Tool Support for Transformational Design of Spatial Databases

Vivian Tambo
March, 2008
Tool Support for Transformational Design of Spatial Databases

by

Vivian Tambo

Thesis submitted to the International Institute for Geo-information Science and Earth Observation in partial fulfilment of the requirements for the degree in Master of Science in Geoinformatics.

Degree Assessment Board

Thesis advisor       dr.ir. Rolf A. de By
                     dr. Javier M. Morales
Thesis examiners    prof.dr. Menno-Jan Kraak
                     dr.ir. Maurice van Keulen
Disclaimer

This document describes work undertaken as part of a programme of study at the International Institute for Geo-information Science and Earth Observation (ITC). All views and opinions expressed therein remain the sole responsibility of the author, and do not necessarily represent those of the institute.
Abstract

With the popularity of Geographic Information Systems, some research efforts have concentrated on producing spatial databases by transforming from conceptual schema into logical schema. This transformation process is achieved by a mapping formalism designed to represent perceptions of the real world to a different formalism designed to provide for data representation in a DBMS that best supports efficient techniques of data management and applications needs.

Relational and object-relational databases have developed these transformation techniques already, but they do not work with spatial constructs. Therefore, it is of great value to develop a transformation process for spatial databases that will work in the domain of methods and behaviour.

Before a transformational design in the spatial domain can be developed, a mechanism is needed to capture the spatial semantics of real world objects in such a way that a direct database implementation of a conceptual model can be achieved. This mechanism was developed in the research project Design of Spatial Templates for Spatial Database Management Systems in SDI context which is carried out by Helen Ghirmai Asfaha.

The main objective of this research is to build a semi-automatic design process for supporting transformational design of spatial databases, based on Design of Spatial Templates for Spatial Database Management Systems in SDI context. In order to achieve this aim a process for transformational design was developed, based on the transformation of conceptual descriptions into syntactic statements of a chosen DBMS. This was done by first, using resources provided by Enterprise Architect 7.0, and creating transformation templates for the spatial constructs. Secondly, an XML parser was implemented in Python, for the transformation of logical to physical schema, that gives as a result a text file with PostGIS DDL statements. PostgreSQL/PostGIS is the selected target DBMS in this thesis.

Keywords

Transformational design, conceptual data modelling, spatial databases, UML Profile, OCL
Contents

Abstract i

List of Tables vii

List of Figures ix

Acknowledgements xi

1 Introduction 1
   1.1 Motivation and problem statement . . . . . . . . . . . . . . . . . 1
      1.1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
      1.1.2 Research problem . . . . . . . . . . . . . . . . . . . . . . . 2
   1.2 Research identification . . . . . . . . . . . . . . . . . . . . . . . 2
      1.2.1 Research objectives . . . . . . . . . . . . . . . . . . . . . . 2
      1.2.2 Research questions . . . . . . . . . . . . . . . . . . . . . . 3
      1.2.3 Innovation aimed at . . . . . . . . . . . . . . . . . . . . . . 3
   1.3 Method adopted . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.4 Outline of thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2 Model Driven Architecture 7
   2.1 The OMG vision and process . . . . . . . . . . . . . . . . . . . . . 7
      2.1.1 The OMG vision . . . . . . . . . . . . . . . . . . . . . . . . 7
      2.1.2 Modelling systems . . . . . . . . . . . . . . . . . . . . . . . 7
      2.1.3 Modelling is evolutionary . . . . . . . . . . . . . . . . . . . 8
      2.1.4 The Object Management Group . . . . . . . . . . . . . . . . 8
   2.2 MDA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
      2.2.1 The basic concepts . . . . . . . . . . . . . . . . . . . . . . . 9
      2.2.2 MDA transformations . . . . . . . . . . . . . . . . . . . . . . 11
      2.2.3 MDA and standards . . . . . . . . . . . . . . . . . . . . . . . 15
      2.2.4 What OMG adopts . . . . . . . . . . . . . . . . . . . . . . . 16
      2.2.5 MDA applications . . . . . . . . . . . . . . . . . . . . . . . . 17
   2.3 Concluding remark . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3 From conceptual to logical design 21
   3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
      3.1.1 Data modelling . . . . . . . . . . . . . . . . . . . . . . . . . 22
      3.1.2 Thematic data structures . . . . . . . . . . . . . . . . . . . . 23
## Contents

3.1.3 Spatial data structures ........................................... 25  
3.1.4 Integrity constraints ............................................. 26  
3.2 From conceptual design to logical design .......................... 27  
  3.2.1 Architecture of the transformation process ..................... 27  
  3.2.2 Transformation rules ............................................ 28  
3.3 UML extensions for object-relational database design ............. 32  
  3.3.1 Previous concepts .............................................. 32  
  3.3.2 UML extensions ................................................ 33  
3.4 Using UML and OCL to maintain the consistency of spatial data  
  in information systems .............................................. 33  
  3.4.1 OCL Language description ..................................... 33  
  3.4.2 Integrating spatial functions in OCL ......................... 34  
3.5 Related work on transformations .................................... 34  
  3.5.1 Transformation of UML into models using concrete syntax  
  patterns ......................................................... 34  
  3.5.2 Transformation process for object-relational databases .... 36  
  3.5.3 Comparisons .................................................. 36  
3.6 Comparison of existing modelling tools ............................. 37  
  3.6.1 Modelling tools ............................................... 37  
  3.6.2 Problems of selected modelling tools ......................... 38  
  3.6.3 Limitations of using EA in transformational design .......... 38  
3.7 Concluding remark ................................................. 39  

4 Method design ....................................................... 41  
  4.1 Design aspects .................................................... 41  
  4.1.1 Design goal ................................................... 41  
  4.1.2 Input ......................................................... 41  
  4.2 Requirements gathering .......................................... 41  
  4.2.1 Enterprise Architect for data modelling ..................... 42  
  4.2.2 UML Profiles ................................................. 42  
  4.2.3 Data Definition Language .................................... 42  
  4.2.4 Adopted Open Standards .................................... 43  
  4.2.5 MDA design approaches ..................................... 44  
  4.3 Transformational design .......................................... 45  
  4.4 Workflow ......................................................... 46  
  4.4.1 From Conceptual to Logical Schema ......................... 47  
  4.4.2 From Logical to Physical Schema ............................. 48  
  4.5 Software requirements .......................................... 48  
  4.6 Summary ......................................................... 49  

5 Implementation of transformational design ........................ 51  
  5.1 Workflow implementation ........................................ 51  
  5.1.1 Data transfer network ....................................... 52  
  5.1.2 Writing transformations in EA ................................ 53  
  5.1.3 Parsing the XML file ........................................ 56  
  5.2 Semi-automatic tool embedded in EA ............................ 60  
  5.2.1 Modelling with EA .......................................... 60
6 Discussions, Conclusions and Recommendations

6.1 Discussion of the transformational design
   6.1.1 Research questions
   6.1.2 Advantages and practicalities of the semi-automatic transformation process
   6.1.3 Limitations and problems

6.2 Future improvements

6.3 Conclusions

Bibliography

A
   A.1 Modelling tools
   A.2 Standards summary
   A.3 Open Source Implementations
   A.4 Software requirements
   A.5 Spatial and non-spatial data types

B
   B.1 UML Notation Elements
   B.2 UML Spatial Data Profile

C
   C.1 Enterprise Architect
      C.1.1 Code Template
      C.1.2 XMI files
      C.1.3 UML Profiles

D
   D.1 Python code
      D.1.1 ParserDOM
      D.1.2 Data Structures

E
   E.1 PostgreSQL/PostGIS
      E.1.1 SPATIAL_REF_SYS table definition

F
   F.1 How to: Transformation User Guide
List of Tables

A.1 Comparison of existing transformation tools 76
A.2 Standards summary 77
A.3 Open Source Implementations 78
A.4 Software used in the transformational design 79
A.5 Correspondence of spatial an non-spatial data types 80
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Method for research project</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>MDA: Marking a model</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>MDA model schema</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Metamodel Mapping Transformation</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Diagram of object type with complex and simple attributes</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Diagram showing a relationship type linking two object types</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Multi-association relationship type</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Diagram of object types connected by Is-a link</td>
<td>26</td>
</tr>
<tr>
<td>3.5</td>
<td>Transformation of multi-association</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Transformation of Is-a links</td>
<td>29</td>
</tr>
<tr>
<td>3.7</td>
<td>Transformation of overlapping links</td>
<td>30</td>
</tr>
<tr>
<td>3.8</td>
<td>Spatial relationships between two simple regions. A simple region is a closed connected point set in a 2-dimensional space $\mathbb{R}^2$</td>
<td>34</td>
</tr>
<tr>
<td>3.9</td>
<td>Process within a single modelling tool</td>
<td>35</td>
</tr>
<tr>
<td>3.10</td>
<td>Features of transformation rules</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Method adopted and Transformation steps</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Transformation steps</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of data transfer paths</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>MDA Transforms in EA</td>
<td>54</td>
</tr>
<tr>
<td>5.3</td>
<td>Model Transformation with ITC-DDL Template in EA</td>
<td>55</td>
</tr>
<tr>
<td>5.4</td>
<td>Conceptual schema for Example Database</td>
<td>61</td>
</tr>
<tr>
<td>5.5</td>
<td>Logical schema for Example Database</td>
<td>62</td>
</tr>
<tr>
<td>5.6</td>
<td>Transformation design: an example</td>
<td>63</td>
</tr>
<tr>
<td>B.1</td>
<td>UML elements for conceptual modelling</td>
<td>81</td>
</tr>
<tr>
<td>B.2</td>
<td>UML Spatial Data Profile</td>
<td>82</td>
</tr>
<tr>
<td>C.1</td>
<td>Transformation Template for ITC-DDL in EA</td>
<td>84</td>
</tr>
<tr>
<td>C.2</td>
<td>XMI file structure</td>
<td>85</td>
</tr>
<tr>
<td>C.3</td>
<td>Example UML Profile in Enterprise Architect</td>
<td>86</td>
</tr>
</tbody>
</table>
“The important thing is not to stop questioning. Curiosity has its own reason for existing. One cannot help but be in awe when he contemplates the mysteries of eternity, of life, of the marvelous structure of reality. It is enough if one tries merely to comprehend a little of this mystery every day. Never lose a holy curiosity.” (Albert Einstein)

I thank God for every day of life that I have. I thank my dearest parents that always give wise advise. My brothers, and sisters that keep me smiling all the time.

I would like to express my thanks to my supervisors dr. Rolf de By, and dr. Javier Morales for their advise and support during this last months on my thesis. I appreciate your support in my studies, and also I appreciate your confidence in my academic ability.

Also, I would like to thank my international friends for the love and encouragement that I received from them. I will keep you all forever in my heart.

Vivian Tambo
Enschede, February 2008
Acknowledgements
Chapter 1

Introduction

1.1 Motivation and problem statement

1.1.1 Motivation

During the past decades, relational databases have been an important topic for researchers. With the development of object-oriented design approach, it can be seen that the diffusion of object-relational databases has increased because of its capability for supporting the data persistence required by present day applications. Object-relational databases offer better support than relational technology for storage and retrieval of complex data types and relationships. As a result, spatial databases that store complex data types and relationships, can be suitably implemented with object-relational databases [16].

Nevertheless, object-relational databases are not by themselves enough to support the extension to spatial databases. It is also necessary to provide methods that support in object-relational database development tasks. Those methods should incorporate the object-relational model, and consider problems like platform-migration, platform-independence and maintenance, among others.

These trends in software complexity are the motivation behind work on industrializing software development. In particular, research in the area of Model-Driven Engineering (MDE) that is primarily concerned with reducing the gap between problem and software implementation domains. This gap can be reduced through the use of technology that supports systematic transformations of problem statements to software implementations [5].

In the MDE vision of software development, models are the engine of development and software developers rely on computer-based techniques to transform those models into running systems. To achieve the MDE vision requires undertaking technical problems, which have been the focus of software engineering the last decades. This technical problems are related to the underlying implementation platforms. An implementation platform may consist of the combination of network computers, libraries of utility functions like graphical user interface, mathematical routines and others. For this reason, to develop MDE technology that automates a portion of software life cycle, needs solution to the different problem platforms [5].

The Object Management Group (OMG), which is an international, open
1.2 Research identification

membership, not-for-profit computer industry consortium, develops and maintains standards for developing complex distributed software systems. The OMG launched the Model-Driven Architecture (MDA) as a framework of MDE standards in 2001 [35].

MDA technology provides more easy integration of new implementation infrastructure into existing designs, generate portions of application-specific code, and other implementation infrastructure artifacts from models. In other words, MDA synchronizes the evolution of models and their implementations as the software evolves [19].

Finally, some researches for combining object-relational databases and MDA technology have been carried out in the last years. A major goal of this researches is to propose a transformation process from conceptual into logical schema, and then to physical schema, using MDA which supports and provides methods that allow object-relational database development tasks.

1.1.2 Research problem

With the popularity of Geographic Information Systems, some research efforts have concentrated on producing spatial databases from conceptual schema into logical schema. This transformation process is achieved by a mapping formalism designed to represent perceptions of the real world to a different formalism designed to provide for data representation that best supports efficient techniques of data management [16].

Relational and object-relational databases have developed these transformation techniques already, but they have not been adopted in the spatial application domain very well. Therefore, it is of great value to develop a transformation process for spatial databases that will work in the domain of methods and behaviour.

1.2 Research identification

In this section, we describe the overall objectives, the research questions to be answered, the innovation aimed at, and an overview of related works.

1.2.1 Research objectives

Main objective

This research aims to build an automatic or semi-automatic design process for supporting transformation design of spatial databases.

Sub-objectives

- To define the conceptual and logical schema, using MDA,
- To define the set that describes the transformation rules,
- To build the transformation process for the particular conceptual schema and the syntax of the target database,
To identify and perform verification and validation techniques for the transformation process.

1.2.2 Research questions

To achieve the above-mentioned research objectives, the following questions need to be solved:

- How to define the conceptual schema?
- How to define the logical schema?
- How to define the set that describes the transformation rules?
- How to build the transformations in a product implementation and combination: Enterprise Architect and PostgreSQL/PostGIS?
- How to verify and validate the transformation process?

1.2.3 Innovation aimed at

The innovation aimed at is the semi-automatic design transformation of spatial database designs using MDA.

1.3 Method adopted

The philosophy of work will be based on the software engineering phases for system development as discussed in [9], combined with the object-relational database design method proposed by [17].

In general, the method proposes six phases: problem definition, analysis, design, implementation, verification and validation, and documentation. The main steps of the method are the analysis, design, implementation, and verification and validation phases, which are an important part of the development life cycle for object-relational database design.

During the analysis phase, the conceptual schema is represented by UML class diagrams. Since UML is the standard language for object-oriented system design, it allows to design the entire system facilitating the integration between different system platforms.

