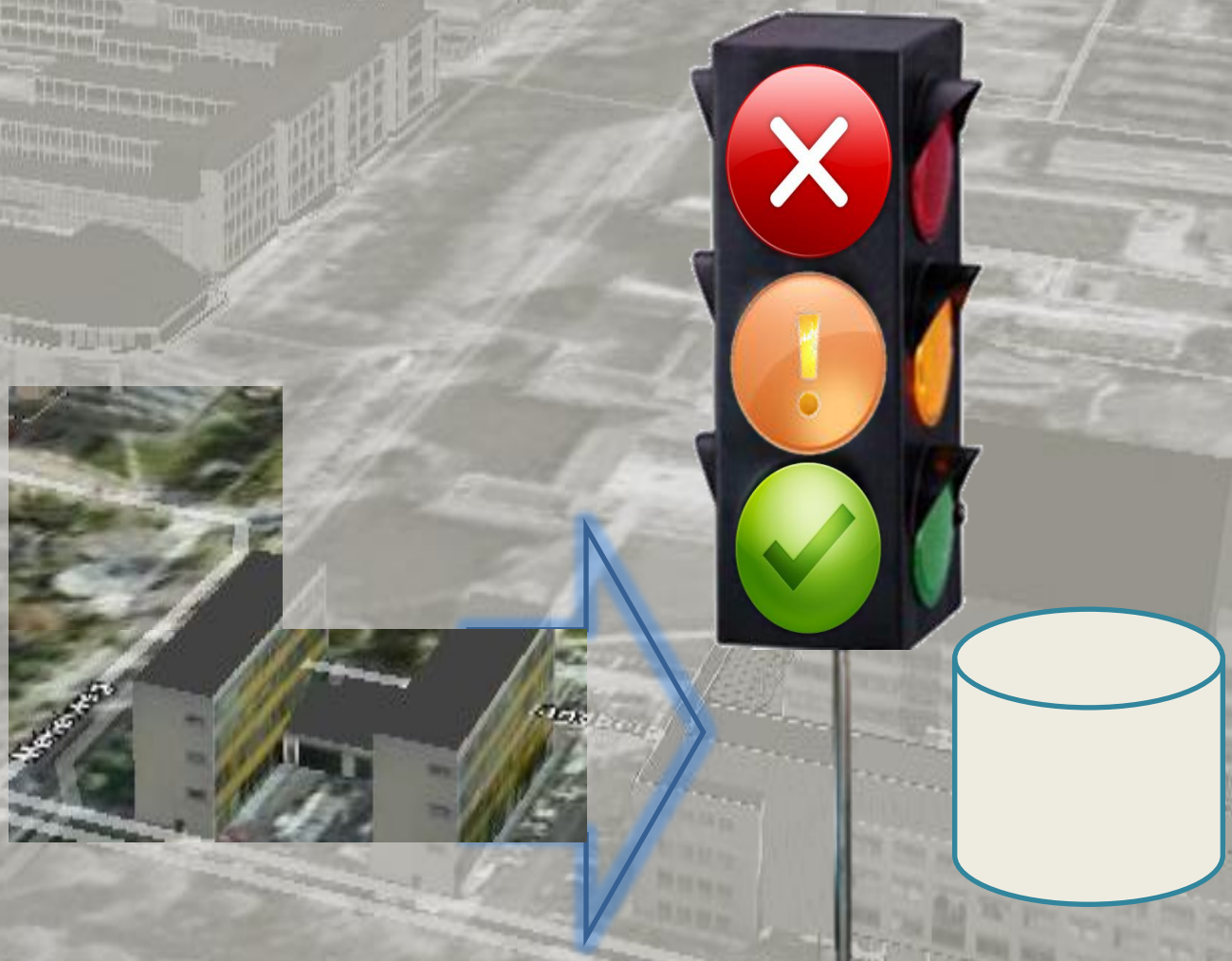
MSc thesis in Geomatics

Design and Implementation of Constraints for 3D Spatial Database - Using Climate City Campus Database as an Example

Daniel Xu

August 2011



TU Delft Delft University of Technology

Department of GIS Technology, OTB Research Institute for the Built Environment

Design and Implementation of Constraints for 3D Spatial Database - Using Climate City Campus Database as an Example

A thesis submitted to Delft University of Technology in partial fulfilment of requirements for the degree of

Master of Science in Geomatics

by

Daniel Xu

August 2011

Supervisors: Prof. dr. Peter van Oosterom
Dr. Sisi Zlatanova
Co-reader: Dr. Jan Hidders

Xu, Daniel: *Design and Implementation of Constraints for 3D Spatial Database - Using Climate City Campus Database as an Example*, MSc thesis, Delft University of Technology, © August 2011.

LOCATION:

Department of GIS Technology
OTB Research Institute for the Built Environment
Delft University of Technology
The Netherlands

TIMEFRAME:

17 December 2010 - 28 August 2011

FINAL PRESENTATION:

1 Sep 2011, at 15:00, Grote Vergaderzaal, OTB

EXAMMINATION COMMITTEE:

Prof.dr. Massimo Menenti
Prof.dr. Peter van Oosterom
Dr. Sisi Zlatanova
Dr. Jan Hidders

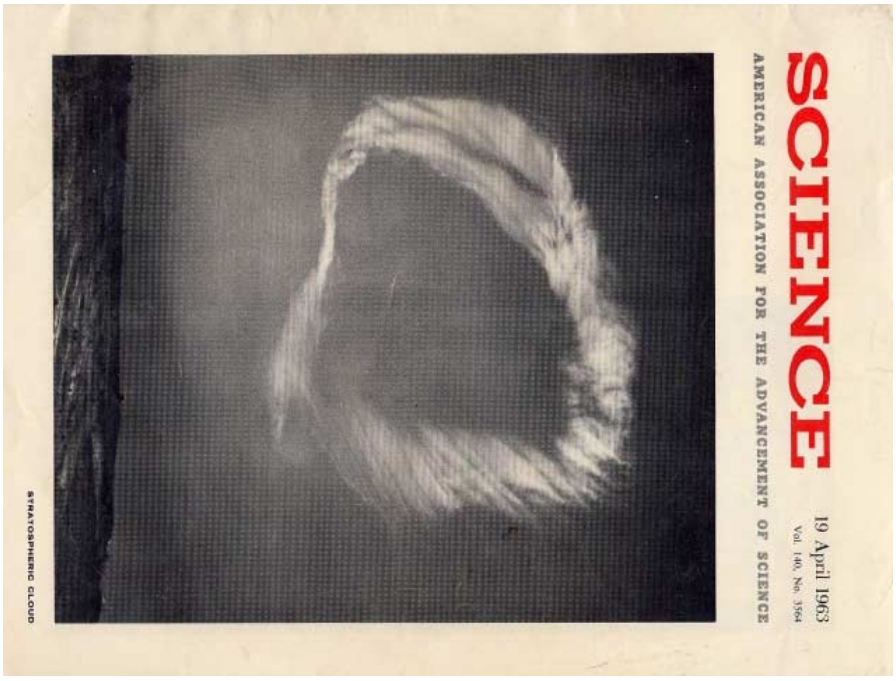
COVER ILLUSTRATION:

The cover background is a snapshot of 3D models about TU Delft campus from Google Earth. The arrow illustrates the 3D data being input of the database. The traffic light with different symbols/colours shows three reactions/interpretations of database constraints toward the input. They are *errors* (in red) which will be stop, *unusual instances* (in orange) which will be kept elsewhere and waiting for a further check, and *correct data* (in green) which satisfies constraints and will be passed. These reactions were implemented by triggers in Oracle Spatial 11g database system during this thesis research.

Trust in the *LORD* with all thine heart,
and lean not unto thine own understanding.
In all thy ways acknowledge *Him*,
and *He* shall direct thy paths.

— *Proverbs* 3:5-6

Dedicated to my beloved Lord Jesus Christ and His Kingdom.



The supernatural cloud appearing over Arizona, USA, Feb 28 1963, was 26 miles high and over 30 miles wide. Notice the likeness of the head of Christ with white hair and beard as spoken of in Rev. 1:14.

ACKNOWLEDGMENTS

First of all I would like to thank Almighty God for His Grace, His Strength, for the Faith He gives me to believe in His Words and for the thankful heart He gives to me. Without Him I would never have been able to carry on to the very end of my research.

There are many thanks to all those that have helped me through the whole period of preparing my thesis. First I would like to state my gratitude to my tutor Dr.ing. Sisi Zlatanova and my supervising professor Prof.dr.ir. Peter van Oosterom. With your enthusiasm, your guidance, your inspiration and your great efforts to explain things clearly and simply, you have helped me to develop the skills in doing research. During my thesis-writing, you have provided encouragement, sound advice, and good ideas. Especially right at the end, your comments were a great help when it came to reshaping my poor technical writing. The things I have learned from you will benefit me in the future and not just for the duration of my thesis research.

I wish to thank others that have helped correct my UML modelling and OCL formulating: João Paulo Hespanha (I will not forget the email you replied to at mid-night), Stefan Werder, Jan van Bennekom-Minnema and Wilko Quak. Although with my limited ability I cannot understand everything you said, your input at the last moment was right on time and was really a great help to me.

I would like to give special thanks to Julia Janssens (who helped me improve my English language), Marinus Benschop (who provided me with a place to stay (a shelter) in the last month of research), Walther Rier (who shared lots of his life experiences with me), Raymond Reid (who encourages me all the time) and many other brothers and sisters in Christ that I will spend eternity with.

I am also grateful to Jan Hidders and Theo Thijssen who gave me a lot of help and stimulated my knowledge with regard to database.

I wish to thank my parents (Dongyuan Xu and Junli Yang) who have sponsored my studying in the Nederlands. You brought me into this world, raised me, supported me and loved me. Without you I would not be here.

Last but not least, there are many others that have helped me to do this thesis research. Words would fail to express my thanks to you in this confined piece of paper.

For all the people that have helped me, I pray: *May God richly bless you all. Amen.*

ABSTRACT

Nowadays the field of geo-information is undergoing major changes, and the transition from 2D to 3D is having a major influence. A significant amount of 3D datasets are stored in the database. Experts are aware that new quality control mechanisms need to be built into the database systems in order to secure and guarantee high-performing data.

Constraints are effective in providing solutions needed to avoid errors and enable maintenance of data quality. Whereas constraints for 2D geographic datasets have already been the subject of several research projects, studies into 3D geo-data constraints are largely unexplored. This thesis research discovers a new approach to model, conceptualise and implement 3D geo-constraints which can function in the database. At the outset, constraints can be formulated using natural language. As natural language is subjective and varies between individuals, expressions can be ambiguous and can easily cause confusion. So spatial constraints are abstracted using geometry that depicts the exact shape, and also topology that reveals the spatial relationship between geometries. This step makes the meaning of a constraint clearer to others. Furthermore, using standardised UML diagrams and OCL expressions, geo-constraints can be formalised to an extent that not only humans, but also machines can understand them. With model-driven architecture supported by various softwares, OCL expressions can be automatically converted to other models/executable codes (e.g. PL/SQL) just by a few clicks. And with small modifications, database triggers can be formulated to carry out constraints check.

A database including various topographic objects (e.g. buildings, trees, roads, grass, water-bodies and terrains) is used as a study case to apply the discovered approach. During this research, a first attempt to formulate 3D geo-constraints in OCL has been made. These expressions can be tested and translated to other models/implementations when the OCL standard is extended with spatial types and operations.

In the implementation stage, the current 3D functions in Oracle Spatial database are found to be insufficient. A new 3D function using existing 2D functions - plus additional code relating to computational geometry - has been developed by the author to bridge the gap. Based upon this function, a large group of spatial constraints which apply to objects in 3D space can be checked.

Bentley Map and Python IDLE are used to test the performance of constraints as well as the visualisation of warning messages to clients. Database error messages are immediately displayed on the front-ends when a modification that does not satisfy a constraint is attempted to commit to the database.

During the case study, new classes of constraints are also discovered. They are higher-level constraints, parameterised constraints, constraints allowing exceptional instances, extra-check rules to detect conflicting constraints and constraints relating to multi-scale representations.

SAMENVATTING

Het geo-informatie vakgebied ondergaat grote veranderingen met de overgang van 2D naar 3D representaties. Een aanzienlijk deel van de 3D-datasets worden opgeslagen in databases. Het is belangrijk dat nieuwe controlemechanismen worden ingebouwd om kwaliteit en consistentie te garanderen.

Geldigheidscondities (constraints) zijn een effectieve oplossing om data fouten te voorkomen en zo de kwaliteit te bewaken. Constraints voor 2D topografische datasets zijn al eerder onderzocht, maar voor 3D data is dit nog niet gebeurd. Het huidige onderzoek gaat hierop in: van conceptualiseren tot implementatie. Er wordt een methode met vier fase voorgesteld. In de eerste fase worden de constraints geformuleerd met behulp van natuurlijke taal. Natuurlijke taal blijkt soms dubbelzinnig en kan leiden tot verwarring. Dus in de tweede fase worden de constraints verder geconceptualiseerd met behulp van goed gedefinieerde geometrische primitieven en hun topologische relaties (volgens ISO 19107, spatial schema). Deze stap maakt de betekenis van een constraint duidelijker voor anderen personen dan de auteur zelf. In de derde fase worden met behulp van Object Constraint Language (OCL) expressies bij de UML-diagrammen, de constraints geformaliseerd zover dat niet alleen mensen, maar ook machines deze kunnen ze begrijpen. In de vierde fase kunnen dan, volgens de model gedreven architectuur (MDA) aanpak, de OCL expressies automatisch worden omgezet naar uitvoerbare programmatuur. Bijvoorbeeld PL/SQL code die triggers en procedures definiëren voor het uitvoeren van de constraint controles in de database.

Een CityGML gebaseerde database met verschillende 3D topografische objecten (gebouwen, bomen, wegen, gras, water en terrein) wordt gebruikt als een case study om de hierboven beschreven methodiek toe te passen. Hiervoor is eerst een uitbreiding van OCL met 3D ruimtelijke typen en operatoren (volgens ISO 19107) nodig. Er is vervolgens een eerste poging gedaan om een flink aantal 3D geo-constraints in OCL te formuleren. Deze OCL uitdrukkingen kunnen worden vertaald naar database omgeving (met dezelfde ISO 19107 uitbreiding) en worden getest met echte 3D data.

Tijdens de implementatie bleek dat de huidige 3D-functies in de Oracle Spatial database nog onvoldoende zijn. Daarom is een nieuwe 3D-functie ontwikkeld voor het vaststellen van topologische relaties in 3D. Op basis van deze functie kan een grote groep van 3D constraints worden gecontroleerd. Bentley Map en Python IDLE worden gebruikt om de implementatie van de constraints te testen. Database foutmeldingen worden weergegeven in de front-end (GUI) als na een edit-operatie de wijziging niet voldoet aan een van de constraints.

Gedurende dit onderzoek zijn ook een aantal nieuwe klassen van constraints ontdekt: 1. constraints op een hoger abstractie niveau (dus bijvoorbeeld op niveau van de generieke klasse city-object en daarmee impliciet ook van toepassing op specialisatie klassen als gebouw en boom), 2. geparameteriseerde constraints (afstand tussen gebouw en boom is ten minste X meter), 3. constraints die zorgen voor consistentie tussen de verschillende detailniveaus (LoD's in CityGML), en 4. goede oplossing voor 'moet' (harde) en 'zou' (zachte) constraints, bij de laatste is het mogelijk dat de gebruiker uitzonderingen toe staat door de betreffende instanties expliciet te markeren. Tot slot is er in het onderzoek aandacht besteed aan het ontdekken en voorkomen van tegenstrijdige constraints, welke anders als gevolg zouden kunnen hebben dat (delen van) de database leeg blijven.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

9IM Egenhofer 9 Intersection Model

CAD Computer Aided Design

CCC Climate City Campus

DBMS Database Management System

DDL Data Definition Language

DML Data Manipulation Language

EA Enterprise Architect

FME Feature Manipulation Engine

FOL First Order Logic

GML Geography Markup Language

GUI Graphic User Interface

HOL Higher Order Logic

IDLE Integrated Development Environment for Python

INSPIRE Infrastructure for Spatial Information in Europe

ISO International Organisation for Standardisation

LOD Level Of Detail

MBR/MBB Minimum Bounding Rectangle/Box

MDA/MDG Model Driven Architecture/Model Driven Generation

OCL Object Constraint Language

OGC Open Geospatial Consortium

OMG Object Management Group

PDA Personal Digital Assistant

PL/SQL Procedural Language/Structured Query Language

TIC Terrain Intersection Curve

UML Unified Modelling Language

XML Extensible Markup Language

1

INTRODUCTION

The aim of the Campus-Climate-City project is to turn the TU Delft campus into a showcase for multi-disciplinary environment/weather research, and one of the best monitored and understood campuses in the world. To help achieve this goal, the in-campus objects that have a strong influence on the local climate, such as buildings, trees, sensors (stationary and mobile), land (e.g. grass fields, water bodies, roads) and terrain are modeled and stored in a database (Oracle Spatial 11g) by a group of students [Geomatics, 2010]. Buildings with certain geometries block the wind and thus change the wind flow conditions in the neighbourhood. Trees create shadows and water vapour, and reduce the speed of wind that goes through its crown. Sensors that record observations, such as humidity, wind speed, carbon dioxide emissions, temperature and rain fall, provides datasets for climate study. Land covered by different ground materials has various stiffnesses and reflectivity coefficients and thus absorbs and emits heat differently. And with the fluctuation of terrain, the ground water flow can be simulated to find the converging place (possibly pond) of rainfall.

With these objects and their climate-influential attributes stored, the database plays the role of a data warehouse (or even a key element of Campus Spatial Data Infrastructure) to store, retrieve and exchange the information of in-campus objects for climate study, e.g. simulation and analysis [Geomatics, 2010]. It will receive input from and give output to a multitude of users that use different kinds of models. Many of the users are not geo-experts and do not control the data quality and integrity at the same level. Any inconsistency or inaccuracy that initially does not seem to be crucial, when retrieved by other users, may turn out to be harmful later on. For example, it maybe enough for someone who is looking at air humidity to know that the sensor is near a water body. But for someone wanting to simulate wind speed, a more accurate distance from this wind speed sensor to the surrounding buildings is required. In order to keep the quality and consistency of dataset, and avoid the conflicts between different levels of accuracy requirements, the demand on data maintenance is high.

Constraints set up rules that must never be violated and thus are used to maintain the data integrity. Literature study on this subject has revealed that many research projects into constraints are still at a theoretical level, such as modelling and conceptualising. Very few implementations at specific GIS application domain have been carried out. And most of them are in 2D or 2.5D. On the other hand, 3D city/urban models are becoming increasingly popular and the need for 3D data models is still growing. If 2D models have already created such a big margin for errors and mistakes, how about adding one more dimension? Some examples of constraints within the 3rd dimension of in-campus objects are: different buildings cannot overlap; tree branches cannot enter buildings; static sensors which are installed in the weather station cannot leave the station; mobile objects, once the route is defined as outdoor (or indoor), cannot suddenly 'jump' into (or out of) the building; each building should have its address

match to the name of a street nearby (i.e. distance < 5m) and so forth. Obviously, there is a big difference between the popularity and reliability of 3D data. An extension of current constraint theory from 2D to 3D would be a great help.

Last but not least, the tools needed to implement constraints are available. For example, the concept schema (UML/OCL) can be drawn in *Enterprise Architecture*. The execution of constraint can be done by database triggers. And even open source toolkit (OCL2SQL) to generate SQL triggers, which was further extended with 2D 9 intersection model, has been developed.

Therefore, with the fundamental theory presented, technical tools available, and user needs pulling, a thesis to design and implement 3D constraints for the campus data model is required. Hopefully this thesis will demonstrate how to handle challenging issues with regard to the quality of 3D city models.

