13.2	-8%	14.4	0.4	6%	15.3
GN	Groningen	5%	21.6	-4%	22.4	0.6	2%	22.9
GV	Zoetermeer	5%	26.0	-10%	28.9	0.8	7%	30.8
LA	Leeuwarden	6%	17.8	-7%	19.1	0.9	5%	20.0
LL	Lelystad	2%	11.0	-10%	12.2	2.9	4%	12.7
MD	Middelburg	2%	18.8	-5%	19.8	0.7	3%	20.3
RM	Roermond	15%	13.0	-6%	13.8	0.4	3%	14.2
RT	Rotterdam	3%	19.1	-5%	20.2	1.1	3%	20.8
UT	Utrecht	4%	21.4	-5%	22.6	1.2	4%	23.4
ZL	Zwolle	13%	22.2	-6%	23.6	0.8	4%	24.6
Average			***18.0***	*-7%*	***19.3***		*4%*	***20.1***

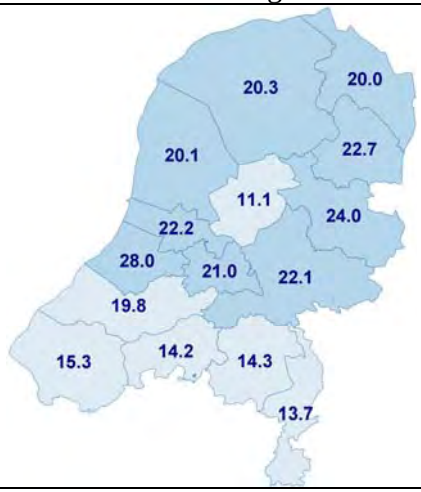
Figure 57 - Overview Survey Points per Cadastral Office (Different Treatment of Outliers)

On the following pages Figure 58 - Figure 65 will visualise the difference between measured and transferred coordinates of connection points, which will be described in the subsequent sections.





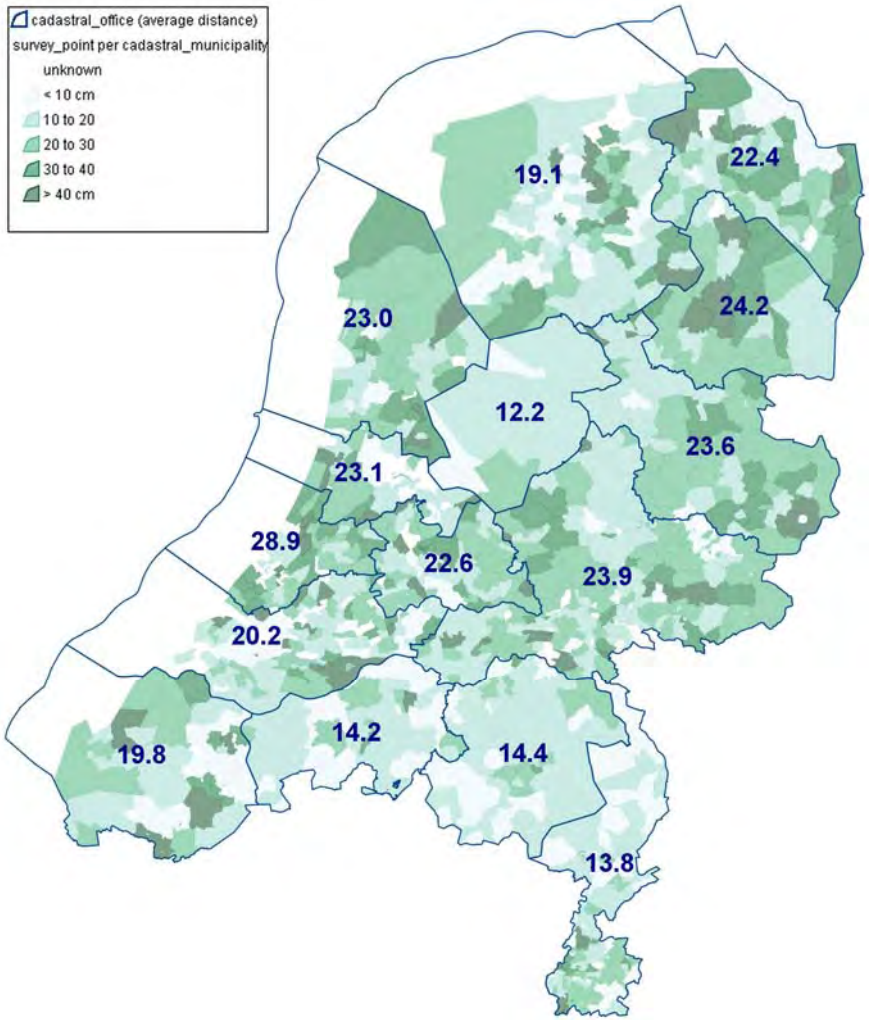
Connection Points based on 'gnss' measurements

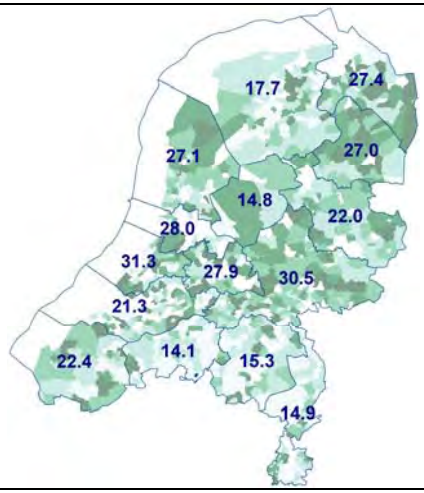


Connection Points based on 'local' measurements

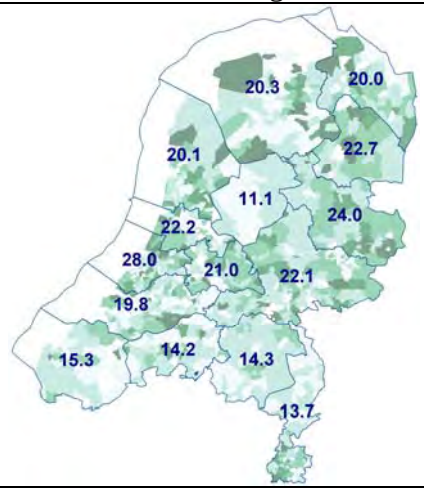
Difference measured and transferred coordinates of Connection Points before and after 2nd phase control point constrained network adjustment (in centimetres)

Figure 58 - Difference Connection Point Coordinates (Aggregated per Cadastral Office)





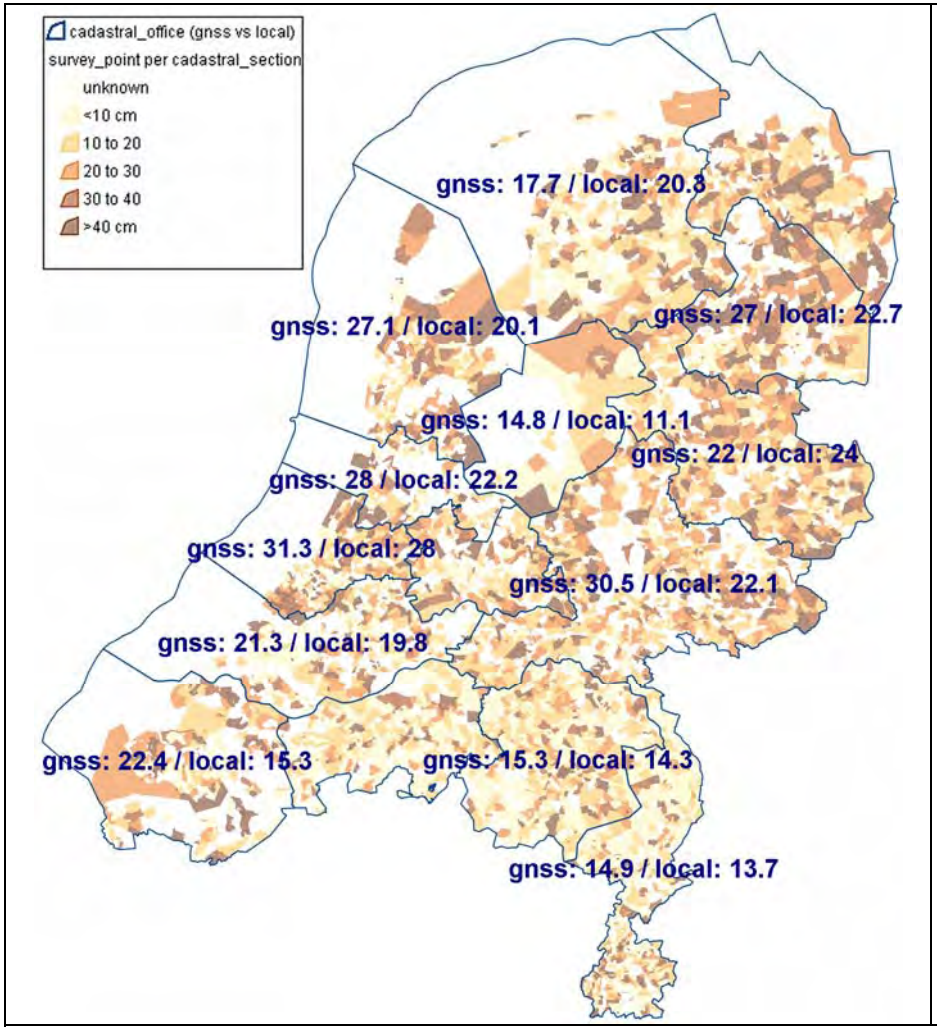
Connection Points based on 'gnss' measurements



Connection Points based on 'local' measurements

Difference measured and transferred coordinates of Connection Points before and after 2nd phase control point constrained network adjustment (in centimetres)

Figure 59 - Difference Connection Point Coordinates (Aggregated per Cadastral Municipality)



Difference measured and transferred coordinates of Connection Points before and after 2nd phase control point constrained network adjustment (in centimetres)

Figure 60 - Difference Connection Point Coordinates (Aggregated per Cadastral Section)

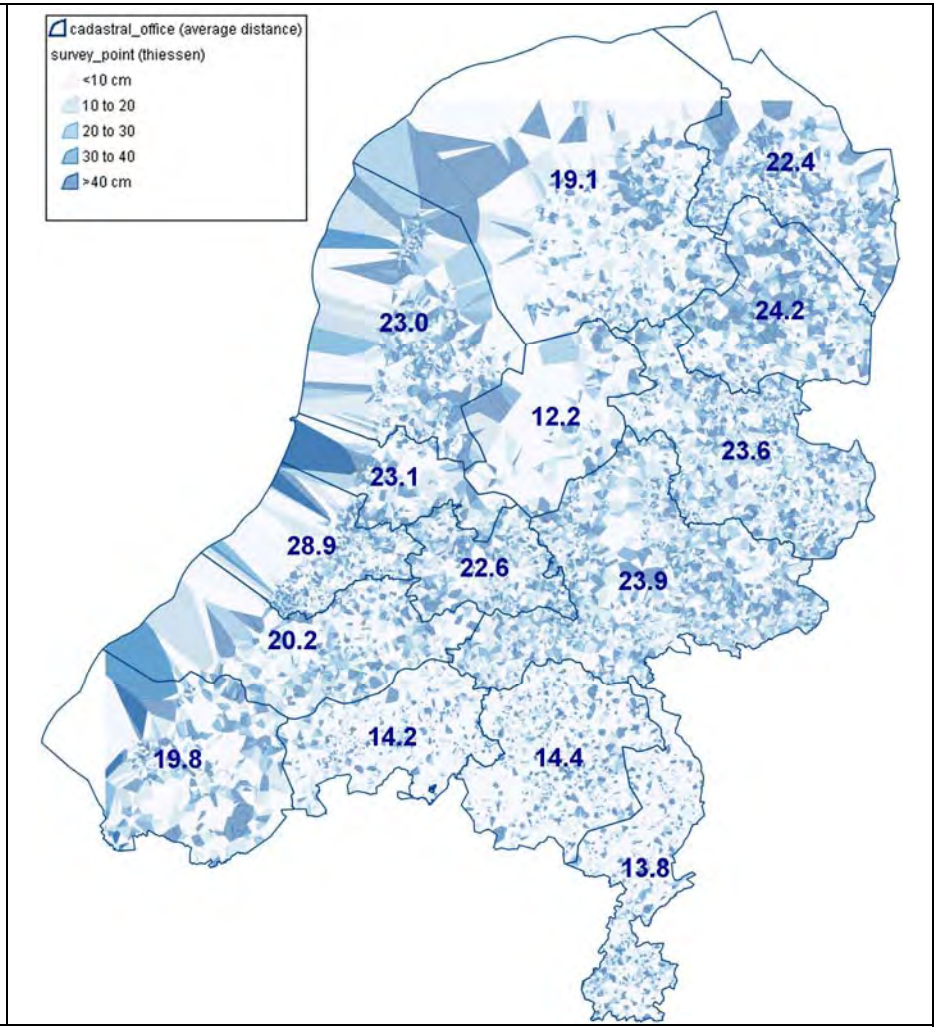
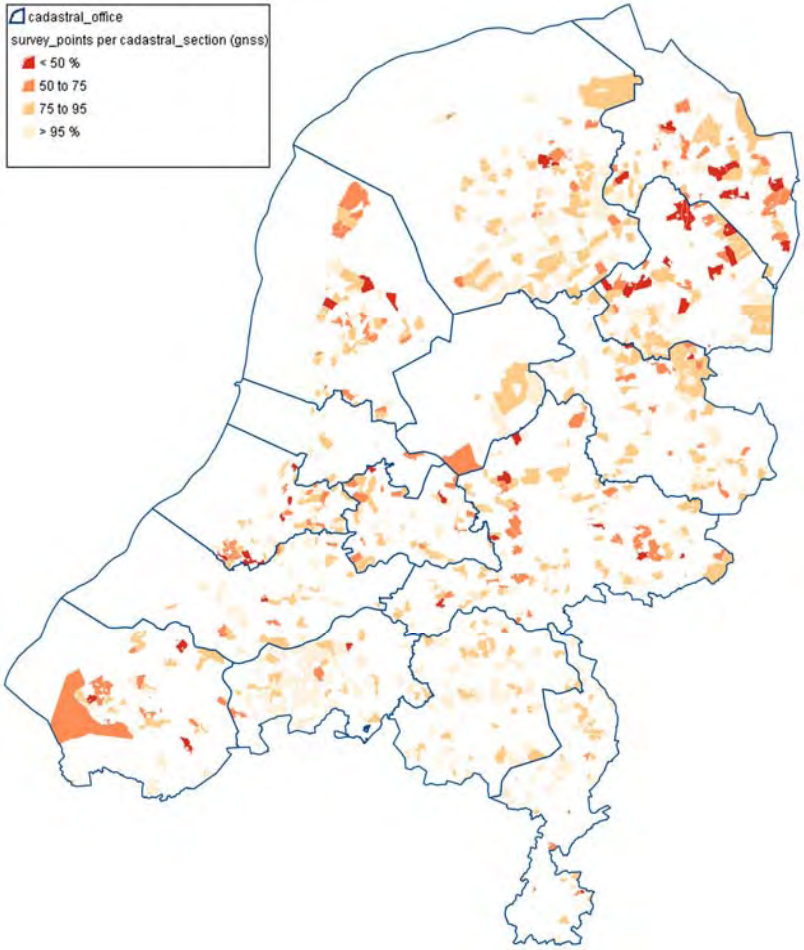
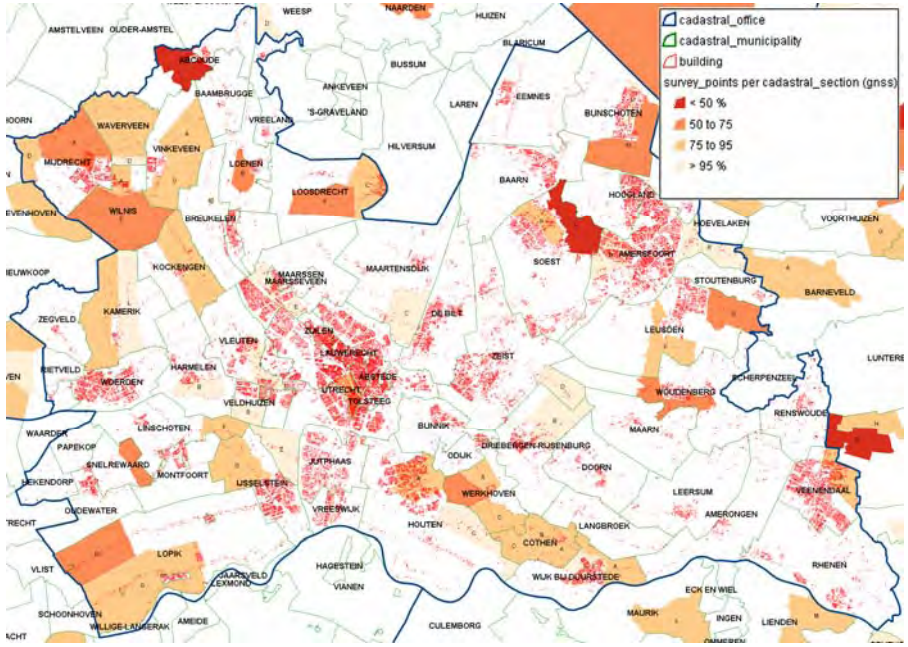


Figure 61 - Difference Connection Point Coordinates (Thiessen Polygons Created from Connection Points)

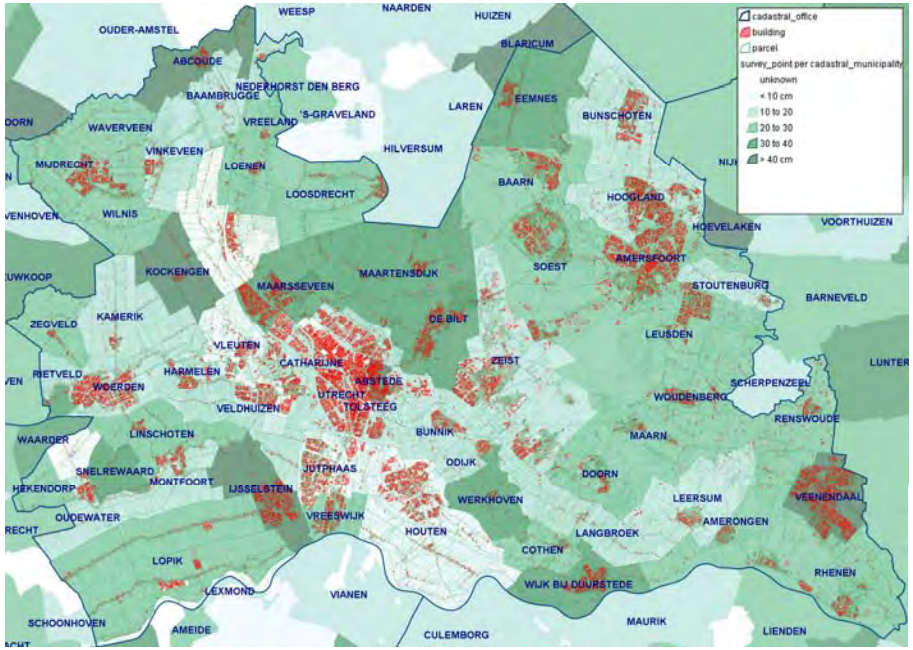


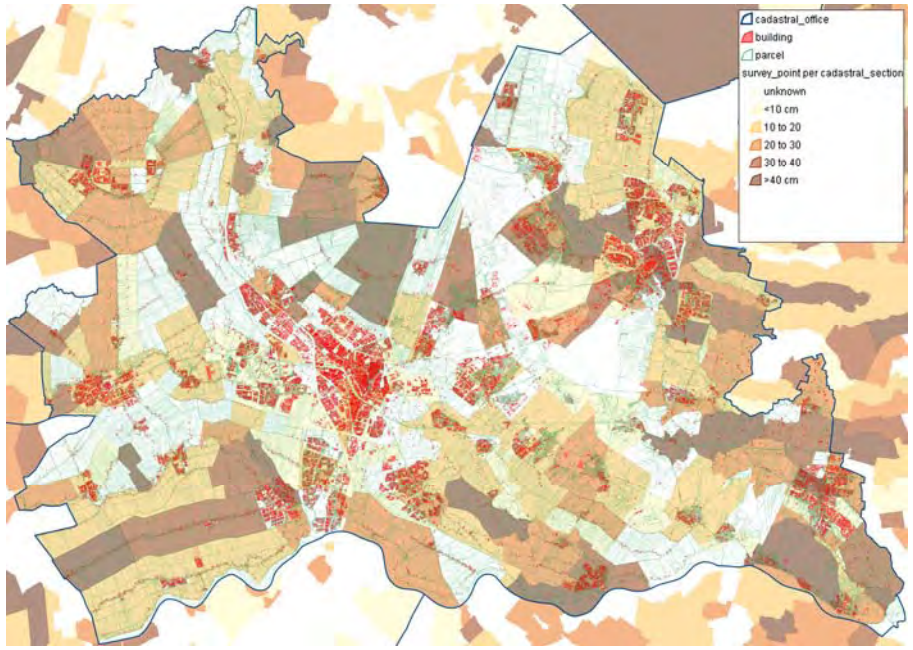


Difference measured and transferred coordinates of Connection Points before and after 2nd phase control point constrained network adjustment (in centimetres) Only Cadastral Sections with 10 or more (measured and transferred) Connection Points (originally measured in 'gnss') are shown.

Figure 62 - Percentage of Connection Points per Cadastral Section (Originally Measured in 'gnss') with a Difference below or equal to 40 cm (The Netherlands)

Figure 63 - Percentage of Connection Points per Cadastral Section (Originally Measured in 'gnss') with a Difference below or equal to 40 cm (Province of Utrecht)





Difference measured and transferred coordinates of Connection Points before and after 2nd phase control point constrained network adjustment (in centimetres) overlaid with Parcels and Buildings (to indicate "urban" and "rural" areas)

Figure 64 - Difference Connection Point Coordinates for Province of Utrecht (Aggregated per Cadastral Municipality)

Figure 65 - Difference Connection Point Coordinates for Province of Utrecht (Aggregated per Cadastral Section)

7.5.2 Aggregation Level: The Netherlands

The *average difference* (distance) between the measured and transferred coordinates of connection points (before and after the 2nd phase control point constrained network adjustment) is ***19.3 cm* for *the Netherlands*** (compared to the μ = 11.55 meter for survey points with all outliers included).

In section 5.3 the use of a similarity transformation to convert measured coordinates to the RDNAP-TRANS spatial reference system has been described. If a distinction is made between connection points, on the one hand originally measured in a local spatial reference system and local RD (i.e. quality = 'local') and on the other hand measured with Global Navigation Satellite System tools (i.e. quality = 'gnss'), the distance between connection points seems to be better for 'local' = 18.3 cm, compared to 'gnss' = 22.6 cm (see Figure 58). Note that all differences for connection points with quality = 'local', have been transformed to RDNAP-TRANS, and that points with a high deviation (rest total > 1.32 m) have been discarded.

7.5.3 Aggregation Level: Cadastral Offices

In Figure 58 the *average differences* (between the measured and transferred coordinate of the connection point) *per cadastral office* are provided, where the differences are categorised with 5 classes (< 10 cm; 10-20 cm; 20-30 cm; 30-40 cm, and > 40 cm).

The distribution of the survey (i.e. connection) points over the cadastral offices is provided in Figure 57, with cadastral offices Arnhem, Breda, Eindhoven, Roermond, and Zwolle taking over 60% of the values (column "*Average Difference (cm)*"). The column shows the average distance/difference between the measured and transferred coordinate of connection points (before and after the 2nd phase control point constrained network adjustment) per cadastral office.

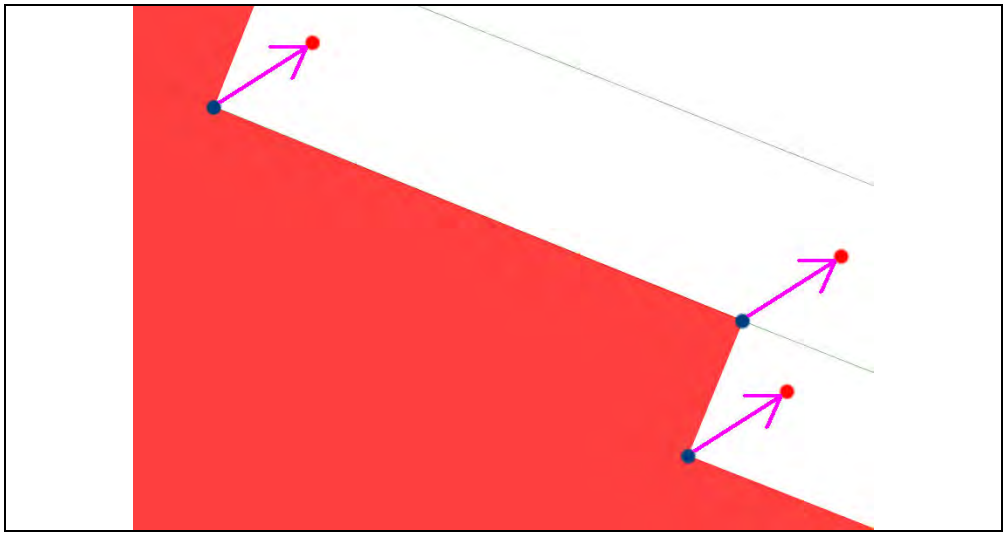


Figure 66 - Difference (Between Measured and Transferred Coordinate of a Connection Point) presented as Vector.

This distance/difference could be represented as a vector from the measured coordinate to a transformed coordinate of a connection point (Figure 66). When taking into account the *direction of the difference vector*, the average differences are considerably lower (column "*Average Difference (cm) [with direction of difference vector]*" in Figure 57), indicating that the cause for the distances/differences is not of a systematic nature (e.g. a shifted map), this could also be analysed at other aggregation levels (e.g. Cadastral Municipalities and Sections). Figure 58 visualises the differences aggregated per cadastral office area.

7.5.4 Aggregation Level: Cadastral Municipalities

In Figure 59 the *average differences* (between the measured and transferred coordinate of the connection point) *per cadastral municipality* are provided, where the differences are again categorised with 5 classes, and a distinction between original 'gnss' and 'local' measurements is made. Both the aggregation at cadastral municipality and cadastral office level, also show the south and central areas of the Netherlands having the smallest differences between measurements and the digital cadastral map. The cadastral office Zoetermeer, presumably classified as urban area, shows remarkable differences over 20 centimetres.

Figure 64 has a focus on the cadastral municipalities of the Province of Utrecht, overlaid with parcels and *buildings*, where the latter gives some idea of the location of *urban* (built-up and town areas).

7.5.5 Aggregation Level: Cadastral Sections

In Figure 60 the *average differences* (between the measured and transferred coordinate of the connection point) *per cadastral section* are provided, where the differences are again categorised with 5 classes. Figure 65 has a focus on the cadastral sections of the Province of Utrecht, overlaid with parcels and buildings.

As mentioned in section 5.3, the differences between the measured and transferred coordinate of a connection point should be smaller or equal to ± 20 cm and ± 40 cm respectively in urban and rural areas. However, no classification of cadastral sections into rural or urban was available, which could be used to assess the quality (accuracy) of the cadastral map.

Kadaster Norm 95%

Kadaster uses a norm that 95% (instead of 100%) of the connection points should be within the 20 and 40 cm limits for urban and rural area [Westerik and Kenselaar, 2004]. Figure 62 shows all cadastral sections with *10 or more connection points*, originally *measured in 'gnss'*. For each cadastral section, it is determined which percentage fall *within the 40 cm difference* (between measured and transformed coordinate), depicted with a very light colour. If less than 95% is within the 40 cm limit, darker colours have been used to indicate cadastral sections that potentially have an 'accuracy' issue. Figure 63 shows the same information for the Province of Utrecht, overlaid with buildings. Further analysis is required to investigate the 'darker' cadastral sections.

7.5.6 Aggregation Level: Connection Points

The connection points have been presented as points in Figure 53. To be able to show the *average differences* (between the measured and transferred coordinate of the connection point) *per cadastral point*, similar to Figure 58 - Figure 60, Thiessen Polygons have been created based on the available connection points. In the centre of Figure 67, 11 connection points are presented, overlaid with the Thiessen polygons with **purple** boundaries.