The design phase is divided into two steps:

- Conceptual design, which provides an object-relational specification independent of implementation platform, and
- Logical design, which is the design using an implementation platform, though with disregard of performance and other more physical system characteristics.

The implementation phase includes the physical design. In this phase, the design obtained from a previous step should be optimized according to the needs and requirements, improving response time and storage capabilities. In the
1.3. Method adopted

current, research such activities will not be performed because are not part of the research topic.

Finally, in the validation phase is determined whether the final software product complies with the needs and requirements for which it is intended. The verification checks the consistency of the final software product, whether the software product adheres to standards, uses reliable techniques, and performs its functions in the correct way.

The proposed method for the research project is shown in Figure 1.1 below.

![Diagram](image)

Figure 1.1: Method for research project. Source: (9) (17)
1.4 Outline of thesis

This section describes the main steps that the project will go through chronologically:

1. Requirements definition
   This step can be called the problem definition phase. The problem needs and requirements are analyzed, and software and hardware requirements are produced.

2. Transformation process design
   In this step, we define the transformation process from conceptual schema into logical schema. The step includes two important activities:
   - The definition of conceptual and logical schema, and
   - A set of transformation rules are produced.

3. Product implementation
   The implementation step includes the programming phase. Once the analysis and design phases of the transformation process are completed, the process can be implemented. The result of this step is a final software product.

4. Verification and validation
   The verification and validation are divided into two steps:
   - The identification of verification and validation techniques, and
   - The verification and validation of the transformation process.

   The results of this step are:
   - A final software product that meets requirements and specifications, and
   - A final software product that produces correct outputs.

5. Integration
   This step is intended for the combination with the research project “Design of Spatial Templates for Spatial Database Management Systems in SDI context” which is carried out by Helen Ghirmai Asfaha.

   The literature review and software learning as well as documentation are recurrent tasks carried out along the entire research project.

   According to the method design, this research is organized into 6 chapters as below:

   Chapter 1 briefly discusses motivation for the work, identify the problem, state the research objectives and questions, and describe the method adopted.
1.4. Outline of thesis

**Chapter 2** analyzes the MDA transformation schema, standards and its advantages of application.

**Chapter 3** is the basic theoretical elements of this thesis. It provides a base for the method adopted and its development and implementation.

**Chapter 4** is the method design part. Based on the MDA transformation process, it deals with some aspects of the transformation templates including the UML elements for conceptual modelling. How to modify the transformation templates, how to combine the MDG file and transformation templates. Finally, works out a workflow for the transformation process.

**Chapter 5** is the implementation part. It realizes the transformation design based on the workflow by developing algorithms and programs. In the meantime, a conceptual model has been designed as a test input, based on which the complete workflow is demostrated. At the end, a semi-automatic process for transformational design is implemented.

**Chapter 6** discusses the practical aspects, existing problems and future improvements of the transformation process. At the end, a summary of the innovation aimed at, and achievements of this research and conclusion of thesis.
Chapter 2

Model Driven Architecture

This chapter introduces the Model Driven Architecture. The knowledge to be gained from this chapter will provide an understanding of the development of the transformation process.

2.1 The OMG vision and process

2.1.1 The OMG vision

Nowadays, organizations realize that their information infrastructure is effectively a distributed computing system. To integrate information resources and use information efficiently, it must be accessible across the company, and more importantly across service suppliers and customers, also this means that computers must be linked to the networks of the world and be capable of sending and receiving information. In other words, computers must be global information tools capable of connecting to a world of information at the application level [19].

Despite the knowledge that every application should be built to be integrated and updated later, most software developers ignore these facts and build only the available specifications. Currently, most software is developed without taking into account the reality of changes in infrastructure and requirements, and most importantly the arrival of new technology. Not only the applications that automate the business processes, but also the data that is collected and organized by those applications ignore these requirements. In general, data integration and application interoperability, continue to be beyond our scope of complete mastery [19].

2.1.2 Modelling systems

Just as in distributed computing system development, the interest in modelling data and applications has increased significantly. In fact, the first step in building software should be to make a clean design work. This means that design not only leads to systems that are easier to develop, integrate and maintain, but also to the ability to automate at least some of the construction process [19].
The Model Driven Architecture (MDA), is based on the idea of separating the specification of a system from the details of implementation. It enables the definition of machine-readable application and data models, which allow flexibility in the software development phases of implementation, integration, testing and maintenance [12].

2.1.3 Modelling is evolutionary

MDA is another evolutionary step in the development of software, and the advantage of software automation from models is truly another level of compilation [35]. In 1954, IBM launched the first high-level programming language known as “FORmula TRANSlating system” (FORTRAN). FORTRAN was designed to simplify the development of software for the IBM machines [19].

Initial resistance to FORTRAN arose from developers that thought they could write more efficient code by hand than code written by a FORTRAN compiler. Since then, the world of programming has opened up to a much larger audience of potential specialists [19].

2.1.4 The Object Management Group

The Object Management Group (OMG) was formed to reduce complexity and costs in the introduction of new software applications in general. The OMG is accomplishing this goal by the introduction of the MDA architectural framework that supports detailed specifications. These specifications will lead the software industry towards interoperable, reusable, portable software components and data models based on standard patterns [35].

The OMG is an international trade association incorporated as a nonprofit corporation in the United States, with affiliate organizations around the world. The OMG receives funding from its diverse membership of hundreds of corporations, universities and standards organizations [35].

The OMG develops enterprise integration standards, including modelling standards as the Unified Modelling Language (UML) and MDA, that allow visual design, implementation and maintenance of software and other processes [35].

2.2 MDA

Starting in 1995, OMG began to adopt industry-specific technology specifications and object modelling. This resulted in the adoption of the Unified Modelling Language (UML). OMG members then began to use UML in technology specification for OMG adoption [19]. In this expanding task, in 2001 OMG adopted a second framework, the Model Driven Architecture (MDA) [19].

At the moment of MDA conception, the idea was to separate the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. As a consequence, the main focus of MDA was to enable definition of machine-readable application and data models to allow flexibility in the software industry development [35].
According to [19], MDA provides an approach for:

- specifying a system independently of the platform that supports it,
- specifying platforms,
- choosing a particular platform for the system, and
- transforming the system specification into one for a particular platform.

Currently, the three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns [19].

### 2.2.1 The basic concepts

According to [19], this section defines a number of basic concepts that are at the core of MDA.

**System**

MDA concepts are expressed in terms of some existing or planned system. That system may include software, a single or combined operating systems, people, and an enterprise or a federation of enterprises.

**Application**

In the following sections, the term “application” is used to refer to functionality of a systems under development.

**Platform**

A platform is a set of systems and techniques that provide a coherent functionality through interfaces and specific patterns. Such elements can be used for any application without concern for details of how the functionality provided by the platform is implemented.

**Model**

A model of a system is a specification of that system and its environment for a certain purpose, and can be represented as a combination of drawings and text. The text can be expressions in a modelling or natural language.

**Metamodel**

A metamodel describes properties of a specific platform. A model is an instance of a metamodel. The notion of instance of a metamodel is itself at least partially intuitive.
2.2. MDA

Class
A description of a group of objects that share the same attributes, operations, relationship types, and semantics.

Metaclass
A class whose instances are classes.

Model Driven
MDA is an approach to system development, and is model-driven because it provides a way for using models towards analysis, design, construction, deployment, operation, maintenance and modification.

Architecture
The architecture of a system is a specification of its parts and connectors, as well as the rules of interaction of the parts using the connectors. The MDA provides certain kinds of model, how those models may be prepared for their use, and the relationships of the different kinds of model.

Viewpoint
A viewpoint on a system is a technique for suppressing selected details to establish a simplified perspective of that system. This can be achieved by using a set of concepts and rules, to focus on particular concerns within that system. The concepts and rules may be considered to form a viewpoint language. As an example, MDA specifies three viewpoints on a system, a computation-independent, a platform-independent and a platform-specific viewpoint.

View
A view of a system is a representation of that system from the perspective of a selected viewpoint.

Platform Independence
Platform independence is the quality that the model is independent of the features of a specific software/hardware platform.

MDA viewpoints
This section describes three MDA viewpoints according to [19]:

- A Computation-Independent Model (CIM) focuses on the environment and requirements of the system, while the details of the structure and functionality are hidden or are left undetermined.
A Platform-Independent Model (PIM) focuses on the operation of a system while hiding the necessary details for a particular platform of choice.

A Platform-Specific Model (PSM) combines the platform-independent viewpoint with an additional focus on the detail of the use of the specific, chosen platform.

### 2.2.2 MDA transformations

This section discusses the MDA transformations on the basis of the literature found in [19].

**Mapping**

An MDA mapping provides specifications for transformation of a PIM into a PSM for a particular platform. The platform model will determine the nature of the mapping. Some mapping techniques are described as follows [19]:

1. A Model Type Mapping specifies a mapping from model using types specified in the PIM language to models expressed using types in a PSM language. In other words, a PIM is prepared using a platform-independent modelling language.

2. A Model Instance Mapping will define marks. A mark represents a concept in the PSM, and is applied to an element of the PIM, to indicate how that element is to be transformed. The marks, being platform-specific, are not part of the platform-independent model.

3. A Combination of Type and Instance Mapping is necessary since a model type mapping is only capable of expressing transformations in terms of rules for things of one type in the PIM, resulting in the generation of some other things of one or more types in the PSM. Nevertheless, without the ability to also mark the model with additional information for the transformation process, the mapping will be deterministic, and will rely completely on platform-independent information to generate the PSM.

4. Marking Models may specify quality of service requirements on the implementation. This means, instead of indicating the target of a transformation, a mark may provide a requirement on the target. The transformation will then choose a target appropriate to that requirement. Marks may need to be structured and constrained. A set of marks, instead of being supplied by a mapping, may be specified by a mark model, which is independent of any particular mapping.

5. Templates may also be included in a mapping. Templates are parameterized designs that specify particular kinds of transformations. These templates are like design patterns to guide the transformation. A set of marks can be associated with a template to indicate instances in a model which should be transformed according to the template.
6. A Mapping Language is used to describe a transformation of one model to another. The description may be in natural language, as algorithm, or in a model mapping language. Model mapping languages are an area for MDA technology adoption.

**Marking a model**

In model instance mapping, the elements of the PIM are marked to indicate the mappings to be used to transform that PIM into a PSM [19].

In the simple case, a PIM element is marked once, indicating that one specific mapping is to be used to transform that element into one or more elements in the PSM. In the general case, several PIM elements are marked to indicate their roles in some mapping, and this mapping is then used to transform those PIM elements into some set of PSM elements [19].

In Figure 2.1, we describe an example for marking a model.

![Figure 2.1: Marking a model. Source: (19)](image)

The example in Figure 2.1 shows the process for marking a PIM. First, a particular platform is chosen, and a mapping technique for this platform is available. The mapping includes a set of Marks. Those Marks are used to indicate elements of the PIM to guide the transformation process itself. The marked PIM is further transformed, using the mapping technique and the set of Marks, to produce the PSM.

**Transformation**

The next step is to take the marked PIM and transform it into a PSM. This can be done in a manual, semi-automatic, or automatic way. Model transformation is the process of converting one model to another model of the same system. The input for the transformation is the marked PIM and the mapping. The output is the PSM and the transformation log [19].
Using model type mapping, transformation takes any PIM for mapping and produces a PSM. Using model instance mapping, transformation takes a marked PIM for mapping as indicated by the marks, and produces a PSM [19].

**Direct Transformation to Code**

Specific software might transform a PIM directly to deliverable code, without producing a PSM [19].

**Transformation log**

The output of transforming a PIM using a particular technique is a PSM and a transformation log. The transformation log includes a map from the element of the PIM to the corresponding elements of the PSM, and shows which parts of the mapping were used for each part of the transformation [19].

**PSM**

The PSM produced by the transformation is a model of the same system specified by the PIM, and also specifies how that system makes use of the selected platform [19].

A PSM may provide more or less detail, depending on its purpose. A PSM will be an implementation, or may be used for further refinement to a PSM that can be directly implemented [19].

A PSM that is an implementation will provide all the information needed to construct a system and to put it into operation. This means a variety of information, which may include program code, program linking and loading specifications, deployment descriptors, and other forms of configuration specifications [19].

**Degrees and Methods of Model Transformation**

Currently, there is software for supporting model transformation. Besides, there are different approaches to transform from PIM to PSM [19]. The following transformation approaches illustrate the set of possibilities:

**Manual transformation** In order to apply the transformation from PIM to PSM, some design decisions must be taken during the process of developing a schema that conforms to engineering requirements on the implementation. Manual transformation is useful since decisions are considered and taken in the context of a specific implementation design. The MDA approach adds value with a explicit distinction between a PIM and the transformed PSM, and a transformation log.

**Transformation of a PIM using profiles** A PIM may be transformed using a platform-independent UML profile. The transformation may involve marking the PIM using marks provided with the platform-specific profile. Transformation rules may be specified using operations, and the specification contained in the UML profile.
2.2. MDA

**Transformation using patterns and marks** Patterns may be used in the specification of a mapping. The mapping of a PIM includes a pattern and marks corresponding to some elements of that pattern. The elements with marks are transformed according to the pattern to produce the PSM.

**Automatic Transformation** There are extensions in which a PIM can provide all the information needed for implementation. As a consequence there is no need to add marks or use data from additional profiles, to generate code. In this case, there is no necessity to add additional information to the PIM, the software interprets the PIM and transforms to program code.

**Inputs for transformation**

According to [19], a number of initial requirement gatherings for transformation is required. Some of them are described in the section below.

**Patterns** Patterns are also important in the description of a group of concepts in one schema that correspond to a concept, or a group of concepts in another schema when specifying a type-based transformation. Software will be responsible for matching the patterns in the source model and using the patterns in the target model as templates for creating the new model.

**Technical choices** Technical choices are necessary to guide the transformation process. Technical choices might also be made by analyzing the software that will work with the PIM, and then used in manual or automatic transformation. Currently, most approaches use a combination of some automated transformation with manual input to the transformation.

**Quality requirements** A set of quality requirements can be used to guide the transformations.

**MDA model schema**

In the MDA model schema, models are instances of metamodels. A metamodel makes it possible to describe properties of a particular platform. In this case, the models that are instances of such a metamodel are said to be platform-specific with regard to this platform [35].

Models and their metamodels may have a semantic relation to one another, this means they describe the same system for the same platform at different levels of abstraction. A mapping between models is assumed to take one or more models as its input and produce one output model. The rules for the transformation that is performed by a mapping are described in a mapping technique [35].

The MDA model schema is shown in Figure 2.2 below.
2.2.3 MDA and standards

OMG has adopted a number of techniques, which together enable the model-driven approach. These include UML, MOF, specific standards, and UML profiles [35].