1.1 MOTIVATION

Constraints can be part of the definition of the object classes, types, functions and their relationships, and the enforcements are not specified. For instance, when a telephone number is defined as type integer, string will not be accepted. A road that is modelled as polyline(s) cannot be assigned by point objects. More examples can be found in some general geographical models such as Geographic Markup Language (GML), X3D, and Industry Foundation Classes (IFC) [Werder, 2009].

Some constraints have extra definition and have specific enforcements. In order to define the constraints, the classification of typical constraints has been studied. The main types of traditional (explicit) constraints include domain constraints, key and relationship structural constraints, and general semantic integrity constraints [Elmasri and Navathe, 2003]. These classical types were then extended by [Cockcroft, 1997] with new ones: topological, semantic and user-defined constraints, which were again elaborated by [Louwsma et al., 2006], and then proven by [van Oosterom, 2006] with broader context. There are different ways to conceptualise these constraints.

A commonly adopted approach suggested by [Casanova et al., 2000] [van Oosterom, 2006] [Duboisset et al., 2005] [Pinet et al., 2007] [Louwsma et al., 2006] is Object Constraint Language (OCL). OCL is a textual language used to describe the constraints applying to objects, and is part of Unified Modelling Language (UML) which is a preferred concept modelling schema. Beside OCL, there are also other ways to express constraints, such as the Constraint Decision Table (CDT) by [Wachowicz et al., 2008], meta data by [Cockcroft, 2004], ontology by [Mas et al., 2005]. OCL formalisation of constraints will be discussed in section 3.3.3 and section 6.1.

Unlike the rich results of theoretical study, literature gives few examples of implementing constraints in the GIS application domain. Some of them are found in a landscape design system [Louwsma et al., 2006], field data capture system [Wachowicz et al., 2008], an agricultural information system [Pinet et al., 2004], a cadastral data maintenance system (TOP10NL) [van Oosterom, 2006], and a traffic flow control system [Reeves et al., 2006]. These examples only deal with 2D constraint. So far, no research on implementing 3D constraints within specific GIS application has been found.

3D GIS has been fast exploding and drawing much attention. Different 3D models, e.g. CAD, KML, GML3 so forth have recently emerged to depict the events and objects in the real world. One of the most quickly adopted and commonly used models is CityGML. It is an open data

model and XML-based format for the storage and exchange of virtual 3D city models. More information about CityGML will be given in section 2.1.

As the spatial model evolves from 2D to 3D, more and more database vendors have their products supports 3D data storage. An example is Oracle. Since Spatial 11g, these being to support 3D object data types, that is 3D (multi) points, (multi) lines, (multi) polygons and (multi) solids. 3D data geometry can be validated according to GML 3.1.1 and ISO 19107 specifications, e.g. geometry validation via `SDO_Geom.Validate_Geometry_With_Context` [Ora, 2010b]. With the hierarchy of single objects and composite objects, it enables storing multi-LODs from CityGML, which can model regions, city/site and architectural interiors. 3D queries concerning 3D visibility, volumetric analysis and spatial/semantic attributes are available in 11g [Ravada, 2008] as well.

Since the real 3D world contains more information than 2D and can be modeled in different ways, the issue of data consistency and integrity in 3D is more sensitive. With SQL assertion, which is specified in SQL-92 standard, the constraint can be coded in a machine-understandable way [van Oosterom, 2006] [Louwsma et al., 2006]. However, despite the undoubted value of assertion, the status that no mainstream DB vendors support direct assertion implementation remains the same since [van Oosterom, 2006, 2006].

Triggers are found to be a proper alternative to implement the constraints. They are used to monitor a database and take action when a condition occurs [Elmasri and Navathe, 2003]. They are described using procedural code of SQL syntax and act upon particular tables or views of a database. More information will be shown in section 2.3.

Nevertheless, in spite of the fact that 3D GIS is fast growing and database is able to implement constraint, so far very little research concerning 3D constraints in the database has been carried out. Some research regarding the validity of geometry [Kazar et al., 2008] and efficient 3D data representation [Arens et al., 2005] has been presented at the conceptual and implementation level. The results of these research projects will be borrowed to establish the 3D constraints implementation within the database in this thesis work.

1.2 RESEARCH QUESTIONS

The final goal of this thesis is *to design, conceptualize and implement constraints for 3D models of the City-Campus-Climate project*. Based upon this final goal, research questions will be:

To design necessary constraints regarding the spatial-temporal-semantic relationships between the objects. Basically, these objects can be grouped as static 3D campus objects and mobile sensor objects:

- 1. Which 3D and sensor objects are needed?
- 2. Which spatial, thematic, temporal and quantity constraints (according to classification of [van Oosterom, 2006]) are necessary?

Once the raw constraints are discovered, the next concern is to conceptualize them:

- 3. What is the best way to formalize these constraints?
- 4. Which tools/software are the most suitable for this work?

To implement the constraints in DB environment. Concerns will be discovered such as:

- 5. What is the best way to implement those constraints in database?
- 6. Which database, PostGIS or Oracle, performs better in realise 3D constraints?
- 7. Is it possible to have some visual feedback to report any error?

1.3 METHODOLOGY

To approach the final goal, the first step is to discover which constraints are relevant. Studying literature to find constraint examples is meaningful at this stage. A close examination of the objects/attributes/relationships is also necessary to find out where errors can come in. The classification method will be inspected so that any discovered constraints can be better structured. This can be seen as a pre-work for specification. After that, different specification and formalisation methods/languages will be checked out, which in turn can help express the constraints in a formal manner. The tools that can help formalisation will be studied as well. The next step is to implement the constraints in the database. On one hand, it involves an in-depth inspection of the spatial part in Oracle Spatial, e.g. 3D geometry functions and spatial objects storage. On the other hand, the mechanisms of the database also need to be studied closely, which will in turn enable the constraints to be checked in run-time, and allow message generation and visualisation. Finally, some front-end clients that are able to visualise error messages from the database should also be tested.

1.4 CHAPTERS OVERVIEW

Chapter 1 is the introduction of this thesis, which formulates the problem setting, motivation, research questions and approaches. In chapter 2 background information gives typical tools/software in aspects such as 3D object modelling (CityGML), a formal representation of conceptual models (UML), databases for both storage of 3D objects and implementation of constraints (Oracle Spatial) and 3D visualisation (Bentley Map). Chapter 3 discusses constraint expression in natural language, a classification of constraints, as well as some implemented constraints in various applications and their approaches found from articles. Chapter 4 analyses the objects, their attributes and relations in the CCC database, which is a pre-step to constraints discovering. Chapter 5 takes a closer look at the constraints needed for the datasets and the approach to express them in a more specific manner with geometry/topology. Based upon the more specific expressions, attempt is given to formulate constraints in an OCL-like manner in Chapter 6. Chapter 6 also discusses an open issue of translating OCL into database language - SQL. Chapter 7 gives the implementation details of several constraints in Oracle database using triggers and procedural code. A new 3D function developed to convey detailed surface topological relationships is also described in this chapter. The visualisation of constraints are tested in Bentley Map and Python IDLE. The snapshots are given in chapter 8. Chapter 9 contains the conclusion, and discusses future work and recommendations.

2

BACKGROUND INFORMATION

To carry out this research, a few tools are important, regarding the object modelling, constraint implementation and visualisation of 3D object. Section 2.2 shows how UML can conceptualise modelling of city objects/classes. Oracle Spatial as a database that store 3D models (as was introduced in section 1.1) and provide means to maintain cleanness of data will be discussed in section 2.3. Bentley map as a graphic CAD/GIS software that can visualise and edit 3D objects is also talked about in section 2.4.

2.1 CITYGML

As an extensible application schema of Geography Markup Language 3 (GML3), CityGML is issued as a standard by OpenGIS. A rich amount of both geometry and semantics information of urban objects could be modeled by CityGML core model, e.g. terrain, building, water body, transportation, vegetation, land use and city furniture (like lamp, electricity pole etc.). For example, trees as city vegetation object not only have the coordinates of their branches and trunks, but also have attributes such as class, function, species, height, trunk diameter and crown diameter. The model will be more meaningful and realistic, if rule as 'tree cannot float in the air but attached on the terrain surface' is given. Since the core modules of CityGML is designed by Unified Modeling Language, it's easy to extend with new classes and attributes, e.g. Application Domain Extensions [CityGML, 2010]. Some extensions have been implemented in concept of CityGML, such as geological features [W. Tegtmeier, 2003] and building information modeling(BIM) [van Berlo and de Laat, 2010].

CityGML also supports multi-resolution modelling with Level Of Detail (LOD), which is applicable from large areas to small local regions and to increase/decrease the complexity in representing 3D objects. Taking the building model as an example (see figure 1). LOD1 building is simply an extrusion of 2D polygon (like block or solid), so the roof and walls are completely flat. LOD2 building has more detailed roof and wall representations (like curve, sharp tower top or any other kind of shape). LOD3 building allows to view the openings, like window and door, and texture of the exterior. LOD4 building gives the furniture, ceiling, room wall and floor in the building interior. Once LODs are involved, the same object can have geometry represented in different detail levels, thus will lead to different definitions on possible/impossible topological relations. For example, a block with flat roof top in LOD1 does not overlap with some other structures overhead, but actual roof in LOD2 would form a spiky tower and overlap with other structure. Section 5.1.4 has more illustrations and discussion about LOD issue.

2.2 UML

Unified Modelling Language (UML) is a standardised modelling language for object-oriented system development. It offers diagrams that specify, visualise, modify, construct, implement, document and analysis the objects and their relationships.

The urban environment is a complex environment contributed by many types of objects/attributes and relationships. When modelling them to the digital system, e.g. database system, class diagram is needed. Class diagram describes the structure of a system by showing the system's classes, their attributes, operations/methods and the relationships amongst the classes. A class is a set of identified objects sharing similar properties. The description in diagram gives the name of class, its subclasses and superclass, instances and their attributes.

The relationships in UML that are useful to for city modelling are generalisation, association, aggregation and composition.

- Generalisation: indicates one class (the sub-class) is taken as a specialised form of the other (the super-class) and the super-class is a generalisation of sub-class. In practice, an instance from the sub-class (sometimes called child) is also an instance of super-class (parent). For example, a house is a type of building; an oak is a type of tree. Moreover, the sub-class can have properties (inheritance) from super-class.

- Association: depicts the static relationship shared amongst the objects from two classes. It can work between two objects or amongst multiple objects. Binary associations can be drawn as a line, with each end connected to an class box. Example 'a professor gives lectures' is a binary association relation. In terms of reading order, association have several different types, bi-directional and uni-directional are the most commonly used. A bi-directional association can be read from both ends, whilst uni-directional must be read in a particular direction. An example of uni-directional relation is 'A person owns a house as his property' or 'A house belongs to an owner'. The marriage relation 'married-to' between a couple applies to both ends (husband and wife) and thus is a bi-direction association. An association can be named, and its ends can be annotated with multiplicity, visibility, ownership indicators and other properties.

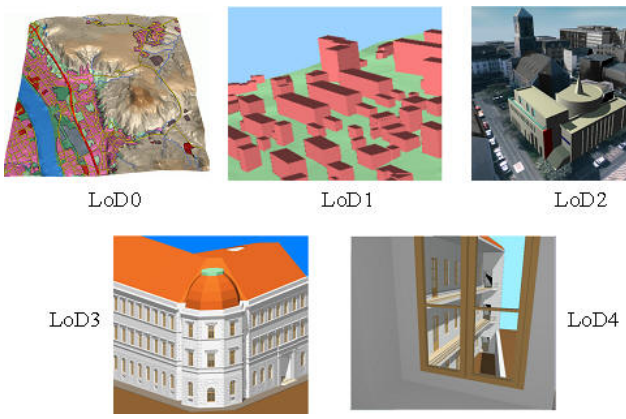


Figure 1.: Building models in different LODs in CityGML from [Kolbe et al., 2009]

- Aggregation: denotes a binary association, in which one side means 'the whole' and the other side 'a part'. An example as such is 'an audio system can be aggregated by CD player, record player, tuner, headphones, speaker and amplifier'.
- Composition: is a sub-type of aggregation but more emphasises the 'owns a' relation. The existence of 'a part' is dependent on the existence of 'the whole'. For instance, a car has an engine is more appropriate to be modelled by composition because a car cannot function without an engine. The audio system example above is better illustrated by aggregation because the individual components of the system can exist and function well without the system.

As presented in figure 3, [Borrebæk and Myrind, 2005] gives a formal topographic modelling schema which defines separate classes for geometric and thematic/temporal attributes (originated from General Feature Model [ISO, 2005]). In the work of [Alberto Belussi, 2004] this method is reformulated to describe the geometric and non-geometric attributes in one class. In figure 3 the spatial attributes of the two feature classes are described as attributes of the classes with geometric domain: both the geometric types have a prefix 'GU_'. The UML *<< BelongsTo >>* stereotype transforms the 'RoadInNetwork' association into a 'BelongsTo' spatial association.

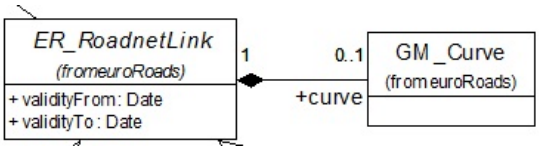


Figure 2.: A proposed UML application schema to model roads (from [Borrebæk and Myrind, 2005]). The geometric attribute and thematic/temporal attribute are modelled in separate classes.

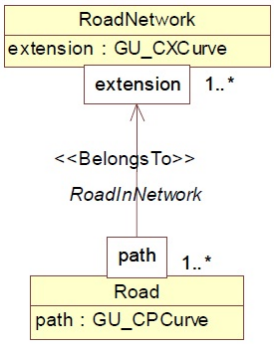


Figure 3.: A proposed UML application schema to model roads from [Alberto Belussi, 2004]. The spatial attribute of a feature type is described within the class.

2.3 ORACLE

As was introduced in the section 1.1, Oracle Spatial 11g shows a growing interests in 3D database. The other characteristic of interest in this thesis about Oracle is its database mechanisms to allow the realisation of constraints. Two mechanisms are important for this research, exception (or run-time error) and trigger.

- Exceptions: can occur because of design faults, coding mistakes, hardware failures, and many other sources [Ora, 2010a]. The user-defined exception (`Raise` and `Raise_Application_Error`) declared in subprogramme or package can be raised when a constraint is found violated.
- Trigger: can react upon a certain event happening in run-time. It can call the procedures (which is coded with user-defined error) to enforce complex business rules, like geographic constraints. Whenever a trigger causes a run-time error, the whole transaction the triggering statement is executing is cancelled, and thus keep the existed data clean.

A simple trigger can fire before or after the data modification event, and whether it runs for each event (statement-level trigger) or for each row (row-level trigger) affected by the event. If two or more triggers with different timing points are defined for the same statement on the same table, then they fire in this order:

- All BEFORE statement triggers
- All BEFORE row triggers
- All AFTER row triggers
- All AFTER statement triggers

When something causes an error, the control in executable part of PL/SQL block stops and then is transfered to exception-handling part. After an exception handler runs, the message is given and control goes to the next statement of the enclosing block. If there is no enclosing block and the exception handler is in a subprogramme (e.g. procedure), then control returns to the invoker, at the statement after the invocation.

One property about trigger is that if a table-trigger involves modifying more than one table, when an error is raised all modifications are cancelled. One other issue to notice is the place to have COMMIT command when DML inside a loop may cause run-time error. In practice, when a DML statement is inside a loop and **COMMIT** command outside, raising of error will roll back the effect from the whole loop. That is, even the correct modification will have to be rejected along with the incorrect.

To avoid the unexpected rejection of correct records, **COMMIT** command should be placed inside the loop so that DML effect is realised one row after the other (see the SQL statement and error message in table 4 from section 7.2.4). One must beware of this property when modifying a set of records. If these records should be seen as elements of one master object (like surfaces that enclose and form a building), it is recommended to place COMMIT outside the loop. If they should be treated separately (e.g. observations from different stationary sensors) then COMMIT should be placed outside the loop.

A scenario of possible data change is that somebody updates the geometry of a building surface. A trigger checks the possible intersection between this surface geometry and other city

objects. Objects in the local neighbourhood (e.g. the trees planted nearby, road that is on the side of the building, and other adjacent building) are most likely to be intersected by this new geometry. To find out these neighbours, a select query should be carried out. But a row-level trigger prevents the querying of the table that the trigger statement (here it is UPDATE) is modifying. One of the adoption is to use compound trigger. The other is to use INSTEAD OF trigger for views, which is not affected by the mutating-table restriction.

One thing to notice is that outside run-time error messages, DBMS_Output package in Oracle database system enables generation of more flexible messages, [Database, 2010]. The package is typically used for debugging, or for displaying messages and reports. The analysis of topological relation, warnings of suspected violation (e.g. the 'should' cases) and any detailed information outside of error message are visualised and retrieved by these functions.

2.4 BENTLEY MAP

Bentley Map is a desktop CAD/GIS software which is used to map, plan, design, build and operate infrastructure. It enhances the MicroStation capabilities in precise geospatial data creation, maintenance and analysis. An important feature when it comes to 3D city modelling using Bentley is that it supports native 3D objects in Oracle Spatial and has smart 3D object editing tools [ben,]. Thus it is often used to visualise and modify the 3D objects that are in Oracle database. One can connect to Oracle via Bentley Map connection (without registering a project) or via Bentley Geospatial Administrator, which uses XML feature modelling (XFM). The latter provides a more interactive graphic application to view the tables, views and attributes that store 3D city models. A feature-locking mechanism ensures a consistency of data under multi-user environments.

In the graphic user-interface of Bentley Map, Oracle error message is visible. One can edit the geometry of an object in the GUI and then 'post' it back to Oracle. This 'post' is equivalent to an 'update' statement in the database and trigger can react upon it. In this manner, when constraint check is coded in trigger, a 'post' of incorrect data modification will launch it and error message of database will be shown in 'Message Center' of Bentley Map. So one will beware of the rules defined in database system.

3

RESEARCH ON CONSTRAINTS

Various research project concerning geographic constraints can be found when studying literature. Although hardly anything is said about 3d geo-constraints, references to general constraints or geo-constraints in 2D/2.5D do exist and they pave the way for extending the current state-of-art of constraint into a higher dimension. Some aspects of expressing constraints in natural language are found in section 3.1. Two ways of stating the same constraints are introduced and a general rule is presented to keep the constraint statements brief. And exceptions to specific constraints will introduce a new class of constraints. Both ways will be addressed in the expression of 3D geo-constraints. Section 3.2 talks about classifying geo-constraints which can group the constraints into a clear structure. And different methods of modelling constraints under some applications are reviewed in section 3.3, in which OCL is found to be the most suited geo-constraint modelling tool.

3.1 CONSTRAINTS IN NATURAL LANGUAGE

Defining geographic constraints by natural language is a challenging task. Natural language is not formalised as is machine language and it is very much dependent on the individual. The words that a person uses to describe the same phenomenon can be very different from the words used by another person. And a word in natural language can mean different things. Not every rule can be expressed as clearly and precisely as some non-geographic rules can, e.g. 'a book from the library can only be lent to one person at a time', or 'the grade of a supervisor must be higher than his supervisees' grade'.

Take the statement 'a road cannot cross a building' as an example. In the real world, there are innovative architectural designs where a building appears to be crossed by a road (see figure 4(a)).

Here, three things need to be clearly defined, 'what is a building?', 'what does cross mean?', and 'what is a road?'. Under the definition:

- A building: is a structure consisting of a set of surfaces that divide the whole space into buildings interior and exterior;
- A road: is a path that allows vehicles to travel along it at a certain speed;

The event this rule means to forbid is rephrased below. Notice the 'through hole' is not on the building 'interior'.