Figure 67 - Connection Points overlaid with Thiessen Polygons

The creation of Thiessen polygons from connection points has been done by first exporting the connection (survey) points to a MapInfo format with the below mentioned FWTools command (URL 22) and then in MapInfo (URL 27) with the function Voronoi.

```
ogr2ogr -f "MapInfo File" MapInfoFolder PG:"dbname='postgis' user='GIMA'" cadastral_survey_points_nl
```

After importing the results back into PostgreSQL/PostGIS, the results can be visualised with uDig (URL 25), as in Figure 61

The Thiessen polygons show a similar image as Figure 60 where the differences aggregated to the level of cadastral sections. The 3 south and 1 central cadastral offices (respectively Breda, Eindhoven, Roermond, and Lelystad) have the lowest difference between measurements and digital cadastral map, cadastral office Zoetermeer the highest.

Survey Point and Buildings

The majority of connection points are related to buildings (instead of parcels) and in Figure 64 and Figure 65 the connection point differences are overlaid with Buildings and Parcels to allow for further analysis. In general, it can be concluded that urban areas, have a lower average difference (between measured and transferred coordinate of a connection point), but contradicting observations can also be found, for example Veenendaal in the lower right corner of Figure 64. In Figure 68, another exception for urban area is shown for the City of Utrecht, where average differences in cadastral section of 66.5 and 96.2 can be found, the latter based on two survey projects 43328 and 43330 respectively with 8 and 6 connection points. Note that the "graphical precision", smaller or equal to ± 20 cm and ± 40 cm respectively in urban and rural areas (Chapter 5) cannot be exactly discriminated, since no attributes exist for the delivered survey points, cadastral offices, municipalities, and section, to indicate the applicable category "urban" or "rural".

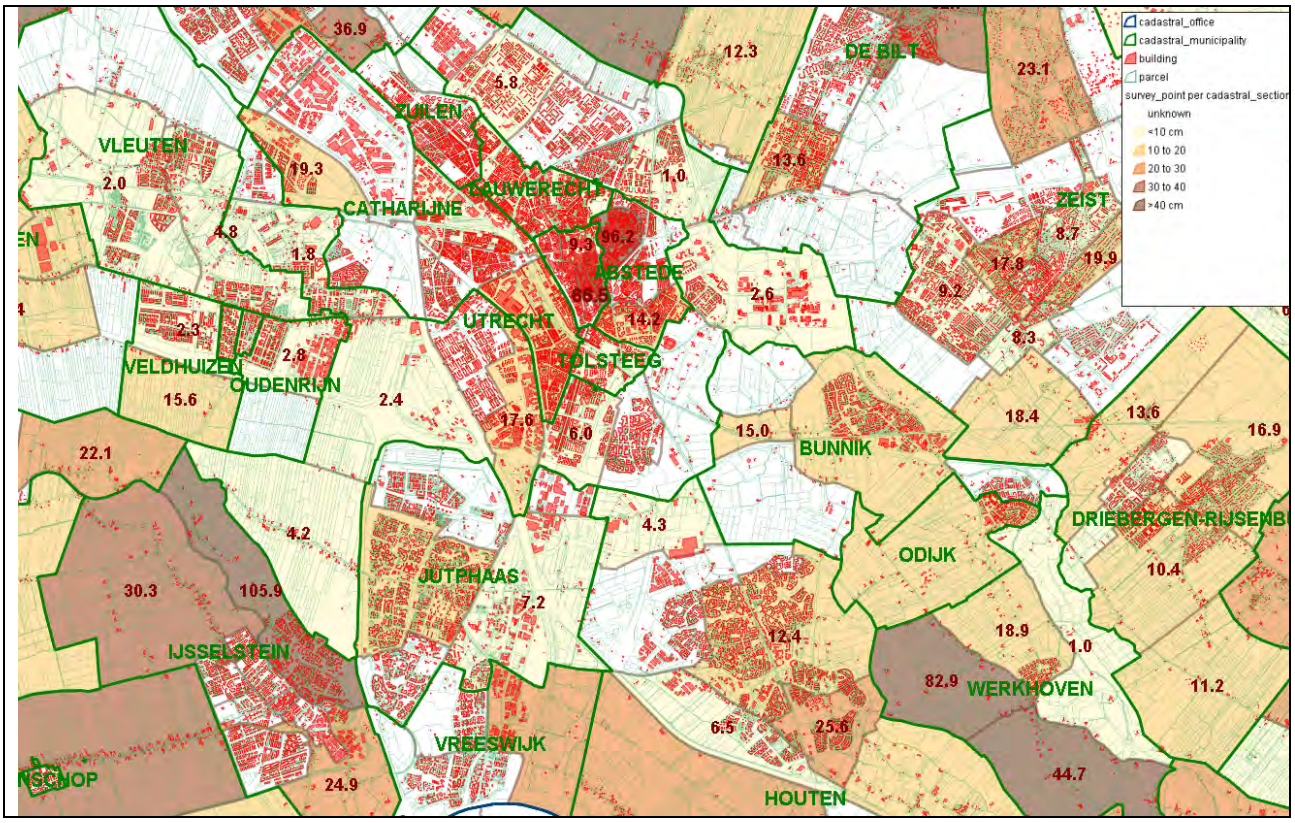


Figure 68 - Detail of Province of Utrecht

7.6 Conclusion

The following open source tools have been used significantly in the master thesis project: the PostgreSQL database (with a PostGIS extension), uDig for visualisation of geographical data (analysis), and FWTools for converting spatial data, which have proven to be suitable and stable for the master thesis project.

The final PSM of the Adapted LADM 'Survey Package' in Enterprise Architect has been the basis for the automatic generation by the MDA prototype of Data Definition Language (DDL) and Data Manipulation Language (DML) scripts, to achieve the actual implementation in PostgreSQL/PostGIS. The DDL/DML scripts create sequences, custom data types, tables, geographic columns, primary key, unique key, and foreign key constraints, check constraints, (spatial) indexes, (OCL) views and a list of OCL constraints on PSM elements. Kadaster has provided data on parcels (as polygons) and buildings (as linestrings) for the Province of Utrecht, the cadastral offices, municipalities, and sections for The Netherlands, and survey projects and measurements (connection points with a measured and a transferred coordinate), on behalf of populating the Adapted LADM 'Survey Package' in PostGIS.

Several comments have been made on the provided data. For example: parcels, cadastral offices, municipalities and sections were found with interior rings (holes) and/or multiple polygons describing them. Furthermore, building linestrings have been found that do not always constitute a closed polygon, and large outliers in the survey connection points have been found. The majority of connection points are related to buildings (instead of parcels). Cadastral sections (part of the unique parcel identification) are not stable, sections have been subdivided, archived, and created of the past 1.5 year. Another observation is that the relation between survey connection points (i.e. through survey project) and cadastral municipality/section is created 'manually' by the TIR user, which is not checked with a geometric check (e.g. ST_Within), about 7% of the connection points are not geometrically within the allocated cadastral section.

The provided data (in MapInfo and ASCII structured format) has been loaded into the Adapted LADM 'Survey Package' in the PostgreSQL/PostGIS database in two steps. In the first step, the provided data is converted into temporary PostGIS tables with a similar data structure, in the second step, the data is converted from the temporary table into the target tables. Various visualisations of the loaded data have been provided with uDig.

A flexible parameterised method for excluding outliers in the connection points (survey points) has been defined, based on a stored function in PostGIS. The selected (included) survey points have been used for an analysis, which showed that the average difference between connection points, before and after the 2nd phase control point constrained network adjustment, is 19.3 cm for the Netherlands (compared to the deviation of ±20 and ±40 cm for urban and rural area). The lowest distance (best quality/accuracy of the cadastral map) is seen in cadastral office Flevoland and

Roermond, the highest is seen in Zoetermeer (presumably classified as urban area), showing a remarkable average difference of over 20 cm).

The reliability of the analysis is influenced by large outliers. The current procedure, of excluding outliers in the connection points, shows different results with different parameters, but the trends (for highest and lowest distance) at cadastral office level are similar for the chosen parameters.

In general, the required "graphical precision" of 20 and 40 cm in respectively urban and rural areas is obtained; however, individual cases exist where these boundaries are exceeded. Further analysis could be done into those individual cadastral sections (Figure 62), and also with regard to specific attributes of the provided data. One of the additionally required data elements, currently not available is a classification of cadastral sections (or municipalities) in either "urban" or "rural".

8 Conclusions and Recommendations

This last chapter presents the conclusions and recommendations as a result of the master thesis project. First, the original research objective, research question and the approach to the project are being reviewed, assessing the achieved goals, as well as the goals and activities that (partially) have not been performed (section 8.1). The conclusions and results with regard to the main topics will be provided in section 8.2 for Model Driven Architecture, Object Constraint Language, and for the analysis of the quality of the Dutch cadastral map. Finally, in section 8.3, the master thesis report will be closed with a number of recommendations with regard to the topics of the master thesis project.

8.1 The Research Objective and Approach Reviewed

The original scope of the master thesis project has been narrowed, based on the result of the mid-term review for this graduation project. The activities, originally set out to be completed within the scope of this master thesis project, were more comprehensive than originally estimated. Therefore, the focus of the master thesis project was set on this part of the objective: "*to gain experience with Model Driven Architecture (MDA) by performing a literature study, and by creating a prototype of the (adapted) LADM Survey Package, based on MDA principles"*. For following activities, related to the part of the objective above, the highest priority was granted:

- *Create an adapted LADM 'Survey Package'*, with two goals: to make this platform independent model suitable as input to the MDA prototype, and suitable to contain the data on connection points, parcels and buildings, cadastral offices, municipalities, and sections, as provided by Kadaster.
- *Design and develop the MDA prototype* (based on the possibilities, offered by the UML/MDA tool Enterprise Architect and its software development toolkit), and test its functions with the adapted LADM 'Survey Package' PIM.
- *Implement the platform specific model* (PSM) for the *Adapted LADM 'Survey Package'* in the target *database PostGIS*, populated with the data provided by Kadaster, as a basis for data analysis.

Other activities and related sub-questions (described in section 1.2) received a lower priority, and have been performed partially, to the extent as described in the following sections, summarised as: the analysis of the differences between measured and transferred coordinates of connection points; the involvement of OCL constraints in the MDA prototype; the extension and improvement of the LADM 'Survey Package'.

8.2 Conclusions

Model Driven Architecture and its main elements have been assessed in Chapter 3, and its relation to standards, constraints and Object Constraint Language in particular has been described in section 3.3 and Chapter 4.

The current definition of the LADM 'Survey Package' has been discussed in Chapter 2. The data handled by Kadaster with regard to the measurements of spatial objects has been discussed in Chapter 5, preparing for an analysis of the quality of the cadastral map, which has been presented in section 7.5.

MDA Prototype Automatically Transforms PIM to PSM to PostGIS
A Model Driven Architecture (MDA) prototype has been built, based on and complaint with the MDA processes and transformations [OMG, 2003], and with help of the transformation possibilities offered by the UML/MDA tool Enterprise Architect (EA, [SparxSystems, 2007]), to automatically transform an object oriented platform independent model (PIM) to a platform specific model (PSM). The Adapted LADM 'Survey Package' is the PIM (i.e. a UML class diagram), and the target PSM is an object-relational PostgreSQL database, with a PostGIS extension for spatial data and functions. The MDA prototype is also capable of automatically generating the DDL scripts to create the Adapted LADM 'Survey Package' objects in the PostGIS database.

Solution for Difference Between O-O (PIM) and Relational DBMS (PSM)
When transforming an object-oriented PIM to a PSM, targeted at a relational database, the difference between object-oriented (O-O) and relational database (RDBMS) definitions has to be resolved. How to implement enumeration classes, or inheritance of attributes and operations? Enterprise Architect offers standard support for the transformations where this difference is small (e.g. class to table, attribute to column), and where the method of implementation is less arbitrary, but more sophisticated transformations (e.g. the implementation of enumeration classes as base table check constraints, and the flattening of class hierarchies into one table) required a considerable custom development to achieve this MDA functionality. This functionality in the MDA prototype is based on a selection and definition of MDA Transformation Rules (as part of the *platform specific transformation specification* depicted in Figure 9), each resolving one of the above mentioned differences between O-O and RDBMS resolve each of the above mentioned . The platform specific transformation specification defines the mapping of the PIM (e.g. [spatial] elements, data types, associations, OCL constraints and operations) to the PSM (e.g. [spatial]

A degree of flexibility was achieved by using tagged values for model elements in the custom developed part (e.g. for flattening class hierarchies), which provides user influence on the transformation to a PSM. Based on the experiments with the MDA prototype, it is expected that the majority of MDA transformation rules, including the ones that have not been considered in the master thesis project, can be performed automatically, provided that the PIM and PSM elements and transformations between them are well defined.

Adapted LADM 'Survey Package' Implemented in PostGIS
The LADM 'Survey Package' was adapted, based on the available Kadaster data, to serve as input to the MDA prototype. The Adapted LADM 'Survey Package' has automatically been transformed to a Platform Specific Model (PSM) for a targeted PostGIS database. Based on the transformed PSM, the MDA prototype has automatically created DDL scripts, with which the PostGIS implementation of the Adapted LADM 'Survey Package' was created, populated with about 7.5 million test records, provided by Kadaster. Open source tools PostgreSQL/PostGIS (object-relational database), uDig (visualisation and analysis of spatial data) and FWTools (conversion of spatial data) have been used extensively and have proven to be suitable for implementation and visualisation activities with regard to the Adapted LADM 'Survey Package'.

Transform and Implement Geometric Data Types and Operations
The custom developed MDA prototype is capable of automatically performing a large portion of the defined MDA transformation rules from PIM to PSM, including handling and transforming a selection of geometric data types (e.g. GM_Point, GM_LineString, GM_Polygon). One of the sub-questions, related to suitability of the MDA prototype for geographic elements of the LADM Survey Package, has been answered for simple geometric data types and operations; however the topological data types and constraints have not used and transformed in the MDA Prototype.

Constraints Assessed and Classified For Implementation
The role of constraints in data modelling has been assessed in Chapter 4. In particular the Object Constraint Language (OCL) has been discussed, and used in examples with regard to the implementation of constraints in relational databases. OCL is a formal language, which has been defined as an extension to UML, to define constraints that cannot be recorded in UML. From an implementation viewpoint, constraints have been divided into constraints applicable to one instance, to multiple instances for one class, or to multiple instances of multiple classes. Relational databases offer functionality to implement constraints such as mandatory columns, default value for column, primary key, unique key, and foreign key constraint, and simple base table check constraint. For other types of constraints, examples of OCL invariants have been defined on the Adapted LADM 'Survey Package' UML class diagram, and used in implementation experiments (section 6.7).

The SQL *assertion* and the *base table check constraint* with sub queries could be used for implementing the OCL invariants, but their functionality is not offered by relational databases like PostgreSQL/PostGIS. Table triggers, firing upon DML actions (i.e. insert, update, delete) at table *row* or *statement* level can offer part of the implementation. However, the development and implementation of a transaction management mechanism is required, to maintain the integrity (based on all, non-violated, constraints) of the relational database.

This transaction management mechanism (section 4.2) will check the constraints at *transaction* level (involving DML on multiple tables), and can be based on "*OCL views*". OCL invariants are transformed into OCL views in the PSM, which query the records that violate the original OCL constraint; a few examples of OCL views have been created (manually) in the PSM of the Adapted LADM 'Survey Package'.

Because these OCL views in principle query all records in the table, proper indexing of these tables and their columns is necessary.

Enterprise Architect offers validation of OCL constraints, but is not capable of transforming or implementing OCL constraints to a relational database, unless functionality is custom developed, as has been done in this master thesis project. The MDA prototype is capable of implementing spatial and non-spatial OCL invariants as based table check constraints. The more complex OCL invariants in the PIM will be transformed to OCL based on PSM elements, as a basis for further manual implementation.

Analysis of Quality of the Cadastral Map at Different Levels
Kadaster has provided data on parcels (polygons) and buildings (linestrings) for the Province of Utrecht, the cadastral offices, municipalities, and sections for The Netherlands, and survey projects and measurements (of connection points). This data has been used to populate the implementation of the Adapted LADM 'Survey Package' PSM in PostGIS. This data has been subject to an analysis with regard to the quality of the cadastral map. This quality is assessed by investigating the differences between the measured coordinate of a connection point (mostly part of a building perimeter), and, the adjusted coordinate of that connection point, i.e. its representation on the (digital) cadastral map (in RDNAP-TRANS spatial reference system), respectively before and after the 2nd phase control point constrained network adjustment (NL: tweede fase aansluitings-vereffening). If the measured coordinate is provided in a local spatial reference system, a similarity transformation has been performed by Kadaster, to transform the coordinate into the RDNAP-TRANS spatial reference system, so that it could be used in the analysis related to transferred coordinates.

The result of the analysis has been presented in section 7.5, partly based on aggregations at the level of cadastral office, municipality and section. In general, the required "graphical precision" of maximum 20 and 40 cm differences in respectively urban and rural areas is obtained at higher aggregations (at national or cadastral office level). However, individual cases at cadastral section level exist where these boundaries seem to have been exceeded, even if the norm is applied that 95% of the measurements should comply with the maximum 20/40 cm differences. The reliability of this analysis is influenced by large outliers, caused by a number of factors, described in 7.5.1. One of these factors is that measured connection points are manually linked to wrong connection points on the cadastral map. The current procedure, of excluding outliers in the survey points, shows different results with different parameters, but the trends remain similar for a range of parameters (e.g. cadastral office with lowest or highest quality/accuracy).

8.3 Recommendations

The following recommendations have been formulated based on the research performed in the master thesis project:

- Improve the Standard MDA Transformations in Enterprise Architect
- Enhance the Current MDA Prototype
- Build MDA Transformation Tool based on XMI
- Extend the OCL with Spatial Definitions
- Further Research into Combination of MDA, OCL, Geometry, and Topology
- Extend the LADM 'Survey Package'
- Further Analysis of Quality of the Cadastral Map
- Implement Improvements with regard to Survey Measurement Handling

Improve the Standard MDA Transformations in Enterprise Architect
During the experiments with the MDA prototype based on Enterprise Architect (EA), the recommendation was formulated to improve the standard possibilities in EA for MDA transformations. With the programming language of the transformation templates (section 6.2.1) and the EA Software Development Kit (section 6.2.2), many MDA Transformations can be done automatically (e.g. with custom development in C#), but only relatively simple MDA transformation rules are offered standard by EA. The current support in EA for OCL is limited to storing OCL and performing some validation of OCL statements. The mentioned improvement should also include the support for OCL (as discussed in section 6.2.3), in terms of validation, transformation from a PIM to a PSM, and also in terms of implementation in a target platform, specifically in the relational database which has the focus in this master thesis project. This implementation should also involve a certain form of a transaction management mechanism (section 4.2), to be able to implement all constraints.

Enhance the Current MDA Prototype
The current MDA prototype, based on EA software, developed to create new implementations of platform independent models, could be expanded to take existing relation databases into account. The transformations and generation of DDL are then able to change existing objects. The prototype could also be extended with regard to other, currently undefined, MDA transformation rules, as well as with regard to support for OCL, which is currently limited to automatic implementation of base table check constraints, for a limited number of (spatial) operations. Current or future developments with regard to OCL (URL 20, URL 28) could be used in this enhancement.

Build MDA Transformation Tool based on XMI
The previous recommendations imply enhancements related to the EA software. This recommendation addresses another approach which is not dependent on commercially supplied software like EA. The development of a tool to support MDA transformations from PIM to PSM is envisioned, based on platform specific transformation specifications, containing a variation of MDA transformation rules for the model elements (UML and OCL), defined for different implementation strategies. The development platform could be Eclipse, and XMI, the OMG standard for exchanging models (i.e. UML and OCL) is the input format, as well as the resulting

output format for this tool. Part of these implementation strategies should focus at an approach for full (OCL) constraint implementation based on row, statement and transaction level DML statements.

Extend the OCL with Spatial Definitions
One of the questions in section 1.2 that has been answered partly is how (spatial) constraints can be specified to the data elements in the LADM 'Survey Package'. Based on the experiments with OCL and spatial data types and operations, and inspired by the availability of spatial standards, now reaching a certain stability and maturity level, a recommendation has been specified. The Object Constraint Language should be extended with spatial data types and operations. At a platform independent level, this could lead for example to the extension of OCL with ISO19107 definitions [ISO/TC211, 2003b], for example the GM_Point, GM_Polygon as used in the MDA prototype, but also topological definitions. At a platform specific level (e.g. SQL and relational databases), this extension could be based on the spatial data types and operations of a standard like ISO/IEC 13249 SQL/MM - Part 3 [ISO/IEC, 2006], for example spatial operation like ST_IsEmpty, ST_Length, ST_Disjoint, ST_Intersects, ST_Crosses, ST_Overlaps, ST_Touches, ST_Buffer, ST_Area, ST_Distance, ST_X, etc.

Further Research into Combination of MDA, OCL, Geometry, and Topology
The MDA prototype, as developed in the master thesis project, is capable of transforming simple geometric data types (e.g. GM_Point, GM_LineString, GM_Polygon, and GM_MultiSurface) and spatial OCL constraints based on geometric operations (e.g. ST_Area, ST_Distance, ST_Within, and ST_Intersects). Future research should be done into the combination of MDA and other geometric data types and operations, as well as topological data types, structures and operations, which have not been assessed in the master thesis project. This could be done in the context of the current MDA tool in EA, or in the context of an XMI based development of a MDA tool, but in any case, also taking into account the possibilities of extending OCL, as previously recommended.

Extend the LADM 'Survey Package'
The extension and improvement of the LADM 'Survey Package' has not been the primary goal in composing the Adjusted LADM 'Survey Package', as explained in section 8.1. However, some errors and improvements have been identified during the course of the master thesis project, as identified in Chapter 2 and in section 5.4, when the Adapted LADM 'Survey Package' was defined as input to the MDA prototype. Further research into the improvement of the Survey Package is recommended, taking the results of this master thesis project into account, as well as the mentioned publications [Ingvarsson, 2005, Lee, 2005, Open Geospatial Consortium, 2006b]. The Extended LADM 'Survey Package' could then be input to MDA based transformations and implementations.

Further Analysis of Quality of the Cadastral Map
The analysis of the differences between measured and transformed coordinates of connection points, indicating the quality (accuracy) of the cadastral map, is recommended to be continued. Further analysis can be done with regard to specific attributes of the provided data, or based on other related attributes, which is currently not provided, nor within the scope of the analysis. One of the sub-questions addressed

the assessment of *urban* and *rural* area, in relation to the outcome of the analysis, which cannot precisely be answered. A classification of, for example, every cadastral section into either "rural" or "urban" sections is required, as well as more analysis of the cause of the outliers, influencing the reliability of the analysis. Furthermore, the analysis of individual cases (cadastral sections) is recommended, within or outside the current study area (Province of Utrecht), to enable a better handling (exclusion) of outliers, and a reliable analysis of the quality of the cadastral map.

Implement Improvements with regard to Survey Measurement Handling
A recommendation is made to include a number of issues in future information system developments of Kadaster. The current survey measurement handling system is based on different and separated applications, which exchange information based on various files in different formats. Not all survey measurement handling related data is preserved (digitally and centrally), and a number of tasks are done 'manually' and left to the discipline, judgement and checks of the user. Permanent storage of the survey measurement (handling) related data in an integrated structured database with constraint validation and handling is recommended. The extension of the LADM 'Survey Package', one of the other recommendations, is highly related to these developments and improvements, which all together can lead to a higher quality and accuracy of the cadastral map.

For example, the current survey measurement handling system is aimed at adjusting ("fitting in") accurate measurements into a less accurate cadastral map. This involves the survey measurements, but also the (meta) data describing what exactly happened to the survey measurement, how were the measurements transformed to the cadastral map, which connection points were used, which connection points were rejected and based on which argumentation, which error checking has been performed, and what was the result. One of the advantages of storing the relation between originally measured coordinates and the transferred coordinates of points on the cadastral map, is that a reverse "fitting" process would be possible; adjusting the (less accurate) cadastral map to the more accurate measurements, eventually leading to a more accurate cadastral map. In relation to this, further research in the possibilities for upgrading the quality of the cadastral map, based on accurate survey measurements is recommended.

9 Appendices

Appendix A: LADM UML Class Diagrams

In the following section figures are presented, showing different parts of the Land Administration Domain Model of ISO 19152 [ISO/TC211, 2008]:

- Figure 69 - LADM Registered Objects (taken from [ISO/TC211, 2008], fig.2)
- Figure 70 - LADM Parcels (taken from [ISO/TC211, 2008], fig.3)
- Figure 71 - LADM Spatial Representation of Parcels and Survey Points (taken from [ISO/TC211, 2008], fig.4)
- Figure 72 - LADM Documents (taken from [ISO/TC211, 2008], fig.5)
- Figure 73 - LADM Enumeration and CodeList classes (taken from [ISO/TC211, 2008], fig.6)

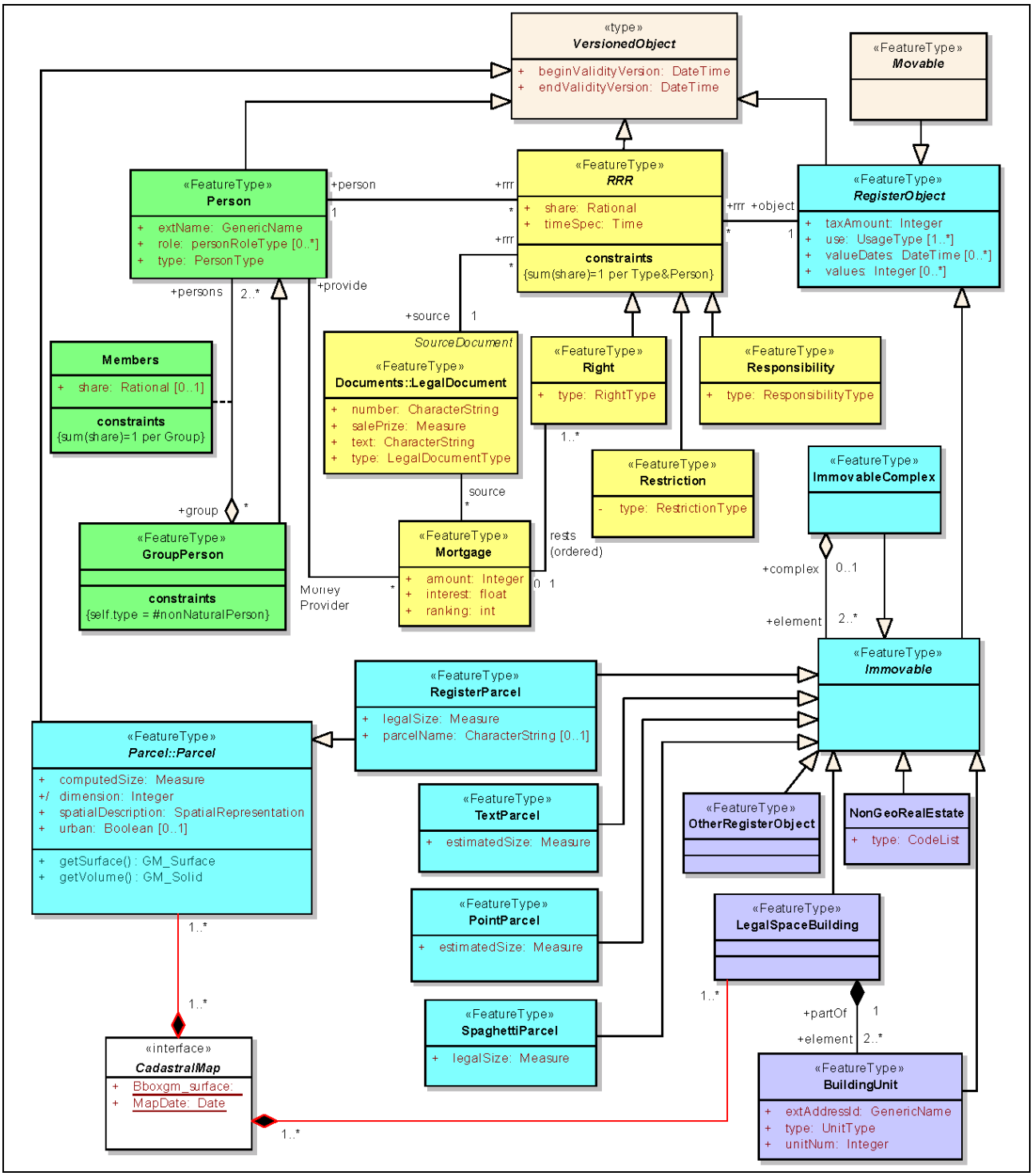


Figure 69 - LADM Registered Objects (taken from [ISO/TC211, 2008], fig.2)