**UML**

The Unified Modelling Language (UML) is a standard modelling language for visualizing, specifying, and documenting software systems, and it is based on the object-oriented paradigm. UML is used for facilitating the communication between developers, since it is a common language for expressing design elements. UML allows to specify the structure and/or behaviour of a system, as well as documenting the important design decisions. Models used with MDA can be expressed using the UML language. UML 2.0 integrates a set of concepts for completely specifying the behaviour of objects, the UML action semantics [22].

**MOF**

The Meta-Object Facility (MOF) provides a model repository that can be used to specify and manipulate models, thus encouraging consistency in manipulating models in all phases of the use of MDA [35].

**Profiles**

Profiles are a UML extension mechanism. A profile applies to a language specification, specifying a new modelling language by adding new language elements or further restrictions to the language [22].
2.2.4 What OMG adopts

OMG adopts specifications that are expressed as models that exploit the MDA patterns to enable portability, interoperability and reusability [19]. According to [19], MDA standard specifications fall into one of the following categories:

Service Specifications For service specifications the term platform, usually refers to middleware, so platform-independent means independent of middleware, and platform-specific means specific to a particular middleware platform.

Data Model Specifications In data model specification, a PIM is independent of a particular data representation, after mapping the derived PSM corresponds to a particular data representation (DBMS).

Language Specification In language specifications, the abstract syntax of the language is specified as a MOF-compliant metamodel.

Mapping Specifications Such specifications contain one or more transformation models and text correspondence descriptions.

Network Protocol Specifications In network protocol specifications, protocols are specified with an appropriate PIM/PSM separation. Such specifications may include the protocol data elements and sequences of interactions as appropriate.

Domain models

The many OMG domain technology adoptions each have an implicit model. This is partly expressed in the IDL specification of the technology. Use of the interfaces depends on an understanding of the implicit model. Domain technologies adopted in the future can be expected to provide explicit platform-independent models [19].

Platform-independent models

OMG and vendors will prepare generic platform-independent models, which will form a library of reusable PIMs [19].

Platform models

MDA platform models may be in the form of UML models, and may be made available in MOF-compliant repositories as UML models, MOF models, or models in extended UML, or other languages specified using the MOF model [19].

UML Family Languages

Extensions to the UML language will be standardized for specific purposes. Many of these will be designed specifically for use in MDA [19].
Chapter 2. Model Driven Architecture

Model and metamodel mappings
Technology adoptions are needed to provide the capability to specify model transformation mapping in complete detail. These are likely to be incorporated into a future version of UML [19].

Conformance testing
To support the MDA, the OMG will also concentrate extra effort on conformance testing and certification of products. In many cases, these efforts will depend on strong relationships with outside organizations with relevant expertise [19].

2.2.5 MDA applications
This section describes other capabilities of MDA according to [19].

Multi-Platform Models
Many systems are built on more than one platform. An MDA transformation can use marks from several different platform models to transform a PIM into a PSM with parts of the system on several different platforms.

Federated Systems
A PIM can specify a system, with several parts, each part under separate control. The transformation of that PIM to a PSM can be made, recognizing that the system is federated.

This approach requires the identification of generic bridges between the platforms. The use of PIMs for specifying the whole system provides generation software with some, or most, of the information needed to perform specific bridging, as long as a generic interoperability mechanism is available. Currently, no standard solutions exist in to this approach.

Multiple applications of the MDA pattern
The MDA pattern includes a PIM, a class of platforms, and a PSM. The PSM is specific to that class of platforms. The PIM is not dependent on any particular platform of that class. The MDA pattern can be applied several times in succession to the PIM. What is a PSM resulting from one application of the pattern, will be a PIM in the next application.

General model to model transformations
The same approaches that enable transformation of a PIM into a PSM can be used to transform any model into another.

The general model to model transformation is shown in Figure 2.3 below.

The figure illustrates the general case of a metamodel mapping transformation. The figure uses a metamodel mapping to illustrate the point. Any of
2.2. MDA

Combination

Combination uses two or more mappings to create a new mapping. The characteristics of the new mapping are determined by the mappings themselves, and by the way they are combined. The effect of the application of a combined mapping is the corresponding combination of the effects of the original mappings.

Ways in which mappings may be combined include sequential and concurrent combinations.

Enabling interoperability

An interoperability mapping uses mappings for two different platforms. These are combined to create a mapping to transform a PIM into a PSM in which some objects are on one platform and others on the second. This mapping is extended to include connectors between the two platforms and specifications for the use of these connectors in a transformation. The resulting mapping is used to transform a PIM into a PSM of a system that makes use of both platforms and provides for the interoperability of the systems on the different platforms.

Nowadays, MDA-based development software is available, and enterprises everywhere around the world are starting their application development by building a PIM instead of writing code.

MDA is in use for sharing valuable design expertise and delivering mission-critical software applications. MDA is used in virtual reality environments, military mission software, communication, and business solutions, among others.
2.3 Concluding remark

In those days of Internet and corporate intranets, companies need a computing infrastructure that allows to choose the best computer for each business purpose, not only with machines within their company, but also with their suppliers and customers.

In the ideal case, computing should be affordable for all companies, without any barriers of operating system, software, hardware, or network compatibility. The new software development, the diversity of applications will never be achieved by settling all this software development to a single operating system, programming language, application framework, or any other single choice. Currently, there exist many platforms, and many and diverse implementation requirements, and it is difficult to agree on a single option in any of these choices. Thus, it is important to coexist by translation, that means by agreeing on schemas and on how to translate between them.

The OMG specification suits this requirement and enables interoperability regardless of operating system, platform, programming language, network hardware and software. Even though the architectural framework of the OMG has changed in recent years, the primary goals of interoperability and portability have not changed. The vision of integrated systems, and applications that can be developed, maintained and integrated with less cost is still within its scope.
Chapter 3

From conceptual to logical design

3.1 Introduction

Object-relational databases offer better support for storage and retrieval of complex data types and relationships. Nevertheless, object-relational databases are not by themselves enough to support the extension of spatial databases because of technical problems like platform-migration, platform-independence and maintenance, among others. By using MDA approach, those problems can be solved since MDA provides methods that support in object-relational database development tasks [16].

Nowadays, there are two types of products available for spatial information systems. One is the traditional database management systems (DBMS) that have been extended to include a partial support for spatial data. Examples are PostGIS, the spatial extension of PostgreSQL, and Oracle. In general, those extensions do not follow a conceptual modelling approach, and their major advantage is that they can handle with the same software the alphanumeric and spatial data simultaneously. On the other hand, there are Geographical Information Systems (GISs), which are dedicated software packages for geographic data management [25].

In order to fill the gap between functionality supported by commercial software and desirable functionality from the user application perspective, software companies have been developing Computer Aided Software Engineering (CASE) tools for conceptual modelling. The term conceptual refers to modelling approaches that use concepts that can be easily understood by user applications and ignore the technical issues of how data is handled by computer systems [5].

To design databases and information systems, conceptual modelling has been adopted by CASE tools because of its user-oriented nature. Currently, UML is a standard for object-oriented design methodology. One advantage is to provide a visual representation of the data structure, allowing the users to define a database schema by using a conceptual schema. The conceptual schema is automatically transformed by the tool into its corresponding logical schema for the targeted DBMS [25].
3.1. Introduction

3.1.1 Data modelling

Data modelling is part of database design, and it is related to the analysis of data objects and their relationships to other data objects. Besides, data modelling implies a transformation process between different abstraction levels. The terms “conceptual”, “logical”, and “physical” are frequently used in data modelling to differentiate levels of abstraction versus detail in the database design [5].

Conceptual design

A data model is conceptual if it enables a direct mapping between the perceived real world and its representation with the requirements of the database design. The representation is handled by a careful analysis of application user requirements [25].

Once the requirements are established, the design of the conceptual schema consist of a rewriting of the natural language specifications into the formal description language associated with the data model. A conceptual data model is free from implementation considerations [25].

According to [29], a conceptual design includes:

- The important entities and the relationship types among them
- No attribute specifications
- No primary key specifications

At this abstraction level, the design attempts to point out the highest-level relationships among the different entities. A conceptual model may include a few significant attributes to enhance the definition of entities. Besides, it may have some candidate keys but they do not represent entity identifiers, since identifiers are logical choices made with deeper knowledge of the context [25].

Logical design

While the goal of conceptual design is defining a schema that describes user requirements about the database design, the goal of the logical design is to define a corresponding schema that conforms to data definition rules of some specific DBMS [25].

Features of logical design include [29]:

- Inclusion of all entity and relationship types among them
- All attributes for each entity type are specified
- The primary key for each entity type is specified
- Foreign keys are specified,
- Normalization occurs at this level.

At this abstraction level, the schema attempts to describe the data in more detail, without regard of how they will be physically implemented.
Physical design

A physical schema is a single logical schema instantiated in a specific database management product (e.g., PostgreSQL/PostGIS, Oracle). The physical schema specifies implementation details which may be features of a DBMS, as well as configuration choices for that database instance [29].

Features of the physical schema include [29]:

- Specification of tables and columns, indexes, partitioning, and clustering files
- Foreign keys are used to identify relationship types between tables
- De-normalization may occur based on user requirements
- Performance considerations or constraints may cause the physical schema to be quite different from the logical schema

At this abstraction level, the schema shows how the logical schema is implemented in a specific DBMS.

3.1.2 Thematic data structures

This term refers to data structures designed to hold traditional administrative alphanumeric data. The word thematic is used to distinguish the application semantic data from the associated spatial data. A number of concepts will be defined in this section like object types, attributes, methods and relationship types on the basis of [25].

Object types

In database applications we describe, denote, relate and manipulate objects with complex information structure. Such objects represent the real-world entity of interest to the application, where an entity is something that exists as a single separate object.

Representation is the outcome of an abstraction process, followed by a classification process. The classes that will result from the classification process are described as object types in a database schema. The term property refers to the attributes and methods of the class.

In general, object types hold a unique name, attributes, and methods. Instances of object types hold an object identifier and a value.

Attributes

Attributes are of two types: simple or complex. A simple attribute is one holding a plain value. A complex attribute conveys a structuring concept that means a name that denotes a set of attributes that are either themselves complex or simple. An object type supporting complex attributes is called a complex object type. Complex object types are supported by object-relational DBMS, but are
3.1. Introduction

not supported by relational DBMS. Object-relational DBMS support constructs that may be used to describe complex attributes.

A diagram of an object type with complex and simple attributes is shown in Figure 3.1 below.

Methods

While attributes convey descriptive information, methods attached to object types, specify the operations that are specifically defined for the object type. A method is specified by defining its name, and its body, which will be executed when the method is required.

Relationship types

Objects are interrelated by relationships that provide paths to complementary information about the object. Relationships in the database represent real-world links that are of interest to the application. Definition of relationship types goes through the same process of perception, abstraction, and classification that is used to identify relevant object types from the observation of the real world of relevance to the application. The outcome of this process is a set of relationship types, where each relationship type conveys a link between two or more object types. There are two kinds of relationship type: association and multi-association.

Association relationships An association relationship type is a relationship type that links two or more object instances without imposing any specific semantics on the link. Associations are non-directed links. They have at least two roles. The only constraint they impose is that for each role the corresponding object type contributes one instance to each instance of the association relationship.

Instances of association relationship types hold a record identifier (rid), a value, and for each role and instance of the linked object type. Keys, composed of attributes and roles, may be defined to serve as an application-based identifier.
Additional constraints are usually stated, such as cardinality constraints that characterize each role of the relationship type. For each role its minimum and maximum cardinalities are defined. These cardinalities define the number of relationship instances that, taken at some arbitrary instant in the life of the database, may link an instance of the object type linked by the role.

A diagram illustrating a relationship type linking two object types is shown in Figure 3.2 below.

![Figure 3.2: Diagram showing a relationship type linking two object types. Source: (25)](image)

**Multi-association relationships** A multi-association relationship type is a relationship type that links, for each role, a non-empty set of instances of the linked object type. In other words, multi-association relationships link group of objects, instead of single objects.

Instances of multi-association relationship types hold a rid, a value, and for each role a collection of instances of the linked object type.

An example of multi-association relationship type is shown in Figure 3.3 below.

![Figure 3.3: Multi-association relationship type. Source: (25)](image)

**Is-a link**

Is-a links are different from relationship types. Relationships usually link two database objects representing different real-world entities, they have an identity and can bear properties. Instead, is-a links tie two instances that are two different representations of the same real-world entity, thus producing a very specific semantics. They have no identity and do not produce properties. Moreover, is-links, because their semantics is that of a classification refinement, are directed links and by definition produce a population inclusion constraint enforcing that every object instantiated in the subtype is also instantiated in the supertype.

A diagram illustrating object types connected by Is-a link is shown in Figure 3.4 below.

**3.1.3 Spatial data structures**

A traditional database schema can work as spatial database schema by including the description of the spatial properties of the real world phenomena that are being represented [25].
3.1. Introduction

The characteristic of spatial databases systems is that they allow positioning objects in space. To position an object means specifying the location that the object occupies with respect to a given framework. Such framework contains all the possible places for an object. In GIS words, the place is known as object extent, defined as the set of points that the objects occupies in space. Extent in space maybe 0-dimensional (points), 1 dimensional (lines), 2-dimensional (surfaces), or 3-dimensional (volumes) [25].

The definition of spatial extents depends on the availability of appropriate value domains. In order to provide an efficient manipulation and query of spatial extents, specific spatial data types have to be available. Without such spatial types a simple query would be difficult to express [25].

The spatial data types come with associated operations and predicates that could be used in writing queries and manipulations. Such spatial operations and predicates concern spatial values [25].

3.1.4 Integrity constraints

Integrity constraints provide a way to define the semantics of data, and establish the quality of a database. Integrity constraints are restrictions for the data that is held in the database, in order to prevent the insertion of data that are obviously incorrect with respect to rules governing the real world and its representation in the database. Different sorts of restrictions can be specified. The simplest form of integrity constraints are value domains. Typical examples include the specification of a limited range of allowed values over the underlying domain, or the explicit enumeration of allowed values. These restrictions are intended to limit the possibility of inserting erroneous, or generating incorrect data by some inappropriate update of existing data [25].

A special form of integrity constraint is the definition of derived attributes. The values of derived attributes are automatically computed by the system according to the derivation formula specified in the description of the attribute. The derivation formula maybe of arbitrary complexity. Several DBMS support the definition of these formulas using the data manipulation language associated with their data model [25]. Before working with spatial data in a database, new kinds of integrity constraint must be embedded in the data model to cope with the spatial characteristic.
Spatial constraints

Spatial databases are familiar with the concepts of partition, covering, connectedness, and containment which are examples of typical spatial constraints. So it makes sense, to embed such constraints into a spatial model. In order to reach this goal it is necessary to reuse the same concepts giving them a spatial connotation, and associating them with aggregation relationships between spatial objects types, rather than with is-a links. By spatial connotation we mean that the constraints do not apply to the set of object identifiers (oids) but to the set of spatial extents associated with the instances of the object types involved in the aggregation relationship [25].