"A road that has a part in the building 'exterior' has the other part entering into the building 'interior', and thereby allows the car to pass through the rooms (imagine a traffic light flashes

in your office)".

Then this figure is not a problem, because the 'hole through the building' is on the exterior and the road does not go through any room in the building (see figure 5).

(a) The road appears to 'cross' the building.

(b) The building appears to be floating in mid-air.

Figure 4.: Conditions that are difficult to describe using natural language.

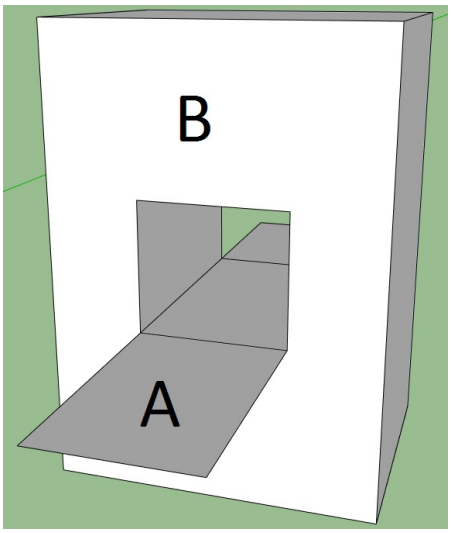


Figure 5.: Geometric model for figure 4(a). A is a road surface 'through' the hole in solid building B. In this case the rule 'road A cannot cross building B' is fulfilled (i.e. TRUE).

Table 1.: Comparison between forced and restricted forms of constraint

Forced statement	Restricted statement
grass that belongs to roof greening must be placed on the roof of a building	grass from roof greening cannot be laid on the ground/underground, on any of: road, water, bridge, park, sport fields, and so on
a road must always go around, above, below, etc. a building	a road cannot cross a building
two trees must be placed with more than 2 meters in between	two trees cannot be placed at a distance of less than 2 meters
any piece of land must have a land-use type	a land cannot be without a land-use

The other example is the relation between building and terrain, 'a building should not float in the air'. For the building that stands on poles, this statement is not very clear (see figure 4(b)). The main body of the building, in other words the rooms (space) that are (is) lived in by people and enclosed by walls, roof and floor appear to be a certain distance above the ground. Someone who interprets the building's location by its rooms' location may perceive it as a floating building. Therefore it is difficult, if not impossible, to avoid confusion when formulating constraints merely by natural language.

3.1.1 *Forced and Restricted Specifications*

There are basically two ways to formulate a constraint in natural language, forced (positive: must be / always have to) and restricted (negative, cannot / must not) [Louwsma, 2004]. Some constraints stated in one way would appear to be much more clear and effective than stating them in the other. For example, the condition 'grass always has to be green' has the same meaning as 'grass cannot have the colour black, white, gray, blue, red, yellow, purple'. The former is a forced way and is much briefer than the latter, the restricted. As you may notice, what matters is the number of defined situations/values. 'Grass colour should be green' only defines one situation, whilst the other way defines many situations. When using the latter, it is also more likely to forget to mention a possibility that must be constrained (e.g. 'orange'), not to mention that there are more restricted cases in the real world than the capacity of enumeration (e.g. more unnamed colours that grass cannot have).

Table 1 gives a comparison of constraints formulated in forced and restricted ways. In example 1, the description is much briefer in the 'forced way' than in the 'restricted way', whilst in example 2 it is vice versa. In example 3 and 4, the two sides do not have much difference in complexity. In order to reduce the complexity of constraint specifications, a constraint design principle is that:

'all constraints should be compared using both methods. The method that requires fewer values should be used'.

3.1.2 *Severity and Exception*

A city can be a complex environment with various kinds of phenomena. Some conditions do not have to be strictly forbidden for the whole city. They may in general be unusual or irregular but in certain cases can nevertheless happen. For instance, in most cases, the condition 'a tree and a building have at least 2 meters distance between them' is satisfied. But in some specific instances, the tree just leans on the wall or stands closer to a building than the allowed distance. It is good to reveal the common distance relation between building and tree and keep the distance, but it is difficult to give a clear-cut rule which describes all situations that have to be rejected.

Exceptions are therefore introduced to make constraints more usable and realistic. Instead of using 'must be' or 'cannot have' and the exact value, this example distance rule can be stated using 'should be' as:

'a tree and a building should be 2 meters apart'.

The 'should be' constraints can accept violating instance when the specific instance is explicitly marked as an 'exception'. In other words, the enforcement of 'should be' constraints is the same as 'must be'. The only difference is that 'should be' can be by-passed when an instance is exceptional. This idea is implemented, as will be described in section 7.2.2.

In order to broaden the results to more generic city models, constraints within this thesis will be divided into two groups, 'do not accept' (i.e. never accept) and 'accept based upon the exception'. The former group consists of constraints that in no way can be excused (see also constraints in the column 'Do not accept' in table II from appendix A.2). The latter group consists of constraints that allow exceptions (see column 'Unusual but accepted' in the same table). In expressions of natural language, the forbidden constraints will be stated with words such as 'must/must not' and 'can/cannot', and the excusable constraints will use words 'should/should not', 'may/may not' and 'is better to be'.

3.2 CLASSIFICATION OF CONSTRAINTS

A single real world object can be described by its geometry, property and appearance in the computer system.

- Geometry: (sometimes also called 'spatial property') indicates an object's location, orientation, size, shape, etc. that can easily be seen by eyes.
- Property: (or non-spatial property) describes the function, usage, meaning, class and so forth. It can be further categorised as:

  Thematic: to specify the meaning of an object and what it stands for (or how to classify and name an object in different application themes).

  Temporal: to describe, represent, reason about time factor. A temporal property can mean a point or a period on the time line.
- Appearance: or visualisation properties, namely texture, colour, lighting and etc., is how close the depicted digital model is to the real object.

For two objects or multi-objects a relationship often delivers useful information. Spatial relationships, which includes topological relationships and metric relationships and temporal, quantitative relationships will be discussed in the next subsection.

Depending on how many objects are involved, objects' relations can be binary relations (two objects), or multiple relations (more than two objects). The object relationships explained in this section only deal with binary relationships and are expressed outside the description of individual classes. The object relationships can also cover aspects of spatial (local neighbourhood), thematic/semantic, temporal, and quantitative properties.

3.2.1 *Spatial Relationships*

The spatial relationship unveils the interrelation and connection between objects in the geometric round. Not only the absolute position of an object is known, e.g. longitude, latitude and height; but also the relative location, namely how it relates spatially to its local neighbours. According to [Egenhofer, 1989], spatial relationship can be formalised by:

Topological relationships: invariant regardless of topological transformations, such as scaling and rotation. Disjoint and neighbour are examples as such. Spatial order and strict order relationships: rely on the definition of the order. For example, *behind* is a spatial order relation with the converse relationship *in front of*. Likewise, *below* is a spatial order relation reversed w.r.t. *above*.

Metric relationships: exploit the existence of measurements, e.g. distances and directions. For instance, '5 meters above the ground' is a distance measurement and 'in the south of city A' is a directional measurement.

Topological and metric relationships are described with more details below.

- Topological relationship
  A commonly seen topology model within the geo-information models uses topological primitives, interior, exterior and boundary to describe the interrelation between objects. The interior, exterior and boundary of object A are denoted as $A^\circ, A^-, \partial A$ and those of object B are $B^\circ, B^-\partial B$, respectively. The combination of pairwise intersection results in 9 intersection as:

$$R(A,B) = \begin{bmatrix} \mathrm{A}^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{bmatrix}$$

  The 9 intersection model can be used for both 2D and 3D. There are many possible relations and some are impossible to be described or distinguished by the 9I model and so they are not discussed here. Figure 6 shows an example that cannot be distinguished by 9I. However, according to [Eliseo Clementini and van Oostero, 1993], the dimension of intersection can be extended from the 9I model with different dimension values in each set. Theoretically this gives 59 topological relationships, although many of which will not exist in reality.

  - $\varnothing$: empty intersection.
  - 0D: point intersection.
  - 1D: line intersection.

- 2D: surface intersection.
- 3D: volume intersection.

Therefore, the two cases in figure 6 can be expressed in 2 different matrices as (first matrix shows the case in the left 'point intersect' and second matrix shows the case in the right 'line intersect'):

$$R(A,B)(left) = \begin{bmatrix} \oslash & \oslash & 2 \\ \oslash & 0 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

$$R(A,B)(right) = \begin{bmatrix} \oslash & \oslash & 2 \\ \oslash & 1 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

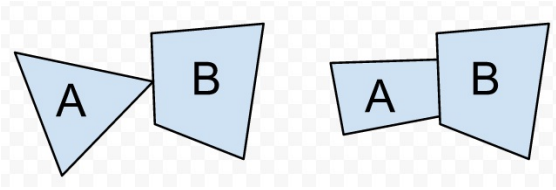


Figure 6.: The two cases cannot be distinguished by the 9I model but can be by the Dimension Extended Method (DIM).

The commonly seen terms (in 9IM) 'disjoint', 'overlap', 'meet' (or touch), 'inside', 'equal', 'cross' between 2D object A and object B can be defined by the following topological primitives (see figure 7):

1. If the interiors and boundaries of objects A and B have no intersection at all, then their topological relation is **disjoint**.

2. If part of the interior and boundary of object A intersects with part of interior and boundary of object B, then these two objects **intersect**.

3. If the boundary of object A intersects with boundary of object B, without any intersection in the interiors, then their relation is **meet (or touch)**.

4. If the complete interior and boundary of object A intersects with only the interior of object B, then object A is **inside** object B.

5. If both intersections of boundary and interior are not empty whilst the two boundary-interior intersections are empty, then the relation is named **equal**.

6. If two objects have common interior and no intersection in the boundaries, then they have **cross** relation. The cross relationship only applies to line-line and line-area situations.

Egenhofer [Egenhofer, 1995] give definitions of terms disjoint, contains, inside, equal, meet, covers, coveredBy and overlap to identify solid-solid relations in figure 8.

- Metric Relationship
  Metric relations depict measurements in space, such as direction and distance. These

relations need a referential origin and scale. The distance is a value measured between two objects by a scale (or unit). In addition to units, a ratio scale of distance can be specified by 'closer than', 'further than', or 'between'. For example, a railway is narrower than 5 meters, a tree is 'taller' than 2 meters, and a dike is 'in between' 2 to 5 meters above the sea level. Or a town hall is 'close to/far from' the city center can also give an additional hint about the distance between the two objects.

Directional relation is the position of an object in comparison to another object. The reference can be to an object in the relation, or to some origin defined within a certain coordinate reference framework. It also requires an azimuth, e.g. in degrees in the range of $[0°, 360°]$, or a description in natural language, North, South, East, West, or a combination of these four primary words. Since 3D objects have height information, the relationship 'below'/'above' is useful in order to specify the relative position in the third dimension.

For point objects it is easy to identify the directional relation and distance. For non-point objects, the directional relation is calculated by the centroid of the objects. The centroid has to be clearly defined, e.g. for a circle object it is the point equidistant from the points on the edge. And the closest distance from any point in one object to any point in the other object is the distance between these two objects.

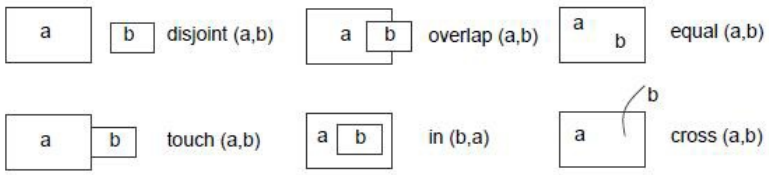


Figure 7.: Topological relations in 2D

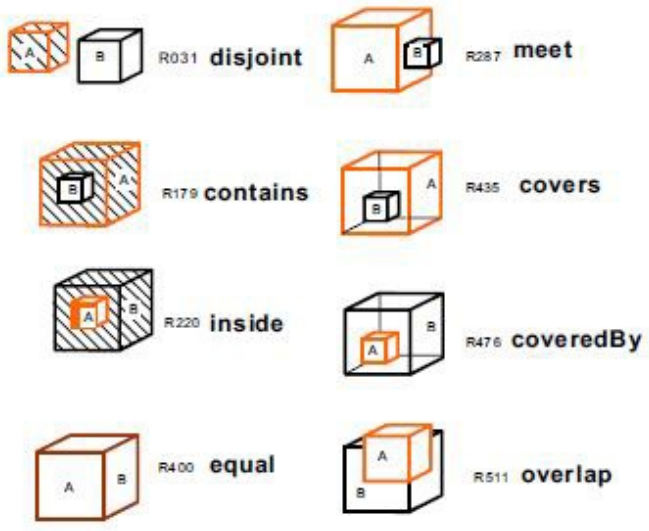


Figure 8.: Topological relations between two 3D solids

3.2.2 *Temporal Relationship*

A time can be either described by a timing point, e.g. the moment when an event takes place or a period of time (time interval), e.g. how long the event lasts. A timing point can be 'before', 'after' or 'the same as / equal to' the other timing point. For a relation between a timing point and a period of time, the timing point can be before the period starts, after the period finishes, at the beginning point of the period, during the period or at the ending point. [Kwon et al., 1999] defined a prototypical model to describe temporal relations between two time intervals (periods). Given two intervals, there are seven distinct ways in which they are related. These relations (known as Allen's relations) are: *before, meets, overlaps, finishes, during, starts* and *equals* (figure 9). To make the time point more clear, the terms 'finishes' and 'during' can be replaced by *finished-by* and *during-until*, respectively.

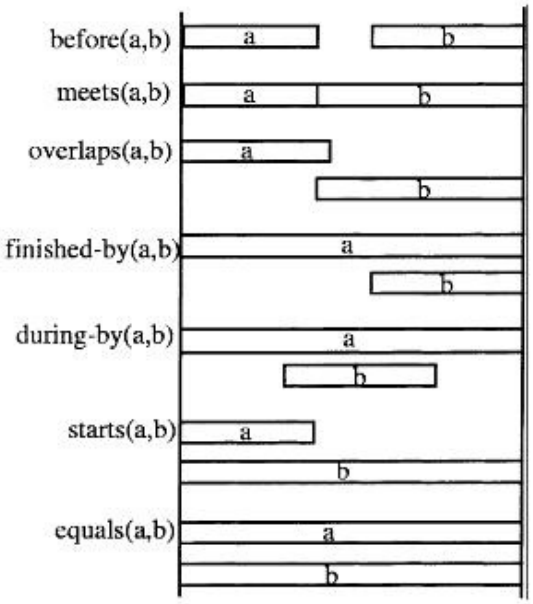


Figure 9.: Temporal relations between two time intervals from [Kwon et al., 1999]

As was found in [van Oosterom et al., 2002] [Jun and Jie, 1998] [Raafat et al., 1994], temporal models can largely help manage the historical data. Take urban design as an example, the land-use type of suburban areas may change through the years. By comparing the land-use map from different years, the urbanisation process could be monitored. A constraint 'A piece of land cannot have different land-use types at the same time' would avoid confusing different historic versions of the same piece of land. Because the time factor has nothing to do with the spatial dimension, the same description about time can apply to objects in both 2D and 3D (e.g. the age of a building).

Examples of other temporal constraints are 'the lifespan of a water pipe is less than 100 years' and 'the duration of the sunshine recorded by a solar panel in the winter time should not exceed 8 hours'.

3.2.3 *Thematic Relationship*

There are relationships that deal with the meaning, function, usage, class, etc. of an object. A building structure could be interpreted differently in various themes. It can be marked as a house in the cadastral registry; a hospital in the emergency calling center; a node in the network inside the system of an electrical plant. The thematic relation only deals with the meaning of an object and its relations to the others in a certain application. If it is modelled in the cadastral theme, then it must have an address containing the name of a road, which is geometrically connected to it.

3.3 MODELLING GEO-CONSTRAINT

There are various ways to model geographical constraints. They are different from the usual (or other types) constraints. Three of them are studied in this section: ontology, GML and OCL. Example rules that are modelled by them are given under each subsection. Some of them come from application cases that are running in real systems, whilst some others come from data specifications at a conceptual level. OCL is found to be the most commonly used approach for modelling geo-constraints. An attempt to use OCL for expressing geo-constraints will also be described in section 6.1.

3.3.1 *Ontology*

Ontology in theory is defined as a 'formal, explicit specification of a shared conceptualisation' [Gruber, 1993]. In information science it is a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts. Common components of ontologies are: individuals (instances), classes (concepts), attributes, relations, function terms, restrictions, rules, axioms and events. The business logics or constraints in ontologies are defined by components *Rules* and *Axioms*. Rules are statements in the form of an 'if-then' sentence that describe the logical inferences that can be drawn from an assertion in a particular form. Axioms are assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application.

In the paper from [Mas et al., 2005], a system in a mobile device to do field data capture is realised with constraint check by ontology. A rule that means 'a clearing is always within a forest' can be described as:

```
<ruleml:imp>
  <ruleml:_body>
        <swrlx:classAtom>
              <owlx:Class owlx:name=''Clearing''/>
              <ruleml:var>x1</ruleml:var>
        </swrlx:classAtom>
        <swrlx:classAtom>
              <owlx:Class owlx:name=''Forrest''/>
              <ruleml:var>x2</ruleml:var>
        </swrlx:classAtom>
  </ruleml:_body>
  <ruleml:_head>
```

```
                <swrlx:individualPropertyAtom swrlx:property=''within''>
                    <ruleml:var>x1</ruleml:var>
                    <ruleml:var>x2</ruleml:var>
                </swrlx:individualPropertyAtom>
            </ruleml:_head>
        </ruleml:imp>
```

3.3.2 *GML*

Geography Markup Language is an XML grammar defined by OGC that has a set of standards for the encoding and transmission of spatially referenced data. It provides a rich set of XML based schema for describing spatial data, including geometry, topology, coordinate systems, coverage and grids, temporal data and observations [ISO, 2010]. In application GML is often extended from the base abstract feature model to a user defined application schema.

In the work from [Reeves et al., 2006], a set of traffic rules regarding geometry is modelled in GML 3.1. They develop an application schema for UK zebra crossing regulations with extended geometry types such as offset lines. The width and min/max value of 'Zebra Stripes' that occur along the center of the road marking in figure 10 are modelled by custom feature level parameters 'StripeWidth', 'sfr:min', 'sfr:max' in GML fragment shown in figure 11. The parameters 'StudOffset' and GivewayOffset' are used to hold the offset distances of the stud lines and giveway lines from the edge of the zebra stripes respectively. A single polyline is to represent the center of zebra crossing, around which the other elements in its geometry are positioned.

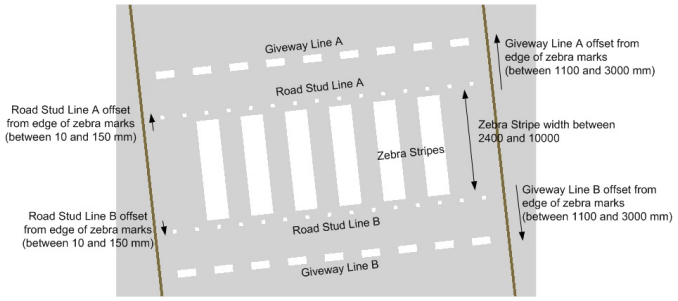


Figure 10.: UK Zebra crossing regulations from [Reeves et al., 2006]

3.3.3 *OCL*

Object Constraint Language (OCL) is a notational language to build and analyse software models. It is a standard 'add on' to the UML diagram standardised by the Object Management Group (OMG). Every expression written in OCL relies on the types (i.e. the classes, interfaces, properties, relationships) that are defined in UML diagrams.