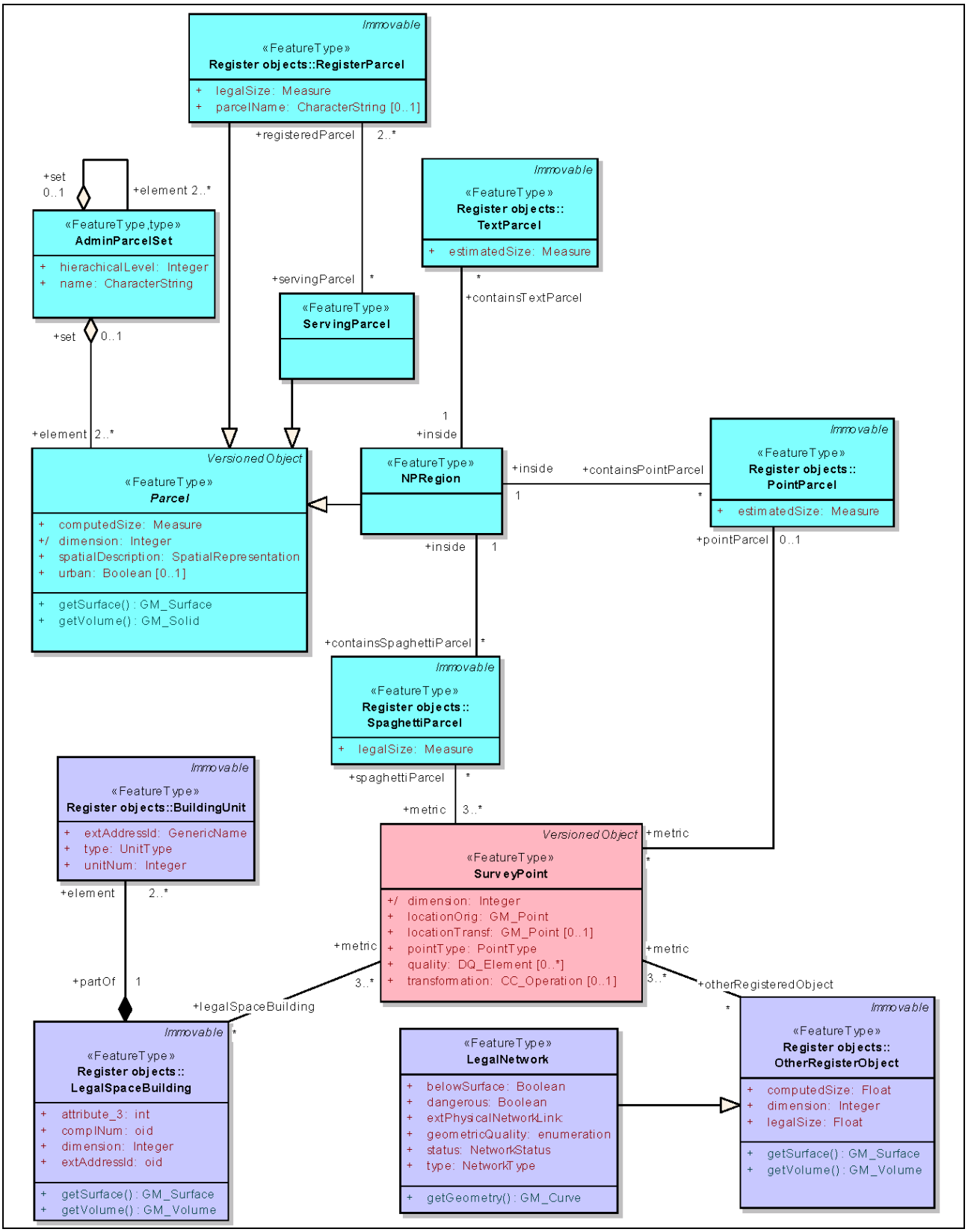


Figure 70 - LADM Parcels (taken from [ISO/TC211, 2008], fig.3)

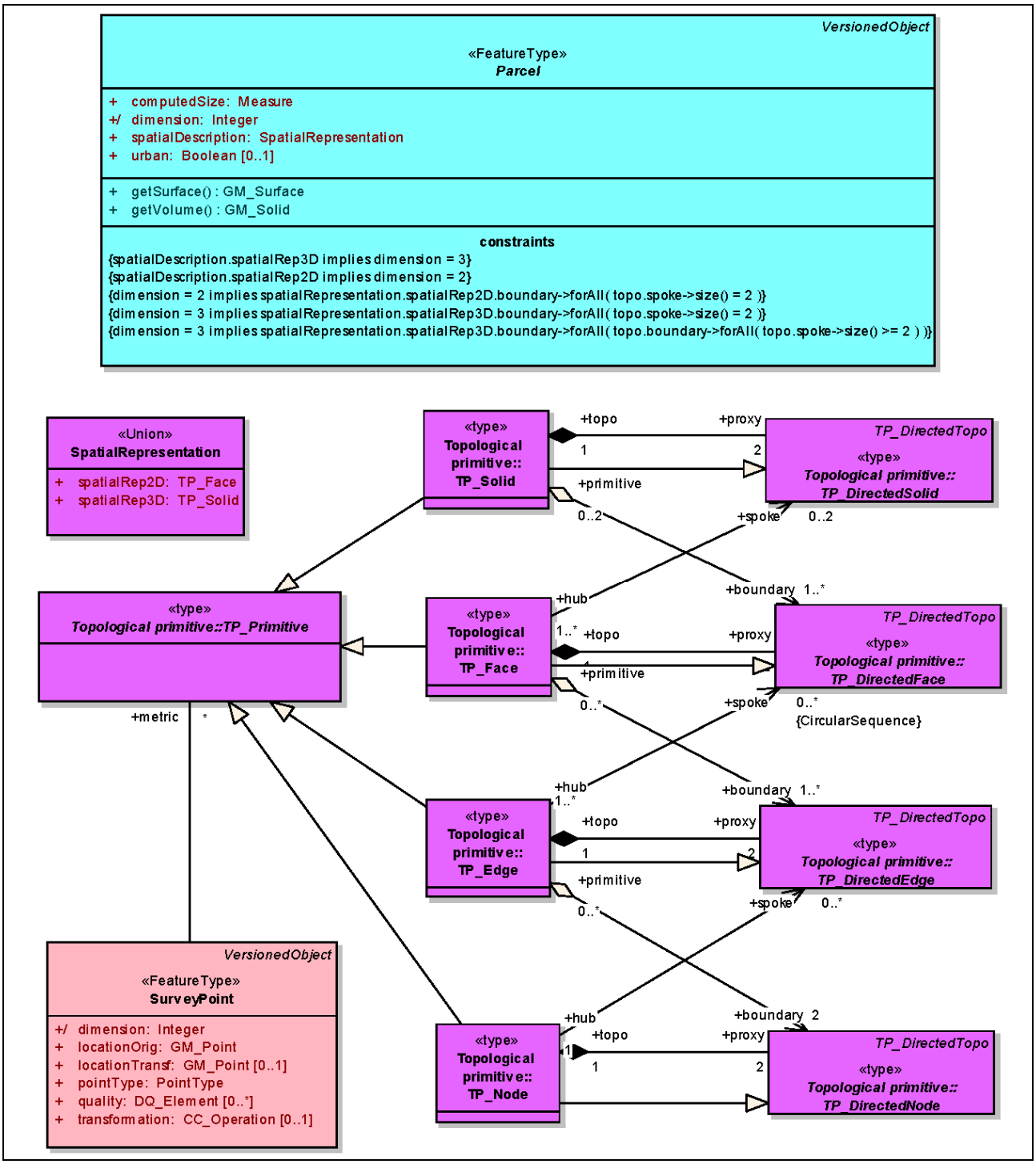


Figure 71 - LADM Spatial Representation of Parcels and Survey Points (taken from [ISO/TC211, 2008], fig.4)

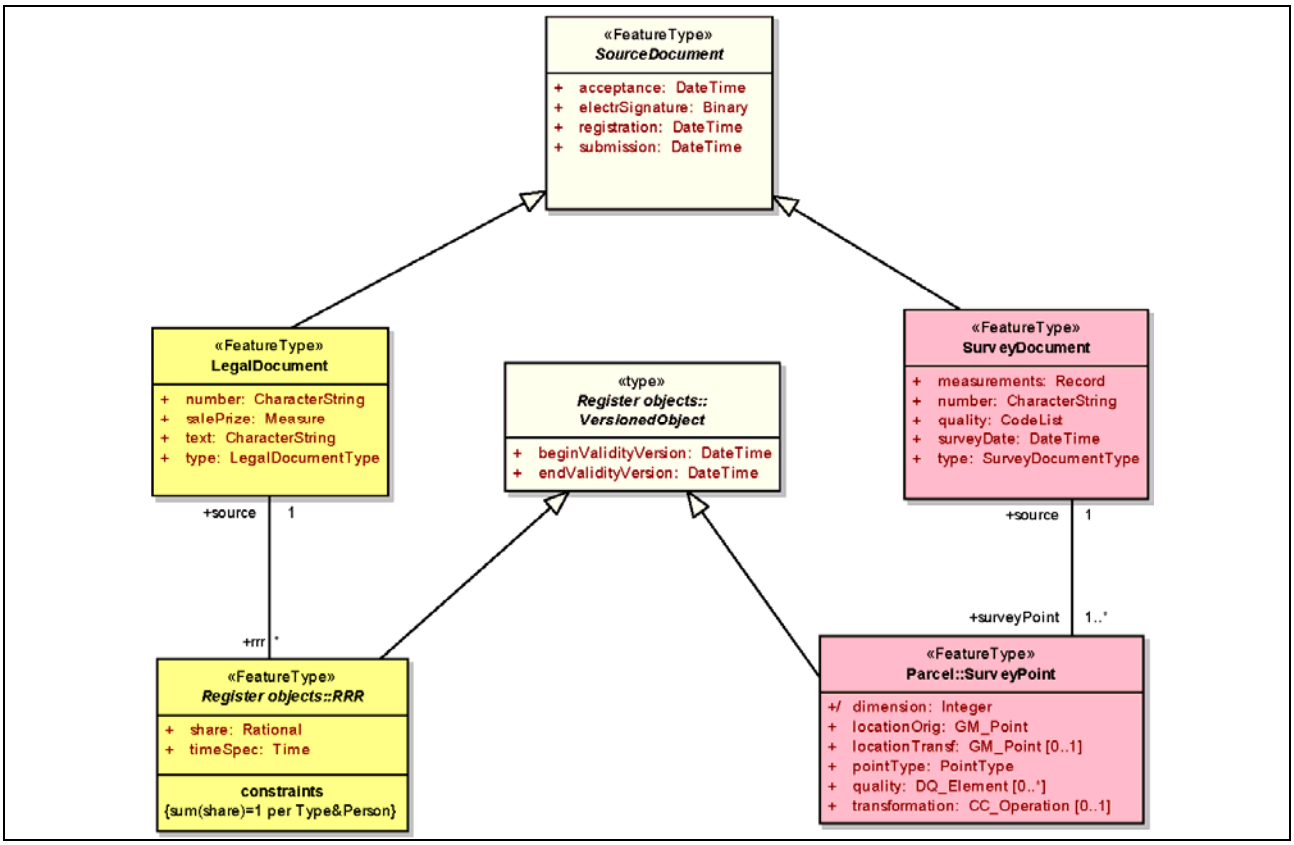


Figure 72 - LADM Documents (taken from [ISO/TC211, 2008], fig.5)

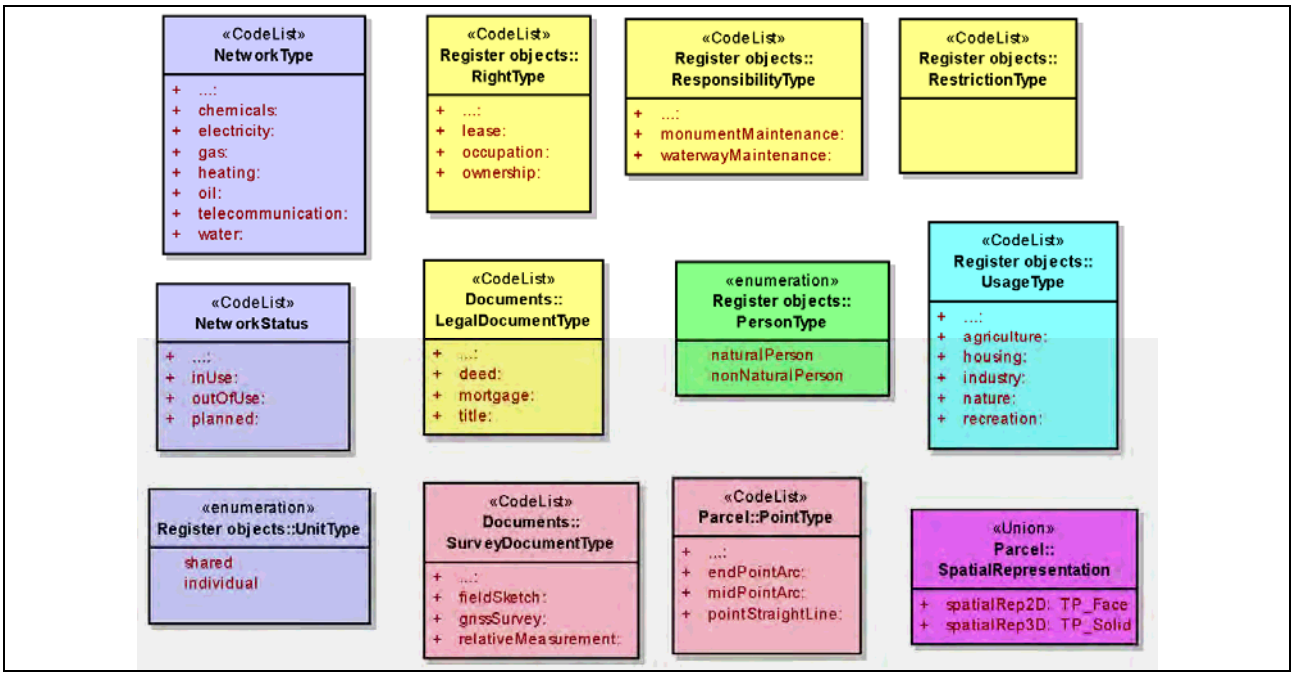


Figure 73 - LADM Enumeration and CodeList classes (taken from [ISO/TC211, 2008], fig.6)

Appendix B: Overview LADM/CCDM/STDM Classes

In this appendix an overview is provided of the classes used in three different publications on the Land Administration Domain Model (LADM), the Core Cadastral Domain Model (CCDM) and the Social Tenure Domain Model (STDM).

Figure 74 - Overview of LADM classes in different articles

CCDM classes [Lemmen and Van Oosterom, 2006, Van Oosterom et al., 2006]	LADM classes [ISO/TC211, 2008]	STDM classes [Lemmen et al., 2007]	Comments
RRR	RRR	SocialTenureRelation	Feature Type
Responsibility	Responsibility		
Restriction	Restirction		View
Right	Right		
LegalDocument	LegalDocument	SocialTenureInventory	
Mortgage	Mortgage	Collateral	Feature Type
Person	Person	Person	Feature Type
NaturalPerson		NaturalPerson	enumeration
nonNaturalPerson		nonNaturalPerson	enumeration
GroupPerson	GroupPerson	GroupPerson	
Members	Members	Members	Feature Type; Association class
MoneyProvidor		MoneyProvidor	
Conveyor		Conveyor	
Surveyor		Surveyor	
RegisterObject	RegisterObject		Feature Type
Immovable	Immovable	SpatialUnit	
ParcelComplex	ImmovableComplex	SpatialUnitComplex	in ISO 19152 replaced by ImmovableComplex ([ISO/TC211, 2008])
PartOfParcel		PartOfParcel	
SpaghettiParcel	SpaghettiParcel	Incomplete SpatialUnit	
PointParcel	PointParcel	PointBased SpatialUnit	
TextParcel	TextParcel	Descriptive SpatialUnit	
		SketchPhoto SpatialUnit	
NonGeoRealEstate	NonGeoRealEstate	FishingRights; OverlappingSpatialUnit	
Building	LegalSpaceBuilding	Building	
Unit	BuildingUnit	Unit	specialised by IndividualUnits and SharedUnits
SharedUnit		SharedUnit	not used in ISO 19152 [ISO/TC211, 2008]
IndividualUnit		IndividualUnit	not used in ISO

CCDM classes [Lemmen and Van Oosterom, 2006, Van Oosterom et al., 2006]	LADM classes [ISO/TC211, 2008]	STDM classes [Lemmen et al., 2007]	Comments
			19152 [ISO/TC211, 2008]
OtherRegisterObject	OtherRegisterObject		
Movable	Movable		
AdminParcelSet	AdminParcelSet	AdminParcelSet	
Parcel	Parcel	Parcel	Feature Type
RegisterParcel	RegisterParcel	RegisterParcel	
ServingParcel	ServingParcel	ServingParcel	
NPRegion	NPRegion	NPRegion	
SourceDocument	SourceDocument	SourceDocument	Feature Type
SurveyDocument	SurveyDocument	SpatialUnitInventory	
SurveyPoint	SurveyPoint	SurveyPoint	Feature Type
	SpatialRepresentation		Union, used in Parcel.spatialDescription
GeomTopoRepresentation	TP_Primitive	GeomTopoRepresentation	
TP_Volume_3D	TP_Solid	TP_Volume_3D	
TP_Face_3D	TP_Face	TP_Face_3D	
TP_Edge_3D	TP_Edge	TP_Edge_3D	
TP_Node_3D	TP_Node	TP_Node_3D	
TP_Face_2D	TP_Face	TP_Face_2D	
TP_Edge_2D	TP_Edge	TP_Edge_2D	
TP_Node_2D	TP_Node	TP_Node_2D	
	TP_DirectedSolid		
	TP_DirectedFace		
	TP_DirectedEdge		
	TP_DirectedNode		
SheetOfRegistry		SocialTenureFolio	Interface object, a.k.a. OwnershipFolio
CadastralMap	CadastralMap	SpatialUnitMap	Interface object, a.k.a. CadMap
	PersonType		enumeration
	UnitType		enumeration
	RightType		CodeList
	ResponsibilityType		CodeList
	RestrictionType		CodeList
	SurveyDocumentType		CodeList
	LegalDocumentType		CodeList
	PointType		CodeList
	UsageType		CodeList
	NetworkType		

CCDM classes [Lemmen and Van Oosterom, 2006, Van Oosterom et al., 2006]	LADM classes [ISO/TC211, 2008]	STDM classes [Lemmen et al., 2007]	Comments
	NetworkStatus		
		TenureType	
	VersionedObject		Type
	Referencable SpatialObject		INSPIRE Base Types
	Referencable VersionedObject		
	LegalNetwork		

Figure 74 - Overview of LADM classes in different articles

In Figure 74 the annotation "Feature Type" indicates that the class is related to ISO features [Van Oosterom et al., 2006].

Appendix C: Examples of Survey Files (Kadaster)

In Figure 75 an overview is provided of files that play are role in handling survey measurements, as well as the files that are provided by Kadaster's "Registration Map Quality" project (see section 5.2.3, 7.4.3, and Figure 101). The differences between measured and transferred coordinates of connection points are provided through the PhaseDifferenceFile (RKK).

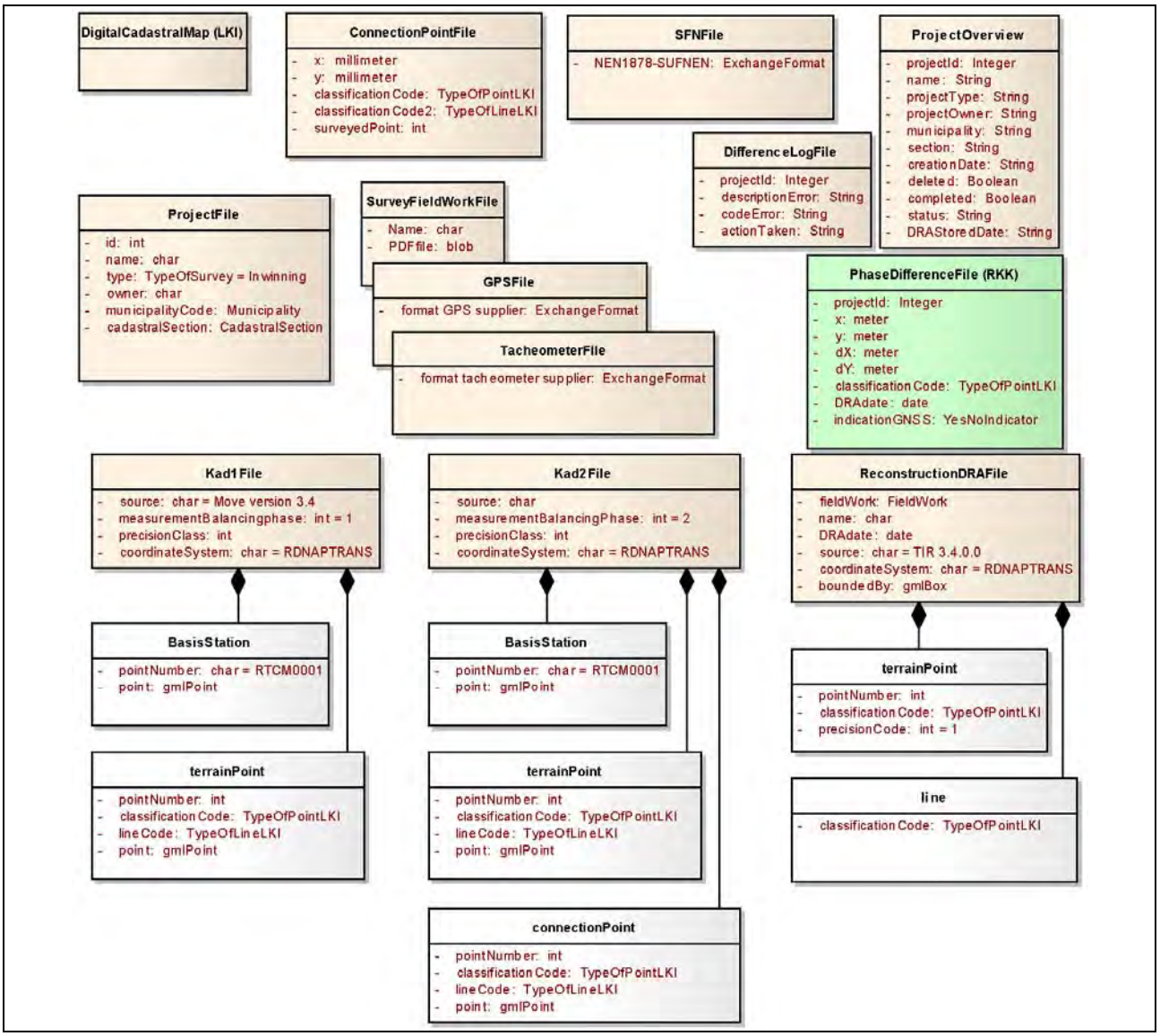


Figure 75 - Files Used during Handling Survey Measurements(LKI, TIR, MOVE3)

Appendix D: Examples of EA Transformation Definition 'PostgreSQL'

The EA Transformation Definition "PostgreSQL" used in the "First Transformation from PIM to PSM-1" consists of a number of transformation templates, as depicted in Figure 76, showing a part of the EA user interface for maintaining Transformation Definitions (i.e. for target object-relational database PostgreSQL). Selected is the transformation template for individual Attributes.

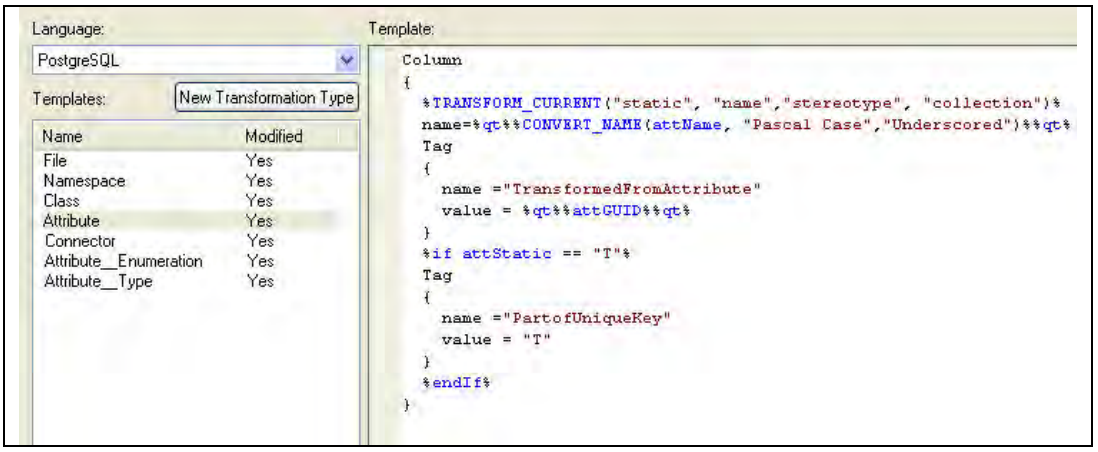


Figure 76 - Overview EA Transformation Definition "PostgreSQL"

Other examples of transformation templates "Class" and "Connector" are provided in:

- Figure 77 - Example EA Prototype: Transformation Template "Class"
- Figure 78 - Example EA Prototype: Transformation Template "Connector"

The conversion templates *File*, *Namespace*, *Class*, *Attribute* and *Connector* have been taken from the standard EA Transformation Definition "DDL", and have been adjusted for the MDA Prototype. The conversion templates *Attribute__Enumeration* and *Attribute__Type* have been created for the MDA prototype.

The result of the transformation templates is written in a temporary text file (see section 6.2.1: "Intermediary File"), which is then used to create EA elements in a PSM.

Class
The transformation template "Class" as used in the MDA prototype, adapted from the standard EA transformation template for target platform "DDL", is provided below (Figure 77). This transformation template transforms PIM classes to PSM tables and types.