Spatial constraints may also be associated with spatial object types. Consequently, there should be the possibility to associate with each spatial object type spatial constraints, like covering, partitioning, or connectedness. There is also the possibility to associate spatial constraints to spatial attributes [25].

Spatial constrains do not need to be embedded as data modelling constructs. They maybe expressed in a specific integrity constraint specification language [25].

3.2 From conceptual design to logical design

To produce from a conceptual schema a logical schema means a transformation process designed to represent perceptions of the real world to a different formalism designed to provide for data representation that best supports efficient techniques of data management [16]. Because of the gap between conceptual and logical schema, the main issue in a transformation process is semantic loss. Semantic loss means that parts of a specification from the initial schema is not presented anymore in the target schema. This specification has been lost in the transformation process, probably because the target schema has no defined concept that allows expressing the same concept [25].

3.2.1 Architecture of the transformation process

The systems architecture for the transformation process is therefore defined as composed of two types of modules as mentioned in [25]:

A transformation module, that performs generic transformations on the input schema. The outcome is a schema that is equivalent to the input schema but only has a limited set of modelling concepts. This result is achieved by applying a set of transformation rules that removes undesired concepts. The module has a library of transformation rules, such rules have to be applied depending on which concepts have to be removed and on the target DBMS.

A set of specialized modules, that depend on the target DBMS. The translation module corresponding to the target DBMS translates the schema produced by the transformation module into a schema expressed with the syntax of the target DMBS. This is basically a language translation process, in other words a rewriting of the specifications.
3.2.2 Transformation rules

In general, transformation rules address structural, spatial and multi-representation concepts. Removal of the use of selected concepts requires applying rules in a given sequence. In general, the order of applying transformation rules starts with the multi-representation, spatial, and structural rules at the end [25]. Such transformation rules are discussed in the section below.

Structural transformation rules

In this section, we discuss transformation rules for the structural dimension, such as multi-associations, is-a links between relationships, among others. The following structural transformation rules are described on the basis of [25].

Transformation of multi-associations

An association is a traditional relationship type linking one object instance per role. In a multi-association, each role of a multi-association instance may link a set of object instances instead of one object instance.

Multi-associations are not directly supported in current DBMSs. Consequently, a process called objectification is performed. Objectification allows to remove n:m association relationships. Objectifying a relationship type is replacing the relationship with an object type. To keep the semantic of the original relationship, the roles of the original relationship type are turned into new relationship types linking the object type linked by the role to the new object type, which is replacing the original relationship type. The cardinality of the roles linking the new object type is set to (1,1). The instances of the new object type must be linked to instances of the other object types, exactly as was the case with the original relationship type.

Objectification can be applied to multi-representations, with the only difference in cardinalities. Objectification transforms every multi-association relationship type into a new object type, and each of its roles into a binary association type.

An example of transformation of multi-association is shown in Figure 3.5 below. The transformation of the multi-association \textit{Corresponds} becomes into an object type and association relationship type.

Transformation of Is-a links

The generalization or is-a link is a typical feature of conceptual schemas. The function of generalization is to link two representations of the same object at different levels of classification. This classification refinement semantics holds an inclusion constraint where the instances in the subtype, the more specific, are instances of the supertype, the more generic. As a consequence, is-a link is used to show property inheritance, from the supertype to the subtype.

The is-a link concept is not available in the logical schema. A proposed technique to remove the is-a hierarchy is to keep the supertype and its subtypes, spreading the key of the supertype to all the subtypes.
In the transformation strategy, each is-a link is replaced with a binary relationship type, and an associated integrity constraint that enforces the oid of the two linked objects to have the same value. The cardinalities of the new relationship type is (0,1) for the supertype and (1,1) for the subtype. Property inheritance is not included, and it may be reconstructed as needed by users through relationship types.

An example of Is-a link is shown in Figure 3.6 below. The transformation of Is-a links into binary relationship types.

**Transformation of the semantics of relationships** To explain this transformation consider as an example an aggregation semantic that is restricted to binary relationship types, and that comes with specific labels for its roles. These labels allow users to identify which object type is the aggregate and which is the component. The aggregation semantic is a normal relationship with a label. The problem is labels can not be translated to the target schema. As a result, they are lost in the transformation process. The objective is keeping them at least as a comment for future
users of the DBMS. In summary, the transformation of the aggregation relationship types replaces the aggregation semantic with a comment.

**Transformation of overlapping links** A overlapping link is a link between two object or relationship types that support that, under certain condition, are in multi-instantiation. Two object or relationship types are in multi-instantiation if their instances have the same object or relationship identifier.

The transformation rule to remove an overlapping link between two object types has the same process as the removal of is-a links. It replaces the overlapping link with a binary relationship type with cardinalities (0,1)-(0,1). This association has no attributes.

An example of transformation of overlapping links is shown in Figure 3.7 below. The transformation of Is-a links into binary relationship types.

![Figure 3.7: Transformation of overlapping links. Source: (25)](image)

**Removing relationships** The rules that have been described so far, turn schemas with relationship types into schemas with only object types and plain binary association relationship types in between. To transform from relationships to binary links is affordable if there is a binary relationship, as links are, and the relationship does not hold information that is attached to the relationship.

In the relational model, a link consists of a pair complemented with a referential integrity constraint. The pair is composed of a primary key, and a foreign key referencing this primary key, and the constraint is that values in the foreign key exist as values in the primary key. As foreign keys are single, a primary key, foreign key link is equivalent to a binary association relationship type, without attributes with a maximum cardinality of 1 for the role corresponding to the foreign key. Furthermore, each relationship type that follows this specification can be implemented as a foreign key.

**Transformation of multivalued attributes** Logical schemas based on relational data modelling do not support multivalued attributes. The transformation strategy for the relational model is removing the multivalued attributes using objectification, this means objectifying the attribute and creating a new binary relationship type between the original attribute and
Chapter 3. From conceptual to logical design

the objectified attribute. This transformation process works if two conditions are satisfied: a) the multivalued attribute derives from an object type. b) the cardinality of the original object type in the new relationship type is the same as the cardinality of the original attribute.

If a multivalued attribute derives from a relationship type, before applying the previous transformation rule the relationship has to be transformed into an object type.

Multi-representation transformation rules

In this section, we discuss multi-representation rules which are based on the assumption that objects or relationships have several representations. First, an object and relationship type may have different attributes, semantics, roles, and populations depending on the perception. Second, an object and relationship type may have a unique representation for a given perception. As a result, the transformation rule for multi-representation transforms an object (or relationship) type that has N perceptions, into N single object (or relationship) types [25].

The definition of these new object (or relationship) types are the definitions of the representations of the original object (or relationship) type for the perceptions. The new object (or relationship) types are all overlapping one another. Besides, they are linked by the same links as the original object (or relationship) types. The original links are replicated for new each object (or relationship) type [25].

Finally, an integrity constraint is generated asserting that the instances of new object (or relationship) types that share the same identity must have for all their common attributes the same values. Also, an integrity constraint for relationship asserts that for any pair of the new relationship types the instances that are sharing the same identity must have for all their common roles, the same linked object instances [25].

Spatial transformation rules

In this section, we describe transformation rules for spatial objects and relationship types, spatial attributes, and topological relationship types. The following spatial transformation rules are described on the basis of [25].

Transformation of spatial object and relationship types The first spatial transformation rule removes spatial object (or relationship) types, turning them into non-spatial object (or relationship) types. This new object (or relationship) type is provided with a spatial attribute, called geometry. The geometry attribute has the spatial representation of the original object (or relationship) type, this means minimum cardinality, domain of values, and set of specifications.

Transformation of spatial attributes The transformation rule to remove spatial attributes is a variant of the structural transformation rule described
in Section 3.2.2 (removing multivalued attributes) and applies objectification of the attribute.

The transformation of spatial attributes first transforms a single spatial attribute of an object type into a spatial object type. The spatial characteristics of the new object type are the same as those of the original attribute (minimum cardinality and set of specifications). The difference between the rule described in Section 3.2.2 and the one described in Section 3.2.2 is that the current one moves the spatial information of the attribute at the object type level, whereas the previous one transfers the attribute to become an attribute of the new object type.

**Transformation of spatial data types** In this section, we describe the rules to transform an attribute with a spatial data type into another, or several other, attribute(s) having spatial data types.

The first rule is for schemas that only support a generic spatial type. It transforms an attribute with a specialized type into an attribute with the generic type named Geotype. The definition of the specific type will be done when the value is created.

For schemas that only support specific types and no generic type like Geotype, an attribute with a generic type named SimpleGeoType is transformed into a complex attribute composed of a simple optional spatial attribute for each possible type of value that can hold this attribute.

**Transformation of topological relationship types** For transformation of topological relationships, a rule is defined to transform the topological relationship type into a simple relationship type and an integrity constraint is specified to hold the topological predicate.

### 3.3 UML extensions for object-relational database design

In this section, we describe a mechanism for object-relational database design, which is based on UML, extended with the required stereotypes for this design task. This mechanism is also proposed for spatial database design.

#### 3.3.1 Previous concepts

The UML extension mechanism allows extending the language in controlled ways by means of stereotypes, tagged values and/or constraints [17].

- **Stereotype**: a stereotype extends the vocabulary of UML by creating new building blocks. A new building block has its own special properties (set of tagged values), semantics (set of own constraints), and notation (own icon that represents the stereotype).

- **Tagged value**: a tagged value extends the properties of a UML building block by adding new information.
• *Constraint:* a constraint extends the semantics of a UML building block by adding new rules or modifying existing ones. A constraint specifies conditions that must be supported by elements to be modeled in order to be well-formed.

Considering previous extensions for database design some specific stereotypes for each of the phases of the development of a relational database are proposed. Such stereotypes are detailed in Chapter 4.

### 3.3.2 UML extensions

In general, the mechanism defines new UML stereotypes for object-relational database design, and proposes general guidelines to transform UML schema into an object-relational one [17]. Such guidelines are based on the UML class diagram as the conceptual modelling technique, the SQL:1999 object-relational model for the logical schema, and PostgreSQL/PostGIS as an example of a DBMS product. The details of this extension mechanism are described in Chapter 4.

### 3.4 Using UML and OCL to maintain the consistency of spatial data in information systems

In this section, we describe the use of the Object Constraint Language (OCL) to model spatial constraints.

#### 3.4.1 OCL Language description

OCL is a formal language used to describe logical expressions over UML schemas. OCL allows specifying invariant conditions (or constraints) over entities representing concepts from the problem domain. Such constraints must hold true for the concepts being modeled [4].

A UML diagram, is usually not refined enough to provide all the important elements of a specification. Among other things, it may be necessary to define additional constraints about the objects in a schema. Those additional constraints are often described in a notation close to natural language, resulting in ambiguities. Formal languages such as OCL have been developed to avoid such ambiguous constraints [20].

OCL can be used for different purposes such as the ones described in [20]:

- As a query language
- To specify invariants on stereotypes, and types in the class diagram
- To specify constraints on operations (pre-conditions and post-conditions)
- To specify derivation rules for attributes in a UML schema.

Currently, OCL is part of the UML standard supported by the OMG, and its role is important for the MDA approach since it provides a platform-independent and generic method to specify constraints [26].
3.5 Related work on transformations

3.4.2 Integrating spatial functions in OCL

It is possible to extend OCL by integrating spatial characteristics, to support operations to describe the spatial constraints that are often needed on spatial data information systems [26]. In Figure 3.8 are shown the eight possible spatial relationships between two spatial regions [29].

Figure 3.8: Spatial relationships between two simple regions. A simple region is a closed connected point set in a 2-dimensional space \( \mathbb{R}^2 \). Source: [29]

To allow the specification of spatial constraints on a specific schema with spatial characteristics, the spatial relationships of Figure 3.8 must be integrated into OCL as predicates. In general terms, this integration is made by a construct that specifies the spatial relationship between pairs of regions. Such constructs are overlaps, disjoint, equal, meet, contains, covers and the inverses of the last two [26].

The integration of the spatial functions into OCL is carried out in the research project “Design of Spatial Templates for Spatial Database Management Systems in SDI context” which is carried out by Helen Ghirmai Asfaha.

3.5 Related work on transformations

Some studies of transformation processes for relational and object-relational databases have been published. Besides, a number of generic transformation tools have also been developed. In the following sections, we discuss some of this research.

3.5.1 Transformation of UML into models using concrete syntax patterns

In general, the transformation process using syntax patterns as described in [33], presents an approach to specify transformations as patterns in the concrete syntax of UML version 2.0. Such patterns are easier to read than usual transformation specifications and use only standard UML version 2.0 constructs.

As detailed in [33], the complete transformation process can be done using a single modelling tool. In Figure 3.9 the process is shown at the abstract level. First, there are two packages that must be created: Pattern and UML2 Model. The Pattern package holds transformation specifications. The Generator uses this pattern to generate a profile and some other data containing transformation and constraints for the extension of the pattern. The resulting profile must be used to apply the pattern to the source model. The extension is realized
by the modifier that takes the transformations and constraints to generate an expanded model. The most abstract parts of the transformation process are Generator and Modifier, which depend on the selected modelling tool.

Figure 3.9: Process within a single modelling tool. Source: (33)

Figure 3.10 shows the diagram for transformation rules that are used in the current approach. The source and target schema are represented in a single UML 2.0 diagram. A transformation rule is specified in a combined Left Hand Side(LHS)/Right Hand Side(RHS) structure. It is also possible to add logic by forms that store OCL constraints. Patterns are specified as parts of UML 2.0 diagram, where every pattern is syntactically typed because of the relationship between a stereotype and its extension.

Figure 3.10: Features of transformation rules. Source: (33)
In general, the presented method specifies a transformation process for any UML 2.0 diagram with no other resources than the standard constructs of UML 2.0. Since transformations are not supported in UML 2.0, it is necessary to make use of profiles for the extension process. At the beginning, the extension process is used to provide a profile to specify transformations with the required stereotypes. Once the transformation specification is complete enough for application and extension, this can be used to generate two different sorts of information: a profile describing and validating the application, and a transformation file with the constraints to perform the required extension [33].

This method shows only the benefits that UML 2.0 profiles offer for MDA.

### 3.5.2 Transformation process for object-relational databases

For object-relational databases, there is for example, the transformation process that proposes the method for the development of object-relational databases in MIDAS [38], a model-driven method for the development of Web Information Systems. In this study, the conceptual schema and the object-relational database are both represented in UML, and it focuses on the formalization of the transformation techniques to derive an object-relational database from the conceptual schema.

The proposed method [38] for object-relational databases development is based on the definition of schemas at different levels of abstraction, as in the MDA development approach. First, the UML profiles for object-relational database modelling must be described, as well as the metamodels from which these profiles were generated. Second, the definition of the mappings between these schemas are proposed. This two steps define the transformation process.