Expressions written in OCL describe the important information of the modelled artefacts. This information cannot be expressed in a UML diagram. With the help of OCL, queries,

referencing values, or stating conditions and business rules in a model can be written in a clear and unambiguous manner [Warmer and Kleppe, 2003].

The link between an entity in a UML diagram and an OCL expression is called the context definition of that OCL expression. Context definition specifies the entity for which the OCL expression is defined. This is usually a class, interface, data type or component. Sometimes it is an operation or an instance. Each OCL expression is written in the context of an instance of a specific type. Some other commonly used notations (operations) are:

- Self: is used to refer to the contextual instance. For example, if the context is a *Flight*, then *self* denotes an instance of *Flight*. The attribute `seat`, which tells how many seats are in a flight, can be denoted as *self*.`seat`.
- Implies: appears as notation 'a implies b'. It states that the result of the total expression is true if the first Boolean operand (a) is true, and the second Boolean operand (b) is also true. If the first Boolean operand is false, the whole implies expression always returns true (does not really check the second operand).

Some predefined functions are:

- isBefore: has syntax a.isBefore(b), in which a and b are both date time type. It returns true is time a is before time b.
- isAfter: is reversed function of isBefore.

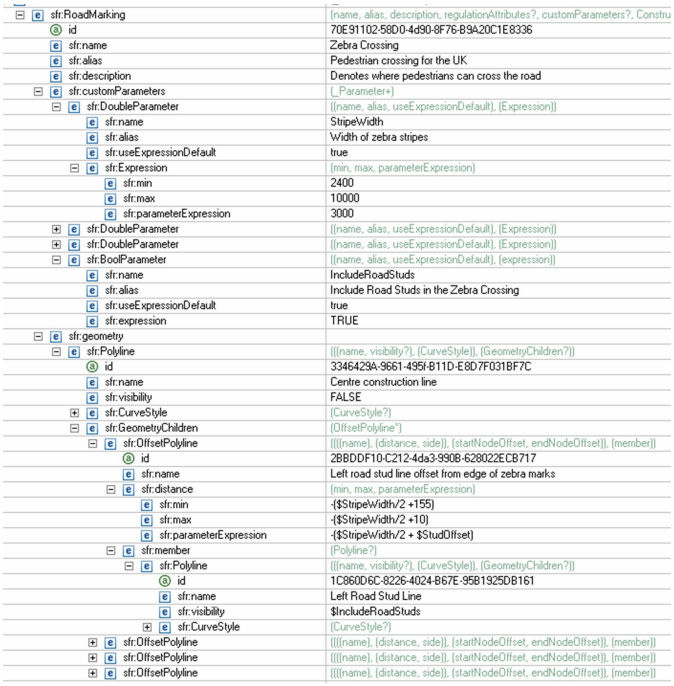


Figure 11.: Representation of the XML instance defining a Zebra Crossing as in UK regulations from [Reeves et al., 2006]

- substring: has syntax substring(str,n,len), which returns the substring of a string str that begins at the nth character and is of length len.

One of the commonly used OCL symbols is ':', which specifies the type casting. For example, p:Integer means p is an integer type. Arrow operator '->' is another commonly seen symbol, which indicates a collection operation. It is often placed before a collection operator, like 'size()' (calculate the number of elements) and 'sum()' (calculate the addition of all elements in the '()'). The following are operations that often appear with '->', which evaluate if the collection of elements passed into them is true for a certain expression. Other operations that work with collection are [Warmer and Kleppe, 2003]:

- Exists(expr): evaluates to true if at least one element that are passed into this notation returns true in the expr.
- forAll(expr): returns true if expr is true for all elements from source collection.
- isEmpty(): returns true if the collection returns no element.
- notEmpty(): returns true if the collection returns at least one element.

Some useful types of the OCL expression are invariants, pre-conditions, post-conditions. [OCL, 2010a]

- Invariant
  An invariant, denoted by **inv**, is a Boolean type expression that returns true if the invariant is met. It can be used to restrict all instances of class, type or interface. An invariant must true all the time. To specify a flight with type 'A02' can only have max. 200 seats, an invariant can be formed as:
  inv: self.type = 'A02' and self.seat <= 200
- Pre-condition
  A pre-condition is a Boolean expression that must be true at the moment when the operation is about to be executed. If a pre-condition is not met the operation will not be executed.
- Post-condition
  A post-condition is a Boolean expression that must be true at the moment when the operation ends its execution. The post-condition must evaluate to true, otherwise the operation has not executed correctly.

A few research papers show the possibility of using OCL+UML as a geo-constraint modelling and formalisation tool with necessary extensions. Examples as such can be found in agricultural sewage sludge monitoring systems from [Pinet et al., 2007], tree planting systems [Louwsma et al., 2006], land administration [Hespanha et al., 2008], part of a spatial data infrastructure GRID project [Werder, 2009]. As well as fundamental concept in [INSPIRE, 8 26].

The paper from [Duboisset et al., 2005] shows they integrate the 9 intersection model into OCL to specify topological constraints. One example is 'each town hall building b associated to a town t must be spatially inside t':

context Town_Hall_Building
inv: self.geometry -> Inside(self.Town.geometry)

This constraint must be satisfied for each Town_Hall_Building instance (denoted by self). And the 'self.Town' returns the Town instance associated to self.

Another extension of OCL called GeoOCL from [Werder, 2009] also gives interesting examples of OCL expression, e.g. having minimum length or maximum curvature. The operator maxCurvature returns the max curvature in degrees.

context Breakline
inv: self.geometry -> length() >= 10
inv: self.geometry -> maxCurvature() < 70

Another constraint states that noise abatement walls are disjoint from street geometries.

context NoiseAbatementwall
inv: Streets.allInstances() -> forAll(s:Street | s.geometry -> disjoint(self.geometry))

In INSPIRE data specifications, which are shared and applied within EU countries, different OCL expressions appear to be spatial-related as well. One example can be seen in the specifications for Addresses [INSPIRE, 4 26a].

A constraint 'AddressCountry' specifies 'An address must have an admin unit address component spatial object whose level is 1 (Country)' is expressed in OCL as:
inv: self.component -> forAll (a1 | exists(a1.parent.oclIsTypeOf(AdminUnitName) and a1.parent.level = 1))

The other example is from the Cadastral parcels specification [INSPIRE, 4 26b] for geometry type. Constraint 'geometryType' specifies that 'Type of geometry has to be GM_Surface or GM_MultiSurface' (ISO19107 spatial types) in OCL which can be expressed as:
inv: geometry.oclIsKindOf(GM_Surface) or geometry.oclIsKindOf(GM_MultiSurface)

4

3D CITY MODEL

As was discussed from last chapter, spatial constraints can apply to geographic objects, their attributes, and relationships between them. These features in the model have to be closely examined before constraints can be defined. This research means to design generic constraints that can serve a broad scope of city modelling. Based on this motivation, the database from Climate-City-Campus project that models general topographic objects in urban environment is selected to design constraints from.

The CCC mission is introduced in section 4.1. In order to achieve this goal, a database is built by students, and the useful objects and their attributes in campus for climate study are stored. The selection of objects will be told in section 4.2. The attributes (spatial and non-spatial) and relationships chosen to design constraints from are discussed in section 4.3. Relating to this section, an attempt of data specification for general city objects, from which the constraints can be developed, is presented in table I - 'Normal Situations' in appendix A.1. This specification attempt can be refined, considered and integrated to CityGML core model.

4.1 CCC MISSION

Urban climate refers to climatic conditions in an urban area that differ from neighbouring rural areas or large scale, i.e. national and global scale. When performing research about urban climate, the study area is limited in several kilometers, namely the local area and micro-scale phenomena. Micro-climatic effects of the city objects can be identified and so are of significant interests. The topography, building materials, plants, roads, people, water body and so forth, greatly influence the local region and its micro-climate situation. When these different scales (micro, local, regional, national and global) come to integrate in a model, the boundary conditions (continuity and consistency) needs to be considered. All these different types of objects, and concerns about modelling a city, raises a need to have storage being capable of handling geometric and thematic characteristics in an efficient manner and consistent.

On the other hand, urban climate is a spatio-temporal entity. Or it can be said it is a 4 dimensional phenomenon (3D space + 1D time). For instance, the temperature at street level would be different from the temperature at the roof level. And the influential parameters vary not only with location but also with time, e.g. the same road would experience different carbon dioxide densities in one day, depending on the traffic flows in the peak time and non-peak time. Collecting climatic data therefore requires a sensor to take measurements with a time factor and a position factor, i.e. a mobile platform with sensors.

CityGML is chosen as a storage model to fulfill all the CCC requirements. It is an information model to represent 3D urban objects, which also considers multi-scales modelling. Although it does not define constraints, the classes and relationships of objects in a city, including their

geometric, topological, semantic and appearance characteristics are modelled (also see the hierarchy in figure 12).

- Spatial Model
  With four levels of detail (LODs) for building and terrain, the same object can have multiple geometries and thus allows climate research in different scales. This exactly fits the CCC mission. For example, a building can be treated as an extrusion (cube) without any distinctions on roof/wall surfaces, or a structure consisting of different surfaces. The cube model would be sufficient to simulate the wind flow passing through local neighbourhood, but not detailed enough to enable study of radiative transfer in different sides of building. Spatial model also describes relationship between different geometric primitives and how they aggregate to more complex geometries in different dimensional levels. Several single points form a multi-point object (0D), likewise do a group of solids form a multi-solid object (3D).

- Appearance Model
  The appearance model provides means to store and manage appearance attributes of an object, namely observable properties of surfaces. Examples of such information are visible properties as texture and colour, and non-visual properties, i.e. material (shininess, transparency, etc.). This is the model to store static climate parameters such as reflectivity, heat capacity and roughness.

- Thematic Model
  The 3D topographic data in a city is categorised by themes in CityGML. One theme models a type of object. For instance, the *building* theme store spatial and non-spatial information about building objects. Other themes are *transportation, waterbody, landuse, vegetation,* etc.

4.2 OBJECT SELECTION

From the study of Geomatics Synthesis Project [Geomatics, 2010], building, tree, terrain, road, grass, water are found that would largely influence climate condition in urban area. Buildings have their multi-geometry in all levels of detail (LOD1-LOD4), seemingly from extrusion to walkable interior environment, in which different roof shapes and surface material is identifiable. A tree has 1 level of detail (LOD2) and can be attached with a certain species. In this LOD tree is a ball-like convex hull, which shows the crown shape. A species has information about its size, age and vegetation parameter i.e. leaf area index (LAI), normalised difference vegetation index (NDVI). Terrain is stored with 2 LODs, the coarser models the whole study area (almost flat) whilst the finer models two small hilly neighbourhoods (near Mekelpark). Road, grass and water are derived from TOP10NL 2D maps and stored as land use theme in the CityGML.

Sensor as an input of the climatic parameters is of significant interests in urban environment study. By the time the synthesis project finished, the university was found having some weather instruments on almost every building roof. Outside the campus, a fully equipped weather station is accessible in the botanical garden. These weather sensing platforms are connected to servers and give real-time data. On the other hand, sensing data from mobile platforms, along with the location and measuring time of each record, were measured and stored using CityGML

model. In section 6.1 a generic rule for sensors will be abstracted to ensure the smaller value change in time.

Beside the positioning and tracking devices that can pinpoint a sensor and moving path of its platform, sensors that could be used for climatic parameters measurements are categorised as:

- Thermometer - temperature sensors: can be either mobile or stationary and can tell the temperature in the air. From the physics point of view, too fast changes of temperature in time or space are not possible.
- Barometer - barometric/air pressure sensors: can be fixed or carried in moving platforms, but usually are installed on fixed/ground-based weather stations. They are used to measure atmospheric pressure in the means of water, air or mercury. Since air pressure is a larger scale (than campus scale) phenomena, sudden pressure change in the campus area may not be correct.
- Hygrometer - humidity sensors: can be put on mobile or static platforms. They measure relative humidity, that is the amount of water vapour in the air. The relative humidity indicates the likelihood of precipitation, e.g. rain, dew, fog, etc. and evaporation and is closely related to the temperature.
- Anemometer and Wind vanes sensors: are found in static weather stations. They measure wind speed and thus shows the wind pressure difference. Wind vanes are devices to tell the direction of the wind and are usually placed at the highest point of a building.

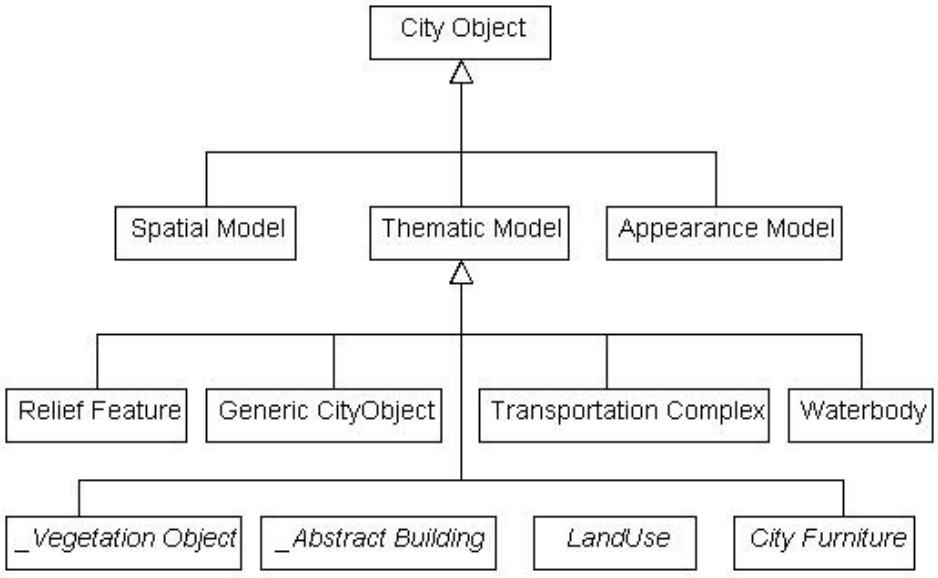


Figure 12.: An overview of CityGML hierarchy from [Kolbe et al., 2009]. Every city object has a geometry (described in *Spatial Model*), a visualisation property (*Appearance Model*) and belongs to a certain class (*Thematic Model*).

- Rain gauge - precipitation sensors: are normally fixed in weather stations. A rain gauge gathers and measures the amount of liquid precipitation over a set period of time. It is important to control the amount of water reaching the ground and whether this amount meets the need for vegetation.
- Disdrometer - drop size sensors: are found in fixed weather stations. A disdrometer is an instrument used to analyse rain by measuring the drop size distribution and velocity of falling hydrometeors.
- Pollution sensors: can be either fixed or mobile. Pollutants are known as substances in the air that are harmful to humans and the environment, e.g. CO_2 and fine dust. The pollutant can either stay in the air as gases, or cause ground contamination when it reaches the ground as solid particle or liquid drop.

4.3 THE CITY MODEL IN CCC

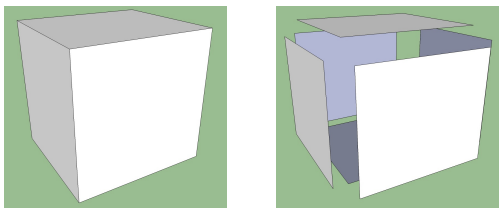
Some object, geometries or properties do not exist in current CityGML, but from the CCC model and constraint point of view, it it necessary to have them. Based upon the concept of influential factors and objects in urban climate, and core model of CityGML, a city model for constraint study is given in the following section. Including description of different geometries that an object is often modelled with, and the generic properties that an object in this class would have.

4.3.1 *Object Geometries*

A real world object can be modelled by different geometric types. For example, mass point, contour line, break lines, triangulated irregular network and grid could all be used to represent the terrain. The implementation of the constraint on the same object would vary when dealing with different geometric types. According to CityGML, the common geometric representations for the chosen city objects are proposed as:

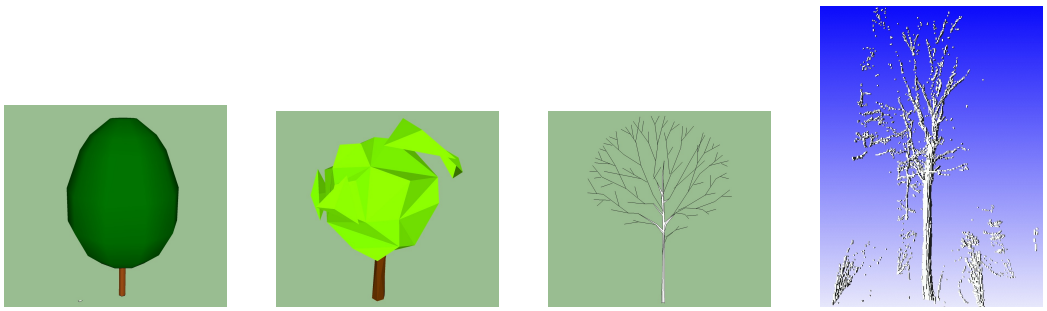
- Building
  A building can be modelled as a solid (in some places called body or polyhedron) or a collection of closed surfaces (multi-surfaces or composite surface) (see figure 13).
- Tree
  A tree can be modelled as a solid or a set of closed surfaces. In some field [Palagyi et al., 2003] and [Gorte and Pfeifer, 2004], network (line) model and point cloud are also applicable for tree (see figure 14). For some constraint about tree knowing the location of a tree is sufficient. So it is assumed that a tree can be simplified as a point (in its trunk bottom).
- Sensor
  There is no sensor object in current CityGML by default. A sensor, e.g. a GPS device, a temperature meter, is modelled in CCC as 3D point. To study the constraint for plane sensor (e.g. solar panel), 3D surface will be used.
- Road
  Road is represented as linear network in LOD0. (see figure 15). For LOD1 and higher the road is modelled as 3D surfaces.

- Grass
  A grass field can be modelled as 3D surface or solid.
- Water
  A water body can be a 3D surface or a solid. In LOD0 it is a top and horizontal surface of water which is often used to model water property. In LOD1 or higher, a waterbody is a solid enclosed by surfaces acting as boundaries between water and air, and between water and ground (see figure 17). When it is river the line geometry can be used.
- Terrain
  Elevation model has various geometric types to model with, triangulated irregular network (TIN), grid, break lines or mass points (see figure 18).



(a) Solid representation. (b) Surface representation.

Figure 13.: A building modelled by solid geometry and surface geometry



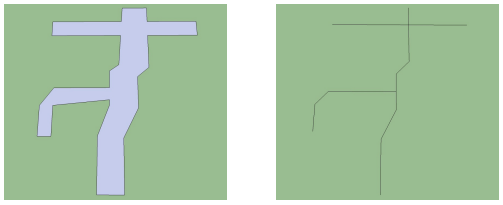
(a) Solid representation. (b) Surface representation. (c) Line representation. (d) Point cloud representation.

Figure 14.: A tree modelled by different geometric types

4.3.2 *Object Properties*

The structure and general properties of CityGML model is give from [Kolbe et al., 2009] (shown in figure 19). Of all the presented classes, the ReliefFeature (for terrain), *_VegetationObject* (for tree), LandUse(for grass, road, water), *_AbstractBuilding* (for building) are used.

Not every property is interesting to set up constraints. The non-spatial properties that will be established with constraints are listed below. Some of them are present in CityGML. Some



(a) Surface representation.

(b) Line (network) representation.

Figure 15.: A road modelled by surface (polygon) geometry and line geometry

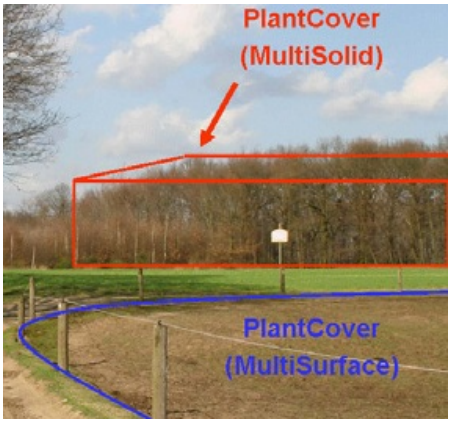


Figure 16.: A grass field modelled by surface geometry and solid geometry from [Kolbe et al., 2009].