For each class the tagged value "TransformToPSM" is evaluated and the template is structured according to:

• Class stereotyped as <<CodeList>>
- Create **Table** in the intermediary text file (e.g. *codelist_surveydocumenttype*) with column "value". - Create tag "TransformedFromClass" and populate the tagged value with the originating Class GUID. - Create the primary key + column - Create tag "property" and populate the tagged value with settings for the sequence for the primary key. - Create (temporary) **Class** in the intermediary text file (to be used in subsequent transformations). - Create tagged value "MarkForDeletePSM" = "T" to indicate the need for deletion of this class after use in sub sequent transformations.
• Class stereotyped as <<enumeration>>
- Create (temporary) **Class** in the intermediary text file (to be used in subsequent transformations). - Create tag "TransformedFromClass" and populate the tagged value with the originating Class GUID. - Create tagged value "MarkForDeletePSM" = "T" to indicate the need for deletion of this class after use in sub sequent transformations. - Call the transformation template "*Attribute__Enumeration*"
• Class stereotyped as <<type>>
- Create (temporary) **Class** in the intermediary text file (to be used in subsequent transformations). - Create tag "TransformedFromClass" and populate the tagged value with the originating Class GUID. - Create tagged value "MarkForDeletePSM" = "T" to indicate the need for deletion of this class after use in sub sequent transformations. - Call the transformation template "*Attribute__Type*"
• Other classes, implemented as table
- Create **Table** in the intermediary text file - Create tag "TransformedFromClass" and populate the tagged value with the originating Class GUID. - Create the primary key + column - Create tag "property" and populate the tagged value with settings for the sequence for the primary key. - Call the transformation template "*Attribute*" - Call the transformation template "*Connector*"

Figure 77 - Example EA Prototype: Transformation Template "Class"

```
$COMMENT="  The class template is used for all elements along with the  "
$COMMENT="  This template transforms classes into stereotype tables  "
$COMMENT="  A %TRANSFORM_REFERENCE()% macro call is required to        "
$COMMENT="  identify the transformed class.                           "

$TransformToPSM = %EXEC_ADD_IN ("PrototypeAddin", "transformToPSM", classGUID)%

-----------------------------------------------------------------------------------------------------------------------------------
$COMMENT="  CodeList stereotypes are transformed as tables "
%if $TransformToPSM =="T" and classStereotype=="CodeList"%
  Table
  {
    %TRANSFORM_REFERENCE("Table")%
    %TRANSFORM_CURRENT("language", "abstract", "name")%
    name = %qt%codelist_%TO_LOWER(className)%%qt%
    language=%qt%PostgreSQL%qt%

    Tag
    {
      name ="TransformedFromClass"
      value = %qt%%classGUID%%qt%
    }

    PrimaryKey
    {
      name = %qt%pk_codelist_%CONVERT_NAME(className, "Pascal Case","Underscored")%%qt%
      Column
      {
        name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName", classGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%

        Tag
        {
          name ="property"
          $COMMENT=" for autonumbering, produce a tag like value =
AutoNum=1;StartNum=33;Increment=11;NotForRep=0;"
          value = %qt%%EXEC_ADD_IN ("PrototypeAddin", "GetSequenceStartIncrement", classGUID)%%qt%
        }
      }
    }
    Column
    {
      name=%qt%value%qt%
      type=%qt%varchar%qt%
    }
  }
```

```
  $COMMENT="  CodeList also generated as class"
  Class
  {
    %TRANSFORM_REFERENCE("Class")%
    %TRANSFORM_CURRENT("language", "abstract")%
    language=%qt%PostgreSQL%qt%
    $COMMENT=" use Tag to know the original class "
    Tag
    {
      name ="TransformedFromClass"
      value = %qt%%classGUID%%qt%
    }
    Tag
    {
      name ="MarkForDeletePSM"
      value = "T"
    }
    $COMMENT="  attributes of enumeration stereotypes are transformed "
    %list="Attribute__Enumeration" @separator="\n" @indent="  "%
  }
%endTemplate%
```

```
$COMMENT="  enumeration stereotypes are transformed as classes "
%if $TransformToPSM =="T" and classStereotype=="enumeration"%
Class
{
  %TRANSFORM_REFERENCE("Class")%
  %TRANSFORM_CURRENT("language", "abstract")%
  language=%qt%PostgreSQL%qt%

  $COMMENT=" use Tag to know the original class "
  Tag
  {
    name ="TransformedFromClass"
    value = %qt%%classGUID%%qt%
  }
  Tag
  {
    name ="MarkForDeletePSM"
    value = "T"
  }
  $COMMENT="  attributes of enumeration stereotypes are transformed "
  %list="Attribute__Enumeration" @separator="\n" @indent="  "%
}
%endTemplate%
```

```
$COMMENT="  type stereotypes are transformed as classes "
%if $TransformToPSM =="T" and classStereotype=="type"%
```

```
Class
{
  %TRANSFORM_REFERENCE("Class")%
  %TRANSFORM_CURRENT("language", "abstract")%
  $COMMENT="  attributes of enumeration stereotypes are transformed "
  %list="Attribute__Type" @separator="\n" @indent="  "%

  $COMMENT=" use Tag to know the original class "
  Tag
  {
    name ="TransformedFromClass"
    value = %qt%%classGUID%%qt%
  }
  Tag
  {
    name ="MarkForDeletePSM"
    value = "N"
  }
}
%endTemplate%

%if classStereotype=="interface" or classStereotype=="Union"%
%endTemplate%
```

```
$COMMENT="  classed implemented as tables "
%if $TransformToPSM =="T"%
Table
{
  %TRANSFORM_REFERENCE("Table")%
  %TRANSFORM_CURRENT("language", "stereotype", "abstract", "name")%
  name = %qt%%CONVERT_NAME(className, "Pascal Case","Underscored")%%qt%
  language=%qt%PostgreSQL%qt%

  $COMMENT=" use Tag to know the original class "
  Tag
  {
    name ="TransformedFromClass"
    value = %qt%%classGUID%%qt%
  }

  %if elemType != "Association"%
  PrimaryKey
  {
    name = %qt%pk_%CONVERT_NAME(className, "Pascal Case","Underscored")%%qt%
    Column
    {
      name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName", classGUID)%%qt%
      type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
```

```
      Tag
      {
        name ="property"
        value = %qt%%EXEC_ADD_IN ("PrototypeAddin", "GetSequenceStartIncrement", classGUID)%%qt%
      }
    }
  }
  %endIf%
---------------------------------------------------------------------------------------------------------------------------------------------------------------
  %list="Attribute" @separator="\n" @indent="  "%
}
%list="Connector" @separator="\n"%

%endTemplate%
```

Figure 77 - Example EA Prototype: Transformation Template "Class"

Connector (Association)

The transformation template "Connector" as used in the MDA prototype, adapted from the standard EA transformation template for target platform "DDL", is provided below (Figure 78). This transformation template transforms PIM associations to PSM relationships and intersection tables, for each connector (UML: association), the type of connector is evaluated, according to the template structure:

- Connector, representing a "Generalisation"
  - Create foreign key + column
- Connector, representing a "Association" or "Aggregation"
  - Determine multiplicity of association ends (source and target) to determine the implementation in a PSM
    - Many to Many associations:
      - Create "intersection" table (e.g. intersection_building_boundary_to_survey_point)
      - Create primary key + columns for "intersection" table
      - Create foreign keys + columns for "intersection" table
    - Many to One & One to Many associations
      - Create foreign key + column

Figure 78 - Example EA Prototype: Transformation Template "Connector"

```
$COMMENT="  The connector template is used for copying connector information "

$COMMENT=" relationship GENERALISATION "

$TransformSourceToPSM = %EXEC_ADD_IN ("PrototypeAddin", "transformToPSM",
connectorSourceElemGUID)%
$TransformTargetToPSM = %EXEC_ADD_IN ("PrototypeAddin", "transformToPSM", connectorDestElemGUID)%
-------------------------------------------------------------------------------------------------------------------------------------------------------
%if connectorType == "Generalization" and $TransformSourceToPSM=="T" and
$TransformTargetToPSM=="T"%
ForeignKey
{
  %TRANSFORM_REFERENCE("General",connectorGUID)%
  Source
  {
    %TRANSFORM_REFERENCE("Table",connectorSourceElemGUID)%

    $COMMENT=" name of the connector/relationship "
    $COMMENT=" based on connector name or destination table "
    %if connectorName != ""%
      name=%qt%fk_%CONVERT_NAME(connectorName, "Pascal Case","Underscored")%%qt%
    %else%
      name=%qt%fk_%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%%qt%
    %endIf%

    multiplicity="0..1"
    Column
    {
      name=%qt%%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorDestElemGUID)%%qt%
      type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
    }
  }
  Target
  {
    %TRANSFORM_REFERENCE("Table",connectorDestElemGUID)%
    multiplicity="1"
    Column
    {
      $COMMENT=" target column is determined by GetPrimaryKeyColumnName "
      name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName", classGUID)%%qt%
      type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
    }
  }
```

```
}
%endTemplate%

$COMMENT=" only deal with Association and Aggregation "
%if connectorType != "Association" and connectorType!="Aggregation"%
%endTemplate%
```

```
$COMMENT=" relationship ASSOCIATION and AGGREGATION "

$COMMENT=" Determine the multiplicity of source and target"
%if connectorDestElemType=="Association"%
  $destMultiple = "TRUE"
  $srcMult = %connectorSourceMultiplicity%
%elseIf connectorSourceElemType=="Association"%
  $sourceMultiple = "TRUE"
  $dstMult = %connectorDestMultiplicity%
%else%
  $srcMult = %connectorSourceMultiplicity%
  $dstMult = %connectorDestMultiplicity%

%if $srcMult != "" and $srcMult != "0" and $srcMult != "0..1" and $srcMult != "1"%
  $sourceMultiple = "TRUE"
%endIf%

%if $dstMult != "" and $dstMult != "0" and $dstMult != "0..1" and $dstMult != "1"%
  $destMultiple = "TRUE"
%endIf%

%if $srcMult == "0..1"%
  $sourceAllowDuplicates = "T"
%endIf%

%endIf%
```

```
%if $sourceMultiple == "TRUE" and $destMultiple == "TRUE" and $TransformSourceToPSM=="T" and
$TransformTargetToPSM=="T"%
  $COMMENT=" Many:Many relationships"
  Table
  {
    %TRANSFORM_REFERENCE("LinkTable",connectorGUID)%

    $COMMENT=" name of the connector/relationship "
    $COMMENT=" based on connector name or destination table "

    %if connectorName != ""%
      $linkTableName="intersection_" + %CONVERT_NAME(connectorName, "Pascal Case","Underscored")%
    %else%
      $linkTableName="intersection_" + %CONVERT_NAME(connectorSourceElemName, "Pascal
Case","Underscored")%+ "_to_" + %CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%
```

```
    %endIf%

     name = %qt%$linkTableName%qt%
    language=%qt%PostgreSQL%qt%

        Tag
    {
      name ="TransformedFromRelationship"
      value = %qt%%connectorGUID%%qt%
    }

    PrimaryKey
    {
      name = %qt%pk_$linkTableName%qt%
      Column
      {
        name=%qt%%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorDestElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
      Column
      {
        name=%qt%%CONVERT_NAME(connectorSourceElemName, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorSourceElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
  }
  $COMMENT=" Many:Many relationships"
  $COMMENT=" FK to Destination Table"
  ForeignKey
  {
    %TRANSFORM_REFERENCE("FK1",connectorGUID)%
    Source
    {
      %TRANSFORM_REFERENCE("LinkTable",connectorGUID)%
      name=%qt%fk_%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%%qt%
      %if $srcMult != ""%
        multiplicity=%qt%$srcMult%qt%
      %endIf%
      Column
      {
        name=%qt%%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorDestElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
    Target
    {
      %TRANSFORM_REFERENCE("Table",connectorDestElemGUID)%
```

```
      Column
      {
        name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName",
connectorDestElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
  }
  $COMMENT=" Many:Many relationships"
  $COMMENT=" FK to Destination Table"
  ForeignKey
  {
    %TRANSFORM_REFERENCE("FK2",connectorGUID)%
    Source
    {
      %TRANSFORM_REFERENCE("LinkTable",connectorGUID)%
      name=%qt%fk_%CONVERT_NAME(connectorSourceElemName, "Pascal Case","Underscored")%%qt%
      %if $dstMult != ""%
        multiplicity=%qt%$dstMult%qt%
      %endIf%
      Column
      {
        name=%qt%%CONVERT_NAME(connectorSourceElemName, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorSourceElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
    Target
    {
      %TRANSFORM_REFERENCE("Table",connectorSourceElemGUID)%
      Column
      {
        name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName",
connectorSourceElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
  }
%elseIf $sourceMultiple == "TRUE" and $TransformSourceToPSM=="T" and $TransformTargetToPSM=="T"%
  $COMMENT=" Source:Target = Many:1"
  ForeignKey
  {
    %TRANSFORM_REFERENCE("FK1",connectorGUID)%
    Source
    {
      %TRANSFORM_REFERENCE("Table",connectorSourceElemGUID)%

      $COMMENT=" default name of relation is destination class name "
      name=%qt%fk_%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%%qt%
```

```
      %if connectorSourceElemType=="Association"%
        name=%qt%fk_%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%%qt%
      %elseIf connectorName != ""%
        $COMMENT=" Use connector/relationship name if not null "
        name=%qt%fk_%CONVERT_NAME(connectorName, "Pascal Case","Underscored")%%qt%
      %elseIf connectorDestRole != ""%
        $COMMENT="Use Destination Role if not null (and if  connector/relationship name is null) "
        name=%qt%fk_%CONVERT_NAME(connectorDestRole, "Pascal Case","Underscored")%%qt%
      %endIf%

      %if $srcMult != ""%
        multiplicity=%qt%$srcMult%qt%
      %endIf%
      Column
      {
        %if connectorDestRole != ""%
          $COMMENT=" Use Destination Role if not null "
          name=%qt%%CONVERT_NAME(connectorDestRole, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorDestElemGUID)%%qt%
        %else%
          name=%qt%%CONVERT_NAME(connectorDestElemName, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorDestElemGUID)%%qt%
        %endIf%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
    Target
    {
      %TRANSFORM_REFERENCE("Table",connectorDestElemGUID)%
      %if $dstMult != ""%
        multiplicity=%qt%$dstMult%qt%
      %endIf%
      Column
      {
        $COMMENT=" Target column doesn't change "
        name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName",
connectorDestElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
  }
   $COMMENT=" %else% "
%elseIf $TransformSourceToPSM=="T" and $TransformTargetToPSM=="T"%
  $COMMENT=" Source:Target = 1:Many"
  ForeignKey
  {
    %TRANSFORM_REFERENCE("FK1",connectorGUID)%
    Source
    {
```

```
      %TRANSFORM_REFERENCE("Table",connectorDestElemGUID)%

      $COMMENT=" default name of relation is source class name "
      name=%qt%fk_%CONVERT_NAME(connectorSourceElemName, "Pascal Case","Underscored")%%qt%
      %if connectorDestElemType=="Association"%
        name=%qt%fk_%CONVERT_NAME(connectorSourceElemName, "Pascal Case","Underscored")%%qt%
      $COMMENT=" Use connector/relationship name if not null "
      %elseIf connectorName != ""%
        name=%qt%fk_%CONVERT_NAME(connectorName, "Pascal Case","Underscored")%%qt%
      $COMMENT=" Use Source Role if not null (and if connector/relationship name is null) "
      %elseIf connectorSourceRole != ""%
        name=%qt%fk_%CONVERT_NAME(connectorSourceRole, "Pascal Case","Underscored")%%qt%
      %endIf%
      %if $dstMult != ""%
        multiplicity=%qt%$dstMult%qt%
      %endIf%
      Column
      {
        $COMMENT=" Use Source Role if not null (and if connector/relationship name is null) "
        %if connectorSourceRole != ""%
          name=%qt%%CONVERT_NAME(connectorSourceRole, "Pascal Case","Underscored")%_%EXEC_ADD_IN
("PrototypeAddin", "GetPrimaryKeyColumnName", connectorSourceElemGUID)%%qt%
        %else%
          name=%qt%%CONVERT_NAME(connectorSourceElemName, "Pascal
Case","Underscored")%_%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName",
connectorSourceElemGUID)%%qt%
        %endIf%
          type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
    Target
    {
      %TRANSFORM_REFERENCE("Table",connectorSourceElemGUID)%
      %if $srcMult != ""%
        multiplicity=%qt%$srcMult%qt%
      %endIf%
      Column
      {
        $COMMENT=" Target column doesn't change "
        name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName",
connectorSourceElemGUID)%%qt%
        type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%
      }
    }
  }
%endIf%
```

Figure 78 - Example EA Prototype: Transformation Template "Connector"

Appendix E: Example EA MDA Prototype Source Code

The EA MDA prototype consists of about 5000 lines of code, divided into 2 main program unit packages:

- **Prototype** for the "First Transformation from PIM to PSM-1"
  See an overview of program units in "Figure 79 - Selected Program Units for First MDA Transformation: Prototype" on page 144, which are used in the EA Transformation Definition, see "Appendix D: Examples of EA Transformation Definition 'PostgreSQL'".

- **Transformation**; a considerable custom development of program units for the "Second Transformation from PSM-1 to PSM-2" and "Third Transformation from PIM OCL to PSM-2". See an overview of program units in "Figure 80 - Selected Program Units for Second and Third MDA Transformation: Transformation" on page 145.

Some examples of individual program units are provided in:

- Figure 81 - Example EA Prototype: GetClassTagValue (2nd & 3rd transformation, page 148); Retrieve tagged values from specific classes.
- Figure 82 - Example EA Prototype: ProcessEnumerationClass (2nd transformation, page 149); transform <<enumeration>> and <<CodeList>> classes from PIM to PSM-1.
- Figure 83 - Example EA Prototype: transformToPSM (page 152); in a transformation template (1st transformation), determine if a class should be transformed from PIM to PSM-1.

Program Unit Package 'Prototype'

This program unit package is based on so-called "Add-in" examples, supplied by Enterprise Architect. The unchanged EA program units are prefixed by "EA", named in a red colour (e.g. EA_Connect). These program units are use by the EA Transformation Definitions, in the "First Transformation from PIM to PSM-1" as described in section 6.2.1 and 6.5. A selection of the program units is provided below in Figure 79, with the program unit "transformToPSM" highlighted, which will be further explained in section "Program Unit: transformtoPSM" on page 152.

```
// Called when EA is started
public String EA_Connect(EA.Repository Repository)

// Called when EA is stopped
public void EA_Disconnect()

// Populates the Menu with our desired selections.
public object EA_GetMenuItems(EA.Repository Repository, string Location, string MenuName)

// Sets the state of the menu depending if there is an active project or not
bool IsProjectOpen(EA.Repository Repository)

// Called once Menu has been opened to see what menu items are active.
public void EA_GetMenuState(EA.Repository Repository, string Location, string MenuName, string
ItemName, ref bool IsEnabled, ref bool IsChecked)

// Called when user makes a selection in the menu.
public void EA_MenuClick(EA.Repository Repository, string Location, string MenuName, string
ItemName)

// Provide default column name for PK
public object GetPrimaryKeyColumnName(EA.Repository repository, object argsObject)

// Provide default column datatype for PK
public object GetPrimaryKeyColumnDataType(EA.Repository repository, object argsObject)

// Provide default sequence start position and incrementation for PK
public object GetSequenceStartIncrement(EA.Repository repository, object argsObject)

// Check if class is marked for transformation (tagged value 'TransformToPSM', Figure 83)
public object transformToPSM(EA.Repository repository, object argsObject)

// read the individual value of the constant (used by 'readPrototypeAddinConstant')
private string readConstantValues ( XmlTextReader myTextReader)

// read the Prototype Constants used during transformation
private void readPrototypeAddinConstant(string xmlFileName, bool displayConstants)
```

Figure 79 - Selected Program Units for First MDA Transformation: Prototype

Program Unit Package 'Transformation'
This program unit package is completely custom developed for the MDA prototype, based on the Enterprise Architect Software Development Kit (EA SDK, section 6.2.2), on behalf of the "Second Transformation from PSM-1 to PSM-2", the "Third Transformation from PIM OCL to PSM-2", and the "Transformation from PSM to DDL (PostgreSQL/PostGIS)", respectively section 6.6, 6.7, and 7.3. A selection of the program units, with a short description, is provided below in Figure 80, with the program unit "*GetClassTagValue*" and "*ProcessEnumerationClass*" highlighted, which will be further explained in section "Program Unit: GetClassTagValue" (page 148), and "Program Unit: ProcessEnumerationClass" (page 149).

Figure 80 - Selected Program Units for Second and Third MDA Transformation: Transformation

```
// Collect all models available in DefaultEAP and fill combobox
void SetComboBoxModelSource()

// Collect all source packages available in chosen model and fill combobox
void SetComboBoxPackageSourceLevel1()

// Collect all source packages available in chosen model and fill combobox
void SetComboBoxPackageSourceLevel2()

// Collect all source diagram available in chosen model and fill combobox
void SetComboBoxDiagramSourceLevel2()

// Collect all source diagram available in chosen model and fill combobox
void SetComboBoxDiagramTargetLevel2()

// Collect all target packages available in chosen model and fill combobox
void SetComboBoxPackageTargetLevel1()

// Collect all Target packages available in chosen model and fill combobox
void SetComboBoxPackageTargetLevel2()

// List the elements of source and target package
void ListPackageElements()

// List the SourcePackage elements
void DumpSourcePackage()

// List the TargetPackage elements
void DumpTargetPackage()

// implement super class in sub classes
void implementInSubClasses (EA.Package myTargetPackage)

// copy source attribute properties to target attribute
void copyAttributeProperties (EA.Attribute sourceAttribute, EA.Attribute targetAttribute)

// Due to EA mistake in first Tranformation, make sure that everything is lowercase
void transformConnectorsToLowerCase (EA.Package myTargetPackage)

// Make sure that generated FK columns are in line with FK cardinality
void updateColumnsAccordingToConnectors (EA.Package myTargetPackage)

// check if there are specific sequence details for tables
void updateSequenceDetails(EA.Package myTargetPackage, string xmlFileName)

// check if provided attribute is in a constraint/operation on the same class
private bool attributeParameterInConstraint (EA.Attribute myAttribute, string
constraintStereotype)

// reorganise attributes: PK, mandatory FK, mandatory, optional FK, and optional columns
void reorganisePositionAttributes ()

// delete classes, attributes, methods that are 'tagged' for deletion
void deleteClassesAttributesOperationsMarkedForDeletion ()
```

```
// getTargetTable based on GUID tag
EA.Element getClassByGUIDTag(string MyClassGUID, string MyTag, EA.Package MyPackage)

// getTargetColumn based on GUID tag
EA.Attribute getAttributeByGUIDTag(string MyAttributeGUID, string MyTag, EA.Package MyPackage)

// replace AttributeNames in Constraints based on input Class and Table
string replaceAttributeNames(string oclConstraint, EA.Element MyClass, EA.Element MyTable,
EA.Package MyPackage )

// replace OCLOperation in Constraints with PostgreSQL functions based on input Table
string replaceOclAttributeOperation(string oclConstraint, EA.Element MyTable )

// change classes and attributes used in OCL to tables and columns
string ChangeClassToTableName(EA.Package MySourcePackageLevel2, EA.Package MyTargetPackageLevel2,
EA.Constraint MyConstraint)

// transform string to MyTransformMethod, e.g. 'Underscored'
string TransformStringTo(string MyTransformMethod, string MyString)

//Loop through all the classes of source Package "+MySourcePackageLevel2.Name + ", and convert OCL
(to check constraint, view, PSM OCL)
void TransformOCLConstraintsClass()

// Loop through all the attributes and perform the required transformations
void PerformTransformation()
```

```
// Get tag value for class (Figure 81)
public string GetClassTagValue(EA.Element myClass, string myTag)
```

```
// set or overwrite class tag value
public void SetClassTagValue(EA.Element myClass, string myTagName, string myTagValue, bool
overwriteTags)

// Get tag value for attribute
public string GetAttributeTagValue(EA.Attribute myAttribute, string myTag)

// set or overwrite attribute tag value
public void SetAttributeTagValue(EA.Attribute myAttribute, string myTagName, string myTagValue,
bool overwriteTags)

// Get tag value for method
public string GetMethodTagValue(EA.Method myMethod, string myTag)

// set or overwrite method tag value
public void SetMethodTagValue(EA.Method myMethod, string myTagName, string myTagValue, bool
overwriteTags)

// Replace PIM data type with PSM datatype
EA.Attribute ProcessAttributeDatatype(EA.Attribute myAttribute)

// Based on the position of classes, the tables will be places similar to classes
void PositionTableAsClass()
```

```
// Create check constraint / lookup table based on enumeration / CodeList (Figure 82)
void ProcessEnumerationClass(EA.Attribute myAttribute)
```

```
// Delete class in package based on class name
private int CleanUpClassByName(EA.Package myPackage, string myClassName)

// Create a "TYPE" table to deal with complex columns (upperbound more than 1)
private void DefineTypeTable(EA.Attribute myAttribute, EA.Element myTypeClass)

// Use lower and upperbound to update (Not) Null checkbox
EA.Attribute ProcessAttributeCardinality(EA.Attribute myAttribute)

// Get type class for current attribute
EA.Element GetTypeClass(EA.Attribute myAttribute)

// Diagram by name within package
EA.Diagram GetDiagram(EA.Package myPackage, string myDiagramName)
```

```
// create a script for generation of all TABLES in target package
private void CreateDDLScript()

// read the individual value of the constant (used in 'readTransformationConstants')
private string readConstantValues ( XmlTextReader myTextReader)

// read the Prototype Constants used during transformation
private void readTransformationConstants(string xmlFileName, bool displayConstants)

// read the DatatypeMapping between PIM and PSM
private EA.Attribute readDatatypeMapping(string xmlFileName, EA.Attribute myAttribute)

// read the OclOperationMapping between PIM and PSM
private string readOclOperationMapping(string xmlFileName, string myAttributeOperation, string
myAttribute)

// Create Unique Keys based on columns that have IsStatic on
private void createUniqueConstraint (EA.Element myClass)
```

Figure 80 - Selected Program Units for Second and Third MDA Transformation: Transformation

Program Unit: GetClassTagValue

An example of a program unit used in the 2nd and 3rd transformation is "GetClassTagValue". The tagged values (section 6.5.1) that are stored for a class, for example *TransformToPSM*, *ImplementedInSubClass*, *MarkForDeletePSM* are retrieved with the program unit GetClassTagValue. This program unit performs the following activities:

- For a given *myClass*, loop through all tagged values (*TaggedValue*).
  - If a tagged value is found for a given tagged value name (*myTag)*, then return the tagged value (*tagFound*).
    - If multiple instances/values of *myTag* are found, then the tagged values will be concatenated, separated by semi colons ";".

```
// Get tag value for class
public string GetClassTagValue(EA.Element myClass, string myTag)
{
    string tagFound = "";
    int countTagFound =0;

    foreach(EA.TaggedValue myClassTag in myClass.TaggedValues)
    {
        if (myClassTag.Name == myTag)
        {
            countTagFound =countTagFound+1;
            if (countTagFound==1)
            {
                tagFound = myClassTag.Value;
                if (myClassTag.Value == "<memo>")
                /* deal with EA way of
                   storing View/Procedure definitions */
                {
                    tagFound = myClassTag.Notes;
                }
            }
            else
            {
               tagFound = tagFound + ";" +myClassTag.Value;
            }
        }
    }

    return tagFound;
}
```

Figure 81 - Example EA Prototype: GetClassTagValue

Program Unit: ProcessEnumerationClass
Another example of a program unit used in the 2nd transformation is "ProcessEnumerationClass", which handles both classes of stereotype <<enumeration>> and <<CodeList>>.

This program unit performs the following activities:

- Loop through all classes within the target package to search for classes that are used to specify the type of a given *myAttribute*.
  - If the found *myClass* is stereotyped as <<enumeration>>
    - then a base table check constraint (*constraintName*) is created based on the attributes/values of the enumeration class.
  - If the found *myClass* is stereotyped as <<CodeList>>
    - then a look-up table (*codeListTableName*) is created
    - a foreign key and a foreign key column are created (*foreignKeyName*)
    - a DML script is created (*fileName_codelist*), which will be populated with attributes/values of the CodeList class. The name of the "CodeListInsertScript" is stored in a tagged value, to be used when the DLL and DML is generated, to create the relational database.