The mapping description may be in natural language, a model in a mapping language, or in an algorithm in an action language.

A set of graph rules must be defined to express the transformation rules in a graph grammar. These rules follow the LHS/RHS structure. Both the LHS and the RHS are graphs: the LHS is the graph to match whereas RHS is the replacement graph. This means that if a match is found on the source schema, then it is replaced by the RHS in the target schema.

The proposed method [38] presents graph transformation rules for persistent classes, attributes, associations, generalizations, and aggregations/compositions.

Finally, this transformation method allows confirming the PIM to PSM transformations are more suitable to automatization process than PIM to PIM transformations. While the former decrease the abstraction level and as a result make the modelling task is easier; the later require higher level of decision making from the user because of the higher abstraction levels of the implied schemas.

### 3.5.3 Comparisons

In the first research, the transformation method of UML into models using concrete syntax patterns, allows a transformation process for any UML 2.0 diagram with no other elements than the standard constructs of UML 2.0. Trans-
formations are not supported in UML 2.0, for that reason the constructs are extended with the use of profiles.

In the second research, the proposed method for object-relational databases development, as well as in the first research the UML profiles for object-relational database modelling must be described, as well as the metamodels from which these profiles were generated.

In most researches, there are two steps that define the transformation process. The extension of standard UML constructs, and the definition of the mappings between schemas. The current research differs from the previous studies and other related work, since they propose transformation techniques adopted for the spatial application domain, with special attention for transformation of the spatial constructs in the designs.

### 3.6 Comparison of existing modelling tools

The following tools are compared on the basis of the latest UML specifications, OCL support, XMI file generation, and MDA support.

#### 3.6.1 Modelling tools

This section presents modelling tools that were selected for their evaluation according to the modules that they contain. Such modules can be used for supporting specific processes (UML components), languages or platforms (Python, C/C++, C#, Java, among others), integration with other tools (Eclipse, Visual Studio Net as an example), or extending functions like design pattern support.

**StarUML\(^{TM}\)**

StarUML\(^{TM}\) as mentioned in [14] is an open source software modelling platform to develop fast and flexible modelling tasks. It is based on UML version 1.4 and provides UML version 2.0 notations. Furthermore, StarUML\(^{TM}\) supports the MDA approach by making available the UML profile concept.

StarUML\(^{TM}\) provides support for saving and loading XMI files. Nevertheless, OCL constraints can only be stored as text. StarUML\(^{TM}\) runs on Windows platforms.

One outstanding characteristic of StarUML\(^{TM}\) is that it provides excellent flexibility and extensibility. It not only provides pre-defined functions but also allows creation of new functions.

**ArgoUML**

ArgoUML as referred in [27] is an open source software package, that is written entirely in Java. Such characteristics allows ArgoUML to run on virtually any platform. ArgoUML is compliant with the OMG Standard for UML 1.4.

Argo UML supports XMI as its standard saving mechanism. Additionally, exporting to XMI file is also possible.
3.6. Comparison of existing modelling tools

ArgoUML supports OCL for UML classes and features. Besides, ArgoUML provides code generation for Java, C++ among other languages. The Java code generation works with the Java reverse engineering allowing basic round-trip engineering.

The company Gentleware offers an ArgoUML commercial extension under the denomination of Poseidon for UML.

**Poseidon for UML**

Poseidon for UML as described in [6] is a software application used to create models on the basis of UML. Originally, Poseidon for UML comes from the ArgoUML project, but after many improvements ArgoUML became into a commercial project and product.

Poseidon for UML is compliant with the OMG Standard for UML 2.0.

**Enterprise Architect**

Enterprise Architect (EA) as mentioned in [37] is one of the most powerful and flexible software design tools available today. The UML 2.1 modelling environment adopts the complete software development lifecycle. EA provides many facilities for software design and systems engineering, requirements management, business process analysis, and some others.

EA provides document generation and reporting tools in many format files. Furthermore, EA supports generation and reverse engineering of source code for languages as C++, C#, Java, VB.Net, Delphi, Visual Basic, PHP and ActionScript.

EA provides support for MDA and OCL. Besides, EA provides options for exporting and importing XMI files.

Finally, EA supports MDA by using, editing, and developing transformation templates. With built-in transformations for DDL it is possible to develop complex solutions from simple PIM into PSM.

### 3.6.2 Problems of selected modelling tools

The most important comparison factors for the selected tools were the latest UML standard specification, OCL support, XMI file generation, and MDA support. The outcomes of the evaluation of the modelling tools are positive for some of them and negative for others. In general, the selected modelling tools are compliant with the OMG standard for UML 2.0, and XMI support for files generation. Two important characteristics that make a difference among them was the MDA support, and OCL definition for classes and features.

The comparison of existing modelling tools is shown in Appendix A.1.

### 3.6.3 Limitations of using EA in transformational design

EA has many important advantages since it provides support for MDA and OCL specification. Furthermore, EA allows editing and developing transformation templates. With transformation templates it is possible to perform these
transformation from PIM to PSM schemas. Nonetheless, EA's limitation is the lack of spatial constructs, and transformation techniques that are adopted in the spatial application domain.

### 3.7 Concluding remark

This chapter has provided a discussion of the modelling concepts that make up a conceptual data model able to handle complex data structures, spatial information and the possibility to represent multiple perceptions on data within the same database.

There is a number of proposed research for transformational design of object-relational databases. Nonetheless, this previous studies and other related work does not provide transformation techniques for the spatial application domain, with special attention for transformation of the spatial dimension in the designs.

Finally, to allow the transformation design in the spatial application domain, it is necessary a modelling tool that allows combining OCL specification, transformation techniques, and the extension of profiles for the required spatial constructs. Some of the currently available modelling tools are powerful and flexible for software design, modelling task, and generation of many format files among other functions. Nevertheless, they need to provide transformation processes that will work in the spatial application domain.
3.7. Concluding remark
Chapter 4

Method design

This chapter describes a method for transformational design of spatial databases by using MDA. The related aspects of the method include requirements gathering, as well as transformation steps definition. Finally, a workflow for the transformation process that includes the integration with the spatial data profile provided by the sister research project “Design of Spatial Templates for Spatial Database Management Systems in SDI context” which is carried out by Helen Ghirmai Asfaha. Synchronously, the workflow is also implemented in the selected implementation platform.

4.1 Design aspects

4.1.1 Design goal

The goal is to develop a practical and operational workflow for transformational design of spatial databases at different abstraction levels, meaning the transformation from conceptual to logical, and then to physical design.

4.1.2 Input

The required input is the UML Spatial Data Profile worked out in the research project “Design of Spatial Templates for Spatial Database Management Systems in SDI context” which is carried out by Helen Ghirmai Asfaha. The mentioned UML Profile is discussed in Appendix B.2.

Before the design effort begins, it is important to mention that we assume a valid input was provided. Therefore, no verification and validation of the input is considered in this research.

4.2 Requirements gathering

To accomplish the requirements gathering it is important to determine the UML elements for conceptual modelling, the format of the XMI file, the method of transformation, and the format of the Data Definition Language (DDL). Besides, the adopted standards must be understood.
4.2. Requirements gathering

4.2.1 Enterprise Architect for data modelling

Enterprise Architect (EA) makes available many facilities for software design, and systems engineering, among others [37]. Besides, it provides a Graphical User Interface (GUI) that is easy to understand and use. In the current research, because of its many functions and facilities, EA is adopted as the data modelling tool for the transformation process.

Some of the characteristics of EA that facilitate the transformation design are:

- Creation of UML Profiles,
- Creation of Transformation Templates, and
- Generation of XMI files.

Finally, EA also provides support for MDA and OCL, which are two important characteristics to consider for the UML Profile extension, and Transformation Template definition.

4.2.2 UML Profiles

UML is the standard for modelling object-oriented applications. When UML was conceived is was not intended for modelling database design specifically. A database modeled in UML presents a problem when its automated implementation is needed. To solve this problem a mechanism is needed to encode the database concepts in a consistent manner [3]. A UML Profile allows such a mechanism.

UML Profiles provide a way of extending the UML language. These extensions are on the basis of additional stereotypes and tagged values that work on classes, attributes, methods, and relationships. A profile is a collection of extensions that all together describe some particular problem domain and facilitate specifying the constructs in that domain [37].

EA provides a generic mechanism for creating UML Profiles. Such UML Profiles are specified in XML format specific for EA. Besides, it makes available resources for importing UML Profiles into EA. Once a UML Profile is imported into a new Enterprise Architect project, the profile elements can be used in the current diagram. Enterprise Architect attaches the stereotypes, tagged values, notes, and default values to the new diagram elements [37].

An example of Enterprise Architect UML Profile is shown in Appendix C.3.

4.2.3 Data Definition Language

A Data Definition Language (DDL) is a computer language for specifying data structures that store data [3].

Structured Query Language (SQL) is a language for sending queries to a DBMS. Besides, SQL can be used to create tables, add or edit data in the tables, define relationships, and perform query operations on the data [3]. The
Chapter 4. Method design

SQL statements used to create the table structures are usually saved as txt file with some specific extension [3].

In general, adopted standards is a compromise between partners who defined it. As a consequence, DBMS designers do not strictly follow the American National Standards Institute/International Organization for Standardization (ANSI/ISO) SQL:1999 standard. Since SQL is not strictly followed by current DBMS, a problem arises that every SQL is different from DBMS to another DBMS [3]. This means that SQL statements that work in one DBMS do not normally work on another one.

In this research, the SQL statements available for the DDL are compliant with PostgreSQL/PostGIS specifications, as we chose that platform in our project set-up as implementation platform.

The PostgreSQL/PostGIS specifications are shown in Appendix E.1.

4.2.4 Adopted Open Standards

In addition to UML for the conceptual modelling, some other standards need to be understood as background knowledge. These include Extensible Markup Language (XML) and XML Metadata Interchange (XMI). The required open standards list is shown in Appendix A.2.

UML elements

The result of data modelling process is a schema at a specific abstraction level that describes the database before it is implemented. The data modelling design begins at the conceptual level and is expressed in a UML diagram. The database community has adopted UML as its modelling standard language. Nevertheless, because of data modelling design was not initially considered during UML conception, is not entirely suited to be the standard modelling language for database applications [3].

In this research, UML is considered the standard for data modelling of the same system at different levels of abstraction. The considered UML elements are class diagrams, class relationships, generalization, association class, and notes. Besides, OCL is also embraced for the specification of constraints over classes and other information content descriptors.

The notation of UML elements for conceptual modelling is shown in the Appendix B.1.

XMI file format

XML is a hierarchical, plain text data format [2]. One of the most important uses is the conversion of data into a standard format for exchange between system platforms. Besides, XML can also be used to represent data in a database [3].

Nowadays, XMI is a standard for encoding UML diagrams into an XML format. The encoding XMI file can be transmitted over a communication protocol, for conversion between different platforms. In general, XMI is not meant for
4.2. Requirements gathering

storage and access of UML data, but rather as a medium for transferring data between platforms [21].

In the transformational design, the format of the XMI file will have some variations, depending on the selected modeling tool: in our case Enterprise Architect. It is important to determine the proper formatting of the XMI file, which will be the input/output between different abstraction levels when the transformation process is performed.

In Appendix C.1.2 we described the structure, notation, and mapping mechanism of XMI files.

4.2.5 MDA design approaches

To determine a method for conversion and proper format of XMI file, two open source similar implementations were identified and studied. The benefit of studying implementations was an understanding of strategies for approaching the transformation of XMI files.

SourceForge.net is the world’s largest Open Source software development website that hosts centralized resources for managing software projects. It has the largest repository of Open Source code and applications on the Internet [36]. Some of the projects that have been held for SourceForge.net are Zenark’s SQL Markup Language(ZsqlML), Language to Structured Query Language (UML2SQL), and XMI2SQL. The mentioned implementations were examined for finding similar algorithms and ideas, or parts of code to reduce development time.

Some of the characteristics of these implementations are explained below:

• **UML2SQL**

  This implementation updates the content of a database in real time as the model is created. At the time, UML2SQL project was developed, no standard UML Profiles were available. For that reason a proprietary UML Profile was used. This proprietary UML Profile helped with the problem of conversion from XMI to SQL, and executed SQL commands for a specific DBMS server [10].

• **Zsqlml**

  Zsqlml is an XML language for SQL. This language includes functions for converting XML to SQL. ZsqlML allows specifying a relational database structure by using an XML object hierarchy [39].

• **XMI2SQL**

  This open source implementation can generate relational tables in SQL from a XML Metadata Interchange (XMI) file [8].

A summary of the open source implementations that were used to help with development of the transformational design is shown in Appendix A.3.
4.3 Transformational design

Separation of concerns is a fundamental principle in software engineering and for solving other problem domains. The separation of concerns state that each concern should be addressed separately by designing a specific solution that is relatively independent from solutions for other concerns. The purpose of working on each concern separately, is to reduce the complexity of the design. Furthermore, separation of concerns is important to facilitate reuse, evolution, and traceability of specific systems [13].

Transformation steps can apply separation of concerns at different levels of abstraction. This means to separate the transformation design in a sequence of transformation steps according to abstraction level. Each transformation step consists of activities. For each activity it is important to define purpose, input and output, difficulties and solutions in the transformational design.

In Section 1.3, we described the method of work for the research. From this method the phases of analysis, design, and implementation correspond with the transformation process. Therefore, those phases are the framework to identify the corresponding transformation steps between abstraction levels.

1. Analysis

The analysis phase corresponds to the conceptual design that represents the system that is being modeled. Activities are:

- Design of a UML class diagram that represents the conceptual schema by using the spatial constructs defined in the UML Spatial Data Profile

2. Design

The design phase corresponds to logical design, and part of the physical design. Activities are:

- Outline a UML class diagram according to the SQL:1999 standard, that represents the conceptual design, which provides an object-relational specification independent of implementation platform, and
- Outline a UML class diagram with its extensions and product specifications for the implementation platform.

3. Implementation

The implementation phase corresponds to the physical design. For this research, in the physical design the arguments and considerations in performance are not considered.

The physical design only considers the DDL in SQL for a specific DBMS. Activities are:

- Generation of SQL statements specific for the selected product implementation.
4.4 Workflow

In general, the implementation phase includes considerations in performance like optimizations according to the needs and requirements, improving response time and storage capabilities. Such considerations are not part of the study topic of this research.

The following transformation steps and their activities were identified:

1. **From Conceptual to Logical schema**

   This transformation step considers the conversion of a UML class diagram, its extensions, and relationship types that represents the conceptual design, to a UML class diagram for the logical design.

   **Input:** A UML class diagram with the spatial constructs, and relationship types for the conceptual design compliant with the database specifications.