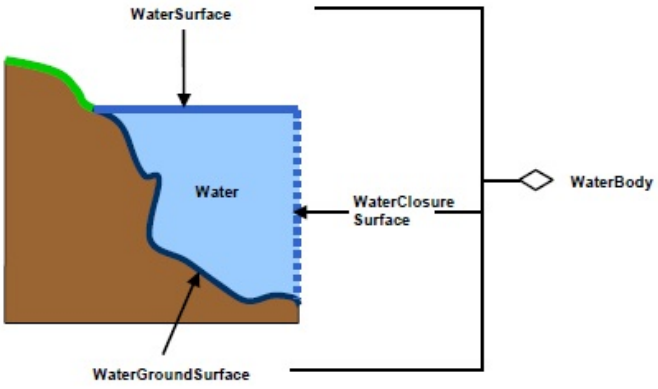
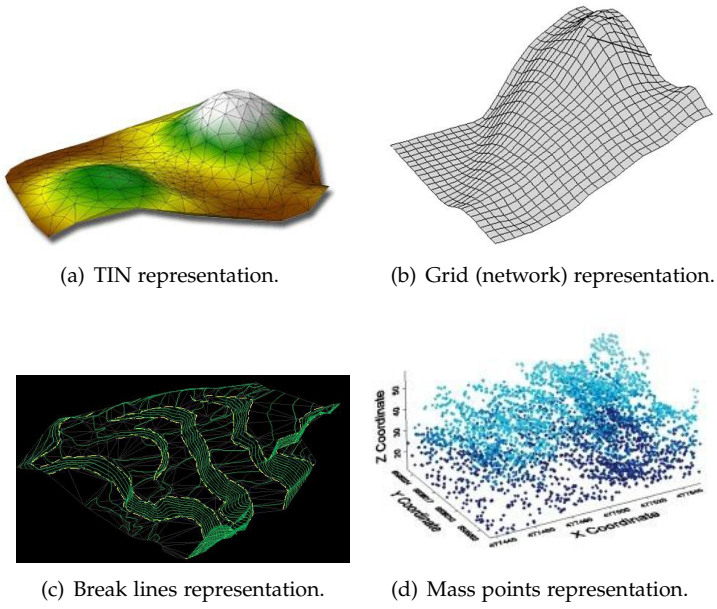


Figure 17.: A waterbody modelled by surface boundaries from [Kolbe et al., 2009].

others are not in CityGML but are common in a city environment. They are extended in CCC for this thesis research. The constraints together with these non-spatial attributes are listed in table II of appendix A.2. Some of them will be tried to formalise in OCL in section 6.1.

- Building
  A building, regardless the theme it is used for, has an address, name, owner, usage/function, construction date and, if it is historical data, demolition date.



(a) TIN representation. (b) Grid (network) representation.

(c) Break lines representation. (d) Mass points representation.

Figure 18.: The elevation modelled by different geometric types

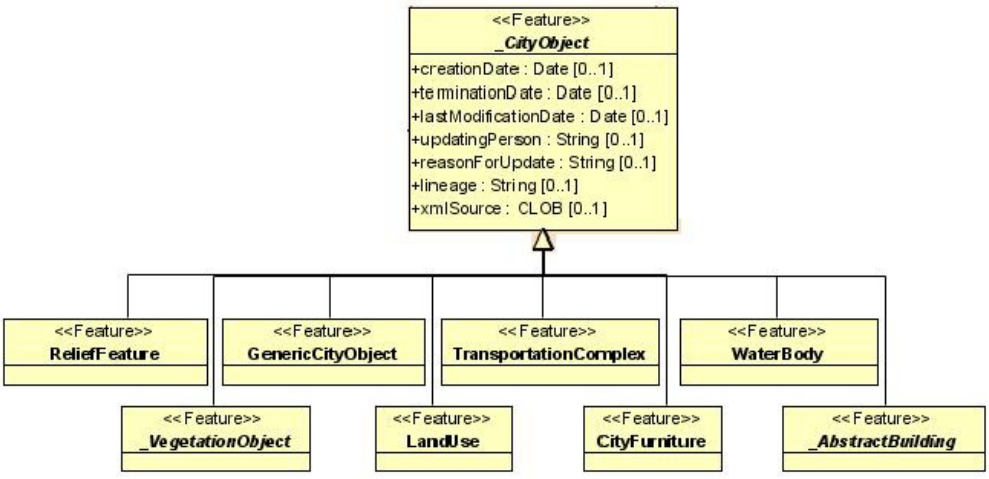


Figure 19.: Structure and general properties of core model in CityGML

- Tree
  Common attributes seen in tree model are species, height, age (or planted date), crown diameter and trunk diameter.

- Sensor
  The proposed general attributes of a sensor are location where the measurement is taken, the time when the value is recorded, sensor type (e.g. thermometer, barometer, wind speed sensor) and mobility (fixed platform or moving platform).

- Road
  General properties of a road can be road name, usage (e.g. highway, bridge, alley), construction date and, if it is historical, also demolition date.

- Grass
  Common attributes are usage (e.g. sport field, urban greening) and area (can be calculated from its geometry).

- Water
  General attributes of a water body can be name (or identifier, if it is not named), usage (e.g. canal, lake or sea) and area (can be calculated from its geometry).

- Terrain
  Digital terrain model means to depict the topography (geometric characteristic) of the ground and thus rarely has any general non-spatial attribute.

4.3.3 *Spatial Properties and Relations in CCC Objects*

Urban environment is artificially-built and has a diversity in objects and their spatial relations. Some relations that do not exist in one city could be seen in another, i.e. buildings touching in an unusual way, or trees planted in a very strange location. When looking at them in the computer model, someone may think it is a mistake and should be restricted, whilst others do not really care or see it as a possible exception. In other word, these arguable relations may disturb the recognition of errors. Before giving the constraints, it is necessary to state what spatial relations are correct from aspect of this research. Later, according to the fine cases, the relations that are very different from them will be marked as constrained cases.

'Table I-Normal Situations' in appendix A.1 includes spatial properties for single object (unary) and spatial relations between two objects (binary) that are considered as fine cases in CCC model. The spatial characteristics that are very different from them are discussed in next section as constraints. The interests include issues of dimension (2D or 3D), multi-geometries (LODs), severity and exception of constraint. The 3D spatial constraints are main subject in this research. So relationship for 2D (or 2.5D) objects, namely terrain, road, grass, water and landuse, is less interesting and is not very developed in this thesis.

5

CONSTRAINTS FOR CCC MODEL

After a close looking of the objects, their attributes and the relationships, a data specification are presented in natural language in table I - 'Normal Situations' (of appendix A.1) from last chapter. The items from this table will be used to develop constraints in this chapter.

From section 3.1 an natural language is found ambiguous to specify geo-constraints. And a certain abstraction of objects, e.g. geometry, seems necessary to achieve a more specific expression. This finding is like a spark for the approach in this thesis to specify geo-constraints. First they are stated in natural language (see subsection 5.1.1). Then because of the ambiguity of natural language in describing geospatial phenomena, the objects (nouns in the sentences) and the relations (verbs in the sentences) are abstracted by geometric primitives and topological relationships, respectively (see subsections 5.1.2 and 5.1.3).

Furthermore, in chapter 6 the objects can be translated into classes in UML class diagrams. Their binary topological relationships can be visualised as links between classes. After that OCL can be used to further formulate the constraints to an unambiguous manner.

5.1 SPATIAL CONSTRAINTS

The stating of spatial constraints are grouped into two classes regarding **strictly forbidden** and **unusual but can accept** (if a specific instance is recognised as exception). Each constraint is given a code in table II - 'Constrained Situations in Natural Language' under different objects and relations. The reason why these rules should be considered is told in subsection 5.1.1.

Subsection 5.1.2 and subsection 5.1.3 will show the geometric and topological abstraction.

Right after the object geometry is an unavoidable consideration of multiple geometries of single object. Subsection 5.1.4 will talk about two interests about it. One is constraints between multiple versions of the same object (LOD multi-geometries) to keep them consistent and in harmony. The other is about how changing geometry from one LOD to the other would influence specific spatial constraints.

5.1.1 *Description in Natural Language*

According to normal spatial characteristics that are proposed earlier, the situations that are very different from them are found and considered to be constrained cases. This includes unary and binary relationship. These constraints are sorted out in two severities, 'do not accept' (must not) and 'unusual but accept' (should not). The latter one deals with the doubtful or exceptional cases (see 2nd and 3rd column in table II in appendix A.2). For example, a normal condition of a land is defined as 'a land has a land use type'. The situations that 'a land has more than one

landuse type at the same time' and 'a land does not have any landuse type' are very different from it. These two situations then need to be constrained. Below is the explanation of the constrained cases (in table II) and the reason why they are considered as forbidden or just unusual.

- Building constraints
  There are 13 constraints discovered for building class. Building is treated as a composition of different parts. Each part can be modelled independently. When putting them together to form a complete building, some junctions may be disconnected and then leave strange openings (i.e. the edge of surfaces does not match). Constraint b11 is to make sure the building boundaries are all together seal a closed volume in LOD1 (extrusion representation). In some other cases, the opening on the inside of a building is necessary. For example, in LOD4 a house (modelled as only one building) consists of two parts and a corridor in between (see figure in constraint b12). For this house, the whole interior is a connected space, which means a person can walk from the left-hand side to the right-hand side. So the walls connected to the passage must have holes on them to allow people's travelling.

  Building objects have different kinds of shape. Giving a neighbourhood, there is only limited space to place a building. It is important to make sure the building do not clash into its neighbours (i.e. other building, tree) or stand on the wrong place (e.g. on the road, water). Many constraints are inspired by this principle, i.e. bb1/bb2, bt1/bt2, br11, bg1, bw1. So a general rule can be given to summaries all these specific constraints. For instance, a building can be adjacent to other building or separate, or touched by a tree. The road can go around, below or above a building. But if a building/tree is inside the other building (bb1 and bt1), or a building standing on the road (br11), there must be something wrong. This rule can further be abstracted into a general level as 'on one location there is only one object'. Notice the situation that two buildings share a point, as will be discussed later, is a bit doubtful in reality but not completely impossible. So it is treated as unusual instead of wrong.

  The address of a building comes from its surrounding road. If a building in the model has a name that cannot be found in its local neighbourhood, this building maybe placed in a wrong location. As a result, a relational constraint is given to make sure a building is put in the right local area (see br12).

  Sometimes a specific value in the constraint is difficult to give, i.e. cases t1 and tt1. A tree can be planted around a building, and the crown can grow as time goes by and get very close (even touch) to the wall. But it is better to keep the tree (especially the root) some meters away from a building. How much interval should be there requires extra knowledge about urban design and plant physiology and is not clear to the author. The value can vary with tree types and specific local conditions. So the exact distance value is not given and the rule is put under the unusual cases.

  Another interesting phenomenon is the roof greening, which has got grass object and building object involved (see bg2 and gt2). Normally the grass should be planted on the ground. When one sees the grass floating in the air (especially when the building is not displayed), he may suspect it as an error. It maybe just because the building has grass on the roof, and the greening layer is modelled as grass object. Therefore bg2 is placed as unusual case with roof greening explicitly permitted.

A common mistake found in 3D urban models is that the building foundation does not always match the underlying terrain. Sometimes holes are found in between the wall/ground floor and the terrain. This would cause problem when a building is used for wind simulation, i.e. the wind goes into a building from its ground floor. To avoid it, a restriction is specified by constraint n1.

When dealing with building-water relation, a special attention is paid to the boat house (see bw2). Usually, a building should not be found in the water. If in the computer model a building is floating on the water surface, it is very likely to be a mistake. Depending on the modelling method, house objects can be treated as building and then the boat house becomes a building. Constraint bw2 is presented to be an unusual case. This research will attempt to recognise it and distinguish it from the actual error.

- Tree
  There are 6 constraints found in tree class. Every species of tree has a different limited size. It is unusual to see some bush growing to the size of an oak tree. So a rule to avoid unrealistic size of tree is given in t1. Tree object has also particular area to be planted in. In other word, it cannot be planted everywhere, i.e. bt1, tr11/tr12 and tw1. Depending on the species, some aquatic plant can grow or even only grow in the water, whilst more others cannot survive if they would have been put in the water. Hence the relation between tree and water object needs to consider the species variety (tw1).

- Sensor
  There are 2 constraints for sensor. Many mobile sensing devices nowadays provide a memorisation function, i.e. memory card, to store a small amount of measurements. The mobile sensor, while is hovering around in the field, sends a set of temporarily-stored data to a database system, which provide better storage mechanism. There are different things that would go wrong during this process. From the aspect of this research, spatial information is of great interest, so a focus here is how correct the location information is modelled.

  Notice the rule for building and mobile sensor (mb1). Giving the building model, assumed that somebody carried a temperature meter together with positioning device, to measure the temperature inside and outside a certain building, as well as the location, and added the measurements to the data (see figure 20). Somehow some points that are recorded from outdoor, especially those that are near the building wall, are understood by machine as from indoor. Then a scientist that wants to study the thermal comfort inside the building may use a spatial query to retrieve the all the measurements from inside the building. He will get some outdoor measurements, which are actually errors for him and will reduce the quality of his research (e.g. in winter time the indoor/outdoor temperature have sharp difference). Thus it is necessary to state that the mobile sensor is not expected to have sudden jump into or out of a building. If this rule is violated, the position of door is necessary to tell if it is really an error. This will be discussed further in implementation from subsection 7.2.4.

  This constraint is a sample to study the spatial constraints of mobile objects in urban environment. The idea and implementation approach can be expanded to generic moving objects in the future, e.g. car, human, bike, boat and etc.

  There are solar sensors (panel) equipped on some weather station and building roof that are now working in the campus to measure solar energy. Usually the solar panels are

installed on the roof or in the outside (wall) of building. Just shifting the panel object into building by 1 meter can make its position become inside the building. This shift may not be very obvious in terms of metric distance, but it definitely breaks the principle of using solar panel, which requires a direct reception of sunshine. Regarding this possible incident the relational constraint sb1 between solar panel (as a stationary sensor) and building is designed.

- Road, Grass and Water
  There are overall 10 constraints for ground objects (including terrain). The grass, road and water features are from land use type. A common relation between these land use features is that they do not overlap each other (think about a land used as road and grass at the same time). This leads to constraints rr2, rg1, rw1, gg2 and l1. An unusual thing of landuse is that in the midst of an area that the usage type is known, there is a gap (like a hole) with no type. It could be a missing of data. So to promote that the landuse has full coverage over a known area, constraint l2 is added and marked as unusual cases. However, how to specify them in a formal way like OCL is difficult.

  Although the ground objects come from land use information (2D), they are stored with 3D coordinate (z value in each point). So the constraints for them will consider all three dimensions (rr2, rg1, rw1, rt1, gt2, wt1). When the road is modelled as polygon, the junction is separately stored so there is no crossing between roads. In this manner, the overlapping of two roads is strange. Therefore rr2 is given in the unusual group to pick up the doubtful overlap.

- Terrain
  The important characteristic of terrain is that the ground objects are laid upon it. According to this role constraint n1 is designed to avoid floating objects. Instead of repeating the match of terrain and different ground objects, enforcing **Not Empty** to the terrain intersection curve (TIC, see [Cit, 2008]) for all objects is must simpler.

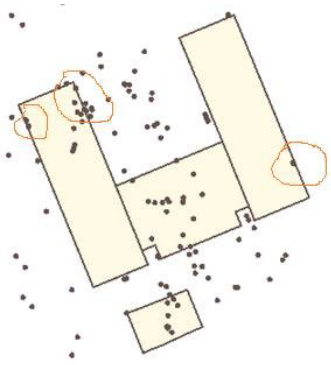


Figure 20.: Some outdoor temperature measurements maybe understood as indoor measurements and may cause discrepancies in the spatial query results.

5.1.2 *Geometric Abstraction for Spatial Constraints*

During the specification of spatial constraints, the various real world city objects and their relations are found complex and difficult to clarify by natural language. An adopted solution is to discuss them in the digital model, which models the objects by clearly-defined geometric primitives (like solid, surface, line, point), and depict the relationship by interaction of topological primitives.

As was explained earlier, one kind of city object can be modelled by different geometric features. The road can be modelled in line geometry in the transportation use. This usage is not included in the current CCC model so will not be discussed here. Since surface and point can be push down to 2D, it is necessary to explicitly mention the dimension, e.g. 3D-point, 2D-point, etc. The 3D geometries selected in the spatial constraints include: solid, surface (in this thesis research is actually 3D-polygon) and point. The city objects that are modelled by them are:

- Solid
  Tree and building.
- Surface
  Building, road, grass, water and terrain.
- Point
  Tree and sensor.

One advantage of such abstraction is that the same constraint can be reused for different city objects. For example, the constraints about building-building (bb1) and building-tree (bt1) both restrict intersection. According to the description of different geometric types of tree and building, they can both be modelled as solid. So one constraint 'solid should not intersect solid' works for both. And the constraint that checks if two road overlaps can be reused for grass-water, since they are all modelled as surface. This kind of higher-level/abstract rules will then be used as generic constraints. To model them, one can put them in the abstract object class in UML, then all other subclasses will inherit. The refinement of generic rules can to be done for specific classes/attributes. For example, when both solids are tree objects, relationship 'intersection' is allowed.

The other advantage of using geometric types is that the same object represented in one type is easier to deal with than using another. For two object that are both building, constraint 'two buildings intersect' (bb1) can be checked by solid-solid, solid-surface or surface-surface relations. The solid-solid relation only compares two single solids. But the surface-surface relation has to be calculated between two sets of surface. Later in implementation the most optimistic geometric primitive (checks constraint with the lest calculation) will be chosen for each object.

The constraints about building obviously need 3D. For instance, a building has a part stretching out overhead some a tree (see figure 21(a)). In 2D it may look like that half of the tree is inside the building, which is forbidden. So is with the case (building overhead a building) shown in figure 21(b).

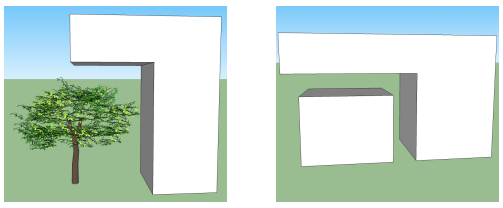
Another example comes with the bridge/flyover (see also figure 22). The bridge or flyover often go over some other objects underneath, e.g. canal, road, tunnel. In 2D model, without the z value, the bridge and canal (or flyover and road) are modelled as crossing in the same location.

If a tree is on the bridge, it will be also seen as on the river. A constraint that prevents tree from standing on the water would tell that it is wrong. This is only a insufficient interpretation of the real world situation. But in 3D, the height information is known and this is not a problem any more.

5.1.3 *Topological Abstraction for Spatial Constraints*

The topological primitives (interior A°, boundary ∂A and exterior A^{-}) introduced in 9 intersection model is found significantly helpful to clarify the spatial relationship. As long as the interior, exterior and boundary for city objects are known, the spatial relations can be made clear by the intersection of them. Once intersection is clear, one can decide what to do with it, i.e. forbid or allow.

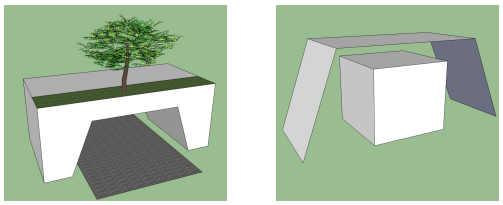
For 3D geometric primitives, the interior, exterior and boundary of the used geometries are defined by [Egenhofer, 1992] below. Notice that in other contexts such as
citeSisiPhD surface can be curved. In this thesis research, 'surface' is used to mean planar surface (3D polygon) only.