Figure 82 - Example EA Prototype: ProcessEnumerationClass

```
         // Create check constraint based on enumeration / CodeList
         void ProcessEnumerationClass(EA.Attribute myAttribute)
         {
              string checkConstraintEnumerationClass = "";
              string scriptCodeList = "";
              string constraintName = "";
              string codeListTableName = "";

              EA.Package MyTargetPackageLevel2 = (EA.Package)
myRepository.GetPackageByGuid(TargetPackagesLevel2[comboBoxPackageTargetLevel2.SelectedIndex].Pack
ageGUID);

              for( short iClass = 0; iClass < MyTargetPackageLevel2.Elements.Count; iClass++ )
              {
                   EA.Element MyClass  = (EA.Element) MyTargetPackageLevel2.Elements.GetAt(iClass);

                   // if the attribute data type is equal to the ENUMERATION class name.
                   if (MyClass.Name == myAttribute.Type && (MyClass.Stereotype == "enumeration" ||
MyClass.Stereotype == "Enumeration"))
                   {
                        ListAdd(" . " + "Enumeration" + " - " + MyClass.Name + " ("+
MyClass.Stereotype+")");
                        constraintName = "check_"+TransformStringTo("Underscored", MyClass.Name);

                        // Construct the checkConstraint based on enumeration
                        for( short iLiteral = 0; iLiteral < MyClass.Attributes.Count; iLiteral++ )
                        {
                             EA.Attribute MyLiteral  = (EA.Attribute)
MyClass.Attributes.GetAt(iLiteral);
                             if (checkConstraintEnumerationClass == "")
                             {
                                  checkConstraintEnumerationClass = myAttribute.Name + " IN ( '" +
MyLiteral.Name + "'";
                             }
                             else
                             {
                                  checkConstraintEnumerationClass = checkConstraintEnumerationClass +
```

```
", '" + MyLiteral.Name + "'";
                        }
                    }
                    checkConstraintEnumerationClass = checkConstraintEnumerationClass + ")";

                    EA.Element myConstraintClass = (EA.Element)
myRepository.GetElementByID(myAttribute.ParentID);

                    // check if this constraint already exists
                    bool methodFound = false;
                    EA.Method existingMethod = null;
                    for( short iMethod = 0; iMethod < myConstraintClass.Methods.Count; iMethod++
)
                    {
                        EA.Method MyMethod  = (EA.Method)
myConstraintClass.Methods.GetAt(iMethod);
                        if (MyMethod.Name == constraintName)
                        {
                            methodFound = true;
                            existingMethod = MyMethod;
                        }
                    }
                    if (methodFound)
                    {
                        existingMethod.Code = checkConstraintEnumerationClass;
                        existingMethod.Notes = "Enumeration " + MyClass.Name;
                    }
                    else
                    {
                        EA.Method NewMethod = (EA.Method)
myConstraintClass.Methods.AddNew(constraintName,"check");
                        NewMethod.Stereotype = "check";
                        NewMethod.Code = checkConstraintEnumerationClass;
                        NewMethod.Notes = "enumeration " + MyClass.Name;
                        NewMethod.Update();
                        myConstraintClass.Methods.Refresh();

                        // Create column for the foreign key
                        EA.Parameter NewParameter = (EA.Parameter)
NewMethod.Parameters.AddNew(myAttribute.Name,"");
                        NewParameter.Update();
                        NewMethod.Parameters.Refresh();
                    }
                    myAttribute.Type = enumerationDataType;
                    myAttribute.Length = enumerationLength.ToString();
                }
                else
                {
                    // if the attribute data type is equal to the enumeration class name
(CodeList).
                    if (MyClass.Name == myAttribute.Type && (MyClass.Stereotype == "CodeList"))
                    {
                        ListAdd(" . " + "CodeList" + " - " + MyClass.Name + " ("+
MyClass.Stereotype+")");

                        codeListTableName = "codelist_"+MyClass.Name.ToLower();

                        // find the codeListTableName, loop through package
                        for( short iCodeListClass = 0; iCodeListClass <
MyTargetPackageLevel2.Elements.Count; iCodeListClass++ )
                        {
                            EA.Element MyCodeListClass  = (EA.Element)
MyTargetPackageLevel2.Elements.GetAt(iCodeListClass);

                            if (MyCodeListClass.Name == codeListTableName)
                            {
                                string fileName_codelist =
"C:\\GIMAPrototype\\Create"+codeListTableName+".sql";
                                TextWriter tw_codelist = openTxtFile(fileName_codelist);

                                // Construct the INSERT script based on CodeListClass
attributes
                                for( short iLiteral = 0; iLiteral < MyClass.Attributes.Count;
iLiteral++ )
```

```
                                    {
                                        EA.Attribute MyLiteral  = (EA.Attribute)
MyClass.Attributes.GetAt(iLiteral);
                                        //  string test =  MyLiteral.Pos.ToString();
                                        scriptCodeList = " Insert into "+codeListTableName + " ("+
PK_Method+ ", value) VALUES ( " + (MyLiteral.Pos+1) +", '" + MyLiteral.Name + "');";
                                        writeLineToFile(tw_codelist,scriptCodeList);
                                    }
                                    // store script in the notes
                                    //  scriptCodeList = scriptCodeList + ");";
                                    MyCodeListClass.Notes = "Values are in stored script
'Create"+codeListTableName+".sql'";
                                    MyCodeListClass.Stereotype = "table";
                                    MyCodeListClass.Update();

                                    this.SetClassTagValue(MyCodeListClass, "CodeListInsertScript",
"Create"+codeListTableName+".sql", true);

                                    // Determine class for current attribute
                                    EA.Element MyAttributeClass  = (EA.Element)
this.myRepository.GetElementByID(myAttribute.ParentID);
                                    // Create foreign key for current class

                                    string foreignKeyName = "fk_"+codeListTableName;

                                    // Mark the ForeignKey for delete if it already exists
                                    for( short iMethod = 0; iMethod <
MyAttributeClass.Methods.Count; iMethod++ )
                                    {
                                        EA.Method myMethod  = (EA.Method)
MyAttributeClass.Methods.GetAt(iMethod);
                                        if (myMethod.Name == foreignKeyName)
                                        {
                                            this.SetMethodTagValue(myMethod, "MarkForDeletePSM",
"T", true);
                                        }
                                    }

                                    EA.Method NewMethod = (EA.Method)
MyAttributeClass.Methods.AddNew(foreignKeyName,"FK");
                                    NewMethod.Stereotype = "FK";
                                    NewMethod.Update();
                                    MyAttributeClass.Methods.Refresh();

                                    // Create column for the foreign key
                                    EA.Parameter NewParameter = (EA.Parameter)
NewMethod.Parameters.AddNew(myAttribute.Name,"");
                                    NewParameter.Update();
                                    NewMethod.Parameters.Refresh();

                                    // update attribute type to integer
                                    myAttribute.Type = this.PK_Datatype; //enumerationDataType;
                                    myAttribute.Update();

                                    // create connector
                                    EA.Connector NewConnector = (EA.Connector)
MyAttributeClass.Connectors.AddNew("list of values","FK");
                                    NewConnector.Type = "Association";
                                    NewConnector.Stereotype = "FK";
                                    NewConnector.Direction = "Source -> Destination";

                                    // Source MyAttributeClass.ElementID
                                    NewConnector.ClientID = MyAttributeClass.ElementID;
                                    NewConnector.ClientEnd.Role = foreignKeyName;
                                    NewConnector.ClientEnd.Cardinality = "*";

                                    // find the name/role of the PK Method
                                    string supplierEndRole = "";
                                    for( short iPkMethod = 0; iPkMethod <
MyCodeListClass.Methods.Count; iPkMethod++ )
                                    {
                                        EA.Method MyPkMethod  = (EA.Method)
MyCodeListClass.Methods.GetAt(iPkMethod);
                                        if (MyPkMethod.Stereotype == "PK") supplierEndRole =
```

```
MyPkMethod.Name;
                                    }
                                    NewConnector.SupplierEnd.Role = supplierEndRole;
                                    NewConnector.SupplierID = MyCodeListClass.ElementID;

// determine cardinality supplier end based on null/not null setting of attribute
                                    switch( myAttribute.AllowDuplicates )
                                    {
                                        case true: // not null
                                            NewConnector.SupplierEnd.Cardinality = "1";
                                            break;
                                        case false: // null
                                            NewConnector.SupplierEnd.Cardinality = "0..1";
                                            break;
                                    }
                                    NewConnector.Update();
                                    closeFile(tw_codelist);
                                }
                            }
                        }
                    }
                }
            }
```

Figure 82 - Example EA Prototype: ProcessEnumerationClass

Program Unit: transformtoPSM

An example of a program unit used in the 1st transformation, as part of the EA Transformation Template "PostgreSQL" is "transformToPSM". This program unit retrieves the tagged value "TransformToPSM" for a class, indicating whether the class should be transformed from PIM to PSM. In Figure 83, part of EA Transformation Definition for "Class" is provided (see conversion template in Figure 77), which checks for a tagged value "TransformToPSM" by calling program unit "transformToPSM" (preceded by '*$*').

```
%if $transformToPSM =="T"%
Table
{
  %TRANSFORM_REFERENCE("Table")%
```

Above: fragment in EA Transformation Definition for "Class"

Below: the C# program unit "transformtoPSM" returning the value of tag "TransformToPSM"

```
// Check if class is marked for transformation ('TransformToPSM')
public object transformtoPSM(EA.Repository repository, object argsObject)
{
    string[] classObject = (string[])argsObject;
    EA.Element myClass = repository.GetElementByGuid(classObject[0]);

    string tagValue = "";

    for( short iTaggedValue = 0; iTaggedValue < myClass.TaggedValues.Count; iTaggedValue++ )
    {
        EA.TaggedValue myTaggedValue = (EA.TaggedValue) myClass.TaggedValues.GetAt(iTaggedValue);

        if (myTaggedValue.Name == "TransformToPSM")
        {
            tagValue = myTaggedValue.Value;
        }
    }
    return tagValue; // True
}
```

Figure 83 - Example EA Prototype: transformToPSM

Appendix F: Details on First Transformation in MDA Prototype (PIM to PSM-1)

The first transformation (section 6.5) is started with the Enterprise Architect application as presented in Figure 84. On the left side, the classes to be transformed are presented. On the right side, the choice for transformation template (i.e. PostgreSQL) is made.

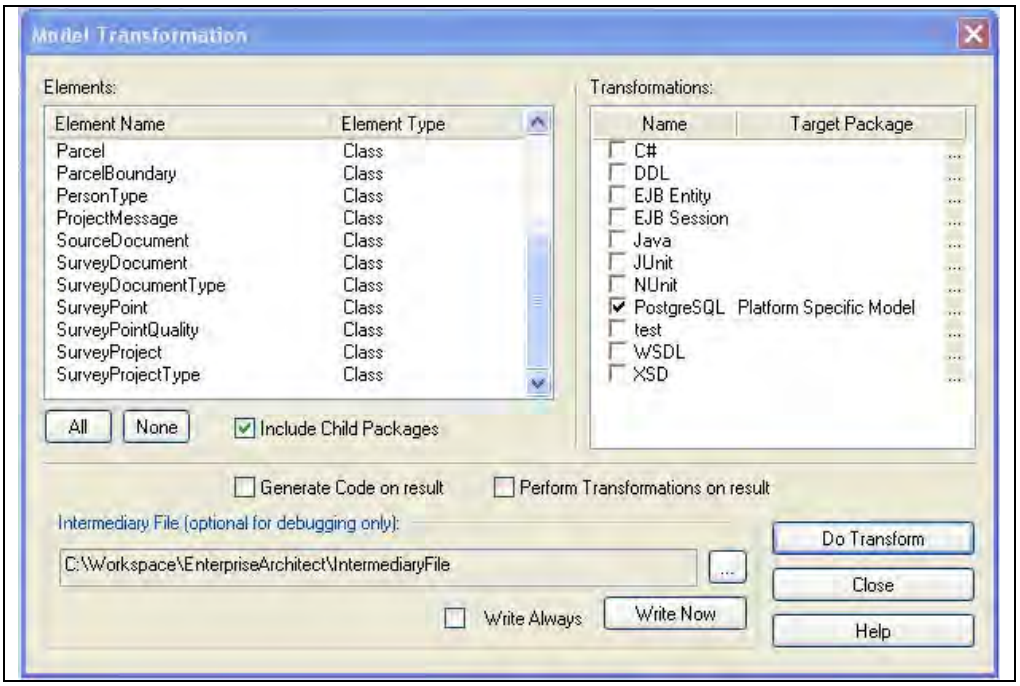


Figure 84 - First Transformation with EA Transformation Definition (EA user interface)

The first transformation offers the following MDA Transformation Rules:

- Create Target Package and Target Platform
- Copy Source Structure to Target Package Structure
- Transform Classes (Stereotyped <<enumeration>> or <<CodeList>>)
- Transform Class to Table
- Transform Attribute to Column
- Generate Primary Key
- Transform Associations to Relationships or Tables

Create Target Package and Target Platform

The platform independent model of the LADM SP can be implemented in several specific platforms, e.g. Oracle, PostgreSQL. The first level in the platform specific models will indicate the target platform, for example PostgreSQL, see Figure 34, showing the structure for both the PIM and PSM.

Input Element	-
MDA Transformation Rule	The main target package carries the name of the target DDL environment, e.g. "PostgreSQL".
Output Element	Main Package
Tool	Conversion template File

Copy Source Structure to Target Package Structure
The structure of packages and namespaces in the PIM will be used in the PSM, see section Figure 34.

Input Element	Package
MDA Transformation Rule	The structure of the source package in the PIM must be maintained and transformed to the PSM package, as an element of the Main Package. N.B. package properties "scope", "abstract", "name", "notes", are not transferred
Output Element	Package
Tool	Conversion template Namespace

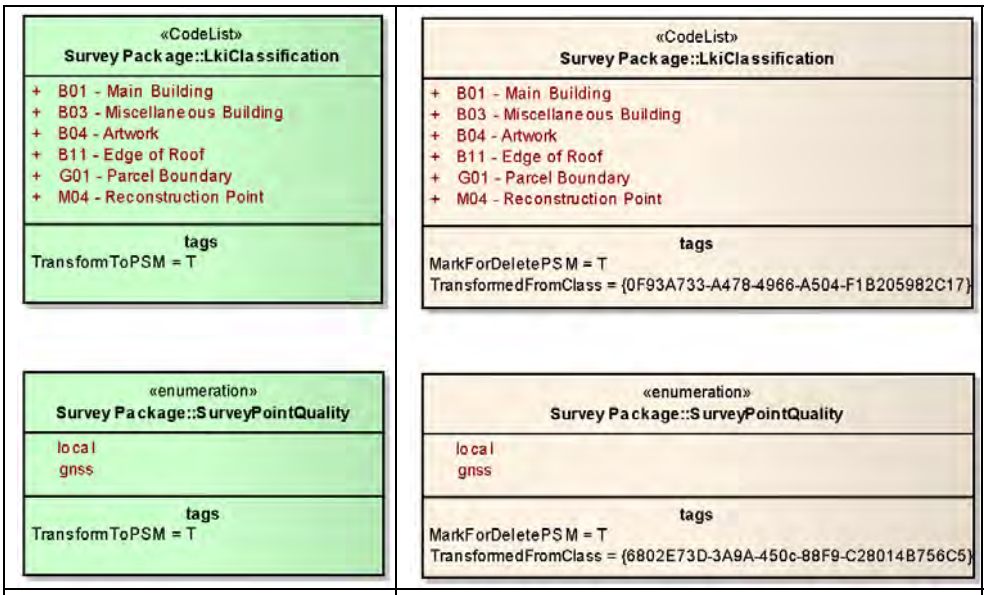


Figure 85 - 1st Transformation (PIM to PSM-1): CodeList & Enumeration Class

Transform Classes (Stereotyped <<enumeration>> or <<CodeList>>)
Classes which are stereotype as <<enumeration>> or <<CodeList>> will be transformed to classes (instead of tables) in the PSM (Figure 85). They will have a short lifecycle and will be used in the second transformation for check constraints and

look-up tables (see section 6.6, "Second Transformation from PSM-1 to PSM-2", for additional explanation).

"MarkForDeletePSM"
This short life cycle is implied by the tagged value "MarkForDeletePSM" with value T (True), and at the end of the second transformation, these classes will be deleted. Below the CodeList class LkiClassification and the enumeration class SurveyPointQuality are shown in the PIM (left side) and the PSM (right side).

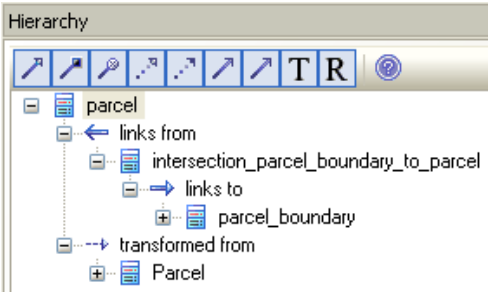
Input Element	enumeration class
MDA Transformation Rule	A class with stereotype "enumeration" or "CodeList", will be transformed as Class. N.B. the subsequent transformation will use the transformed classes as input.
Output Element	Class (stereotype enumeration or CodeList)
Tool	Conversion template Class

Also the enumeration and CodeList values (attributes) will be transformed to attributes in the PSM.

Input Element	enumeration values
MDA Transformation Rule	The attributes of a class with stereotype "enumeration" or "CodeList", will be transformed as attributes. N.B. attribute properties "collection", "constant", "containment", "ordered", "static", "volatile", "initial" ,"precision", "scale", are not transformed, because are not applicable in the context of the prototype.
Output Element	Attribute
Tool	Conversion template Attribute__Enumeration

Transform Class to Table
The core classes of the PIM, which are marked with tagged value "*TransformToPSM*" with value "T" (True) will be transformed to tables. In Figure 86, it is shown that the names of classes are changed to underscored, lowercase table names, and OCL constraints are not transformed to the PSM. An application prefix as the first part of the table is not supported by the MDA prototype.

"TransformedFromClass"
The tagged value "TransformedFromClass" will show the GUID (globally unique identifier) of the originating class. The EA user interface does show the relation between PIM and PSM elements in the so-called Hierarchy, but this hierarchy and its XREF codes, cannot be accessed in the EA SDK, so a solution with



tagged values was implemented to be able to address the relation between PIM and PSM elements in the prototype (see Figure 85, after 1st transformation).

Input Element	Class
MDA Transformation Rule	A class (not stereotype in "enumeration", "CodeList") will be transformed to one class with stereotype table. N.B. A choice has been made for a 1:1 Class to Table mapping. N.B. table properties "language", "stereotype", "abstract", are not transformed. N.B. Class names are assumed to be in "Pascal Case", i.e. words in the name start with capital letter, transformed in a table name, in which the words are separated by underscores.
Output Element	Table
Tool	Conversion template Class

Transform Attribute to Column

The attribute names are changed to underscored, lowercase column names. Note: the attribute data types remain the same in this transformation, see Figure 86. The initial (default) values of the attributes in the PIM are also transformed to default values of columns in the PSM.

Input Element	Attribute
MDA Transformation Rule	The attributes of a class (not stereotype in "enumeration", "CodeList", "Union"), will be transformed to columns. N.B. attribute properties "stereotype", "collection", "constant", "containment", "ordered", "static", "volatile", "precision", "scale", are not transformed. N.B. Attribute names are assumed to be in "Pascal Case", i.e. words in the name start with capital letter, transformed in a column name, in which the words are separated by underscores. N.B. if the "IsStatic" property of the attribute is "T" (True), that a tagged value "PartofUniqueKey" = "T" (True) will be added, to be used in subsequent transformations, see section 6.6.
Output Element	Column
Tool	Conversion template Attribute

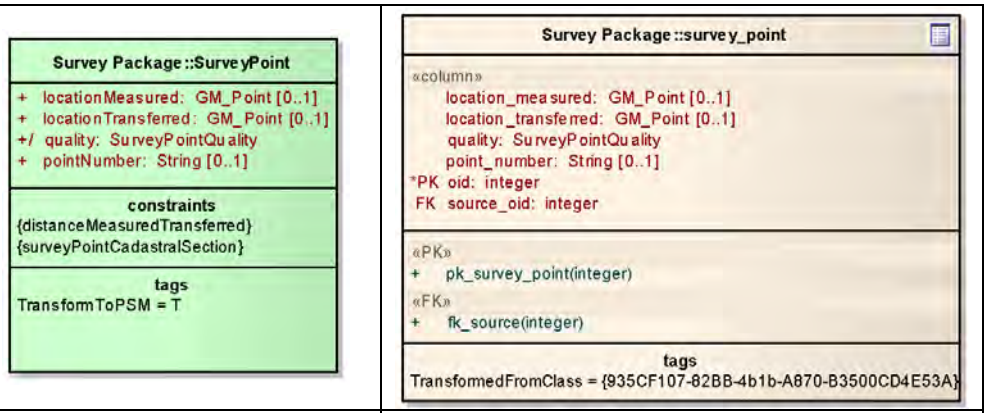


Figure 86 - 1st Transformation (PIM to PSM-1): Class to Table

Generate Primary Key

For each table a primary key and column are being created, based on the prototype constants PK_Method, and PK_Datatype (Figure 87), respectively with value "oid", and "integer", see pk_survey_point in Figure 86.

```
PrimaryKey
{
  name = %qt%pk_%CONVERT_NAME(className, "Pascal Case","Underscored")%%qt%
  Column
  {
    name=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnName", classGUID)%%qt%
    type=%qt%%EXEC_ADD_IN ("PrototypeAddin", "GetPrimaryKeyColumnDataType", classGUID)%%qt%

    Tag
    {
      name ="property"
      value = %qt%%EXEC_ADD_IN ("PrototypeAddin", "GetSequenceStartIncrement", classGUID)%%qt%
    }
  }
}
```

Figure 87 - Prototype Constants Primary Key Name and Data Type, and Tagged Value for Sequence

Input Element	Class
MDA Transformation Rule	For each class, stereotyped "table", an operation, stereotyped "PK" is created, based on a newly created column named "oid", with PostgreSQL data type = "integer". N.B. The PrototypeAddin.GetPrimaryKeyColumnName determines the name of the PK column (default "oid").
Output Element	Primary Key Column Operation (stereotype "PK")
Tool	Conversion template Class

Sequences

For a sequence on the primary key column, EA makes an alternative use of a *tagged value* "Property" is being created with a *value* like "*AutoNum=1; StartNum=1; Increment=1; NotForRep=0;*", which will eventually result in for example a sequence "survey_point_oid_seq" starting at 1, and with an increment of 1 at each requested next value, see the transformation template for primary key in Figure 87. During the loading of Kadaster test data into the LADM SP prototype, the need to keep the original unique oid values in the Kadaster data was identified. The communication on specific data cases would be facilitated by using the oid's as delivered by Kadaster. This would require the sequences for those tables to start at an integer value, higher than the highest oid value. For example, over 7700 cadastral sections were delivered, which resulted in the need for the sequence "cadastral_section_oid_seq" to start at start 7800. This has been realised with a setting in xml file *PrototypeConstants.xml*, resulting in a tagged value "*AutoNum=1; StartNum=7800; Increment=1; NotForRep=0;*", see also Figure 100 for an example of the EA user interface for maintaining the tagged value for sequences.

```
<Sequence_cadastral_section Class="CadastralSection" Column="oid"
Start="7800" Increment="1" />
```

Alternative primary key column data type and name

In the course of the developing and testing the prototype, other information system development has been done partly based on the MDA prototype (Land Information System in Ghana). In this land information system, the primary key columns generated were of data type "GUID" (globally unique identifier) and name "gid". GUID's are used when primary keys are required for records which have to be unique globally, which enables easy consolidation of information from different databases. For example the collection of data from distributed regional databases with similar data model, into a national central database. An example of a GUID value is "*05550709-f059-4643-8b02-734c32e6b9d8*". PostgreSQL offers the data type UUID (Universally Unique IDentifiers) which is a data type similar to GUID. The "GUID" primary key columns with name *gid*, were generated by a different setting in prototype constants file PrototypeConstants.xml.

Transform Associations to Relationships or Tables

For associations, different situations may occur. For example with regard to Generalisations, an MDA transformation rule exists:

Input Element	Generalization
MDA Transformation Rule	For each relationship, stereotyped "Generalization" an association, stereotyped "FK" is created, based a newly created foreign key column named according to the referring class "OID", with PostGIS data type = "integer".
Output Element	Foreign Key Column Operation (stereotype "FK")
Tool	Conversion template Connector

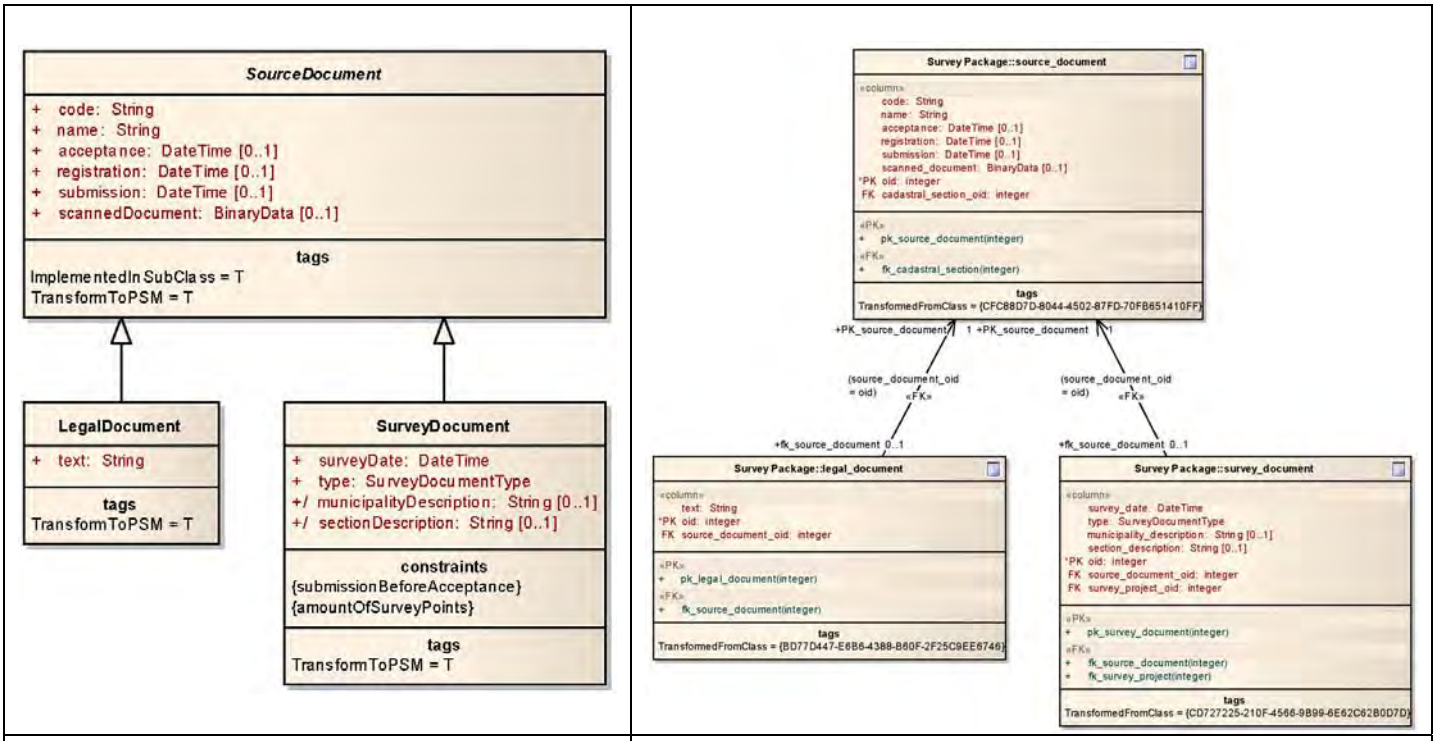


Figure 88 - 1st Transformation (PIM to PSM-1): Generalisation

Note that the *direction of the association* is transformed as well (Figure 88), although it has no function in the PSM. The direction has been added for model readability purposes, pointing in the direction of the "master" table, providing its primary key column(s) value for the dependent foreign key column(s).

Based on tagged value "ImplementedInSubClass", columns and relationships will be inherited by sub classes (tables), see section 6.6.1

Input Element	Association
MDA Transformation Rule	Based on the multiplicity of the source and target of the connector / relationship, one of the following scenarios can be executed: • If the multiplicity of both source and destination of relationship > 1; create intersection table to hold the many-to-many relationship • If the multiplicity of the source OR the destination of relationship > 1; create a foreign key.
Output Element	Intersection table Foreign Key Column Operation (stereotype "FK")
Tool	Conversion template Connector

For associations two situations are being handled: many-to-many relationship and 0 or 1 to-many relationship.

many-to-many relationship
Associations with a multiplicity larger than 1 on both sides will be implemented by "intersection" tables, see table "intersection_parcel_boundary_to_parcel" in Figure 89. If the associations would require attributes to describe them, they should be replaced by association classes, which are not handled in the prototype.

The term "intersection" maybe confusing in a spatial environment, where it is used to indicate the spatial operator to identify geometric objects that intersect, however, the term intersection tables is used to indicate tables that implement a many-to-many relationship between two tables.

0 or 1 to-many relationship
Associations with a multiplicity larger than 1 on only one side, will be implemented as foreign key and column, for example "fk_cadastral_municipality" and column "cadastral_municipality_oid" in Figure 90. Note that 1-to-1 relationships (Associations with no specified multiplicity, or multiplicity 1 on both sides, will be implemented as foreign keys with the foreign key column on the "target" side of the association. Aggregations and compositions have not been handled in the prototype, e.g. for example the association between SurveyProject and SurveyDocument. This association could be depicted as a composition, which would be implemented as a foreign key, with database functionality (i.e. row level table triggers) to prevent the change of the value of the foreign key column (to point at another SurveyProject).