   **Output:** A UML class diagram that represents the logical design.

   **Activities:**
   - Design of a conceptual schema by using the UML class diagram, UML Spatial Data Profile, and relationship types
   - Execution of the transformation from conceptual into logical schema by using the transformation templates of EA

2. **From Logical to Physical schema**

   This transformation step considers the conversion of a UML class diagram that represents the logical design, to a set of SQL statements for a specific DBMS.

   **Input:** A UML class diagram for the logical design.

   **Output:** DDL as a list of SQL statements.

   **Activities:**
   - Generating XMI files from the logical schema by using EA
   - Parsing XMI files
   - Generating output DDL

   The analogy between the adopted method for the research project and the transformation steps are shown in Figure (4.1).

   In the next section, Figure (4.2) exposes the workflow that outline the steps for the transformational design.

### 4.4 Workflow

In general, the transformation steps will transform conceptual descriptions into syntactic statements of a chosen DBMS [3]. In the context of this research, the transformation steps implies the conversion from data model into DDL for a specific DBMS.
The workflow describes steps for translating a conceptual schema represented in a UML class diagram to SQL statements for a PostgreSQL/PostGIS database. One of the benefits of this intent is the use of UML Profiles, XMI files, and some other open standards.

In the MDA model schema, models are instances of metamodels. A metamodel describes properties of a particular platform. A mapping between models is assumed to take one or more models as its input and produce one output model [35]. In terms of MDA, the transformation process designed for this research, considers two elements: the input and output model. Both models should be compliant with their corresponding metamodel, since a representation of the mapping between models is proposed in terms of their metamodels. The transformation engine, which is based on the mapping, will be used to produce an output model (DDL). Such output is a transformation of the input model (UML class diagram).

### 4.4.1 From Conceptual to Logical Schema

The transformation from conceptual to logical schema, converts a UML class diagram that represents the conceptual design to a UML class diagram for the logical design.

Enterprise Architect resources will be used for this transformation step. One important resource is the Model Driven Generation (MDG) file that allows for a logical grouping of resources related to specific techniques to be packed into Enterprise Architect. MDG provides the option of importing UML Profiles, and Code Templates among other resources into an Enterprise Architect view.

The first activity of this transformation step is generating the MDG file. Next, the definition of the Transformation Templates using EA. Finally, the integration through importing UML Profile that contains the spatial constructs (input from “Design of Spatial Templates for Spatial Database Management...
4.5 Software requirements

The required software for the development of the transformational design is listed below.

1. Enterprise Architect Version 7.0
   A software design tool on the basis of UML specification, was selected as the modelling tool because of characteristics mentioned in Section 3.6.1.

2. Python 2.5
The programming language is Python Version 2.5, with Python's Integrated DeveLopment Environment (IDLE) Version 1.2.

3. PostgreSQL/PostGIS 8.2
The selected DBMS is PostgreSQL/PostGIS 8.2, which is an open source relational database system. Some characteristics of PostgreSQL/PostGIS 8.2 [28], is that it runs on several operating systems like Linux, UNIX, and Windows. It has full support for foreign keys, joins, views, triggers, and stored procedures.

4. SQuirreL SQL Client 2.5.1
A GUI that provides an environment for administrating PostgreSQL/PostGIS.

5. PostgreSQL JDBC driver
This driver provides the connection between PostgreSQL/PostGIS and SQuirreL SQL Client.

A summary of the software used to help with development of the transformational design is shown in Appendix A.4.

4.6 Summary
According to [3], UML is the standard for object-oriented applications. Nevertheless, it was not conceived for database modelling. In order to encode database concepts in UML, a mechanism must be available. UML Profiles provide this mechanism, allowing the encoding of database concepts in a consistent manner.

EA supports the generation of MDG file that allows for a logical grouping of resources related to a specific technique to be packed into this modelling tool [37]. EA provides XMI file generation, and UML Profile definition. Besides, the MDG file provides the option of importing UML Profiles into EA.

Separation of concerns is a fundamental principle in solving any kind of problem, and it fits very well for software engineering design. According to [13], the purpose of separating concerns is to reduce complexity of design, reuse, evolution and traceability of specific systems. In the transformational design, three important sections in which the notion of separation of concerns were considered important were the separation of the conceptual design from the logical design from the implementation. In general, the conceptual design is related to object-relational specification of information content independent of implementation platform. Whereas the logical design is related to the design of data structure using an implementation platform.
4.6. Summary
Chapter 5

Implementation of transformational design

In this chapter, the implementation of the workflow presented in Chapter 4 for transformational design is described. Besides the implementation aspects, considerations and algorithms are presented.

In the proposed workflow, the transformational design is separated in two parts: the transformation from conceptual to logical, and the transformation from logical to physical. EA is used in transforming from conceptual to logical schema by using EA Transformation Templates incorporated in a Model Driven Generation (MDG) file. In the transformation from logical to physical schema, a conversion algorithm was written in Python, an object oriented language with XML support.

5.1 Workflow implementation

The transformational design implementation was separated in two parts for its implementation: the transformation from conceptual to logical schema, and from logical to physical schema.

- From Conceptual to Logical schema
  
The UML class diagram that represents the conceptual design is transformed into its logical schema by using EA resources.

  EA provides a way of converting a PIM from one domain to another. A transformation is defined using EA’s code generation template transformation, and involves no more than writing a template to create a simple intermediary source file. EA reads the source file and links that to the new PSM. EA also creates internal bindings between the created PSM and the original PIM [37].

  The mentioned process is completely performed by EA, the data format of its input/output for the transformation processes are internally administrate.
5.1. Workflow implementation

- **From Logical to Physical schema**

  The UML class diagram that represents the logical schema, and that is the product of the transformation of the conceptual schema by applying EA resources, is input for the transformation from logical to physical schema. The first stage of this transformation phase is to encode the logical schema in an XMI file by using resources provided by EA. The next stage is the conversion from XMI to SQL for PostgreSQL/PostGIS, which is performed by a conversion algorithm that was written in Python, an object oriented language with XML support.

  DOM is the method to perform object-based parsing \[18\]. First, the conversion algorithm includes the parsing of the XMI file by using DOM. Next, the XMI file is extracted into temporary data structures, which are linked with the data contained in the XMI file. Finally, an XMI to SQL conversion algorithm converts into SQL (DDL) commands for PostgreSQL/PostGIS.

5.1.1 Data transfer network

In this section, we present a data transfer network for the transformational design. The network gives an overview of the data transfer paths of information, starting with the conceptual schema and ending with the generation of SQL (DDL) for PostgreSQL/PostGIS.

The nodes of the network show the information paths for the transformation between the different abstraction levels that were explained in previous sections. The data network that represents the data transfer paths between data communication nodes for the transformation process is shown in Figure 5.1 below.

![Figure 5.1: Overview of data transfer paths](image)

---
In the data transfer network of Figure 5.1, we present the principal and alternative transfer paths of information. We select the following path for the transformational design:

- Start with the conceptual schema (A) represented in a class diagram

- The conceptual schema (B) is transformed into its logical representation by using transformation templates of EA, and spatial constructs defined in the UML Spatial Data Profile. The result is a UML class diagram representing the logical schema (C)

- The logical schema is exported to an XMI file (D) using EA's resources

- The XMI file is converted into SQL DDL (E)

- End with the generation of SQL DDL for PostgreSQL/PostGIS (F)

5.1.2 Writing transformations in EA

EA supports MDA transforms. Such transforms provide a way of converting model elements from one domain to another. This implies converting PIM to PSM elements. A single PIM element can be responsible for the creation of multiple PSM elements across multiple domains [37].

EA defines transformations by using a code generation template language. Such a process is nothing more than writing a template to create a simple intermediary source file. EA reads the source file and links that to the new PSM. Besides, EA creates links between each created PSM and the PIM [37]. An example of EA’s code template is shown in Appendix C.1.1.

Transformations reduce the burden of manually implementing classes and elements for a particular implementation domain. EA includes some basic built-in transformations like PIM to Data Model, Java, and C#, among other transformations [37]. An important remark is that EA uses the abbreviation DDL for its Data Model Transformation. Therefore, in the rest of the document, this DDL Transformation will be used for data model transformations in EA.

An example of MDA transforms is shown in Figure 5.2 below.

The DDL Transformation converts PIM class elements to PSM table elements [37]. In this research, the Data Model Transformation provided by EA with a number of extensions for spatial constructs is used.

Transformation templates

One of the purposes of using templates is to reduce implementation time by enabling the use of concepts that are valid for a specific problem domain, rather than developing them from the beginning. EA provides a mechanism to create transformation templates, and add new stereotypes into such templates [37]. An example of transformation template is shown in Appendix C.1.1.

EA comes with a number of built-in transformation types, as mentioned before. These transformations have been designed to be reusable and customizable according to the transformation’s requirements [37]. In this way, it is pos-
5.1. Workflow implementation

Figure 5.2: MDA Transforms in EA. Source: [37]

It is possible to write personalized transformation templates, that allow fast and simple applications development, focusing on the design rather than technical issues.

EA not only allows customization of the transformation templates but also provides an option for selecting which transformations to perform on the PIM [37]. In this research, the ITC-DDL is the transformation template created for the spatial constructs that are contained in the UML Spatial Data Profile (“Design of Spatial Templates for Spatial Database Management Systems in SDI context” which is carried out by Helen Ghirmai Asfaha).

**Incorporate Model Templates in MDG**

A Model Driven Generation (MDG) file is an important facility of EA, that allows for a logical grouping of resources related to a specific function to be packed into EA. The MDG file provides the option for importing UML Profiles, Code Templates among other resources into an EA view [37].

The DDL Transformation template ITC-DDL can be added into an MDG file by selecting some available options in EA. The first step is to create the MDG file. Next, one needs to create or include the templates in the MDG file [37]. This process is performed by using EA’s GUI or by inserting the following code in the MDG file:
<ModelTemplates>
  <Model name="ITC-DDL"
    description="DDL Transformation Template"
    location="MDG-Transformation.xml"
    default="yes"
    icon = "34"
    filter="Filter Name" />
</ModelTemplates>

Where:

- **Model Name** represents the name of the template
- **Description** represents the description of the template
- **Location** contains the name and path of the location of the MDG file
- **Default** indicates whether the template is or is not selected by being executed on the schema that needs to be transformed
- **Icon** contains an index to EA's icons list (29 = Use Case, 30 = Dynamic, 31 = Class, 32 = Component, 33 = Deployment, 34 = Simple)
- **Filter** is used to group templates under a common name that is used as a filter

The ITC-DDL transformation template added into the MDG file is shown in Figure 5.3 below.

![Figure 5.3: Model Transformation with ITC-DDL Template in EA. Source: [37]](image-url)
5.1. Workflow implementation

**DDL Transformation**

The purpose of a DDL Transformation is to create a logical schema from the conceptual schema. EA supports and uses an Intermediate Language for a number of database specification concepts like Table, Column, Primary Key, and Foreign Key [37].

- **Table**
  Mapped one-to-one into class elements

- **Column**
  Mapped one-to-one into attributes

- **Primary Key**
  List all columns involved, and create the primary key for them

- **Foreign Key**
  List all columns involved in the source and destination class, and create the primary key and foreign key, respectively.

The result of the transformational design from conceptual (Figure 5.4) to logical schema (Figure 5.5) for an Example Database is discussed in Section .

Not only the DDL Transformation is used for transforming conceptual to logical schema, it can also be used to automatically generate the DDL statements to run in one of the EA supported DBMS products. Instead of this process, here in this work the logical schema will be exported to an XMI file for its conversion to SQL commands for PostgreSQL/PostGIS. This process is explained in the section below.

5.1.3 Parsing the XML file

The input for this step is an XMI file encoding of the logical schema that results from the transformation of the conceptual schema. The first activity in the transformation from the logical schema into implementation, is parsing the input XMI file.

The XMI file that encodes the logical schema, contains a very large amount of information that is irrelevant to the data model. The transformation process makes available a function to separate the meaningful information from the XMI file. These items are added in temporary data structures. Once the important information is extracted from the XMI file, relationships between the data structures and the XMI file are generated, and the database is mapped out. With the completion of the internal data structure process, parsing is complete.

In order to generate the output, a function is available that is applied for each item of the temporary data structure allowing the generation of SQL statements for PostgreSQL/PostGIS.
Chapter 5. Implementation of transformational design

Parsing XML with DOM

The Document Object Model (DOM) defines an Application Program Interface (API) for accessing and manipulating XML documents as tree data structures. The DOM is defined by a set of World Wide Web Consortium (W3C) Recommendations that describe a special programming language object model used to store hierarchical documents in memory [7].

The transformational design implements a set of Python functions to convert an XML representation of a schema into SQL statements for PostgreSQL/PostGIS. The Python code for parsing the XMI file was written reusing the proposed code exposed in [18].

The Python code for parsing the XMI with DOM is shown in Appendix D.1.

Conversion Algorithm

The algorithm first parses the XMI file. Then it extracts the XMI file into temporary data structures. The data structures are implemented by using the dictionary data type provided in Python. Next, the structures are linked by cross-referencing with the data contained in the XMI file. Finally, a set of rules is used to convert the content of the data structures into SQL statements.

The algorithm for converting XMI to SQL statements is shown in the section below.