(a) A building has a wing extending over a tree. In 2D half of the tree may be inside the building.

(b) A building has a wing extending over another building. In 2D the building underneath may appear to be inside its neighbour.

Figure 21.: Examples of building constraints that have to be checked in 3D.



(a) When a tree is planted on a bridge that goes over a road, in 2D the tree may appear to be standing on the road.

(b) When a road goes above a building, in 2D the road may appear to be crossing the building.

Figure 22.: Examples of constraints applicable to roads that have to be checked in 3D.

- Point
  A point in the space has only interior and no boundary.
- Line
  A line in the space has two or more disconnected boundaries. The part between boundaries is interior. Line object is not in the CCC model so will not be further discussed in this thesis.
- Surface
  A surface has its boundary (line segments) enclosing an area, and dividing the space into interior and exterior.
- Solid
  A solid consists of surfaces that enclose a space and form the boundary. The space closed inside is the interior and outside is the exterior.

The terms used in specification from tables in appendices A.3 and A.4 use Egenhofer 9I model. Many relations between solid-surface and surface-surface, as presented in [Zlatanova, 2000], do not have names yet. During the research these relations are investigated. Some relations can exist but in reality they do not occur so often. But in solid-surface and surface-surface relations, a surface can be touched on its boundary or its interior. When it is accompanied with the meaning (semantics), these two cases can be treated differently in constraints. The former one can be fine and the latter may be wrong. For instance, a solid touches a surface in the surface interior could be a building standing on the road (see figure 23(a)), which should not happen in an regular urban environment. But if a solid touches a surface at the surface boundary, it could be a road laying next to a building (figure 23(b)), which is fine.

Therefore, a name 'strong-touch' is proposed here to distinguish touching a surface on its interior from touching surface merely in the boundary. The expression 'A strong-touches B' (A can be surface or solid but B has to be surface) is equal to the 9I matrix in matrix below. It does not matter whether or not boundary A has intersection with boundary B (can be 0 or 1 in that set). This term is also detectable, as is developed and described in section 7.2.1. Plus the meaning of objects, whether or not two city objects, e.g. two different buildings, have a restricted relationship is also implemented in this research, as you will see in section 7.2.1.

$$R(A,B)(Astrong-touchesB) = \begin{bmatrix} \oslash & \oslash & 1 \\ 1 & \oslash/1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In order to easily rephrase each constraint by the geometric and topological primitives, the two columns 'Do not accept' and 'Unusual but accepted' in table II are separately discussed in two tables, 'table III-Situations that cannot be accepted' (appendix A.3) and 'table IV-Situations that are unusual but acceptable' (appendix A.4) respectively.

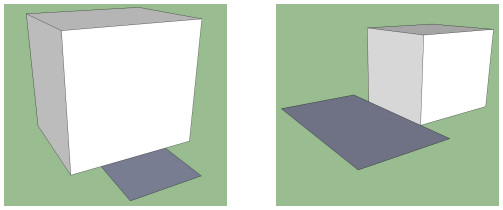
Here some examples are given about how the spatial constraints in natural language are foramalised by geometric and topological primitives. Look at the constrained br11 in 2nd column of table III. Giving that a building is modelled as a solid and a road a surface, spatial relation br11 is represented as 'surface A strong-touches or intersects solid B'. When building is modelled as surfaces, the representation becomes 'two surfaces intersect'. In practice, this check needs to be applied to every surface from building, or the most likely intersected surfaces from the building, to make sure there is absolutely no interior intersection between a road

and a building. So is with the bg1, bw1 constraints. Either the closest building surface to the grass/water should be pick up first, or the same check should be run for every building surface. However even if all surfaces give 'disjoint' it is still not the moment to say two solids really 'disjoint' (e.g. a tree floating inside a house). This will be discussed in implementation (see section 7.2.1).

Building class has two unary constraints (b11 and b12) that are difficult to be formalised by topological terms. The one about closeness (b11), when a building is treated as single solid, can be checked by geometry validity function from Oracle. When a building is modelled as a composition of surfaces, these surfaces should altogether seal a valid solid (be it according to a specific definition, like Oracle). In other word, their orientation should be correct and must not have faces intersecting or adjacent faces with some distance. Otherwise the formulating of solid will not result in a sealed volume. The constraint b12 calls to inspect if a surface is with or without hole. If there is a passage between different building structures, when the building boundary seals a volume, on the detailed interior structure there must be a hole on the wall surface to allow passing.

The other example of abstracted translation is about tree-road relation. The tr11 relation 'tree stands on the road' can be translated to 'a point is on the surface', when a tree is modelled by its ground point and road as surface. Under the same geometric primitives, the tr12 relation about distance between tree and road becomes 'distance from point A to surface B smaller than value n'. Here a general issue is also drawn that constraints can have parameters which must be specified by user.

Some constraint cannot be rephrased only by geometry because they requires information from other categories (thematic, temporal and quantitative). A thematic information is required in building-solar panel constraint (sb1). When both building and solar panel are surface objects, the check calls for a selection of the building boundary surface first. If the selected surface is a wall inside the building, the check will be pointless. The address matching (br12) is another example. The road that is within a distance of a building should be first found out. Then the matching of the road name and building address can tell if there is something wrong with the address. The exceptional cases also require thematic checks. Examples can be found in roof greening (bg2 and gt2) in table IV, bridge (rw1) and tree species (tw1) in table III.



(a) Building stands on the road, which is a type of 'touch' relation but should be avoided.

(b) Building stands next to the road, which is also a type of 'touch' relation but is acceptable.

Figure 23.: Here are two situations reflecting 'a solid touches a surface'. These two cases can be treated differently in constraints.

The quantitative check can be used to specify spatial constraint also. Notice that the spatial characteristics about size of tree is already modelled in its attributes, height, crown diameter and trunk diameter. The spatial constraint can be specified by restricting the values for these properties. Therefore, situation t1 is translated as 'attributes (height, crown diameter or trunk diameter) exceeds a certain value'.

Still, there are some constraints difficult to be formalised by geometric and topological primitives in this stage, because the algorithm is not fixed yet. For example, if we would consider a mobile sensor as point object and building a solid. For situation mb1, knowing the relation between an individual point and the building is not sufficient. A method is to collect the points recorded in one go. By comparing the position of its previously-measured point and later-recorded point, one may be able to tell if mb1 situation happens. In geometry, it means a set of points are marked in a time sequence, which actually means a spatial-temporal line. And then by checking the line-solid (sensing route-building) relation the jumping points may be exposed. The algorithms will be discussed in the implementation (see section 7.2.4).

The other constraint difficult to specify is the landuse constraint l2. An idea to check it is to select the local area and use a board of puzzles. Then fill all pieces of 'puzzle' (lands) in and see if there is a hole somewhere. But how to find this hole, and how to let an individual 'puzzle piece' understand its neighbour is missing is open for effective algorithm.

A more formalised view of object constraints is unveiled in UML diagrams, together with pseudo-OCL expressions in section 6.1.

5.1.4 *LOD Issue for City Objects*

Since a single building object can be represented in multi-scale geometries (LODs), one consideration falls on the consistency between different LODs of the same object. Consistency of multi-scale spatial objects is often discussed under contexts such as maps generalisation (terrain continuity/connectivity [Kolar, 2004]. And is found in various researches such as multi-representations in geo-database [Sheeren et al., 2005], [Breunig et al.,], consistent geometry between different dimensions [GROGER and PLUMER, 2005]) and (dynamic) visualisation in computer graphics ([Malik, 2000], [Godse, 2009]). A recent study about the topological consistency between building LODs is presented in [Ghawana et al., 2010]. The issue of keeping consistent data is also mentioned in specifications of INSPIRE ([INSPIRE, 8 26]), which gives general suggestions and requirements to keep data from different themes and regions consistent and coherent. As is originated from [Sheeren, 2005], when comparing data at two levels of detail, it is necessary to distinguish between difference and inconsistency. Difference is a result that are expected and can be found in the data specification (level of detail), whilst inconsistency is difference that is not explained by the specification.

The data specification from General Conceptual Model [INSPIRE, 8 26] defined LOD by:

- the type of information (the spatial object type and its properties)
- the selection rules (explaining which entities of the real world are represented in the data set)
- the accuracy of the attributes
- the semantic granularity (e.g. CityGML LODs)
- the type of geometries (3D, 2.5D, or 2D; volume, surface, curve or point)
- the accuracy of the geometries

To tell if two levels of detail are consistent, an analysis on the relationships between spatial objects in these levels is necessary. These relationships can come from:

- aggregation: for example the built-up area at coarser level is composed of buildings at finer level.
- generalisation/type hierarchies: a spatial object at coarser LOD is represented in finer LOD by several spatially connected objects.
- object selection: a set of spatial objects of a class in coarser LOD depicts a selection of the main spatial objects of a larger set at finer LOD (e.g. the road or river network).
- simplification/reduction of geometric dimension: for example a river from surface to curve, or a building from surface to point.

Only when the formalisation of theoretical relationships between different LODs are identified will the detection of inconsistencies become possible. The data specification of CityGML does not specify the change between different LODs in such a formal manner. The original description [Cit, 2008] about building LODs says:

'LOD1 is the well-known blocks model comprising prismatic buildings with flat roofs. In contrast, a building in LOD2 has differentiated roof structures and thematically differentiated

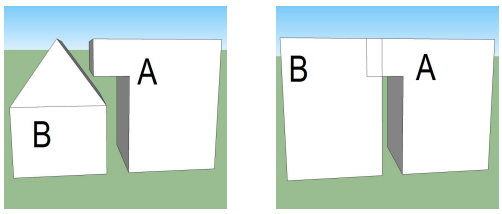
surfaces. LOD3 denotes architectural models with detailed wall and roof structures, balconies, bays and projections. High-resolution textures can be mapped onto these structures. LOD4 completes a LOD3 model by adding interior structures for 3D objects. For example, buildings are composed of rooms, interior doors, stairs, and furniture.'

In this research the 3D geometric attributes is of interests. Therefore, based upon this description, certain rules concerning geometry changes of the same building object are abstracted and proposed to exemplify the identification of inconsistencies. (Again the parameters can vary along with the modelling process, a different data specification and so forth.)

- Locations should be the same: deviation of locations from different LODs must be less than, say 1 meter, in campus level.
- Heights/Widths/Lengths of the main part of building should be the same: in this sense the small building extensions (e.g. fence, flag pole, antenna and etc.) are not included in the calculation.
- Volumes of building from LOD3 and LOD4 should be the same: this is based upon the fact that LOD4 is simply exterior from LOD3, plus the interior structures.

Another consideration is about the influence of multi-geometries in spatial relationship between objects. Because an object can have different geometric representations, it is necessary to clarify which representation(s) should be checked when it comes to a specific constraint. This consideration is shown in table III and table IV (see appendix A.2) with column 'Sensitive LOD'. In practice, the LODs mentioned in this column will be checked before one can tell whether the model really has constrained situations from first column (natural language) or not.

One example of why LOD matters is that the change of roof shape affects a building-building relation (figure 24) (see also table III). In real world, a building A has a part stretching out and hanging over its neighbour B, which has a sharp roof (figure 24(a)). This relation is clearly depicted in LOD2. However, in LOD1, building B is only an extrusion (a cube) with a flat roof (24(b)). Their upper parts appear to be intersecting, which is forbidden by constraint 'two building cannot intersect'.



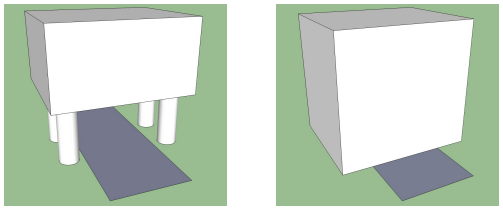
(a) Real world situation: sharp roof.

(b) LOD1 representation: flat and intersecting roof.

Figure 24.: Relationship between two buildings changes with LOD.

Another consideration of how LOD matters is about the supporting structures (i.e. pillar) of building. A building standing on the poles often has some other city object laying below, i.e. road (br1), grass (bg1) or water (bw1). The poles in LOD1 sometimes are neglected, and the building is only extruded from its 2D footprint. Thus it appears that a building is interfering

the object below, i.e. building stands on the road, which is not true (see figure 25). In LOD2 or higher, the pillar is modelled so this problem does not exist anymore.



(a) LOD2, or a higher level, models the pillar.

(b) LOD1 may miss the pillar.

Figure 25.: Poles of buildings may be missed in LOD1 thereby causing an inaccurate relationship between a building and the object(s) below.

The last two examples only talk about difference between LOD1 and LOD2. When dealing with the building interior, i.e. connectivity of a complex building consisting of different parts, LOD2 is not enough. In constraint b12, a building has two parts connected by a passage (see figure 26). To know if the interior structure is correctly modelled, a useful constraint is to check if there are holes on the walls allowing people to travel through. The figures 27 show how this building is modelled in all 4 LODs. Only in LOD4 is the hole on the wall recognisable.

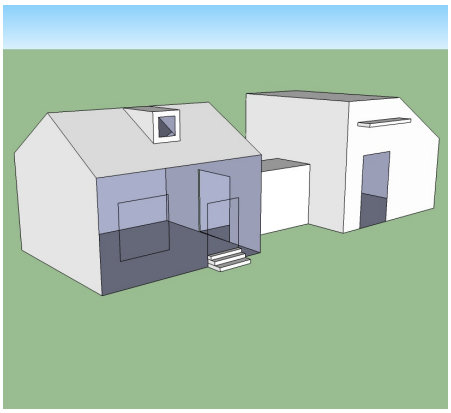


Figure 26.: A complex building modelled as a single volume, which has three structures, i.e. two rooms plus a connecting passage.

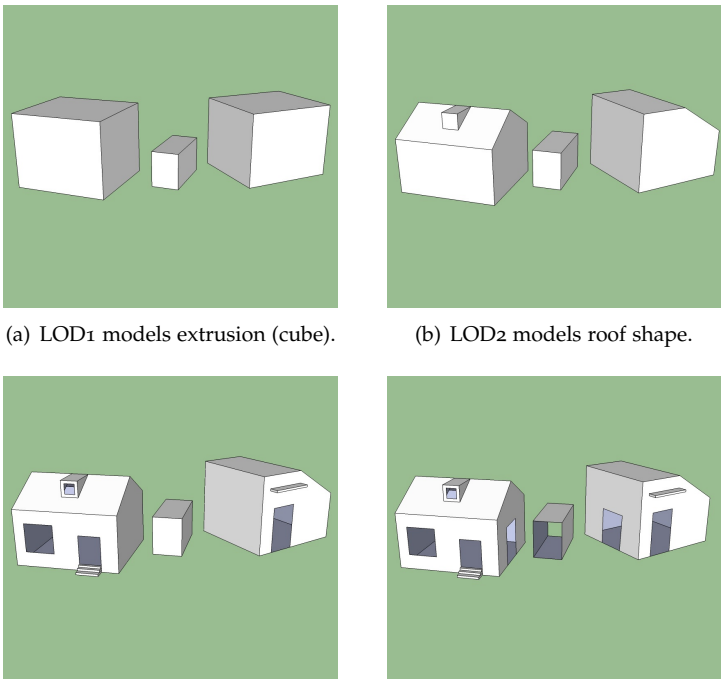
5.2 NON-SPATIAL CONSTRAINTS

The 3D spatial constraints in this thesis work are more specific. The non-spatial constraints are more common and used elsewhere, so they are not very developed here. The temporal, quantitative and thematic constraints are less ambiguous than 3D spatial relationship so they have only one severity: cannot have/must not. Most of them are dealing with unary relationship (see also non-spatial constraints in table II - 'Constrained Situations in Natural Language').

- Temporal Constraints
  The temporal constraint means to avoid confusion in time factor. Constraints b1 and r1 are in this class. As shown in section 4.3.2, the artificial structures, namely building and road, that have construction date and demolition date, share a common constraint 'the construction date must be earlier than demolition date'. This would be useful when dealing with historical dataset.

  The other interesting temporal constraint is the subsequent version of the same object must have matching times, e.g. old_version.EndTime = new_version.StartTime. This is not shown in table II for specific objects but can be abstracted for generic objects. The current CityGML storage does not assign any id to different versions of an object in time. And there is not a clear hint to indicate that two records in the database actually mean the same object in two different periods of time. Therefore it is difficult to specify this constraint. A solution could be to compare the objects at the same location and see if they match in shape. If they do then they may mean different versions in time for the same object.

- Quantitative Constraint
  Sensing data, which includes all different kinds of climate observations (see section 4.2) and sensitive to erroneous value, is the main subject that desires for quantitative constraint (see constraint s1 for example). The min/max value, unit and space/surface density (also



(a) LOD1 models extrusion (cube). (b) LOD2 models roof shape.

(c) LOD3 models openings. (d) LOD4 models interior passages.

Figure 27.: LOD1-LOD4 representations of two halls connected by a passage.

spatial) are the main properties to be constrained. Because the constraint of sensing data requires very detailed information about different types of sensing device, which is not the main goal of this research, they are not discussed in detail here.

How a quantitative constraint can help is exemplified by temperature measurement. Given temperature meter that records the air temperature with only units Fahrenheit and Celsius. A useful constraint is to avoid misusing these two units, e.g. 'every temperature measurement should have a unit, either Fahrenheit or Celsius'. The min/max temperature can also be specified, depending on the region, season and historical record. For a region that never has temperature higher than 25 Celsius degree from November to February, the value 30 $^{\circ}C$ is impossible, which may be an erroneous measurement of device or a true value but actually using Fahrenheit degree. Constraint s1 shows the attention to too fast change in the temperature value. The threshold again may differ from one location to the other and needs to be specified by user.

Beside sensing data, quantity constraint can also apply to other city objects, i.e. tree size, road width, building height and grass area.

- Thematic Constraint
  Some of the discovered thematic constraints are mixed with spatial, temporal and quantitative characteristics. For example, every building should have the address the same as name of road close by (say, dist<3m) is a spatio-thematic (as shown in constraint br12, table II, appendix A.2). Constraints under this category are very much developed in other fields so not many of them are designed in this thesis work.

6

FORMALISATION OF CITY OBJECT CONSTRAINTS

As a result of reading various articles on formalisation approaches applicable to geo-constraints (see section 3.3.3), the UML diagram together with OCL notation has been found to be a suitable tool to express the designed constraints, together with the 3D geometries defined in ISO19107 standard [ISO, 2008]. An advantage of this approach, comparing it to its text-only competitors (such as GML, XML, Ontology), is that it unveils the relationships between objects visible in diagrams. The OCL notations give formal (well defined and unambiguous) meanings that can be understood for both humans and machines. And with the flexibility of OCL, which can be extended with spatial data types and operations, the geometric features and topological terms can be written in a formal manner.

A first attempt to formulate the designed geo-constraints (proposed in section 5.1.1) using UML+OCL is made in section 6.1. Section 6.2 will discuss the possible contradictions between constraints and show a possibility of detection by rewriting constraints in first order logic. Section 6.3 will show different tools that can be used to translate OCL expressions and discuss their pros and cons in dealing with geo-constraints.

6.1 FORMALISATION USING UML/OCL

Section 2.2 has indicated that the UML class diagram is useful to describe topographic objects. Figure 28 displays the city model which includes attributes and relationships that are useful to study constraints.