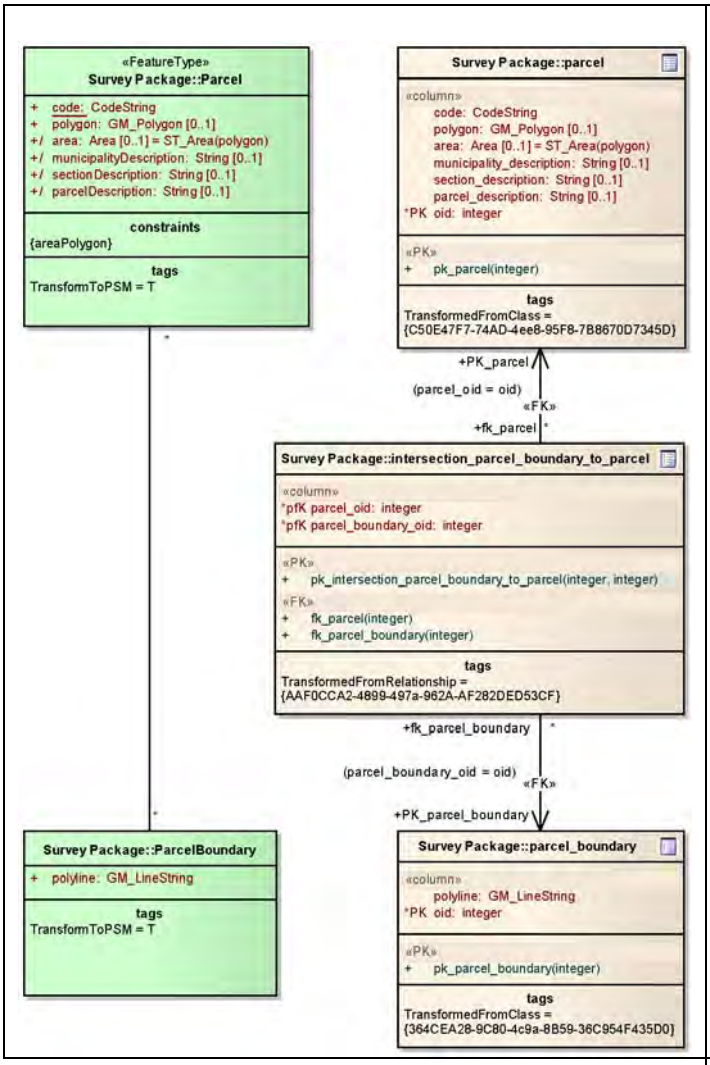


Figure 89 - 1st Transformation (PIM to PSM-1): Many-to-Many Associations

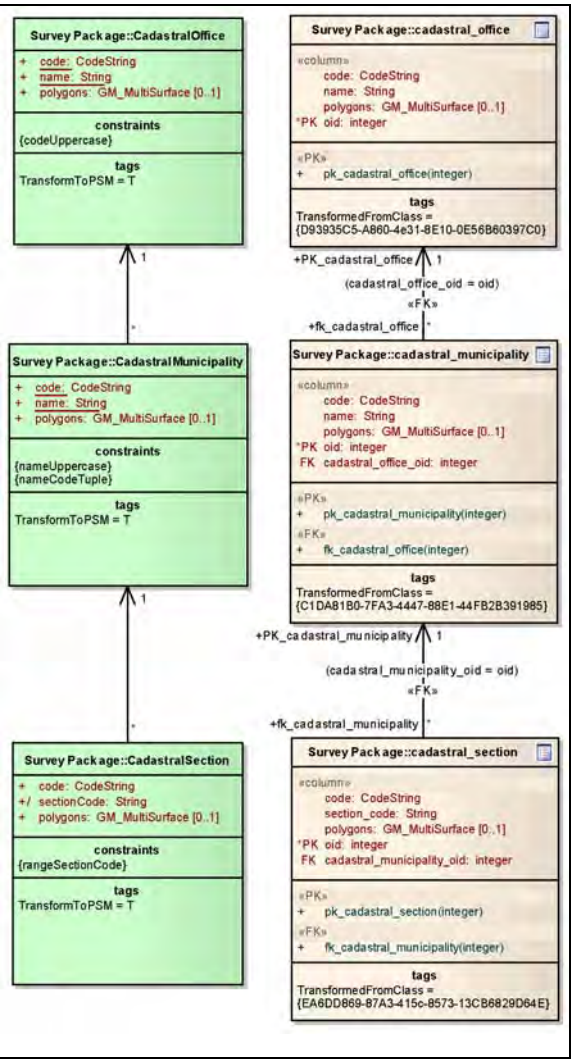


Figure 90 - 1st Transformation (PIM to PSM-1): One-to-Many Associations

Appendix G: Details on Second Transformation in MDA Prototype (PSM-1 to PSM-2)

The MDA Prototype user interface is presented in Figure 91. In the top section, the EA project file, as well as source and target packages, are presented. Below on the left side, the transformations tasks, build for the prototype are presented, for example the selected "3 Transform PSM -> PSM (2nd transformation)", see section 6.6. In the window "Report on Transformation Task" the feedback to the MDA user is presented.

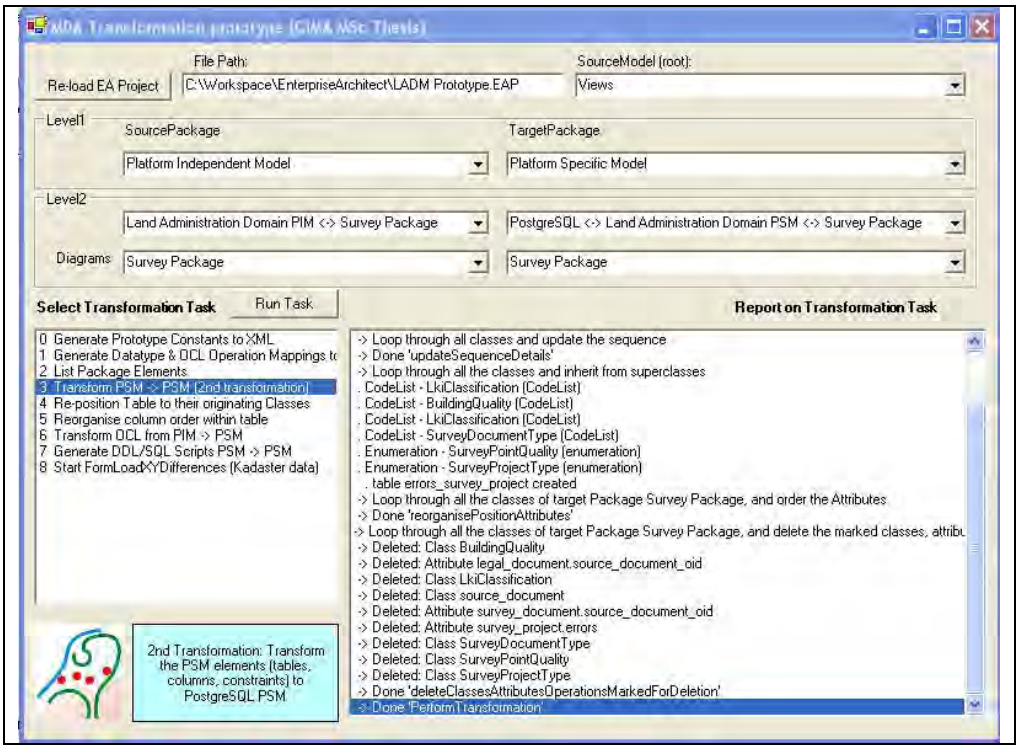


Figure 91 - Prototype Report after 2nd transformation from PSM-1 to PSM-2

The second transformation offers the following MDA Transformation Rules:

- Set Column "Not Null" property
- Process columns defined by <<Type>> classes
- Transform Attribute
- Transform Classes, stereotyped <<enumeration>> and <<CodeList>>
- Create Uniqueness Constraint
- Re-organise Order of Columns within Table
- Implement Super Class in Sub Class (Table)

Set Column "Not Null" property

In the prototype, the Lower and Upper Bound properties of attributes are used to set the "Not Null" column property. Note that the default in EA for lowerbound..upperbound is [1..1] In Figure 92, the changes for survey_project are shown, for example start_date with [lower bound and upper bound 1] is set as "Not

Null" = True (indicated in Figure 92 with an asterix [*]) and end_date [0..1] is set as "Not Null" = False.

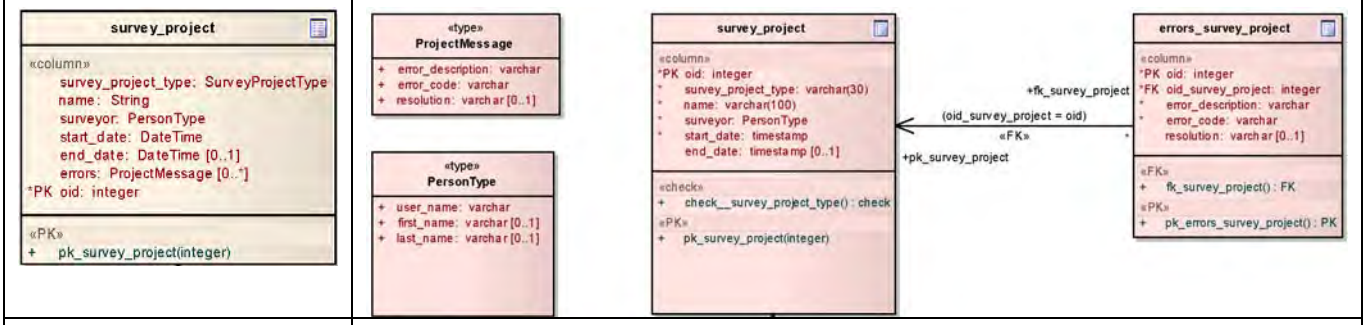


Figure 92 - 2nd Transformation (PSM-1 to PSM-2): Column Cardinality

The "Not Null" property can also be set, based on the cardinality of the foreign key (association). If the Foreign Key is optional [0..1], the foreign key column is set to "Not Null" = False.

Input Element	Column & Foreign Key Cardinality
MDA Transformation Rule	The Cardinality (through parameters Lower and Upper bound of a Collection) of an attribute in PIM is used to set NULL or NOT NULL characteristics of a Column, already in the PIM. NB setting "0..1" leads to "NULL", setting "1..1" leads to NOT NULL, other values lead to the message "*Warning: COLUMN has Upper bound *, must be modelled as Table with constraint.*" in the "Report on Transformation Task" message window.
Output Element	Column "Not Null" setting
Tool	PrototypeAddin.updateColumnsAccordingToConnectors & PrototypeAddin.ProcessAttributeCardinality

Process columns defined by <<Type>> classes

Columns that have a data type, corresponding to a class, stereotyped <<type>> can be handled in two ways, based on the attribute cardinality (indicated with Lower bound and Icebound). For example, the column "survey_project.surveyor" has a data type "PersonType" (referring to class PersonType) and a *Lowerbound..Upperbound [1..1]* (Figure 92), which will result in the generation of a type object in the PostgreSQL database. Another example is the column "survey_project.errors" (Figure 92) with data type "ProjectMessage" and a *Lowerbound..Upperbound [0..*]*. This will result in the generation of a table "errors_survey_project", based on class PersonType, capable of holding multiple instances of project messages, with a foreign key to survey_project in which the column "errors" now is deleted.

The drawback of this solution is that it creates an extra table which has to be joined with implication for query performance. Another possible implementation would have been to create an array of data type "ProjectMessage", PostgreSQL allows

columns of a table to be defined as variable-length multidimensional arrays, although the cardinality of columns can not be checked (e.g. error_description and error_code are mandatory, while resolution is not), and would require additional constraints to enforce this.

Note that, for experimental purposes, the prototype functionality with regard to attribute cardinality (e.g. [0..*]) has been created in relation to and in combination with complex types, such as type *ProjectMessage* and attribute **SurveyProject.errors**. For attributes with a fixed cardinality (e.g. [3]), the implementation could either be the creation of 3 columns to hold these values, or the creation of a child table, similar to **errors_survey_project**, with a constraint that enforces the fixed amount of records in this table. This has not been implemented in the prototype, but would not have large programming consequences.

Input Element	Attribute Data type "type"
MDA Transformation Rule	If an attribute's data type is not present in the file DatatypeMapping.xml, and that name is equal to a class name, stereotyped as "type", two situations may occur: 1) If the attribute cardinality is equal to "0..1" or "1..1", the Data type will remain the same, a type will be generated in the target database, see section 7.3. 2) If the attribute cardinality is "0..*" or "1..*", a Type Table will be generated (Name: AttributeName_Datatype), with attributes equal to the type. A foreign key from this table to the tables/class for the original attribute will be created The relevant column, replaced by the new table , will be deleted. N.B. this functionality is realised with EA Tagged Values
Output Element	1) Column data type or 2) Type Table Operation (stereotype "FK") deleted Column
Tool	PrototypeAddin.UpdateAttributes ProcessAttributeCardinality GetTypeClass DefineTypeTable GetAttributeTagValue

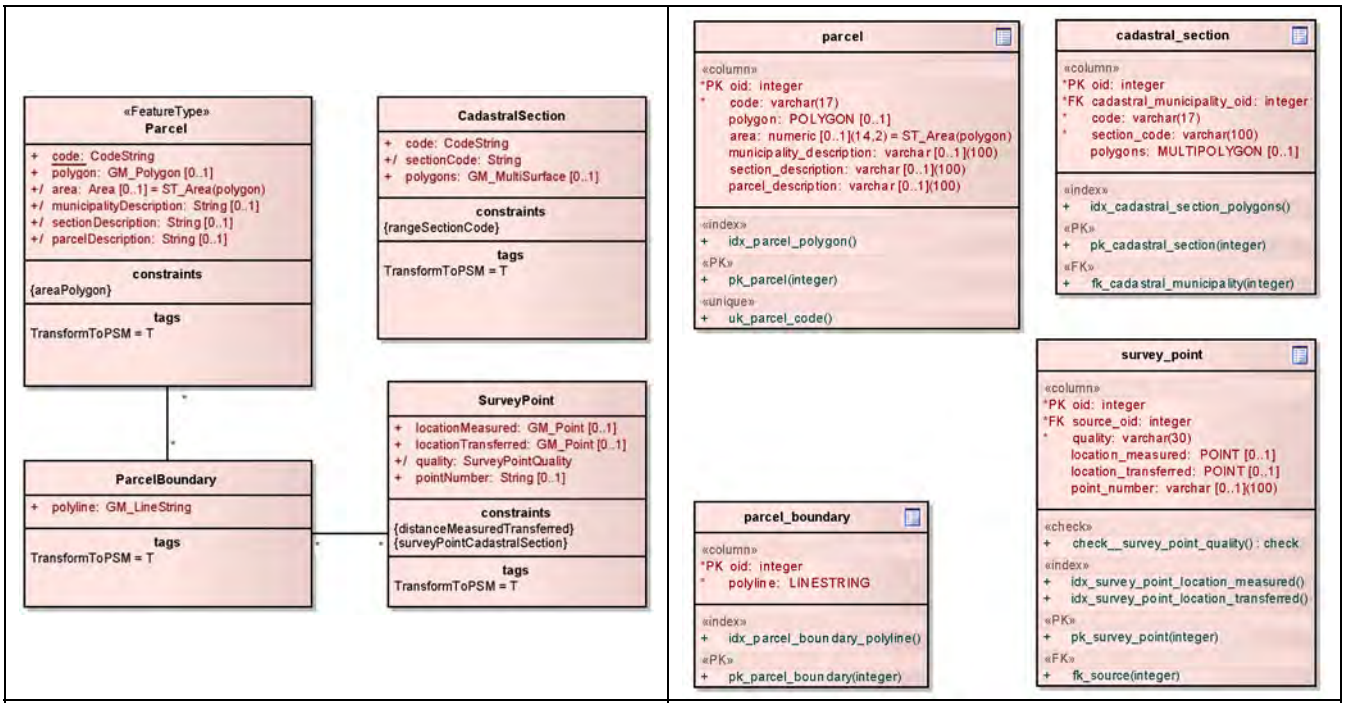
Transform Attribute Data type

In the first transformation based on EA transformation templates a conversion of PIM data types into PSM (PostgreSQL/PostGIS) data types would have been possible, but additional flexibility in this mapping process was acquired through using the xml file DatatypeMapping.xml as shown in Figure 33, to address the Model Type Mapping as described in the MDA Guide [OMG, 2003]. In the MDA prototype an implementation choice has been made to handle the attribute data types in the second transformation deals with attribute data types.

In this XML file a number of mappings of PIM data types to PSM data types are shown, which are used in this "Second Transformation from PSM-1 to PSM-2". In addition to the transformation of data types between the PIM and the PSM, an implementation with regard to other (PSM) properties of the attribute has been defined. For example a PIM data type "CodeString" will be converted to a PSM "varchar" data type of length 17, and a PIM data type "Number" will be converted to a PSM "numeric" data type of precision 15, with a dimension of 3 positions after the decimal sign.

Another approach to influence PSM columns could have been to use PIM data types such as CharacterString, which is data type defined with integer attributes for size and maximum length, and the actual character elements

For example, Figure 93 shows the MDA transformations for data types CodeString to varchar(17); GM_Polygon to POLYGON (PostGIS), and String to varchar(100).



Before 1st Transformation | After 2nd Transformation

Figure 93 - Transformation (PSM-1 to PSM-2): Attribute -> Column Data type

Input Element	Attribute Data type, Precision, Scale, Length
MDA Transformation Rule	Based on definitions of mapping between LADM/PIM data types (e.g. defined in ISO/INSPIRE standards and profiles), and the PostgreSQL/PostGIS data types, precisions, scales, and length (in DatatypeMapping.xml), the attribute data type will be transformed to PostgreSQL data types.
Output Element	Column data type, precision, scale, length
Tool	PrototypeAddin.ProcessAttributeDatatype

Transform Classes, stereotyped <<enumeration>> and <<CodeList>>
The stereotype classes <<enumeration>> and <<CodeList>> are both used in the LADM (PIM) for indicating a list of allowed values. For <<enumeration>> a check constraint is added on the attribute and hard coded in the check are the allowed values.

Input Element	Class stereotyped "enumeration"
MDA Transformation Rule	If an attribute's data type is not present in the file DatatypeMapping.xml, and that name is equal to a class name, stereotyped as "enumeration", a check constraint will be generated for the relevant column. The data type and length of the column will be changed based on constants "enumerationDataType" and "enumerationLength" in the file PrototypeConstants.xml NB The attribute names of the "enumeration" class will be used to generate the check constraint.
Output Element	Column data type, length Operation (stereotype "check")
Tool	PrototypeAddin.UpdateAttributes & PrototypeAddin.ProcessEnumerationClass

For <<CodeList>> a (look-up) table with the allowed values is defined, as well as a foreign key constraint from the original table to the look-up table.

Input Element	Class stereotyped "CodeList"
MDA Transformation Rule	If an attribute's data type is not present in the file DatatypeMapping.xml, and that name is equal to a class name, stereotyped as "CodeList" (defined in the UML Profile for INSPIRE data specifications), a lookup table will be generated. A foreign key for the relevant column to the lookup table will also be generated.
Output Element	Lookup Table Operation (stereotype "FK")
Tool	PrototypeAddin.UpdateAttributes & PrototypeAddin.ProcessEnumerationClass *(also for CodeList)*

Create Uniqueness Constraint
In the first transformation, based on the "IsStatic" property of the attribute, tagged values "PartofUniqueKey" = "T" (True) have been added the transformed column, which will now be used to create a unique key. For example the attribute Parcel.code, has the "IsStatic" property set, visualised by underlining of the attribute name (Figure 94). During the first transformation, this has resulted in the tagged attribute value "PartofUniqueKey" = "T" (True), which in the second, current transformation is used to create a unique key uk_parcel_code.

Note that EA uses the same attribute properties in the PIM for different purposes as in the PSM [SparxSystems, 2007] [Chapter 16], for example PIM IsStatic is used to indicate columns, part of a Unique key. By setting IsStatic for an attribute in the PIM,

and MDA transformation rule will convert it to a Unique Key constraint (operation) in the PSM. Other examples are PIM:IsOrdered (PSM: Primary Key), and PIM:AllowDuplicates (PSM:NotNull).

The prototype will process all attributes with IsStatic = True, and create one unique key based on all columns. Another implementation (for example based on tagged values), would be to create a unique for each attribute with IsStatic = True. This has not been implemented in the prototype, but would be relatively easy to program.

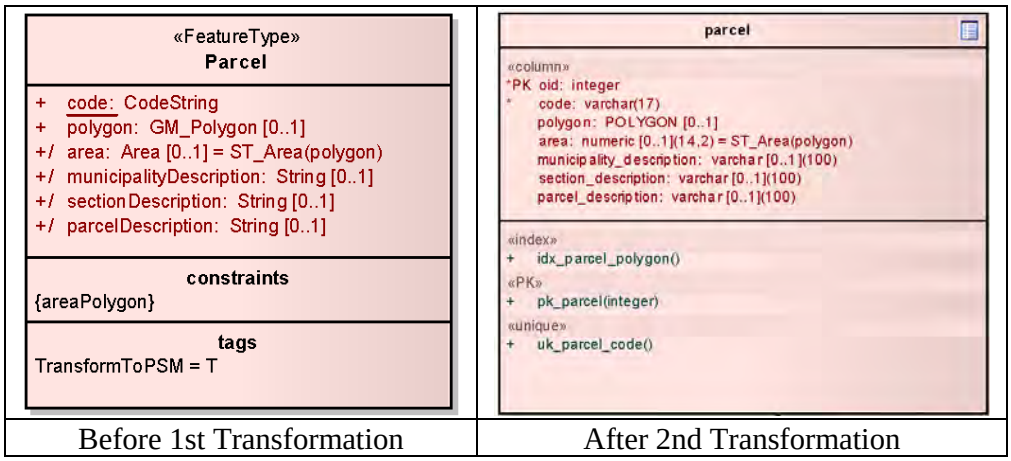


Figure 94 - 2nd Transformation (PSM-1 to PSM-2): uniqueness constraints

Input Element	column with tagged value "PartofUniqueKey" = "T"
MDA Transformation Rule	Create a unique key constraint based on columns with tagged value "PartofUniqueKey" = "T" (attributes with "IsStatic" = "T")
Output Element	unique key constraint
Tool	createUniqueConstraint()

Re-organise Order of Columns within Table

In the first transformation, EA creates primary and foreign key columns as the *last* columns within the table, see Figure 95 (source_document and survey_document). A MDA transformation rule was defined to change the position (order) of columns to a more logical one. The order that has been defined for implementation in the PostgreSQL database is first the Primary Key columns, then mandatory Foreign Key columns, the mandatory columns, the optional Foreign Key columns, and finally the remaining optional columns. Figure 95 shows on the right side the table survey_document, which has inherited attributes of source_document (as part of flattening of class hierarchy), after which the columns have been ordered according to the MDA transformation rule.

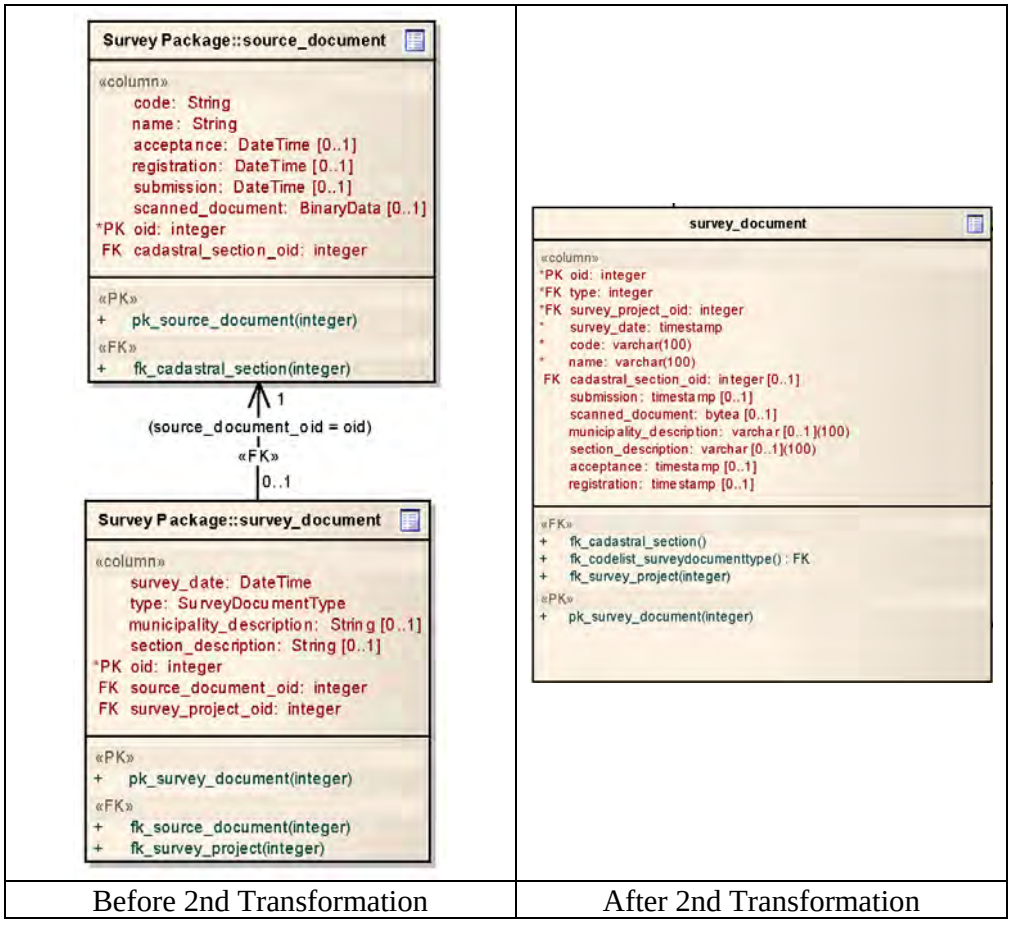


Figure 95 - 2nd Transformation (PSM-1 to PSM-2): order of columns within a class

Input Element	columns, cardinality, primary, unique and foreign keys
MDA Transformation Rule	Change the order of columns: • Primary Key columns • mandatory Foreign Key columns • mandatory columns • optional Foreign Key columns • optional columns
Output Element	columns
Tool	reorganisePositionAttributes()

Implement Super Class in Sub Class (Table)
In the first transformation each class in the PIM is converted to one table in the PSM for each class in the PIM (see Figure 88, Figure 95, left side). Based on tagged value "ImplementedInSubClass", attributes, operations and associations will be inherited by sub classes, the super class (table) will be removed from the PSM.

Input Element	super and sub classes (table)
MDA Transformation Rule	Based on the tagged value "ImplementedInSubClass" with value "T" (True), the attributes, operations and associations of the super class are inherited in the sub classes (tables).
Output Element	sub classes (tables)
Tool	implementInSubClasses()

Appendix H: Details on Third Transformation in MDA Prototype (PIM OCL to PSM-2)

In Figure 96, below on the left side, the transformations tasks, build for the prototype are presented, for example the selected "6 Transform OCL from PIM -> PSM", see section 6.7. In the window "Report on Transformation Task" the feedback to the MDA user is presented.

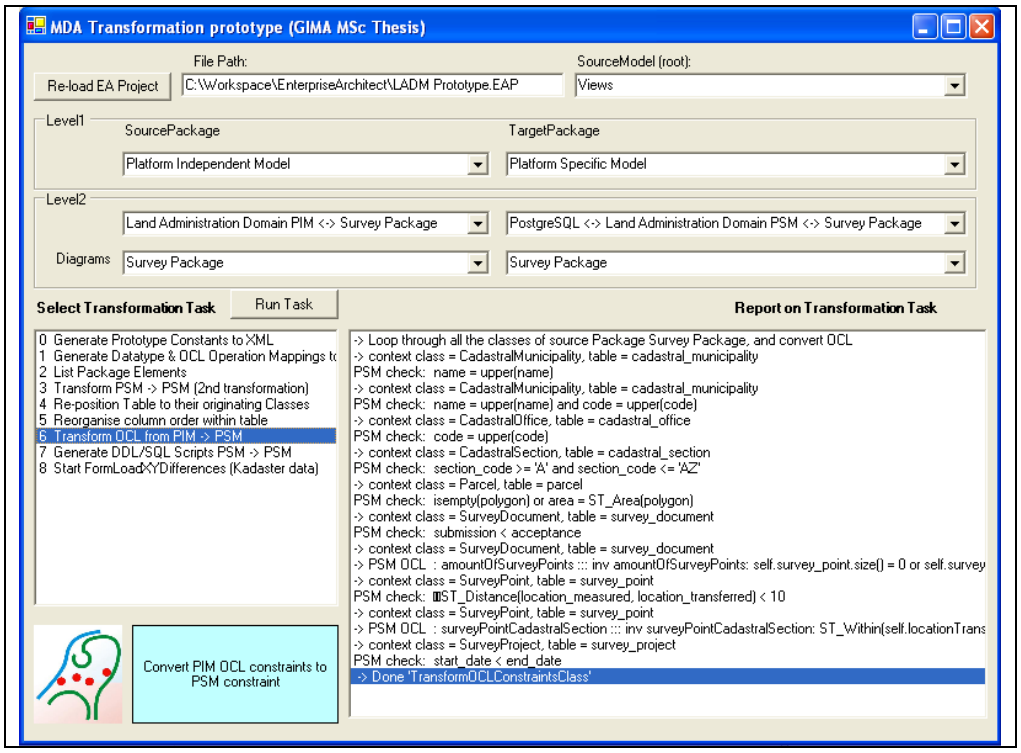


Figure 96 - Prototype Report after 3rd transformation from PIM OCL to PSM-2 OCL

The following types of constraints are discussed:

- Constraints Applicable to One Instance of One Class
- Constraints Applicable to Multiple Instances of One Class
- Constraints Applicable to Multiple Instances of Multiple Classes

Constraints Applicable to One Instance of One Class

Constraints that can be checked by addressing/knowing only attributes of one instance (tuple) of one class, fall into the category "One Instance of One Class". Examples are provided for the following types of attribute constraints:

- Mandatory Attribute (Not Null)
- Maximum Attribute Length
- Range (OCL, example alphanumeric range)
- Domain (list of possible values)
- Autonumber
- Format (OCL, example: Upper)
- Tuple (OCL, example: IsEmpty & ST_Area)

Mandatory Attribute (Not Null)
The attribute must have a value, i.e. "Not Null". See section 6.6, section "Set Column "Not Null" property" on the prototype implementation. This constraint could be defined with OCL, but it is perceived more clear to implement the constraint in the UML model with the Lower- and Upper bound properties.