Algorithm DoConversion

Description: Performs the conversion from XMI to SQL statements
Input: A non-empty XMI file
Output: The SQL code corresponding to the XMI file

- Parse the XMI file by calling the function **ParserDOM**
- Set **XMLDoc** to the result of parsing the XMI file
- Get the first element of the XMI file
- Set **XmlRootNode** to the first element of the XMI file
- Match **XmlRootNode** elements that contain the expression **"xmi:Extension"**
- Set **XmlModelNode** to **XmlRootNode** matching elements
- Create temporary tables **StereoType**, **TaggedValue** for stereotype and tagged value nodes
- Get a list of all stereotypes and tagged value nodes by searching in **XmlModelNode** the matching elements with expression **"uml:stereotype"** or **"uml:tagdefinition"**
- Set **SearchList** to the **XmlModelNode** search results
- For every element **N** of **SearchList**
  - Check if **N.ID** and **N.Name** attributes are valid and not null
  - If **N** is a valid element then
    - Check if the **N** element is a **"uml:stereotype"** then
      - Add **N** to **StereoType** temporary table
    - Check if the **N** element is a **"uml:tagdefinition"** then
      - Add **N** to **TaggedValue** temporary table
- Create temporary tables **DBNode**, **TableNode** for database and tables
5.1. Workflow implementation

- For every element \( N \) of SearchList
  Check if \( N.ID \) and \( N.Name \) attributes are valid and not null
  If \( N \) is a valid element then
    Check if \( N.Type \) is a \texttt{uml:stereotype} and its \( N.IDRef \) is valid
    If \( N \) is a valid element then
      Check if \( N \) is a database then
      Add \( N \) to DBNode temporary table
      Check if \( N \) is a table then
      Add \( N \) to TableNode temporary table
- Create a temporary table DataSourceTable
- Get a list of all nodes of XmlModelNode that represent datatypes by matching
  XmlModelNode elements that contain the expression \texttt{uml:class} or \texttt{uml:DataType}
- Set \( DataSourceTable \) to the XmlModelNode that contains the datatypes
- Call the function CreateSQL with the arguments \( DBNode, Stereotype, DataSourceTable, TaggedValue \)
- Set theSQL to the result of CreateSQL function
- Return theSQL

Algorithm CreateSQL
Description: Creates the SQL statements
Input: DBNode that refers to the database nodes,
  Stereotype that refers to the stereotype nodes,
  TaggedValue that refers to the tagged values nodes,
  DataSourceTable that contains all datatypes in the XMI file
Output: A SQL statement

- Set the SQL Statement to null
- For every element \( N \) of DBNode
  Set TempDatabase to \( N \) node
  Set NameDB to the attribute TempDatabase.Name value
  Find the tables associated with TempDatabase
  Set TempTableNodes to the list of tables associated with TempDatabase
  Set TempTableSet to null
  Concatenate theSQLStatement to the \texttt{CREATE DATABASE}+NameDB
  For every element \( M \) of TempTableNodes
    Get the first element of TempTableNodes
    Set \( S \) to the first element of TempTableNodes
    Check if \( S.ID \) attribute is valid and not null
    If \( S \) is a valid element then
      Create TempTable of data structure ClassTable*
      Set TempTable.Name to \( M \).Name value
      Set TempTable.Element to \( M \) node
      Get a list of all nodes matching the expression \texttt{uml:Property}"
      Set AttributeNodes to the list of matching nodes
      For every element \( A \) of AttributeNodes
Create a **TempAttribute** variable
Set TempAttribute.Name to A.Name
Set TempAttribute.Type to A.Type
Find attributes of A matching with expression “uml:stereotype”
Set **StypeStrings** to the list of matching attributes
Check if StypeStrings matches expression “PK”
If StypeStrings matches “PK” then
  Set TempTable.PK to TempAttribute list
If StypeStrings matches "FK" then
  Set **TempFK.LocalAttribute** to TempAttribute.Name value
Find attributes of A matching with “uml:tagdefinition”
Set **ValueNodes** to the list of matching attributes
For every element **TempTaggedValue** of ValueNodes
  Set TempTypeNode to the tag values in TaggedValue
  Set TempValueNode to TempTaggedValue
  Check if TempTypeNode.Name attribute is equal to “table”
  If TempTypeNode.Name is equal then
    Set **TempFK.Table** to TempValueNode.InnerText
  else
    Set **TempFK.ForeignAttribute** to TempValueNode.InnerText
Set **TempTable.FK** to TempFK value
Add TempTable item to TempTableSet
For every element T of TempTableSet
  Call function **ToStringTable** with parameter T
  Set the result of function ToStringTable to **sqlTable** text variable
  Concatenate the SQL statement to the **sqlTable**
– Return **theSQLStatement**

In the previous algorithm CreateSQL, the type ClassTable was used to define a variable that holds the name, list of attributes, and list of primary and foreign keys for a table structure. The class definition for ClassTable is shown in Appendix D.1.

**UML Profile, XMI, and SQL for PostgreSQL/PostGIS**

The UML Profile for spatial data types, the XMI file defined for EA and the SQL for PostgreSQL/PostGIS have an important role for the transformation process, and its correspondence must be well defined.

The spatial data types available in the UML Profile must be listed, and their correspondence with the PostgreSQL/PostGIS geometric object clearly defined. At the same time, the XMI elements for UML must be listed and their correspondence with the PostgreSQL/PostGIS data types, and SQL syntax must be made clear.

The list of available spatial constructs, XMI elements, and PostgreSQL/PostGIS data types and syntax is shown in Appendix E.1.
5.2 Semi-automatic tool embedded in EA

Due to the time limitations of the M.Sc. research work, the transformation process is not completely automatic. Besides, a number of modules and refinement functions have not been completed for all cases. Therefore, we made available a semi-automatic transformational design that composes of two parts: Modelling with EA, and conversion of XMI file to SQL statements for PostgreSQL/PostGIS.

5.2.1 Modelling with EA

The process starts with a requirement to model a system in fast and simple way. The first step is the data modelling using EA.

The data modelling with EA consists of the steps described below.

- Modelling the conceptual schema of a specific system by using EA.
- Transforming the conceptual schema by using the transformation function provided in EA.
- Saving the logical schema that is the result of the transformation.
- Exporting the logical schema to its XMI file representation.

The result of the modelling with EA is an XMI file that encodes the logical schema of the system that is being modeled.

5.2.2 XMI file to SQL

The conversion from XMI file to SQL statements consists of the steps described below.

- Running the conversion program
- Opening, and executing the conversion function on the XMI file
- Saving the output SQL file

A guide for using the transformation resources provided by EA is shown in Appendix F.1.

5.3 Transformational design: an example

This example shows the result of the combination of both parts of the research: spatial constructs and transformational design.

- The conceptual design shows a UML class diagram for the conceptual schema. A class Airport with attributes like Name, City, and Location, where the data type of Location uses the defined spatial construct: Point(4392).

The conceptual design for Airport is shown in Figure 5.4 below.
A conceptual schema for Airport is available. The next step is transforming to its logical schema by using EA’s transformation templates. In Figure 5.5, a UML class diagram shows the logical schema resulting from the transformation.

The physical design shows the resulting schema in the SQL of the selected DBMS PostGIS/PostGIS.

The SQL statement is shown in the section below.

```sql
DROP TABLE IF EXISTS airport;
CREATE TABLE airport (airportID INT4 NOT NULL, name varchar (50), city varchar (50));
SELECT AddGeometryColumn('', 'airport', 'location', 4326, 'POINT', 2);
CREATE INDEX airportID ON airport using GIST('location');
ALTER TABLE airport ADD CONSTRAINT PK_Airport PRIMARY KEY (airportID);
```

In a general view, the transformation process is shown in Figure 5.6 below.
5.4 Summary

The implementation of the workflow for the development process of the transformational design was discussed in this section. The transformational design implementation was separated in two parts: the transformation from conceptual to logical schema, and from logical to physical schema.

In the transformation from conceptual to logical schema, EA is used as a modelling and transformation tool. The UML class diagram that represents the conceptual schema is transformed to its logical representation by using EA transformation resources.

In the transformation from logical to physical schema, EA and a semi-automatic tool written in Python are considered. EA for exporting the XMI file that encodes the logical schema, and the semi-automatic tool for converting the XMI file to SQL statements.

Figure 5.5: Logical schema for Example Database (PSM)
Figure 5.6: Transformation design: an example

DDL: SQL FOR PostgreSQL/PostGIS

DROP TABLE IF EXISTS airport;
CREATE TABLE airport (airportID INT4 NOT NULL,
name varchar(50), city varchar(50));
SELECT AddGeometryColumn(“”,’airport’,’location’,4326,
‘POINT’,2);
CREATE INDEX airportID ON airport using GIST(location);
ALTER TABLE airport ADD CONSTRAINT PK_Airport
PRIMARY KEY (airportID);
Chapter 6

Discussions, Conclusions and Recommendations

In this research project, a semi-automatic process for Transformational Design of Spatial Databases has been developed. The workflow for the transformation process has been implemented under a number of considerations related to the modelling software package, and a UML Spatial Data Profile. This chapter discusses the practical aspects, existing problems and the improvements that can be done in the future for the transformation process. Finally, it provides a summary and conclusion of this research.

6.1 Discussion of the transformational design

In order to verify if the research questions defined in Section 1.2.2 have been addressed and the research objective has been achieved, the discussion is divided over three sections according to the research questions, practicality of the semi-automatic process, and a number of implementation considerations.

6.1.1 Research questions

Definition of the conceptual schema

In Chapter 3, we discussed modelling concepts that allow a conceptual data model to handle complex data structures, spatial information and the possibility to represent multiple perceptions on data within the same database.

To allow modelling of spatial application domains, it is necessary to combine OCL specifications, and the extension of UML Profiles for the required spatial constructs. Therefore, we require a UML Profile that allows modelling the conceptual schema of spatial databases by using spatial constructs. Such UML Spatial Data Profile is worked out in the research project “Design of Spatial Templates for Spatial Database Management Systems in SDI context” which is carried out by Helen Ghirmai Asfaha.
6.1. Discussion of the transformational design

Definition of the logical schema

The logical schema is obtained from the conceptual schema by applying transformation functions implemented in EA as was discussed in Section 5.1.2. Such transformations, and other resources like the UML Spatial Data Profile are packed into a MDG file. The MDG file groups the resources that allow us to perform the transformations on a conceptual database schema.

Definition of set of rules that describes the transformation process

As discussed in Chapter 3, to allow the transformation design in the spatial application domain, a modelling tool is necessary a modelling tool that allows combining OCL specification, transformation techniques, and the extension of UML Profiles for the required spatial constructs.

EA provides transformation templates, and uses an Intermediate Language for database specifications, that make it possible to insert new transformation rules for the spatial constructs discussed in Section 6.1.1.

The set of transformation rules that can be inserted in such transformation templates are discussed in Section.

Building the transformations in a product implementation and combination: Enterprise Architect and PostgreSQL/PostGIS

To build the transformation, a workflow was discussed in Section 4.4. The transformational design implementation was separated in two parts: the transformation from conceptual to logical schema, and from logical to physical schema.

In the transformation from conceptual to logical schema, EA is used as a modelling and transformation tool. The UML class diagram that represents the conceptual schema is transformed to its logical representation by using EA transformation resources.

In the transformation from logical to physical schema, EA and a semi-automatic tool written in Python are described: EA for exporting the XMI file that encodes the logical schema, and the semi-automatic tool for converting the XMI file to SQL DDL statements for a PostgreSQL/PostGIS database.

Evaluating the transformation process

This section briefly describes the evaluation mechanism of the transformational design by analyzing the obtained schema at each step of the transformation process. The analysis consist of two parts:

From conceptual to logical schema The obtained schema of the transformation from conceptual to logical schema, was verified by making the manual analysis by applying normalization techniques on the input schema of a database. Next, the resulting tables as obtained from manual analysis were compared with the schema that resulted from the transformation by using EA.
From logical to physical schema The SQL statements for PostgreSQL/PostGIS obtained of the transformation from logical to physical schema, were verified by executing the obtained SQL statement. The SQL statement was executed by using a GUI that provides an environment for administrating PostgreSQL/PostGIS (SQuirreL SQL Client 2.5.1). The created tables, and their spatial columns, primary keys, and foreign keys were verified to exist in a schema of the DBMS. Finally, a number of rows were inserted in the tables, and a number of queries were executed on them.

In Section 5.3, an example of the application of transformational design was discussed.

6.1.2 Advantages and practicalities of the semi-automatic transformation process

In this section, some advantages and practicality of the semi-automatic process Support for Transformational Design of Spatial Databases are discussed.

- As far as we know, there is no work related to transformational design of databases with spatial representation.

- EA provides support for transformation of object-relational databases. The transformation from conceptual to logical schema can be easily executed by using EA’s GUI, and the resulting schema from the transformation process can be encoded in an XMI file.

- The programming language (Python 2.5), supports XML functions like parsing XML files, and other functions that reduce implementation efforts. Besides, Python provides data type structures that are suitably to work with for implementing the conversion process like dictionaries, among others.

- The keywords for parsing the XMI file, are saved in a text file from where they can be read. Therefore, the algorithm conversion from XMI to SQL can be extended for other modelling tools that can export their logical schema to an XMI file.

- It is possible to extend the conversion process from XMI to SQL for another selected DBMS with extra programming efforts.

- Parametrization and modularity of the implemented functions provide easy extension, and programming of new functions

6.1.3 Limitations and problems

Some important considerations of this research are that a specific modelling tool was selected. Therefore, the transformational design must consider standards, functionality and some other resources that the selected tool provides, to define the method of transformation and its transformation steps.
6.2 Future improvements

In this section, we discuss some improvements for the transformation process and implementation.

- Make the complete process automatic. This means embed an option in EA that allows running the complete transformation process after the conceptual schema is completed and the option is executed.
- Improve the conversion algorithm from XMI to SQL statements.
- Extend the conversion algorithm to another DBMS.

6.3 Conclusions

In this research project, a semi-automatic process for transformational design of spatial databases was developed. From the review of the available transformational design researches, this research is the first attempt to develop a process to model spatial databases by using the MDA approach.

To allow transformational design in the spatial application domain, we need a modelling tool that allows combining OCL specification, transformation techniques, and the extension of profiles for the required spatial constructs. Some of the currently available modelling tools are powerful and flexible for software design, modelling tasks, and generation of many format files, among other functions. Nevertheless, they need to provide transformation processes that will work in the spatial application domain. The innovative part is that the semi-automatic process for transformational design allows spatial modelling by using spatial constructs defined in a UML Profile.

From the discussion and the research achievements, the following conclusions can be drawn:

- Separation of concerns, three important sections in which the notion of separation of concerns were considered important were the separation of the conceptual design from the logical design from the implementation. In
general, the conceptual design is related to object-relational specification independent of implementation platform. Whereas the logical design is related to the design using an implementation platform.

The conceptual design, logical design, and implementation provide separation of concerns to enhance robustness and reusability to change, architectural extensibility and flexibility in developing applications.

- One important challenge in the implementation of new methods and technology are the speed in making operational such new developments. Even with rapid development of applications or rapid prototyping, technology changes are faster, thus spatial information systems may be obsolete before they are completed. Therefore, generation and/or replication of data models for a field of application by using reference models or set of templates can be a rapid and controlled mechanism for developing spatial information systems.