Almost every class has its thematic attribute, except terrain surface. Stationary sensors and dynamic sensors are generalised to an abstract sensor class. Both sensor groups have 'sensorType' attribute. The stationary sensor can either be surface (e.g. 'solar panel') or point type (e.g. 'temperature meter'). A building can be treated as a composite solid formed by different building parts which are solids. Both 'Building_Parts' class and 'Building' class have temporal attributes from their 'starting_date' (construction) and 'end_date' (demolition). Grass is modelled by surface and trees by solid. They inherit from abstract class 'vegetation'. The grass class has a theme that shows its function/usage (e.g. roof greening), which can be used in constraint check. Tree class has attribute 'species' to help identify its species-related behaviour (e.g. size, living environment - aquatic or non-aquatic). Together with road, terrain and water, they inherit from the most abstract class *CityObject*.

Currently, there is not an association link in UML2.2 to indicate pure constraint. When two object classes associate, be it thematic, temporal or spatial (see also section 2.2 for spatial associations), it is not too difficult to mention a relationship constraint, since the constraint can be attached to the association. But when two object classes do not directly associate in space, or in theme/time, the method to mention constraints about their relationships could lead to

discussion. For example, in general a tree class and a building class may not be considered to be associating with each other. However, when a tree instance and a building instance come close in space, some spatial association such as distance (from which the distance constraints can be developed) between them may need to be considered.

In this research an attempt is made to notate the spatial constraints using OCL and attach them in UML class diagram. The current UML standard does not have a specific link to describe constraints. The lack of a constraint link makes it difficult to express constraints in OCL. A reason for this is that OCL itself does not support constraints for multiple classes. In normal OCL formula, to specify the distance rule between a building instance and a tree instance, the class 'Tree' must be available in the context 'Building'. In other words, the class 'Tree' should have a property that is of 'Building' type, or 'Building' class should have a property that is of 'Tree' type. But in this example of a distance constraint, neither 'Building' nor 'Tree' has a property to type (typify) each other. So an expression using multiple classes would be:

'context Tree
inv: Building.allInstances() -> forAll(b | distance(b.geometry, self.geometry) > minDistTreeBld)'

Therefore an extension of OCL is necessary to enable the use of multiple classes. Examples expressions using multiple classes can be found in the pseudo OCL listings 6.5, 6.4, 6.7, 6.8, 6.12 and 6.12.

The association link with a different colour 'brown' is thus used to depict constraints and distinguish constraint links from normal association links. As a sequence of 'borrowing' association, a consideration is where the multiplicity should be. No doubt it must be mentioned in OCL expressions. But should it be also attached to the UML class diagram? Attaching it to the UML makes the multiplicity visible. However, when it exists in both UML diagram and OCL expressions, it must be guaranteed that the multiplicity from these two places is consistent.

To avoid the possible inconsistency of multiplicity between a class diagram and its attached OCL expressions, multiplicity will only exist in OCL. And for a brief and tidy visualisation of constraints in the diagram, the association together with constraint names written on it is used to describe constraints in binary relationships (instead of using floating notes that explicitly show the content of constraints). When a spatial constraint acts on relation between two instances from the same class, an association link coming out from and going into that class is used (reflexive association) (see also 'oneobj_oneloc' from figure 28 and 'MinDist2Tree' from figure 30).

Because all the designed constraints are always 'True' for all instances of the same class, and do not vary before or after any operation (e.g. DML statement insert/update/delete), the static Boolean expression 'invariant' is used. The constraints in the resultant UML diagrams are quoted by a pair of curly brackets '{ }'.

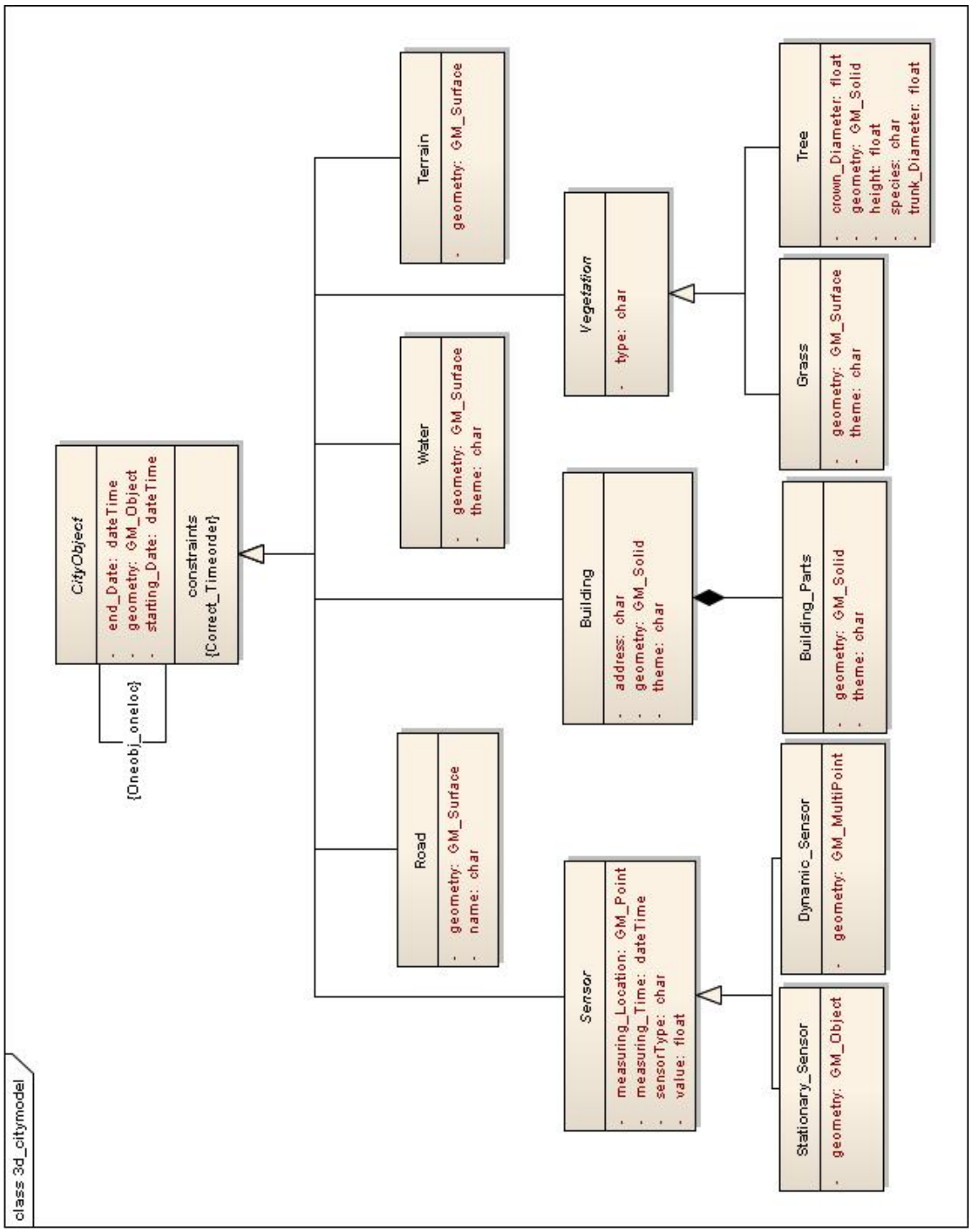


Figure 28.: UML class diagram for constraints study in general city modelling.

An initial formulation of OCL regarding some constraints in table II - Constrained Situations in Natural Language (see appendix A.2) is included below. To ensure the correctness of syntax and semantics they still need to be tested, together with the UML model, in OCL parsers like Dresden OCL2 toolkit [Wilke et al.,] (see also future work in section 9.2). An extension for the use of multiple classes is also needed in OCL to complete the parsing. Each constraint together with its unique code can be found in table II.

Beside the standard types in OCL (see section 3.3.3), the well-defined 3D geometric primitives from ISO19107:2003 standard [ISO, 2008] are used as spatial types in these pseudo OCL expressions, including:

- GM_Point
- GM_Curve
- GM_Surface
- GM_Solid

and the aggregational and compositional types of them.

Extensions of spatial operators are borrowed from ISO19107 as well. Since later on the implementation of constraints will be carried out in Oracle Spatial, some spatial functions in the database are useful and so are added here.

ISO operators:

- distance(GM_Object, GM_Object): a number valued operator that returns the distance between two GM_Objects. This distance is defined to be the greatest lower bound of the set of distances between all pairs of points from each of the two GM_Objects. This function is also supported in Oracle [ISO, 2008].

Oracle operators:

- inside(GM_Object, GM_Solid): a Boolean valued operator that returns true if a GM_Object is 'inside' (Egenhofer 9IM) the GM_solid [Ora, 2010b].

Some operators are important in this research but do not exist in neither ISO standard or Oracle specification, they are:

- interrupt(Object, Object): a Boolean valued operator that returns true if two city objects have geometric intersection that should be forbidden. Defining the forbidden intersection involves a check on the meaning of input objects, and is difficult to specify merely by geometry types (see also section 5.1.2 about different reactions on tree-tree intersection and building-building intersection in this constraint). Later on, it is realised for building-building intersection in section 7.2.1.
- composeValidSolid(GM_MultiSolid): returns a Boolean value if the input collection of solids altogether form a valid (according to Oracle definition) solid.
- jump(GM_MultiPoint, GM_Solid): returns true if a multipoint object, esp. mobile sensor, is found to be jumping in and out of a building solid during movement. This function is also realised and will be described in section 7.2.4.

- Abstract Constraints
  Some constraints can actually apply to all the instances of city objects. Instead of repeating them everywhere, and also to keep the UML diagram clean and brief, they are abstracted to a general level and apply to the highest abstract class City_Object (see also figure 28).

  1. Oneobj_oneloc: ensures that at one location there is only one object. Self-defined function 'interrupt' is used here. This summarises constraints bb1, bt1, br11, bg1, bw1, tr11, rg1, rw1, rr2, rg2 and gw1. Obj1 and obj2 here are two different CityObject instances.

Listing 6.1: Oneobj_oneloc

```
context CityObject
inv Oneobj_oneloc: self.allInstances() -> forAll(obj1, obj2 | obj1 <> obj2
    Implies interrupt(obj1, obj2) = 'FALSE')
```

2. Correct_Timeorder: (generalised from b1 and r1) checks that the starting time of an object does not occur after its end time.

Listing 6.2: Correct_Timeorder

```
context CityObject
inv Correct_Timeorder: self.starting_Date < self.end_Date
```

- Building Constraints
  Building constraints are attached to the UML diagram in figure 29.

  1. ValidSolid: (expression of b11) checks if geometries (solids) of all building parts that belong to one building together form an (Oracle) valid solid. This presumes conditions such as all the building parts are topologically correct (no gaps between adjacent ones and also no intersections).

Listing 6.3: valid solid

```
context Building
inv validSolid: self.Building_Parts -> composeValidSolid(geometry) = 'TRUE'
```

2. MinDist_TreeBld: (expression of bt1) checks if between one tree and one building the distance is larger than a minimum threshold. The min distance can be specified by the user.

Listing 6.4: Min distance between tree and building

```
context Building
inv MinDist_TreeBld: Tree.allInstances() -> forAll(t | distance(t.geometry,
    self.geometry) > minDistBld2Tree)
```

3. Address_matches_roadname: (expression of br12) checks if within a certain distance to a building there exists a road which has a name that precisely matches the whole or part of the building address.

Given one building instance 'self', this expression selects all the road instances that are in the neighbourhood. Then in the comparison of string, exactly one road instance must have its name equal to part (or the whole) of the building address. Here a standard OCL function 'substring' is used to fetch the road name from the building address. The parameters 'lowpos' and 'highpos' are depending on the format. For example, if an Address would be 'Jaffalaan9, Delft, NL', then lowpos = 1 and highpos = 10.

Listing 6.5: Building address matches road name

```
context Building
inv Address_matches_roadname: Road.allInstances() -> one(r | distance(self.
    geometry, r.geometry) < maxDist AND self.Address.substring(lowpos, highpos)
     = r.Name)
```

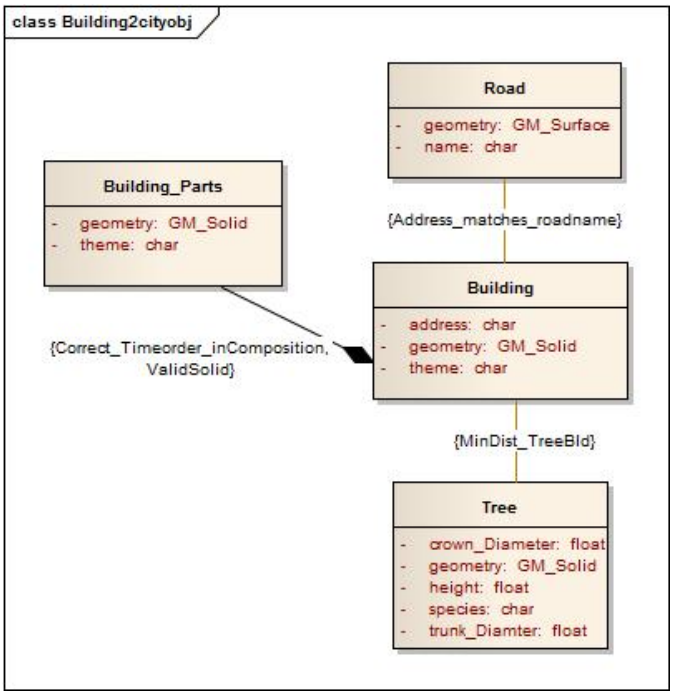


Figure 29.: UML model of constraints relevant to building object class.

4. Correcttimeorder_in_composition: (expression of b1 in composition of building) checks the time order between a building assembly and its parts. It defines that different building parts compose a building, so a part could not exist before the assembly being there first. In other words, the lifespan of building parts should be within that of the assembly.

Listing 6.6: Correcttimeorder_in_composition

```
context Building
inv: self.Building_Parts -> forAll(b | b.starting_Date >= self.starting_Date)
inv: self.Building_Parts -> forAll(b | b.end_Date <= self.end_Date)
```

- Tree Constraints
  1. MinDist2Tree/Road: (expressions of tt1 and tr12) ensures min distances between two tree instances and between a tree instance and a road instance. Each distance rule can have a different min distance threshold (i.e. the min distance of tree-tree 'minDistT2T' would be different from tree-road 'minDistT2R').

Listing 6.7: Tree distance rules

```
context Tree
inv MinDist2Trees: self.allInstances() -> forAll(t1, t2 | t1 <> t2 Implies
    distance(t1.geometry, t2.geometry) > minDistT2T)

context Tree
inv MinDist2Road: Road.allInstances() -> forAll(r | distance(self.geometry, r.
    geometry) > minDistT2R)
```

2. Aquatic_Close2Water: (expression of tw1) checks if an aquatic plant is close enough (comparing to a user-defined threshold) to water.

Listing 6.8: Aquatic_Close2Water

```
context Tree
inv Aquatic_Close2Water: Water.allInstances() -> exists(w | self.species = '
    Aquatic' AND distance(self.geometry, w.geometry) <= maxDist_Aquatic2Water)
```

3. Limited_Size: (expression of t1) restricts the size of a tree. The species-maxCrownvalue/Height/Trunkvalue are constants for each species. But the value can differ from one species to another.

Listing 6.9: Limited size of tree

```
context Tree
inv: self.crown_Diameter < SpeciesmaxCrownvalue
inv: self.height < SpeciesmaxHeight
inv: self.trunk_Diameter < SpeciesmaxTrunkvalue
```

- Sensor Constraints
  The inheritance from generalisation is helpful in optimising the property constraints for sensors (static and mobile types). The necessity of having measurements of location and time for all sensors and smooth value changes is specified in the abstract type *sensor* so that they can be automatically inherited by mobile sensors and static sensors (see figure 31).

1. SmoothValueChange: (expression of s1) restricts too fast a change in the measurement values of a sensor within a period of time. The expression ensures that between two sensor instances from the same type the value change is within a threshold. The threshold can be derived from change per time interval (can be second, minute, hour, etc.) and vary with sensor type. Since this rule applies to every sensor object, it is added to the abstract class.

Listing 6.10: SmoothValueChanges

```
context Sensor
inv SmoothValueChange: self.sensorType = 'Userdefined_Type' Implies self.
    allInstances() -> forAll(s1, s2 | s1 <> s2 Implies abs((s1.value - s2.value
    ) / (s1.measuring_Time - s2.measuring_Time)) <= Userdefined_maxChangeRate)}
```

2. Solarpanel_Outside: (expression of sb1) ensures that the solar panel is not inside any building. The geometry of solar panels is GM_Surface.

Listing 6.11: Solar panel outside

```
context Stationary_Sensor
inv Solarpanel_outside: Building.allInstances() -> forAll(b | self.sensorType =
    'Solar Panel' Implies Inside(self.geometry, b.geometry) = 'FALSE')
```

3. No_Jump: (expression of mb1) checks if a mobile sensor is jumping in and out of a building. The mobile sensor is treated as multipoint that is in a series of time and the input building is a solid. The distance function selects the building instances b that are in the neighbourhood of this multipoint first. And then function Jump() checks if this multipoint is jumping w.r.t. b. See also the realisation of function Jump in section 7.2.4.

Listing 6.12: No_Jump

```
context Dynamic_Sensor
inv No_jump: Building.allInstances() -> forAll(b | distance(b.geometry, self.
    geometry) < maxDist Implies jump(self.geometry,
b.geometry) = 'FALSE')
```

- Road, Grass, Water and Terrain
  The main check within road, grass and water objects is to avoid their intersection. This is already abstracted in the general constraint 'Oneobj_oneloc' in pseudo OCL expression 6.1. The constraint relative to terrain in urban area is to match with ground objects (constraint n1). This can simply be achieved by saying 'terrain intersection curve (TIC, see [Cit, 2008]) of a CityObject instance cannot be empty' and thus is not formulated as an OCL constraint here.

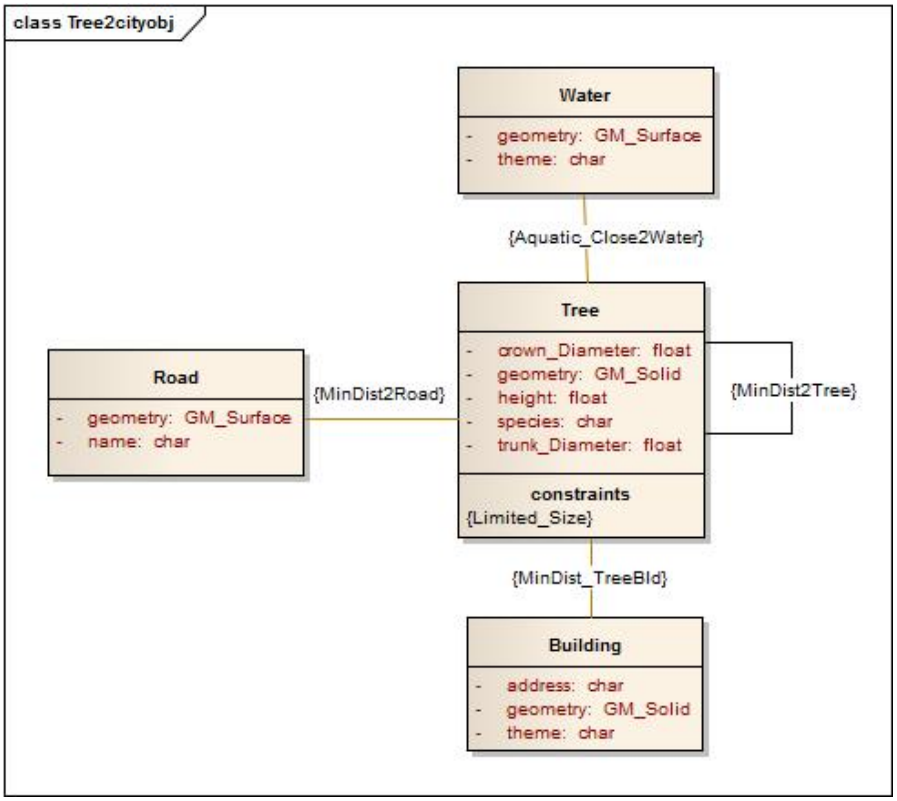


Figure 30.: UML model of constraints relative to tree object class.