Type	Mandatory Attribute
PIM OCL	Not used
	context Parcel inv: oid.notEmpty()
PIM UML	In the prototype the Lower bound / Upper bound properties of an attribute are used for this constraint.
PSM	the "Not Null" column property checkbox.
Implementation	ALTER TABLE parcel ALTER COLUMN oid SET NOT NULL;

Maximum Attribute Length
The attribute has a maximum length in number of characters. See section 6.6, section "Transform Attribute " on the prototype implementation based on file *DatatypeMapping.xml* with lengths of attributes.

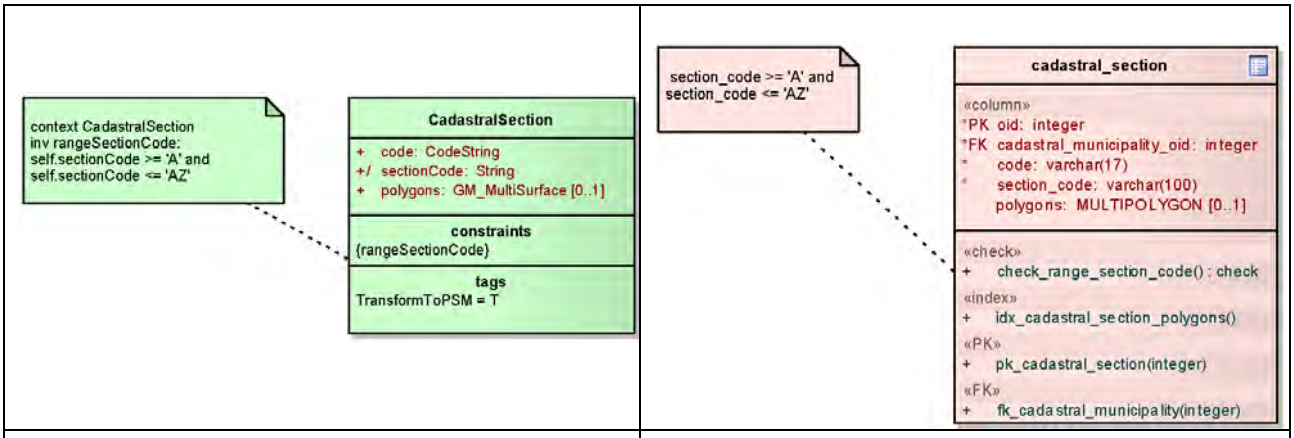
Type	Maximum Length
PIM OCL	Not used
PIM UML	in the prototype the use of data types, e.g. CodeString, String, and NotesString, results in to certain column lengths, resp. varchar(17), varchar(100), varchar(2000).
PSM	the "Length" column property
Implementation	ALTER TABLE parcel ADD COLUMN code varchar(17);

Data type
The attribute must be of a certain data type (e.g. String, Integer, GM_Point, GM_Polygon). See section 6.6, section "Transform Attribute " and "Geometry " on the prototype implementation based on file *DatatypeMapping.xml*. In these sections, the use of SFA-SQL (Simple Features) data types Point, Linestring, Polygon, and Multipolygon [Open Geospatial Consortium, 2006c] is described, resulting in implemented data types and automatically generated spatial constraints.

Type	Data type
PIM OCL	Not used
PIM UML	PIM data types
PSM	the data type column property
Implementation	ALTER TABLE parcel ADD COLUMN code varchar(17); select addgeometrycolumn ('survey_point','location_measured',28992,'POINT',2);

Range (OCL, example alphanumeric range)
The attribute value must be between a defined range of a begin and an end value (Figure 97). An example has been created for class CadastralSection and attribute sectionCode. This constraint has been transformed to PSM using the transformed table and column names, i.e. cadastral_section and section_code. Subsequently the OCL constraint has been implemented as table check constraint, i.e.

check_range_section_code, based on the "Status" property of the constraint, which is set to "PSM check", see also Figure 38.



Before 1st Transformation	After 3rd Transformation

Figure 97 - 3rd Transformation (from PIM OCL to PSM-2): Implement Range Constraint

Type	Range
PIM OCL	context CadastralSection inv: self.sectionCode >= 'A' and self.sectionCode <= 'AZ'
PIM UML	Not used
PSM	table check constraint
Implementation	`ALTER TABLE CONSTRAINT check_range_section_code` `CHECK (section_code >= 'A' AND section_code <= 'AA');`

Domain (list of possible values)

The attribute value must be one of the items in a pre-defined list of (look-up) values.

Type	Domain / enumeration
PIM OCL	Not used
PIM UML	classes stereotyped <<enumeration>>, and <<CodeList>>
PSM	resp. table check constraint or look-up table

Implementation	`<<enumeration>>`
	`ALTER TABLE survey_point`
	`  ADD CONSTRAINT check_surveypointquality`
	`  CHECK (quality = ANY (ARRAY['local', 'gnss']));`
	
	`<<CodeList>>`
	`CREATE TABLE codelist_surveydocumenttype`
	`(`
	`  oid serial NOT NULL,`
	`  value character varying NOT NULL,`
	`  CONSTRAINT pk_codelist_survey_document_type PRIMARY KEY (oid)`
	`)`

In the prototype, both classes stereotype <<enumeration>> and <<CodeList>> are used to generate respectively table check constraints and look-up tables, see section 6.6, section "Transform Classes, stereotyped <<enumeration>> and <<CodeList>>".

Autonumber

The value of the attribute should by default come from a table specific sequence/increment. In the PIM, no method exists to specify this. In the prototype all classes have a "oid" column (primary key), with a sequence attached, see "Appendix F: Details on First Transformation in MDA Prototype (PIM to PSM-1)", section "Generate Primary Key" for a description on how sequences are created based on a setting in the EA transformation templates and in the xml file PrototypeConstants.xml.

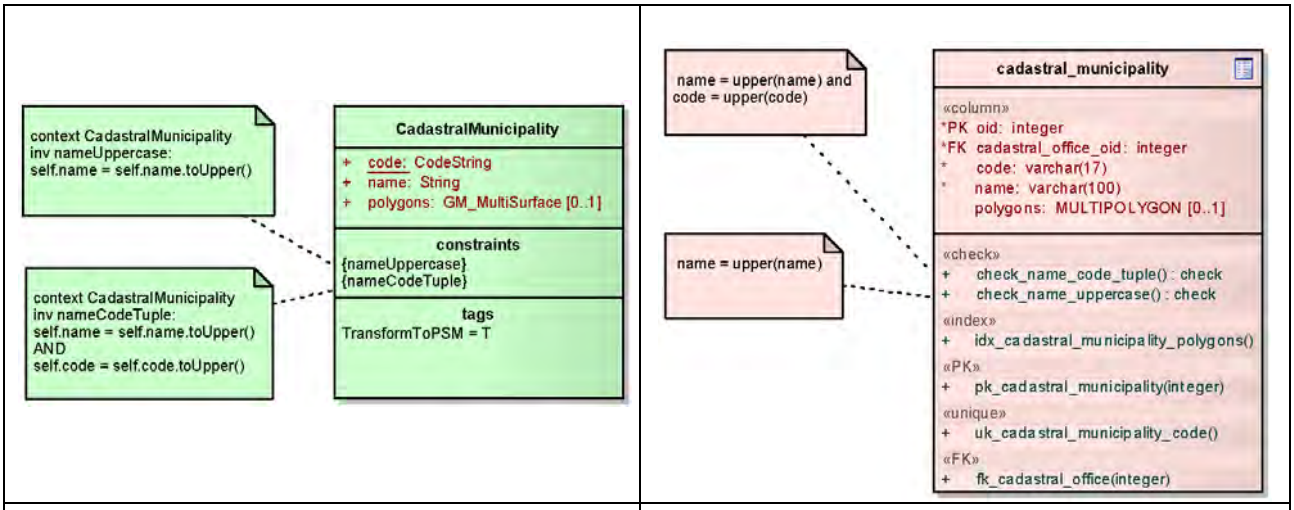
Type	Autonumber, sequence
PIM OCL	Not used
PIM UML	Not used
PSM	the column properties: AutoNumber, StartValue, Increment
Implementation	CREATE SEQUENCE parcel_oid_seq INCREMENT 5 START 1000; CREATE TABLE parcel (oid integer DEFAULT NEXTVAL('parcel_oid_seq') NOT NULL)

Format (OCL, example: Upper)

The format of the attribute value is constrained, the value should be in capital characters (uppercase, Figure 98), a zip code consists of 4 digits and 2 characters, the 11-test for bank account numbers.

Type	Format
PIM OCL	context CadastralMunicipality inv nameUppercase: self.name = self.name.toUpper()
PIM UML	Not used
PSM	table check constraint
Implementation	ALTER TABLE cadastral_municipality ADD CONSTRAINT check_name_uppercase CHECK (name = upper(name));

Note that both constraints nameUppercase and nameCodeTuple are "overlapping" constraints, where the former is implied by the latter. These constraints are maintained as an example.



Before 1st Transformation; OCL constraints	After 3rd Transformation; Check Constraints

Figure 98 - 3rd Transformation (from PIM OCL to PSM-2): Implement Format Constraint

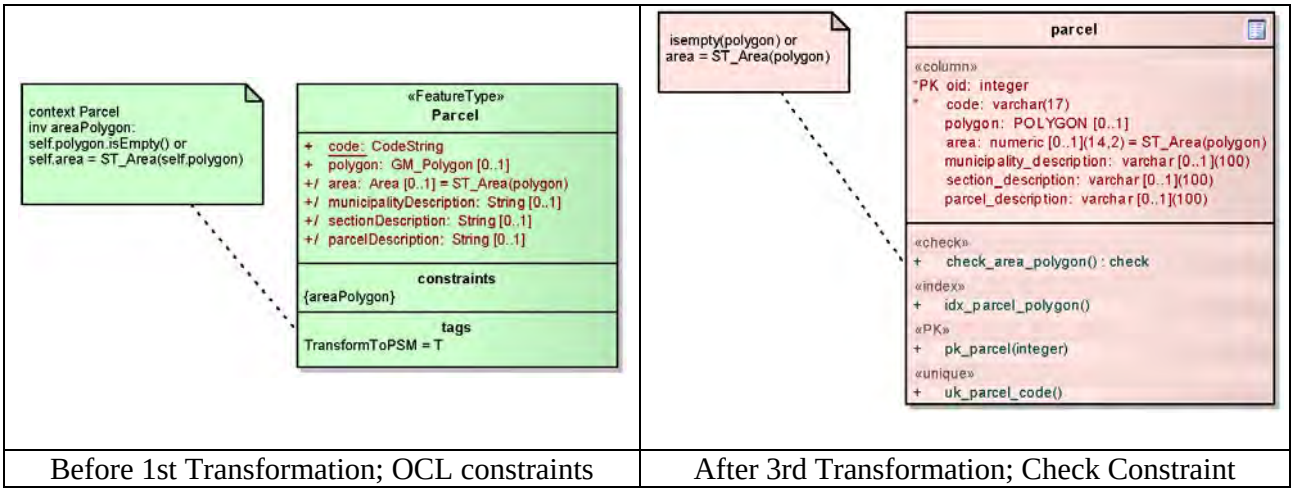
Tuple (OCL, example: IsEmpty & ST_Area)

Tuple rules involve two or more attributes of the same instance, which can also be implemented as base table check constraints, see Figure 99, where the transformation of a tuple constraint based on attributes Parcel.area and Parcel.polygon is shown.

Type	Tuple constraint
PIM OCL	context SurveyProject inv startDateBeforeEndDate: self.startDate < self.endDate
PIM UML	Not used
PSM	table check constraint
Implementation	ALTER TABLE cadastral_municipality ADD CONSTRAINT check_start_date_before_end_date CHECK (start_date < end_date);

Type	Tuple (spatial) constraint
PIM OCL	context Parcel inv areaPolygon: self.polygon->isEmpty() or self.area = ST_Area(self.polygon)
PIM UML	Not used
PSM	table check constraint
Implementation	ALTER TABLE parcel ADD CONSTRAINT check_area_polygon CHECK (isempty(polygon) or area = ST_Area(polygon));

Type	Tuple (spatial) constraint
PIM OCL	context SurveyPoint inv distanceMeasuredTransferred: ST_Distance(self.locationMeasured, self.locationTransferred) < 5
PIM UML	Not used
PSM	table check constraint
Implementation	ALTER TABLE survey_point ADD CONSTRAINT check_distance_measured_transferred CHECK (ST_Distance(location_measured, location_transferred) < 5); *or* ALTER TABLE survey_point ADD CONSTRAINT check_distance_measured_transferred CHECK (ST_DWithin(location_measured, location_transferred, 5)



Before 1st Transformation; OCL constraints	After 3rd Transformation; Check Constraint

Figure 99 - 3rd Transformation (from PIM OCL to PSM-2): Implement Tuple Constraint

Constraints Applicable to Multiple Instances of One Class

Constraints that can be checked by addressing/knowing attributes of multiple instances of one class, fall into this category. Not that the previous category (one instance of one class) in a sense falls within this category, with the subtle difference that category, cannot be checked by looking at just one instance, and therefore not implemented with a check constraint.

Examples are provided for the types of constraint:

- Primary Key Constraint
- Unique Key Constraint
- Other (OCL, example: overlapping parcels)

Primary Key Constraint

An example of a constraint concerning multiple instances of one class is the primary key constraint. The value(s) of the primary key attribute(s) must be unique within the class and this must be checked against other instances (objects) of the same class.

Type	Primary Key
PIM OCL	not used
PIM UML	not used primary key columns ("oid") are generated in first transformation PIM to PSM.
PSM	column property "primary key" and "not null" (automatically set) result in a primary key constraint
Implementation	ALTER TABLE parcel ADD CONSTRAINT pk_parcel PRIMARY KEY(oid);

Unique Key Constraint

The value(s) of the unique key attribute(s) must be unique within the class and this must be checked against other instances of the same class.

Type	Unique Key
PIM OCL	not used
PIM UML	column property "static"
PSM	column property "unique key" result in a unique key constraint
Implementation	ALTER TABLE cadastral_municipality ADD CONSTRAINT uk_cadastral_municipality_code UNIQUE(code);

Other (OCL, example: overlapping parcels)

Other constraints involving one or more attributes for multiple instances of the same class can have many variants, an example with regard to a geographic constraint is provided below.

Type	Other "Multiple Instances of One Class": spatial constraint
PIM OCL	context Parcel p1 inv noOverlappingParcels: p1.polygon->notEmpty() implies not exists (p2: Parcel | ST_Intersects(p1.polygon, p2.polygon)
PIM UML	not possible
PSM	constraint view showing records in violation
Implementation	create view v_ocl_no_overlapping_parcels as select p1.polygon from parcel p1 where not exists (select 1 from parcel p2 where ST_Intersects(p1.polygon, p2.polygon));

Constraints Applicable to Multiple Instances of Multiple Classes
The remaining type of constraints have to be checked by querying the attributes of multiple instances of multiple classes. Examples are provided for the types of constraint:

- Foreign Key Constraint
- Relationship Cardinality (OCL)
- Derivation (OCL)
- Other (OCL, example: ST_Within)

Foreign Key Constraint
The value(s) of the foreign key attribute(s) must exist in the primary key attribute values of the related table, which is checked by reviewing instances of one other class.

Type	Foreign Key
PIM OCL	not used
PIM UML	Associations with [0..1] or [1] multiplicity on "1" side and [*] on the "many" side..
PSM	operation stereotyped "FK" result in a foreign key constraint
Implementation	ALTER TABLE survey_document ADD CONSTRAINT fk_survey_project FOREIGN KEY (survey_project_oid) REFERENCES survey_project (oid)

Relationship Cardinality (OCL)
For a relationship a cardinality can be defined on both sides. Cardinality refers to the quantity of the instances the user can select from a relationship, e.g. [0..1] , [1], [*], see previous section on Foreign Key.

Note that a single cardinality can be indicated in UML, but in the prototype a choice has been made for OCL, also because combinations of cardinality can be defined, for Example if a SurveyDocument must have either 0 or more than 2 SurveyPoints:

Type	Relationship Cardinality
PIM OCL	context SurveyDocument inv amountOfSurveyPoints: self.SurveyPoint->size() = 0 or self.SurveyPoint->size() > 2
PIM UML	not used:
PSM	view

Implementation	`create view v_ocl_amount_of_survey_points as`
	`select self.oid, count(spt.source_oid)`
	`from survey_document self`
	`, survey_point spt`
	`where self.oid = spt.source_oid`
	`group by self.oid`
	`having not (count(spt.source_oid) = 0 or count(spt.source_oid) > 2);`

Derivation (OCL)

Derivation of attribute values, although not a constraint, is discussed here because it can be described in OCL as part of the Adapted LADM 'Survey Package', for example the value for class SurveyDocument, attributes municipalityDescription and sectionDescription:

Type	Derivation
PIM OCL	context SurveyDocument::sectionDescription derive: CadastralSection.code
PIM UML	not used
PSM	view

Implementation	`create view v_ocl_derive_section_description as`
	`select csn.code`
	`from survey_document self`
	`, cadastral_section csn`
	`where self.cadastral_section_oid = csn.oid;`

Related to derivation, is the initial (default) value, for example:

Type	Derivation
PIM OCL	context parcel::area derive: ST_Area(polygon)
PIM UML	not used
PSM	view
Implementation	`before insert trigger` `:new.area = ST_Area(new:polygon)`

In PostgreSQL, the default expression to fill an initial value, cannot be based on other columns. For example, the following statement, based on a default value for column parcel.area is not possible:

```
ALTER TABLE parcel ALTER COLUMN area SET DEFAULT ST_Area(polygon)  ;
```

The implementation of this OCL constraint would be to construct code to derive the "ST_Area(polygon)", and used that in BEFORE INSERT UPDATE triggers.

Other (OCL, example: ST_Within)

Other OCL constraints can be constructed by navigating classes and associations. For example, to make sure that a SurveyPoint is within the CadastralSection to which it is allocated, an OCL rule can be defined.

Type	Other: spatial constraint
PIM OCL	context SurveyPoint inv surveyPointCadastralSection: ST_Within(self.locationTransferred, self.SurveyDocument.CadastralSection.polygons)
PIM UML	not used
PSM	view

Implementation	`create view v_ocl_survey_point_cadastral_section as`
	`select csn.code`
	`from survey_point self`
	`, survey_document sdt`
	`, cadastral_section csn`
	`where self.source_oid = sdt.oid`
	`and sdt.cadastral_section_oid = csn.oid`
	`and`
	`not ST_Within(self.location_transferred, csn.polygons);`

Appendix I: Details on the Generation of DDL Scripts in MDA Prototype (PSM-2 to PostgreSQL/PostGIS)

The fourth transformation is from the final PSM in Enterprise Architect to the actual implementation in PostgreSQL/PostGIS (section 7.3). The following scripts from PSM to target platform PostgreSQL/PostGIS have been defined for the actual implementation in PostgreSQL (in the specified order):

- Delete Objects
- Create Sequences
- Create Types
- Create Tables
- Create Geographic Columns
- Create Primary Key Constraints
- Create Constraints
- Create Indexes
- Create Views
- Populate Look-up Tables
- Present OCL Constraints

These generated scripts are available at URL 30.

Delete Objects
All objects, to be created in subsequent scripts, will first be deleted.

Input Element	All Classes
DDL Script	Before creating types, sequences, tables, geometry columns, constraints, indexes and views, these database will be cleaned up, by deleting the objects to be created (with drop cascade statements).
Output Element	DeleteObjects.sql
Tool	PrototypeAddin.CreateDDLScript
Example	`DROP SEQUENCE cadastral_office_oid_seq CASCADE;`

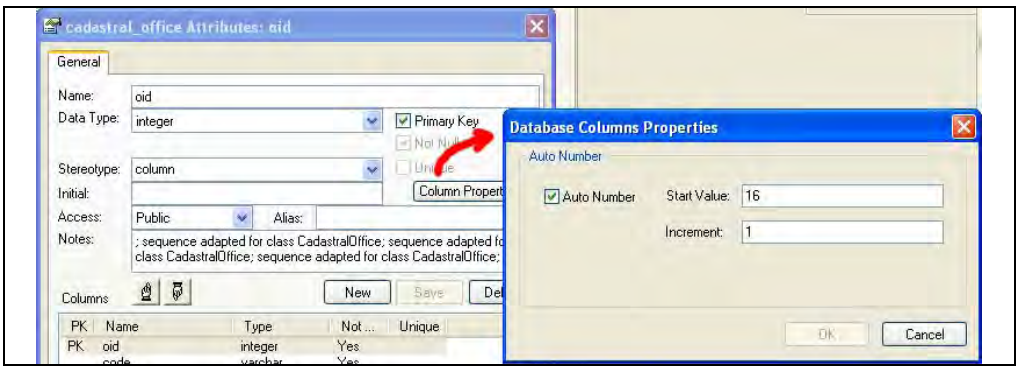


Figure 100 - Define a Sequence in Enterprise Architect

Create Sequences
In Enterprise Architect a sequence can be defined through the dialogbox presented below. The values entered are stored as a Tagged Value for the relevant column, either by the EA user interface (Figure 100) or by the MDA prototype, see section 6.5 on primary keys and sequences.

Input Element	Attributes with Autonum Tagged Value, for example "AutoNum=1;StartNum=16;Increment=1;NotForRep=0;"
DDL Script	Attributes with a sequence defined will be used to generate a sequence to populate oid (primary key) columns.
Output Element	CreateSequences.sql
Tool	PrototypeAddin.CreateDDLScript
Example	`CREATE SEQUENCE cadastral_office_oid_seq INCREMENT 1 START 16;`

Create Types
When columns have a datatype that is referring to a class, stereotyped <<type>>, and the column is not implemented as a child table, then a type will be generated in the PostGIS database, see section 6.6, section "Process columns defined by <<Type>> classes".

Input Element	Classes, stereotyped "type"
DDL Script	All classes with stereotype "type" will be used to generate a DDL script with "create statements" for the type.
Output Element	CreateTypes.sql
Tool	PrototypeAddin.CreateDDLScript

Example	`CREATE TYPE PersonType AS (`
	`  user_name  varchar,`
	`  first_name  varchar,`
	`  last_name  varchar`
	`);`

Create Tables
Tables will be created, based on the classes, stereotyped <<table>>, with the exception of geographic columns, see next section on "Create Geographic Columns".

Input Element	Classes, stereotyped "table"
DDL Script	All classes with stereotype "table" will be used to generate an DDL script with "create statements" for the table.
Output Element	CreateTables.sql
Tool	PrototypeAddin.CreateDDLScript

Example	`CREATE TABLE survey_point (`
	`  oid  integer  DEFAULT`
	`  NEXTVAL('survey_point_oid_seq') NOT NULL,`
	`  source_oid  integer  NOT NULL,`
	`  quality  varchar(30)  NOT NULL,`
	`  point_number varchar(100)  NULL`
	`);`

Create Geographic Columns
In PostGIS it is possible to create tables including the geographic data types POINT, LINESTRING, etc. However, the PostGIS metadata table geometry_columns as specified in SFA-SQL [Open Geospatial Consortium, 2006c], will not be populated automatically. Therefore an "alter table statement" is generated and used to create geographic columns in the PostGIS database based on spatial reference system (e.g. 28992 for Dutch "Rijskdriehoek"), spatial data type (e.g. "Point"), and dimension (e.g. "2").

Input Element	Attributes, stereotyped "column" and geographic data type
DDL Script	All classes with stereotype "table" and with columns of geographic data type (e.g. POINT, LINESTRING, POLYGON) will be used to generate a DDL script with "alter statements" for the table. N.B. on behalf of the meta-data table geometry_columns, PostGIS requires the geographic columns to be added/alter after table creation
Output Element	CreateGeometry.sql
Tool	PrototypeAddin.CreateDDLScript
Example	`select addgeometrycolumn ('survey_point','location_measured',28992,'POINT',2); select addgeometrycolumn ('survey_point','location_transferred',28992,'POINT',2) ;`

Create Primary Key Constraints
For all primary key columns a primary key constraint will be generated. This has to be conducted before the creation of other constraints (see next section "Create Constraints"), such as foreign key constraints, which will refer to the primary key constraints.

Input Element	Operations, stereotyped <<PK>>
DDL Script	A primary key constraint will be generated for the primary key columns.
Output Element	CreatePkConstraints.sql
Tool	PrototypeAddin.CreateDDLScript
Example	`ALTER TABLE parcel ADD CONSTRAINT pk_parcel PRIMARY KEY ( oid );`

Create Constraints
All other constraints (foreign key, unique key, and check) will be generated.

Input Element	Operations, stereotyped <<unique>>, <<check>>, or <<FK>>
DDL Script	Unique key constraints, check constraints and foreign key constraints will be generated, based on operations with above mentioned stereotypes.
Output Element	CreateConstraints.sql
Tool	PrototypeAddin.CreateDDLScript

Example	`ALTER TABLE cadastral_office ADD CONSTRAINT uk_cadastral_office_code UNIQUE ( code );`
	
	`ALTER TABLE cadastral_office ADD CONSTRAINT check_code_uppercase CHECK ( code = upper(code) );`
	
	`ALTER TABLE cadastral_section ADD CONSTRAINT fk_cadastral_municipality FOREIGN KEY ( cadastral_municipality_oid ) REFERENCES cadastral_municipality ( oid );`

Create Indexes

PostgreSQL automatically creates indexes for primary, unique and foreign key columns. Any other index that is explicitly defined in the PSM on one or more columns, will be created by this script, including the geographic indexes, see section 6.6, section "Transform Attribute " on indexes.

Input Element	Operations, stereotyped <<index>>
DDL Script	Explicitly described indexes will generated, besides the automatically indexes generated for primary, unique and foreign key columns.
Output Element	CreateIndexes.sql
Tool	PrototypeAddin.CreateDDLScript

Example	`CREATE INDEX idx_survey_point_location_measured ON survey_point USING GIST ( location_measured );`
	
	`CREATE INDEX idx_survey_point_location_transferred ON survey_point USING GIST ( location_transferred );`

Create Views

Classes, stereotyped <<view>>, for example as a result of handling, transforming and implementing OCL constraints, will be generated

Input Element	Classes, stereotyped <<view>>
DDL Script	Views will be generated, based on classes, stereotyped <<view>>.
Output Element	CreateViews.sql
Tool	PrototypeAddin.CreateDDLScript
Example	`CREATE OR REPLACE VIEW "v_ocl_survey_point_cadastral_section" AS SELECT csn.code FROM survey_point self , survey_document sdt , cadastral_section csn WHERE (((self.source_oid = sdt.oid) AND (sdt.cadastral_section_oid = csn.oid)) AND (NOT st_within(self.location_transferred, csn.polygons)));`

Populate Look-up Tables

Based on the classes stereotyped <<CodeList>>and their attributes, insert scripts with the allowed values will be generated, see section 6.6, section "Transform Classes, stereotyped <<enumeration>> and <<CodeList>>". Note that currently the first column *oid* is generated based on the sequence of the attributes within the originating enumeration class, and the short code (e.g. B01) is combined/concatenated with the description of the code (e.g. Main Building), stored in column *value*.

Input Element	Class stereotyped "CodeList"
DML Script	The attribute names of the "CodeList" class will be used to generate an SQL script with "insert statements" for the lookup table.
Output Element	Fore example: Createcodelist_buildingquality.sql Createcodelist_lkiclassification.sql Createcodelist_surveydocumenttype.sql
Tool	PrototypeAddin.CreateDDLScript
Example	`Insert into codelist_buildingquality (oid, value) VALUES ( 1, 'D0'); Insert into codelist_buildingquality (oid, value) VALUES ( 2, 'D1'); Insert into codelist_buildingquality (oid, value) VALUES ( 3, 'D2');`
	`Insert into codelist_lkiclassification (oid, value) VALUES ( 1, 'B01 - Main Building'); Insert into codelist_lkiclassification (oid, value) VALUES ( 2, 'B03 - Miscellaneous Building');`
	`Insert into codelist_surveydocumenttype (oid, value) VALUES ( 1, 'fieldSketch'); Insert into codelist_surveydocumenttype (oid, value) VALUES ( 2, 'gnssSurvey'); Insert into codelist_surveydocumenttype (oid, value) VALUES ( 3, 'relativeMeasurement');`

Present OCL Constraints

Some of the PIM OCL constraints have been transformed to PSM and then implemented for example as base table check constraints, or via "constraint views". When an OCL constraint has not automatically been implemented, it will be transformed to PSM and listed in the overview OCLSpecification.ocl, serving as a basis for manual implementation in the PostgreSQL and PostGIS environment.