The research problem and questions have been solved to some extent. As a result, the objective to develop a semi-automatic process for Transformational Design of Spatial Databases has been achieved.
6.3. Conclusions
Bibliography


Appendix A

A.1 Comparison of existing transformation tools
A.2 Standards summary
A.3 Open Source Implementations
A.4 Software used in the transformational design
A.5 Correspondence of spatial an non-spatial data types
### Table A.1: Comparison of existing transformation tools

<table>
<thead>
<tr>
<th>Criteria</th>
<th>StarUML 5.0</th>
<th>ArgoUML (Release 0.24)</th>
<th>Posidon For UML 6.0</th>
<th>Enterprise Architect 7.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Open Source Development Project</td>
<td>Open Source Development Project</td>
<td>GentleWare</td>
<td>Spark Systems</td>
</tr>
<tr>
<td>Release Type</td>
<td>Open Source</td>
<td>Open Source</td>
<td>Limited Edition Community</td>
<td>Commercial</td>
</tr>
<tr>
<td>Standard UML</td>
<td>UML 2.0</td>
<td>UML 1.4</td>
<td>UML 2.0</td>
<td>UML 2.x</td>
</tr>
<tr>
<td>OCL</td>
<td>Text</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>XML support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MDA support</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SysML</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reverse engineering</td>
<td>C++, C#, Java</td>
<td>Java</td>
<td>Java</td>
<td>Yes</td>
</tr>
<tr>
<td>IDE integration</td>
<td>No</td>
<td>No</td>
<td>Eclipse</td>
<td>Microsoft Visual, Borland Delphi, Java</td>
</tr>
</tbody>
</table>
## Table A.2: Standards summary

<table>
<thead>
<tr>
<th>Standard</th>
<th>Author/Maintainer</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>W3C’s XML Recommendation: <a href="http://www.w3.org/TR/REC-xml">http://www.w3.org/TR/REC-xml</a></td>
</tr>
<tr>
<td>Open Source Implementation</td>
<td>Author</td>
<td>Primary Resource</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>UML2SQL</td>
<td>Hannes Kegel and Ingo Kegel</td>
<td><a href="http://sourceforge.net/projects/uml2sql">http://sourceforge.net/projects/uml2sql</a></td>
</tr>
<tr>
<td>Zsqml</td>
<td>Zenark Ltd.</td>
<td><a href="http://sourceforge.net/projects/zsqml">http://sourceforge.net/projects/zsqml</a></td>
</tr>
<tr>
<td>XML2SQL</td>
<td>Eric Raymond Hartford</td>
<td><a href="http://sourceforge.net/projects/xml2sql">http://sourceforge.net/projects/xml2sql</a></td>
</tr>
</tbody>
</table>

Table A.3: Open Source Implementations
<table>
<thead>
<tr>
<th>Software</th>
<th>Author</th>
<th>Primary Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL-PostGIS</td>
<td>PostGIS Development Group</td>
<td><a href="http://postgis.refractions.net">http://postgis.refractions.net</a></td>
<td>Provides the connection between PostgreSQL and Python, allowing for SQL-like access to spatial data.</td>
</tr>
<tr>
<td>PostgreSQL-Oracle</td>
<td>Oracle Corporation</td>
<td><a href="http://www.oracle.com">http://www.oracle.com</a></td>
<td>Provides the connection between PostgreSQL and Oracle databases.</td>
</tr>
<tr>
<td>PostgreSQL-PostgreSQL</td>
<td>PostgreSQL Global Development Group</td>
<td><a href="http://www.postgresql.org">http://www.postgresql.org</a></td>
<td>Provides the connection between PostgreSQL databases.</td>
</tr>
<tr>
<td>PostgreSQL-SQL</td>
<td>PostgreSQL Global Development Group</td>
<td><a href="http://www.postgresql.org">http://www.postgresql.org</a></td>
<td>Provides the connection between PostgreSQL databases.</td>
</tr>
<tr>
<td>PostgreSQL-SQLite</td>
<td>SQLite</td>
<td><a href="http://www.sqlite.org">http://www.sqlite.org</a></td>
<td>Provides the connection between PostgreSQL databases.</td>
</tr>
<tr>
<td>PostgreSQL-MySQL</td>
<td>MySQL</td>
<td><a href="http://www.mysql.com">http://www.mysql.com</a></td>
<td>Provides the connection between PostgreSQL databases.</td>
</tr>
<tr>
<td>PostgreSQL-DB2</td>
<td>IBM</td>
<td><a href="http://www.ibm.com">http://www.ibm.com</a></td>
<td>Provides the connection between PostgreSQL databases.</td>
</tr>
<tr>
<td>PostgreSQL-MS SQL Server</td>
<td>Microsoft</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
<td>Provides the connection between PostgreSQL databases.</td>
</tr>
</tbody>
</table>

Table A.4: Software used in the transformational design
### A.5. Spatial and non-spatial data types

Table A.5: Correspondence of spatial and non-spatial data types

<table>
<thead>
<tr>
<th>SPATIAL DATA TYPES</th>
<th>DATA TYPES</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML Profile</td>
<td>PostgreSQL/PostGIS</td>
</tr>
<tr>
<td>ST_Point</td>
<td>Geometry</td>
</tr>
<tr>
<td>ST_LineString</td>
<td>Point</td>
</tr>
<tr>
<td>ST_Polygon</td>
<td>LineString</td>
</tr>
<tr>
<td>ST_MultiPoint</td>
<td>Polygon</td>
</tr>
<tr>
<td>ST_MultiLineString</td>
<td>MultiPoint</td>
</tr>
<tr>
<td>ST_MultiPolygon</td>
<td>MultiLineString</td>
</tr>
<tr>
<td>ST_Geometry_Collection</td>
<td>MultiPolygon</td>
</tr>
<tr>
<td>ST_Geometry</td>
<td>GeometryCollection</td>
</tr>
<tr>
<td>ST_Curve</td>
<td>real</td>
</tr>
<tr>
<td>ST_LinearString</td>
<td>smallint</td>
</tr>
<tr>
<td>ST_LinearRing</td>
<td>text</td>
</tr>
<tr>
<td>ST_MultiCurve</td>
<td>timestamp</td>
</tr>
<tr>
<td>ST_MultiSurface</td>
<td>varchar</td>
</tr>
<tr>
<td>ST_Polyhedral</td>
<td></td>
</tr>
<tr>
<td>ST_Surface</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

B.1 UML Notation Elements

Figure B.1: UML elements for conceptual modelling. Source: (22)

B.2 UML Spatial Data Profile
B.2. UML Spatial Data Profile

Figure B.2: UML Spatial Data Profile. Source: research project "Design of Spatial Templates for Spatial Database Management Systems in SDI context" carried out by Helen Ghirmai Asfaha
Appendix C

C.1 Enterprise Architect

C.1.1 Code Template

EA's Code Templates are written as plain text. Basically, the template syntax is based on three constructs: literal text, macros, and variables [37].

Literal Text

The text that is contained in a code template that is not part of a macro or a variable, is considered literal text. With the exception of blank lines, literal text is directly substituted from the template into the generated code.

For example: %PI=""

The %, $ characters have special meaning in the template syntax and cannot always be used as literal text.

Macros

Macros provide access to element fields in the UML model. Besides, they are also used to structure the generated output.

Macros are enclosed within percent (%) signs, and they are performed by executing determined templates that are available in EA.

The syntax is as follows: %< TemplateName >% where < TemplateName > can be one of the templates listed below.

<table>
<thead>
<tr>
<th>AttributeDeclaration</th>
<th>ClassParameter</th>
<th>NamespaceBody</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeNotes</td>
<td>File</td>
<td>NamespaceDeclaration</td>
</tr>
<tr>
<td>FileImpl</td>
<td>Class</td>
<td>ImportSection</td>
</tr>
<tr>
<td>Operation</td>
<td>ClassImpl</td>
<td>ImportSectionImpl</td>
</tr>
<tr>
<td>OperationBody</td>
<td>ClassBase</td>
<td>InnerClass</td>
</tr>
<tr>
<td>OperationBodyImpl</td>
<td>ClassBody</td>
<td>InnerClassImpl</td>
</tr>
<tr>
<td>OperationDeclaration</td>
<td>ClassBodyImpl</td>
<td>LinkedAttribute</td>
</tr>
<tr>
<td>OperationDeclarationImpl</td>
<td>ClassDeclaration</td>
<td>LinkedAttributeNotes</td>
</tr>
<tr>
<td>OperationImpl</td>
<td>ClassDeclarationImpl</td>
<td>LinkedAttributeDeclaration</td>
</tr>
<tr>
<td>OperationNotes</td>
<td>ClassInherits</td>
<td>LinkedClassBase</td>
</tr>
<tr>
<td>Parameter</td>
<td>ClassInterface</td>
<td>LinkedClassInterface</td>
</tr>
<tr>
<td>ClassNotes</td>
<td>Namespace</td>
<td>Attribute</td>
</tr>
</tbody>
</table>

83
Variables

Template variables provide a way of storing and retrieving data within a template. The definition of a Variable is as follows:

\[ \text{<name>} = \text{<value>} \]

where: <name> describes the name of the variable <value> is a value derived from a macro or another variable

A simple example definition would be:

$\text{foo} = \%\text{className}\%$

The ITC-DDL Template is illustrated in Figure C.1 below.

C.1.2 XMI files

According to [37], XMI files are composed of three parts: XMI element, documentation, and extensions elements.

The XMI class is for default the container for XMI documents metadata and contents. The attributes of the XMI class are: version, documentations, differences, and extensions.

XMI files for EA has the structure shown in Figure C.2 below.

C.1.3 UML Profiles
Figure C.2: Example of XML file structure. Source: (37)
Figure C.3: Example UML Profile in Enterprise Architect. Source: (37)
Appendix D

D.1 Python code

D.1.1 ParserDOM

The ParserDOM Python function is used in the conversion process from XMI to SQL statements for PostgreSQL/PostGIS. The parser is written in Python Version 2.5. The source code in shown in the section below.

```python
+ # ————————————————————————-
+ # Python Function ParserDOM
+ # Description: Performs the XML parsing using DOM
+ # Params: XML file
+ # Return: Parser file
+ # ————————————————————————-
+
+ def ParserDOM(f):
+     # Calls the parsing function xml.dom.minidom provided by Python
+     doc=xml.dom.minidom.parse(f)
+     as_=doc.getElementsByTagName('a')
+     # The parsing XML with DOM
+     for a in as_:
+         value = a.getAttribute('href')
+         if value:
+             newtext=doc.createTextNode('(((%s)))'%value)
+             a.parentNode.insertBefore(newtext,a)
+
+ # Class to write out the text lines
+ class UnicodeStdoutWriter:
+     def write(self,data):
+         sys.stdout.write(data.encode('utf-8'))
+
+ # Show the result of parsing the XMI file
+ doc.writexml(UnicodeStdoutWriter())
```
An example how to call the function ParserDOM in shown in the section below.

+ # ————————————————————————-  
+ # Python Function DoConversion  
+ # Description: Performs the conversion from XMI to SQL  
+ # Params: A non-empty theXMI file  
+ # Return: The SQL code corresponding to the XMI file  
+ # ————————————————————————-  
+  
+ # Importing Python libraries for XML parsing  
+ import xml.dom.minidom,urllib,sys  
+  
+ def DoConversion():  
+    # Calls the parsing function ParserDOM  
+    theXMI=open('C:/Thesis/EA/xmi/xmi.xml')  
+    ParserDOM(theXMI)
D.1.2 Data Structures

```python
# ————————————————————————-
# Data structure definition
# Class FK
# Class Attribute
# Class Table
# ————————————————————————-

# Class FK definition
class FK:
    def __init__(self):
        self.LocalAttribute, self.ForeignAttribute, self.Table = ""

# Class Attribute definition
class Attribute:
    def __init__(self):
        self.Name, self.Type = ""

# Class Table definition
class Table:
    def __init__(self):
        self.Name = ""

    def ToStringTable():
        result = ""
        result += "CREATE TABLE " + Name + cRetLin + "(" + cRetLin
        for Attribute in Attributes:
            result += a.Name + " " + a.Type + " NOT NULL," + cRetLin
        result += cRetLin
        result += "PRIMARY KEY (" + cRetLin
        for p in PK:
            result += p + ","
        result = result.Substring(0,result.Length-1) + cRetLin
        result += cRetLin
        for f in FK:
            result += "FOREIGN KEY (" + f.LocalAttribute + " REFERENCES " + f.Table + "(" + f.ForeignAttribute + ")" + cRetLin
        result += "") + cRetLin
        return result

Readers interested in received full source code contact with dr. Rolf de By.
```
D.1. Python code
Appendix E

E.1 PostgreSQL/PostGIS

E.1.1 SPATIAL_REF_SYS table definition

According to [28], the SPATIAL_REF_SYS table definition is as follows:

```
CREATE TABLE SPATIAL_REF_SYS ( SRID INTEGER NOT NULL PRIMARY KEY,
    AUTH_NAME VARCHAR(256), AUTH_SRID INTEGER,
    SRTEXT VARCHAR(2048), PROJ4TEXT VARCHAR(2048) )
```

Where:

- **SRID**: Identifier of the Spatial Referencing System (SRS) within the database
- **SRTEXT**: The text representation of the Spatial Reference System

According to [28], the GEOMETRY_COLUMNS table definition is as follows:

```
CREATE TABLE GEOMETRY_COLUMNS ( F_TABLE_CATALOG VARCHAR(256)
    NOT NULL, F_TABLE_SCHEMA VARCHAR(256) NOT NULL,
    F_TABLE_NAME VARCHAR(256) NOT NULL,
    F_GEOMETRY_COLUMN VARCHAR(256) NOT NULL,
    COORD_DIMENSION INTEGER NOT NULL,
    SRID INTEGER NOT NULL, TYPE VARCHAR(30) NOT NULL )
```

Where:

- **F_GEOMETRY_COLUMN**: The name of the geometry column in the feature table
- **COORD_DIMENSION**: The spatial dimension of the column
- **SRID**: Identifier of spatial reference system used for the coordinate geometry in this table. It is a foreign key reference to the SPATIAL_REF_SYS
- **TYPE**: The type of the spatial object

The following types are available in PostGIS:

---

91
POINT            LINestring
POLYGON          MULTIPoINT
MULTILINestring  MULTIPoLYGON
GEOMETRYCOLLECTION POINTM
LINESTRINGM      POLYGONM
MULTIPOINTM      MULTILINestringM
MULTIPoLYGONM    GEOMETRYCOLLECTIONM

Inserting a GIS object into a database

Create a normal table, and then add the column that corresponds to the geometry.

# DROP TABLE my_spatial_table;
# CREATE TABLE my_spatial_table ( ID int4,
    NAME varchar(20) );
# SELECT AddGeometryColumn("','my_spatial_table',
    'the_geom',4326,'LINESTRING',2);

Inserting a geometry into the table:

# INSERT INTO my_spatial_table (ID, NAME, THE_GEOM)
# VALUES (1,'First Geometry',
    GeomFromText('LINESTRING(1 1, 4 4)',-1));
Appendix F

F.1 How to: Transformation User Guide

The following section describes the steps to realize in EA to transform the conceptual schema into an XMI file:

- Create a new project in EA
- Import the XMI file that corresponds to the UML Spatial Data Profile
- Design a conceptual schema for an example database by using UML class diagram with the spatial constructs
- Transform the example conceptual schema
  - Select from EA's menu of options: PROJECT -> TRANSFORMATION
  - Execute the transformation
- Save the result of the transformation: logical schema
- Select from EA's menu of options: PROJECT-IMPORT/EXPORT-EXPORT PACKAGES TO XMI
- Execute the exportation of the logical schema to its encoded XMI file
- Save the XMI file in a directory

So far, we have an XMI file that corresponds to the conceptual schema of a database.

- Execute the file DoConversion
- DoConversion will present a menu of options where we have to select the XMI file to transform
  - Select the XMI file that was exported in previous step.
  - Run the transformation
- Save the result of the transformation in a file with extension .sql
- Close DoConversion