Input Element	Class constraints
Text File	OCL constraints that have not been converted to check constraints or other means of implementation, will be reported as input/specification of a manual implementation
Output Element	OCLSpecification.ocl
Tool	PrototypeAddin.CreateDDLScript

Example	`\* OCL constraint amountOfSurveyPoints *\`
	`context survey_document`
	`inv amountOfSurveyPoints:`
	`self.survey_point.size() = 0 or`
	`self.survey_point.size() > 2`
	
	`\* OCL constraint surveyPointCadastralSection *\`
	`context survey_point`
	`inv surveyPointCadastralSection:`
	`ST_Within(self.location_transferred ,`
	`self.survey_document.cadastral_section.polygons)`

Appendix J: Load Data into Adapted LADM 'Survey Package' PostGIS Database

The loading of data (section 7.4), provided by Kadaster is described in the following sections of this Appendix:

- Conversion MapInfo to PostGIS (temporary tables)
- Conversion ASCII Files to PostGIS (temporary tables)
- Conversion temporary tables into LADM SP (PostGIS)

Conversion MapInfo to PostGIS (temporary tables)
One of the tools that were used for conversion of spatial data from one format to another is FWtools (section 7.2.3, URL 22), offering commands like below to convert a MapInfo *.tab file into the PostGIS database:

```
ogr2ogr -overwrite -nlt POLYGON -a_srs EPSG:28992 -f PostgreSQL
PG:"dbname='postgis' user='GIMA'" ut_vlak.TAB
```

About 7.5 million records were converted in this manner into the prototype PostGIS database into temporary tables, later to be used to populate the LADM SP tables:
- Buildings (ut_gebw): 680,157 records
- Boundaries (ut_grns): 1,656,077 records
- Basic Points (ut_grns, NL: Grondslag Punten): 956 records
- Lines (ut_lijn): 3,685,521 records
- Parcel Numbers (ut_prnr): 429,107 records
- Symbol(ut_symb): 432,557 records
- Parcels (ut_vlak): 429,107 records
- Annotations (ut_text): 257,018 records

Note that the MapInfo tables had no SRID (Spatial Reference IDentifier) defined for the data, which could not be solved by specifying it in the command described above (EPSG:28992). In PostGIS this had to be altered to the 2D spatial reference system "Amersfoort / RD New"; EPSG:28992, with a command like:

```
select UpdateGeometrySRID('ut_vlak','wkb_geometry',28992);
```

Indexes have been manually added in PostGIS to improve performance on the (spatial) data manipulation with a command like:

```
CREATE INDEX idx_vlak ON ut_vlak USING GIST (wkb_geometry);
```

The Annotations in MapInfo format in the table ut_text (mostly categories Z01: Parcel Number, and Z06: Streetname) have not been transformed by the FWTools to PostGIS, the attributes of the table have been converted, but the text itself at its orientation could not be converted, which is a common conversion problem for data with annotations. A similar observation could be made for ut_prnr (with classification Z01: parcel number).

The buildings (ut_gebw2nd) were delivered as linestrings, so the PostGIS function ST_POLYGONIZE was used to create polygons from linestrings, to be able to populate the buildings table with polygons (Figure 41). The function returned about 250,000 building polygons after about a day of processing.

```
SELECT geom AS geom
FROM st_dump((
        SELECT st_polygonize(wkb_geometry) AS geom
        FROM ut_gebw2nd));
```

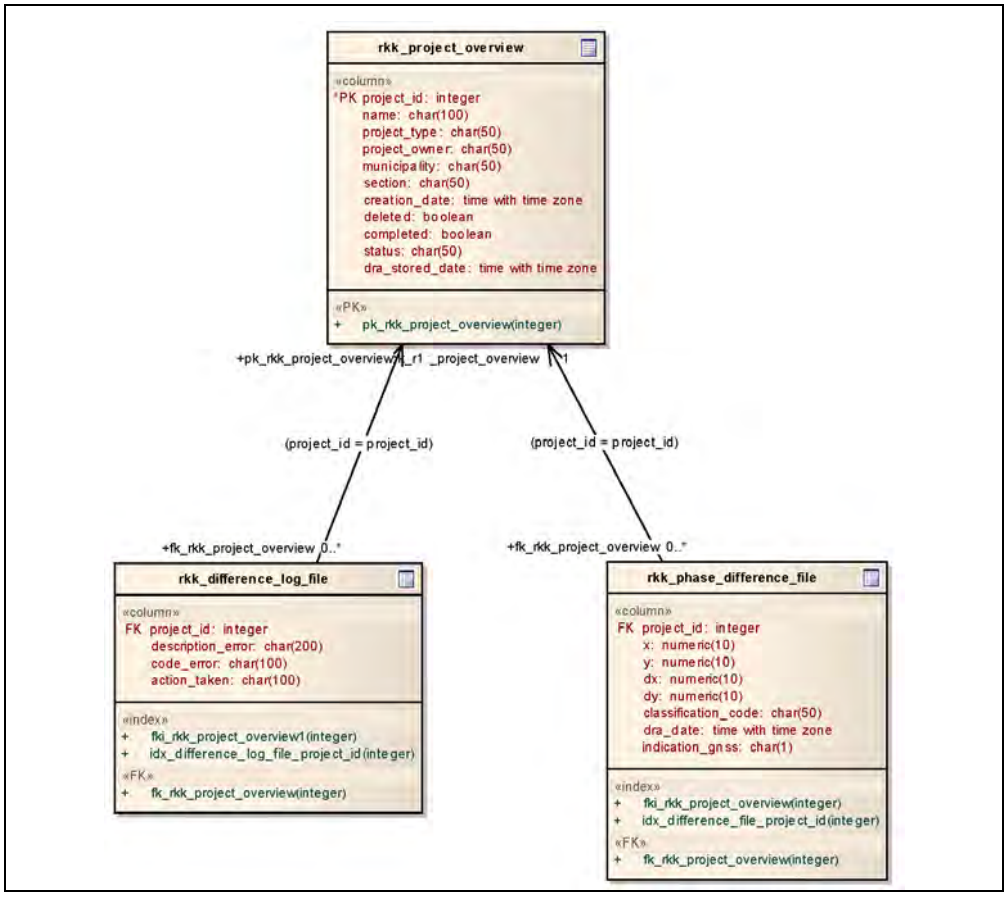


Figure 101 - Temporary Tables Containing Survey Projects and Connection Points

Conversion ASCII Files to PostGIS (temporary tables)

The text files with differences between coordinates before and after the 2nd phase control point constrained network adjustment (see section 5.2.2), as well as the information on survey projects and error logging were converted into an insert script, with a C# program, part of the MDA prototype (section 6.3). The insert script loads the (temporary) tables rkk_phase_difference_file, rkk_difference_log_file, and rkk_project_overview (Figure 101). Figure 102 shows the MDA prototype user interface with the total records in the period April 2006- December 2007. These temporary tables function as an intermediate storage of the text files with the goal of converting all information in the text files to a PostgreSQL database. Stored functions in the PostgreSQL database will be used to convert this data to the prototype tables.

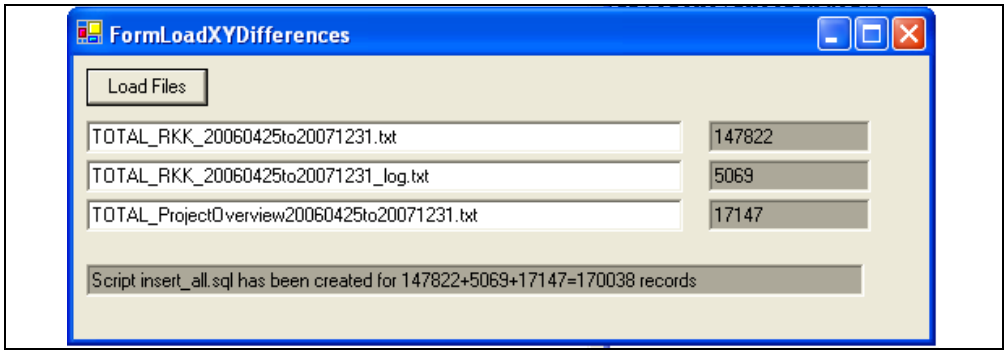


Figure 102 - Prototype User Interface to create DML/SQL insert scripts for 3 tables

Conversion temporary tables into LADM SP (PostGIS)
Once the source data has been loaded from PostGIS external formats into temporary PostGIS tables, these temporary tables have been converted to the *Adapted LADM 'Survey Package'* tables. A selection of the stored functions that have been created to populate the tables and to manipulate data in these tables:

- load_building
- load_building_intersection
- load_cadastral_municipality
- load_parcel
- load_parcel_intersection
- load_survey_document
- load_survey_point (Figure 103)
- load_survey_point_analysis (Figure 104)
- load_survey_point_intersection
- load_survey_point_intersection_parcel

An example of one of the PostGIS stored function that was created for populating the table survey_point is provided in Figure 103 below. The stored function *load_survey_point()* is structured as:

- Clear the table with warnings and errors (i.e. error_messages), which will be populated during the course of this stored function *load_survey_point()*
- Loop through all survey point records of table *rkk_phase_difference_file* (Figure 101), ordered by survey project
  - Maintain a *survey point number counter* within the survey project.
  - Create point geometry based on x and y coordinate-elements for *locationMeasured* and *locationTransferred*.
  - determine pointQuality, defined as either 'gnss' or 'local' measurements
  - Insert a new record in table *survey_point*, based on the values that have been prepared for individual columns.
- Report the number of created records in table *survey_point*.

Figure 103 - Example of PostGIS load function: load_survey_point()

```
CREATE OR REPLACE FUNCTION load_survey_point()
  RETURNS text AS
$BODY$
DECLARE
   c_rkk_phase_difference_file RECORD; -- declare a generic record to be used in a FOR LOOP
   iProject integer;
   locationMeasured geometry; --before 2nd phase
   locationTransferred geometry; -- after second phase
   pointQuality survey_point.quality%type;
   previousProject int;
   counter int;
   pointCounter int;
   totalSurveyPoint integer;
BEGIN
   delete from error_messages;
   previousProject = 0;
   select count(*)+1 into counter from survey_point;

   /* process records in rkk_phase_difference_file */
   FOR c_rkk_phase_difference_file IN select * from rkk_phase_difference_file order by 1,2,3 LOOP
-- loop through all records
      counter = counter +1;

      if previousProject = c_rkk_phase_difference_file.project_id
      THEN
         pointCounter = pointCounter + 1;
      ELSE
         pointCounter = 1;
      END IF;

      insert into error_messages values (c_rkk_phase_difference_file.project_id, counter);
      locationTransferred
      = GeomFromText(
      'POINT('||c_rkk_phase_difference_file.x||' '||c_rkk_phase_difference_file.y||')',28992);
      locationMeasured
      = GeomFromText(
      'POINT('||c_rkk_phase_difference_file.x+c_rkk_phase_difference_file.dx||'
             '||c_rkk_phase_difference_file.y+c_rkk_phase_difference_file.dy||')',28992);

      if c_rkk_phase_difference_file.indication_gnss = 'J'
      THEN
         pointQuality = 'gnss';
      ELSE
         pointQuality = 'local';
      END IF;
```

```
      begin
    insert into survey_point
        ( oid
        , source_oid
        , quality
        , point_number
        , location_measured
        , location_transferred
        )
    values
        ( counter
        , c_rkk_phase_difference_file.project_id
        , pointQuality
        , pointCounter
        , locationMeasured
        , locationTransferred
        );
      end;
      previousProject = c_rkk_phase_difference_file.project_id;
   END LOOP;

select count(*) into totalSurveyPoint from survey_point;

return 'ready; '||totalSurveyPoint||' records in survey_point';
END;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;
ALTER FUNCTION load_survey_point() OWNER TO "GIMA";
```

Figure 103 - Example of PostGIS load function: load_survey_point()

Appendix K: Stored Function to Select Survey Points for Analysis

The PostgreSQL function *load_survey_point_analysis()* as given below (Figure 104), has been used to exclude outliers from the survey point data. The stored function is started with the parameters *max_distance*, *max_arithmetic_mean*, and *sigma_multiplier*, as explained in section 7.5.1.

- Clear the table with warnings and errors (i.e. error_messages), which will be populated during the course of this stored function *load_survey_point_analysis()*.
- Exclude survey_points with a distance over *max_distance* (e.g. 5 meter)
- Loop through all survey project records (distinctly selected from *survey_point*, via view *v_survey_point_transferred*).
  - Determine number of survey points within the survey project
  - Determine the average distance (μ ~ arithmetic mean) for survey project
  - Determine the standard deviation (σ) for survey project
  - Loop through all survey point records (selected from view *v_survey_point_transferred*).
    - Exclude or include survey points, dependent on the *arithmetic_mean* in relation to the parameter *max_arithmetic_mean*; and the number of survey points (*number_of_values*) in a survey project; and the distance between connection point coordinates (before and after the 2nd phase control point constrained network adjustment), in relation to the product of *sigma_multiplier* and *standard_deviation*
- Report the number of excluded records in table *survey_point*, accompanied with the function parameter settings.

Figure 104 - Example of PostGIS load function: load_survey_point_analysis()

```
CREATE OR REPLACE FUNCTION load_survey_point_analysis(max_distance numeric, max_arithmetic_mean
numeric, sigma_multiplier numeric)
  RETURNS text AS
$BODY$
DECLARE
   c_survey_project RECORD;
   c_survey_point RECORD; -- declare a generic record to be used in a FOR
   number_of_values integer;
   arithmetic_mean numeric ;
   standard_deviation numeric ;
   excluded_max_distance_survey_points integer;
   count_excluded_survey_points integer;

   totalSurveyPoint integer;
   readyText varchar;

BEGIN
   delete from error_messages;
   count_excluded_survey_points=0;

   /* count survey points above max_distance */
   select count(*) into excluded_max_distance_survey_points
   from survey_point
   where ST_Distance(location_measured,location_transferred)>= max_distance;

   /* disable survey_points with a distance over max_distance (e.g. 5 meter) */
   update survey_point set exclude = 'Y'
   where ST_Distance(location_measured,location_transferred)>= max_distance;

   /* loop through survey projects */
   FOR c_survey_project IN
      select distinct (source_oid) from v_survey_point_transferred order by 1 LOOP

     /* determine number of values */
      select count(*) into number_of_values
      from v_survey_point_transferred
      where source_oid = c_survey_project.source_oid
      and distance < max_distance;

     /* determine average, arithmetic mean */
      select avg(distance) into arithmetic_mean
      from v_survey_point_transferred
      where source_oid = c_survey_project.source_oid
      and distance < max_distance;
```

```
     /* determine standard deviation */
      select sqrt(sum(power((distance-arithmetic_mean),2))/number_of_values) into
standard_deviation
      from v_survey_point_transferred
      where source_oid = c_survey_project.source_oid
      and distance < max_distance;

      insert into error_messages values (0, 'project:  '||c_survey_project.source_oid||'
arithmetic_mean:  '||arithmetic_mean|| ' count: '||number_of_values|| ' stddev:
'||standard_deviation|| ' boundary: '||arithmetic_mean + sigma_multiplier*standard_deviation);

      /* loop through survey points per project */
      FOR c_survey_point IN
         select * from v_survey_point_transferred where source_oid = c_survey_project.source_oid
and distance < max_distance order by 1 LOOP -- loop through all projects

         if arithmetic_mean > max_arithmetic_mean then
         /* deal with all survey points where arithmetic_mean > 1.32m */

            if number_of_values > 5 then
            /* deal with more than 5 survey points per project */

               if c_survey_point.distance > arithmetic_mean + sigma_multiplier*standard_deviation
then
                  /* disable survey_points */
                  count_excluded_survey_points = count_excluded_survey_points + 1;
                  update survey_point set exclude = 'Y' where oid = c_survey_point.oid;
                  insert into error_messages values (0, 'project:
'||c_survey_project.source_oid||' ignored :  '||c_survey_point.distance);
               else
                  /* enable survey_points */
                  update survey_point set exclude = 'N' where oid = c_survey_point.oid;
               end if;
            else
            /* deal with 5 or less survey points per project */
                if c_survey_point.distance > arithmetic_mean + standard_deviation then
                  /* disable survey_points */
                  count_excluded_survey_points = count_excluded_survey_points + 1;
                  update survey_point set exclude = 'Y' where oid = c_survey_point.oid;
                  insert into error_messages values (0, 'project:
'||c_survey_project.source_oid||' ignored :  '||c_survey_point.distance);
               else
                  /* enable survey_points */
                  update survey_point set exclude = 'N' where oid = c_survey_point.oid;
               end if;
            end if;
         else
            /* enable survey_points */
            update survey_point set exclude = 'N' where oid = c_survey_point.oid;
         end if;
```

```
      END LOOP; -- c_survey_point

   END LOOP; -- c_survey_project

   readyText = ' ready; max_distance='||max_distance
      ||', max_arithmetic_mean ='||max_arithmetic_mean
      ||', sigma_multiplier ='||sigma_multiplier
      ||', excluded for max_distance='||excluded_max_distance_survey_points
      ||', excluded for sigma_multiplier='||count_excluded_survey_points;
   insert into error_messages values (0, 'end script: '||to_char(now(), 'YYYY-MM-DD HH24-MI-
SS')||' ->'||readyText);
   return readyText;

END;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;
ALTER FUNCTION load_survey_point_analysis(max_distance numeric, max_arithmetic_mean numeric,
sigma_multiplier numeric) OWNER TO "GIMA";
```

Figure 104 - Example of PostGIS load function: load_survey_point_analysis()

Relevant Internet Pages (URL's)

The following URL's have been referred to in this report; all URL's were checked and available at May 30, 2008.

URL 1 Geographical Information Management and Applications (**GIMA**, http://www.msc-gima.nl)
URL 2 The Netherlands' Cadastre, Land Registry and Mapping Agency (**Kadaster**) (http://www.kadaster.nl)
URL 3 Geo-Database Management Centre (**GDMC**), the research and development centre for Geo-Information technology of the Delft University of Technology (http://www.gdmc.nl)
URL 4 International Institute for Geo-Information Science and Earth Observation (**ITC**, http://www.itc.nl)
URL 5 Open Geospatial Consortium (**OGC**, http://www.opengeospatial.org)
URL 6 The Object Management Group (**OMG**, http://www.omg.org)
URL 7 International Federation of Surveyors (**FIG**, http://www.fig.net)
URL 8 International Organization for Standardization (**ISO**, http://www.iso.org)
URL 9 The United Nations Human Settlements Programme (**UN-Habitat**, http://www.unhabitat.org)
URL 10 Infrastructure for Spatial Information in Europe (**INSPIRE**, http://inspire.jrc.it)
URL 11 Unified Modelling Language (**UML**) Resource page (OMG, http://www.uml.org)
URL 12 OMG: **Companies, committed to MDA** and their products (http://www.omg.org/mda/committed-products.htm)
URL 13 **Oracle** (http://www.oracle.com)
URL 14 **PostgreSQL/PostGIS** (http://www.postgresql.org & http://postgis.refractions.net)
URL 15 **.NET** Framework by Microsoft. (http://msdn2.microsoft.com/en-us/netframework/default.aspx)
URL 16 **Java** 2 Enterprise Edition by Sun Microsystems (http://java.sun.com/javaee)
URL 17 **XML** definition by World Wide Web Consortium (W3C) (http://www.w3.org/XML and http://www.w3.org/XML/Schema)
URL 18 UML & MDA tool **Enterprise Architect** (http://www.sparxsystems.com)
URL 19 **EA Add**-in Samples http://www.sparxsystems.com.au/EAUserGuide/index.html?availableresources.htm
URL 20 **Dresden OCL Toolkit** (http://dresden-ocl.sourceforge.net)
URL 21 **MOVE3** (http://www.grontmij.nl/site/nl-

nl/Diensten/GIS+en+ICT/Softwareproducten/MOVE3+-+English.htm)
URL 22 **FWTools** (http://fwtools.maptools.org/)
URL 23 Geospatial Data Abstraction Library (**GDAL**) (http://www.gdal.org)
URL 24 **OGR** Simple Feature Library (http://www.gdal.org/ogr)
URL 25 **uDig** (http://udig.refractions.net/)
URL 26 Snowflake **GML viewer** (http://www.snowflakesoftware.co.uk)
URL 27 Pitney Bowes **MapInfo** (http://www.mapinfo.com)
URL 28 The **Eclipse** open source community (http://www.eclipse.org)
URL 29 "Rijksdriehoeksmeting" (**RD**) and "National Ordnance Datum" (**NAP**) (http://www.rdnap.nl)
URL 30 MDA prototype details (http://www.rgi-otb.nl/geoinfoned/mda and http://www.janvanbennekom.nl/mscthesis.html)

References

AUGUSTINUS, C., LEMMEN, C. H. J. & VAN OOSTEROM, P. J. M. (2006) Social Tenure Domain Model - Requirements from the Perspective of Pro-Poor Land Management. *In: Proceeding of the 5th FIG regional conference for Africa : Promoting Land Administration and Good Governance, 8-11 March 2006, Accra, Ghana.*, pp. 1-52.

BRÄUER, M. (2007) Design and Prototypical Implementation of a Pivot Model as Exchange Format for Models and Metamodels in a QVT/OCL Development Environment. Großer Beleg (MSc Diploma-Thesis).

COCKCROFT, S. (1997) A Taxonomy of Spatial Data Integrity Constraints. *The Information Science Discussion Paper Series*.

DEMUTH, B., HUSSMANN, H. & KONERMANN, A. (2005) Generation of an {OCL} 2.0 Parser. *Proceedings of the MoDELS'05 Conference Workshop on Tool Support for OCL and Related Formalisms - Needs and Trends, Montego Bay, Jamaica, October 4, 2005},* EPFL Technical Report LGL-REPORT-2005-001**,** 38--52.

GROOTHEDDE, A., LEMMEN, C., VAN DER MOLEN, P. & VAN OOSTEROM, P. (2008) A standardized Land Administration Domain Model as part of the (Spatial) Information Infrastructure – (S)II (Chapter 9). *Creating Spatial Information Infrastructures - Towards the Spatial Semantic Web,* CRC Press, Taylor & Francis, Boca Raton**,** pp 129-150.

GYLSETH, S., DAVELAAR, S. & VAN KOOTEN, T. (2000a) Requirements Modeling Using Oracle Designer. *CDM Standards and Guidelines Library,* Volume 1.

GYLSETH, S., JELLEMA, L., MULLER, S., DAVELAAR, S. & VAN KOOTEN, T. (2000b) Design and Generation of Multi-Tier Web Applications. *CDM Standards and Guidelines Library,* Volume 2.

HEIDENREICH, F., WENDE, C. & DEMUTH, B. (2007) A Framework for Generating Query Language Code from OCL Invariants. *7th OCL Workshop at the UML/MoDELS Conferences*

HESPANHA, J., VAN BENNEKOM-MINNEMA, J., VAN OOSTEROM, P. & LEMMEN, C. (2008) The Model Driven Architecture approach applied to the Land Administration Domain Model version 1.1 - with focus on constraints specified in the Object Constraint Language. *FIG Working Week 2008 – 14-19 June, Stockholm, Sweden*.

INGVARSSON, T. M. (2005) CCDM and Open Source Applications in Context of Implementing Cadastre in Iceland.

ISO/IEC (2003) International Standard ISO/IEC 9075-2:2003 Information Technology -- Database Languages -- SQL -- Part 2:

Foundation (SQL/Foundation).

ISO/IEC (2006) International Standard ISO/IEC 13249-3 Information technology - Database languages - SQL multimedia and application packages - Part 3: Spatial.
ISO/TC211 (2003a) Final Draft International Standard ISO/FDIS 19115 Geographic information - Metadata.
ISO/TC211 (2003b) International Standard ISO19107 Geographic information - Spatial schema.
ISO/TC211 (2006) Draft International Standard ISO/DIS 19136 Geographic information - Geography Markup Language (GML).
ISO/TC211 (2007) International Standard ISO19111 Geographic information - Spatial referencing by coordinates.
ISO/TC211 (2008) ISO/NP 19152 Geographic information - Land Administration Domain Model (LADM). *(under development)*.
JACOBSON, I., BOOCH, G. & RUMBAUGH, J. (1999) *unified software development process UML,* Boston etc., Addison-Wesley.
KAUFMANN, J. & STEUDLER, D. (2001) *Cadastre 2014 : a vision for a future cadastral system,* S.l., International Federation of Surveyors (FIG).
LARSSON, G. (1991) *Land registration and cadastral systems : tools for land information and management,* New York, Longman Scientific & Technical.
LEE, Y.-H. (2005) Design of the Survey Record Management System (SRMS) to support LIS in South Korea. *MSc Thesis, ITC*.
LEMMEN, C., AUGUSTINUS, C., VAN OOSTEROM, P. & VAN DER MOLEN, P. (2007) The Social Tenure Domain Model – Design of a First Draft Model. *Proceedings FIG Working Week 2007, May, Hong Kong***,** 23 p.
LEMMEN, C. H. J. & VAN OOSTEROM, P. J. M. (2006) Version 1.0 of the FIG core cadastral domain model. *FIG 2006 : Proceedings of the conference : Shaping the change, XXIII FIG congress, Munich, Germany, 8-13 October 2006. Frederiksberg: International Federation of Surveyors (FIG), 2006.. .* 18 p.
LOUWSMA, J., ZLATANOVA, S., LAMMEREN, R. & VAN OOSTEROM, P. (2006) Specifying and Implementing Constraints in GIS - with Examples from a Geo-Virtual Reality System. *GeoInformatica,* Volume 10, 4**,** pp. 531-550.
OMG (2003) MDA Guide Version 1.0.1. omg/2003-06-01.
OMG (2005) MOF 2.0/XMI Mapping Specification, v2.1.
OMG (2006a) Meta Object Facility (MOF) Core Specification Version 2.0.
OMG (2006b) Object Constraint Language Version 2.0.
OMG (2007a) Unified Modeling Language: Infrastructure version 2.1.1.
OMG (2007b) Unified Modeling Language: Superstructure version 2.1.1.
OPEN GEOSPATIAL CONSORTIUM, I. (1999) OpenGIS Simple Features Specification For SQL. Technical Report Revision 1.1.
OPEN GEOSPATIAL CONSORTIUM, I. (2002) OpenGIS® Geography Markup Language (GML) Encoding Specification Version: 3.00.
OPEN GEOSPATIAL CONSORTIUM, I. (2006a) Geography Markup Language (GML) simple features profile, version 1.0.
OPEN GEOSPATIAL CONSORTIUM, I. (2006b) Observations and Measurements, version: 0.14.7. *Category: OpenGIS® Best Practices*.
OPEN GEOSPATIAL CONSORTIUM, I. (2006c) OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option, version 1.2.0.
OPEN GIS CONSORTIUM, I. (1999) OpenGIS Simple Features Specification For SQL. Technical Report Revision 1.1.
PINET, F., DUBOISSET, M. & SOULIGNAC, V. (2007) Using UML and OCL to maintain the consistency of spatial data in environmental information systems. *Environmental Modelling & Software,* 22**,** pp. 1217-1220.

PINET, F., KANG, M. & VIGIER, F. (2005) Spatial Constraint Modelling with a GIS Extension of UML and OCL: Application to Agricultural Information Systems. *Metainformatics***,** 160-178.

POLMAN, J. & SALZMANN, M. A. (1996) Handleiding voor de Technische Werkzaamheden van het Kadaster *(Manual for Technical Operations of the Kadaster, internal report).*

SPARXSYSTEMS (2007) Enterprise Architect Version 7.0 User Guide.

VAN BENNEKOM-MINNEMA, J. (2007) Selecting a project management and information system development method for a (geospatial) information system development project (unpublished paper).

VAN BUREN, J. (2006) Projectvoorstel: Registratie Kaart Kwaliteit (internal report).

VAN OOSTEROM, P. (2006) Constraints in Spatial Data Models, in a Dynamic Context. *J. Drummond, R. Billen, E. Joao and D. Forrest (Eds.); Dynamic and Mobile GIS: Investigating Changes in Space and Time,* **,** pp. 104-137.

VAN OOSTEROM, P., LEMMEN, C., INGVARSSON, T., VAN DER MOLEN, P., PLOEGER, H., QUAK, W., STOTER, J. & ZEVENBERGEN, J. (2006) The core cadastral domain model. *Computers, Environment and Urban Systems,* Volume 30**,** pp. 627-660.

WANG, F. & REINHARDT, W. (2007) Extending Geographic Data Modeling by Adopting Constraint Decision Table to Specify Spatial Integrity Constraints. *The European Information Society,* Leading the Way with Geo-Information Lecture Notes in Geoinformation and Carography, Springer.

WESTERIK, A. & KENSELAAR, F. (2004) Precisiekenmerk Kadastrale Kaart *(precision characteristic, internal report).*