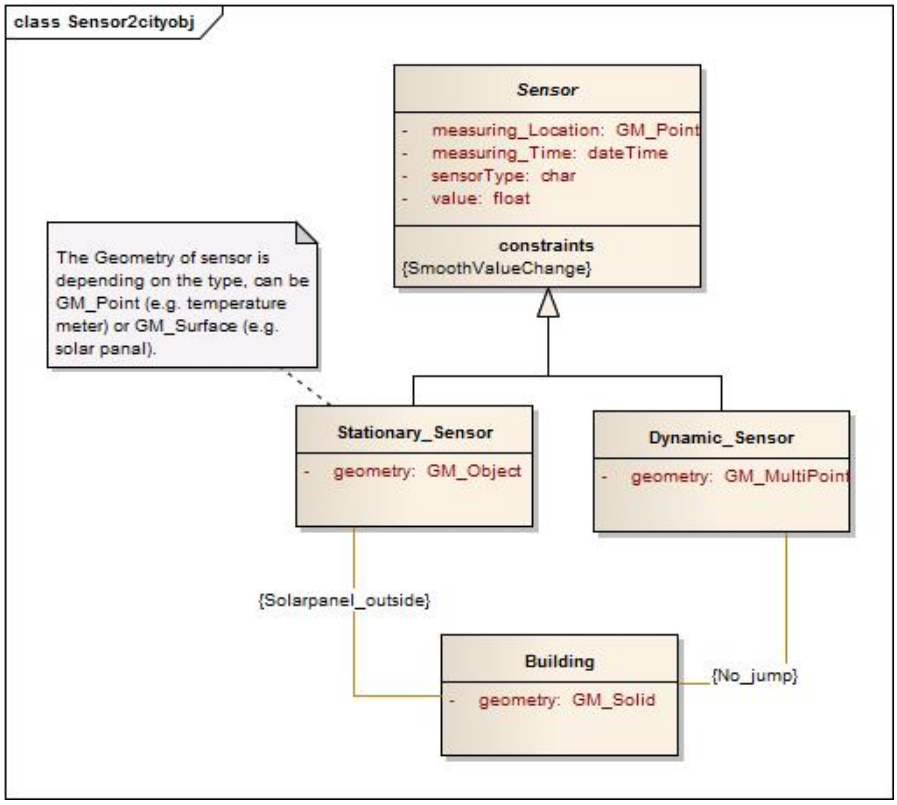


Figure 31.: UML model of constraints relative to sensor object class.

6.2 CONTRADICTING CONSTRAINTS

As you may have already noticed, there are quite a few constraints which hold for the same object class, either about single instance or about multiple instances. When the amount of constraints gets larger, it may happen that some rules contradict, which means no solution can be found to satisfy them all. In practice, no instance of a certain class can be stored in the database (some tables will always be empty). Although the currently implemented constraints are proposed carefully by human reasoning and the amount is not that large, in the future when more rules are established, this issue has to be considered.

Checking constraint consistency is a research topic by itself. During this thesis research, several aspects that are useful for geographic database constraints are inspected, that is, an unambiguous definition of Geo-DB constraints inconsistency, specification of additional check-rules and tools for automated constraints comparison.

At the beginning, 'contradict' seems intuitively clear at a glance, that is, several constraints together cause an impossible solution. Later as more content about general constraints consistency issue is studied, which exists in fundamental mathematics, it is found necessary to give a clear definition to distinguish consistency of geo-database from the rest.

The explanation of what 'contradict' means can be borrowed from usual mathematical notion, which states: The truth of a formula is defined w.r.t to a certain mathematical structure that describes the world, which is usually called a model. A database can be treated as a model, so is any particular table therein which represents a class of real world objects. A constraint is a formula. A statement like 'the model M satisfies the formula F' means basically that constraint F is satisfied in a table or several tables M. If formula F1 and formula F2 contradict, it means that F1 AND F2 $->$FALSE. This is equivalent to 'no model M exists to satisfy both F1 and F2'. So there is no object from table(s) M that can have F1 and F2 both satisfied. Generalised to a set of formulas/constraints it becomes: a set of constraints is inconsistent if there are no real world object that satisfies all of them. Therefore, 'unsatisfiable' is a better definition of what 'inconsistent' really means in this research.

An example will be made up to test how the database system actually reacts upon unsatisfiable constraints (see section 7.2.5). Three parametric rules regarding distances amongst tree, building and road are proposed as:

- Tree-Building: tree must be placed within a distance of 5 meters from building.
- Building-Road: there must always be at least 10 meters distance between a building and a road.
- Tree-Road: tree must be put within a distance of 2 meters from a road.

An illustration is given in figure 32. Assume that the building is already stored in database. Then a road object that is 10 meters from the current building is added. As you can see, after that it becomes impossible to have a tree object anymore, because it will either be close to the building (fulfills Tree-Building rule) but far away from the road (satisfies Tree-Road rule) or the other way around. The result is the tree table will always be empty.

Despite this strange outcome, the database can exist and keep functioning under such situation, just that it will not have tree object. When taking a second thought, what this outcome really contradicts is a common sense that can be expressed as 'it must be possible to have tree objects in the database'. More generally, every object class (table) the database schema is

designed to model must be allowed in the database. Explicit statement of this type of rule (let's call it extra check-rule), which reflects human intuition or common sense, is subtle (easy to neglect) and yet important for constraints reasoning.

More specifically, to avoid the possible conflict from the example distance rules, an extra check-rule to gurrantee the existence of tree object can be:

'between building and road that are more than 10 meters apart and have no other building or road in between, it must be possible to have tree(s)'.

Note that this rule also contains two other conditions that can be expressed separately as:

'it must be possible to have buildings' and

'it must be possible to have roads'.

This logic can be expressed as formulas in OCL but is difficult to be enforced as a standard constraint. A reason is due to the characteristic of constraint. It is easy to formalise and enforce a constraint that holds for all time. But a rule like 'tree is possible to exist' does not say when. The possibility/existence can be manifested at any moment and is difficult to specify.

One can of course choose to just specify this extra check-rule as it is now in UML model for database designer to read. A better way is to input a series of constraints, plus extra check-rules, and then run a comparison to find out the counter-rules. Phrasing the extra rule in a machine-understandable way can pave the way, which is found in logic programming, e.g. first-order logic [Jeremy Avigad, 2007]. First-order logic (FOL) allows rewriting a statement, in our case a geo-constraint, into several predicates (or axioms) as input in theorem prover. Each predicates can be examined according to definition of them (from user), and then brought together to form a more complex rule. Then the theorem prover, i.e. Prover9 [pro, 2009], can be used to compare the FOL axioms and find out the counter-rules. For example, to describe the possible existence of tree, a formula F_t can be F_t = Exists x (tree(x)). Given the rules that are to be checked in the theorem prover, F_1 ... F_n, if (F_1 AND F_2 ... F_n AND F_t $->$ FALSE is calculated, which is equivalent to (F_1 AND F_2 ... AND F_n $->$ NOT(F_t), then the constraints are conflicting and disallowing expected tree object.

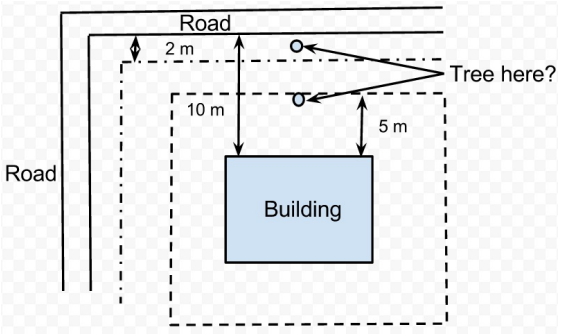


Figure 32.: A possible scenario of inserting a tree when tree-building-road distance constraints apply.

The three example distance rules can be expressed as formulas in FOL:

```
F_tb = ForAll b, t (IF Building(b) AND Tree(t) THEN dist(b, t) <= 5)

F_br = ForAll b, r (IF Building(b) AND Road(r) THEN dist(b, r) >= 10)

F_tr = ForAll r, t (IF Tree(t) AND Road(r) THEN dist(t, r) <= 2)
```

Also the extra check-rule can be rewritten in a FOL-like manner and then together with the three formulas evaluated by theorem prover to find out counter-rules.

```
Exists b, r(
  building(b) AND
  road(r) AND
  dist(b, r) > 10 AND
  NOT(
    Exists x(
      between(x, b, r) AND
      (building(x) OR road(x))
    )
  ) AND
  Exists t(tree(t) AND between(t, x, r))
)
```

Notice in FOL the notation blah(x, y) can mean two things, either a predicate (or proposition/statement in some contexts) that says blah is true for (x, y) combination, or a function (or operator) that denotes the result of applying blah to (x, y) combination. In the example expressions here the terms Building(), Tree() and Road() are unary predicates saying that the object within the bracket '()' belongs to the object class, e.g. Building(b) holds if b is a building object. And dist() is a function about binary relationship that denotes the result of distance calculation between two objects within the bracket, e.g. dist(b,r) returns the distance between object b and r. And between() is a ternary predicate showing that the 2nd object is spatially between the 1st and the 3rd object, e.g. between(b,x,r) means object x is spatially between object b and object r.

To conclude, consistency of geo-database constraints in this research means a series of constraints do not forbid object that should be possible to exist in database. Depending on the number and relation (or structure) of constraints, the inconsistent constraints can be very obvious to human, as is exemplified by distance rules. Or they can be hidden and difficult to find out. How several different constraints applying upon the same object would influence each other and what the outcome might be is a more complicated reasoning issue.

Nevertheless, to achieve the evaluation of possible contradiction, extra check-rules have to be explicitly specified so that both human and machine reader can understand. They can be specified as formulas in OCL as was shown, but almost impossible to address as constraints like a usual database constraint can. A workaround is to rewrite them in first-order logic. And then with well-defined structure of axioms, it is possible to do automated comparison in theorem prover. However, FOL does not understand the spatial predicates/functions yet, e.g. 'distance()', 'overlap()', 'buffer()', etc. How much workload has to be given to translate a single geo-constraint to axioms so that the FOL theorem prover can understand is not tried yet. A

general conception is the FOL theorem provers hardly support geographic rules. If human knowledge input necessary in FOL in order to 'automate' the reasoning requires more effort than doing the comparison by human reasoning, then it is not 'automatic' as one really means.

6.3 TRANSLATION OF CONCEPTUAL MODEL

The Model Driven Architecture principle, being supported by Object Management Group (OMG), provides a framework to define how models in one domain-specific language can be translated to models in the other languages ([Kleppe et al., 2003]). Considering that the constraints will finally be implemented in the database system, it is of great help if UML/OCL model can be automatically translated to SQL script. Some useful tools are explored and their strengths and weaknesses are given below.

- Enterprise Architect
  Enterprise Architect (EA) is used in this study to draw UML diagrams. With DDL (Data Definition Language), a UML model can be translated to SQL script and create a corresponding database schema. It provides OCL writing as was used in a research project presented in [Hespanha et al., 2008]. But except the referential key (PK/FK) constraints and check constraints, the current version does not support the translation of OCL constraints to DDL. In the research of this thesis, the syntax checking of EA is found to be rather weak and unreliable. Statements which are obviously incorrect in syntax can still be validated successfully.

- Hibernate (Spatial)
  Hibernate is a model translation library for Java language. It has a SQL-like dialect called Hibernate Query Language to translate an object-oriented domain model to a traditional relational database [hib, 2011a]. One of its primary function is mapping from Java classes to database tables. Hibernate has an extension, Hibernate Spatial, to handle the geographic data [hib, 2011b]. With the help of HQL, it can translate the spatial features coded in Java to spatial database tables/attributes. It is more like a Java library and does not give a graphic user interface to do translation as EA or Eclipse supports. Using it calls for a Java programming skill. For people that are not familiar with Java and just want to drag-and-draw it does not seem to be a good choice.

- Dresden OCL2
  Dresden OCL toolkit works in Eclipse environment and provides a set of tools to parse and evaluate OCL constraints on various models, i.e. UML, EMF (Eclipse Modelling Framework), Java metamodel and XML [Wilke et al.,]. Basically, it converts a rule to SQL query that finds out all the records that do not satisfy this rule. The input of code generator includes UML diagram of the appointed model and the associated OCL notations. Giving a UML model (or EMF or XML) together with correctly parsed constraints in Eclipse system, within several clicks an executable SQL script can be generated.

  From the actual use of OCL2SQL tool, it is found out that whilst UML drawing is standardised and always the same regardless of platform, the UML file (XMI) is not well understood or shared amongst different software/platforms. An output XMI from EA UML model cannot be understood by Eclipse, not saying the attached OCL expressions. And the constraints have to be separately coded than attached to the UML model. An

alternative is to draw UML (or EMF) model in native Eclipse environment which also can avoid UML translation from one platform to another.

To conclude for model translation, for one that does not want to do much Java programming, Dresden OCL2SQL tool seems the easiest to generate SQL code from OCL expressions of non-spatial constraints. Within few clicks SQL code can be achieved, if the core model (like UML) is understood by Eclipse, and OCL expressions correctly parsed.

For spatial constraints, the 3D geometries standardised in ISO19107 and 9I topological names are not yet included in OCL library. Articles from [Pinet et al., 2007] and [Werder, 2009] show the possible extensions for 2D objects and 2D topological relationships. To achieve automatic SQL generation the extensions toward 3D spatial types and operators are still required.

Furthermore, although Oracle supports ISO19107 spatial predicates and types, they are mainly 2D. If we would have the spatial constraints in OCL translated into SQL, still not many corresponding 3D operators/functions in DBMS can be used to immediately implement them (see the 3D functions in table 6 from appendix B.1). Therefore an immediate implementation of 3D geo-constraints is to do programming in database. This will be described at next chapter.

7

IMPLEMENTING CONSTRAINTS

As was discussed in the last chapter, the current model translation technique does not support immediate and automatic translation from spatial OCL to SQL. Plus the fact that pseudo OCL expressions from section 6.1 may not be perfect, an optimistic way to implement the 3D geo-constraints is to programme them in database with PL/SQL. In this sense, the power of data maintenance and geospatial functions from database can be combined to have 3D geo-constraints integrated in database seamlessly.

An in-negligible fact is that the existed 3D functions in Oracle Spatial are relatively new and not widely extended. Many constraint checks can not be immediately implemented yet. To be able to compute the detailed topological relationship, as required from the designed geo-constraints in section chapter 5, new functions need to be developed. Section 7.1 will take a close look at implementation of a new 3D spatial function. This function is able to tell a more detailed topological relationship between two surface objects so that constraint about 'Oneobj_oneloc' in listing 6.1 (from section 6.1) can be realised for two building instances. The using of this function to carry out the constraint will be described in subsection 7.2.1.

The realisation of constraint that allows exceptional instances is achieved by different triggers and will be discussed in subsection 7.2.2. A non-spatial constraint regarding small change of sensor observation value is implemented in section 7.2.3. The 'No_Jump' constraint (constraint mb1 from table II in appendix A.2) about moving pattern of mobile object is realised in subsection 7.2.4. And the function to tell event 'jump' (proposed in pseudo OCL in section 6.1) is also discussed in that subsection. After that is a realisation of three contradicting constraints (about distances amongst tree-building-road) proposed from section 6.2. This implementation shows how the trigger mechanism would react upon conflicting constraints. And the last implementation, as you will see in section 7.2.6, is about LOD consistency of single object.

7.1 A NEW GEOMETRIC FUNCTION: SDO_SURFACERELATE

As was discussed in section 5.1.2, distinction between 9IM 'touch' and 'intersect' is necessary to realise constraint such as 'Oneobj_oneloc' from section 6.1. In 'touch' relationship the refinement to see 'strong-touch' differently from 'touch' is also necessary. The most related function in Oracle Spatial database for this work is a 3D function `SDO_AnyInteract`. It is able to detect if two 3D objects are 'disjoint' or not. But it does not tell more details about what is happening in the 'non-disjoint' side. That is to say, two geometries which have 'touch' and 'intersect' do not make any difference to it. Considering the demand of necessary discrimination in topological relationship, a new function named '3D_SurfaceRelate' is developed here.

As the building objects in database are currently modelled by separate surfaces, a possible relationship between two surfaces from different buildings is 'overlap' (9IM). There are two

variants which need to be treated differently under this situation. One variant is two buildings share a common face because they are adjacent and have connection (a fine case). The other is one building is 'coveredBy' (9IM) by the other. This variant is seen as 'a building intersecting another building' which violates constraint 'Oneobj_oneloc' (a wrong case). A close examination of the surface storage indicates that these two variants can be discriminated by checking the orientation of surface vertices. If 'overlap' is caused by adjacent buildings, the orientations of two overlapping surfaces will be opposite. If it is caused by 'coveredBy' situation the orientations will be the same. The former will be called 'overlap-opposite' and the later 'overlap-parallel'.

Therefore, function 3D_SurfaceRelate is designed so that given two planar surfaces as input, it will be able to tell if two geometries 'intersect', 'overlap-opposite', 'overlap-parallel', 'strong-touch', 'touch' or 'disjoint' (see figure 33 for its I/O). It combines self-developed algorithm from computational geometry with SDO_AnyInteract and existed 2D operators/functions of Oracle.

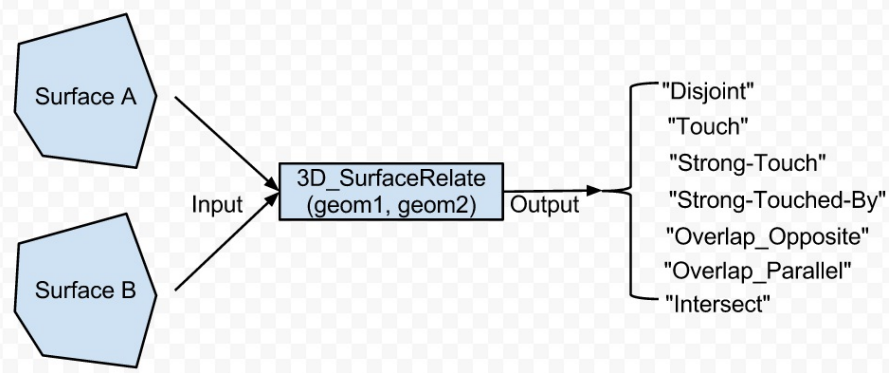


Figure 33.: The I/O of new function '3D_SurfaceRelate'.

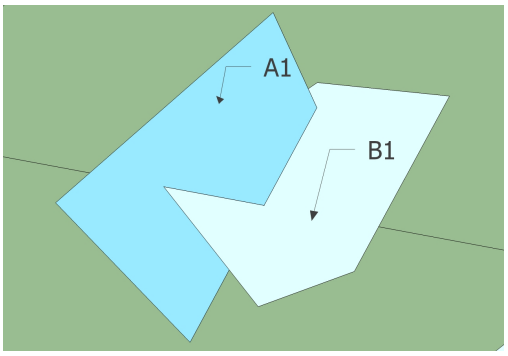
The principle of this algorithm is that each planar surface belongs to a plane (infinite boundary). When two planes are not parallel, they meet in a line of intersection. If any surface from one plane should have any contact spatially with a surface from the other plane, it can only be somewhere along the line of intersection. If two surfaces intersect, this line will cross both interiors (see figure 34(a)). If A strong-touches B, the line will cross B's interior and go through A's boundary, and vice versa (figure 35(a)). If they merely touch, this line will only intersects the boundaries (figure 36(a)).

Algorithm details are explained below (see also code in listing C.1 presented in appendix C.1):

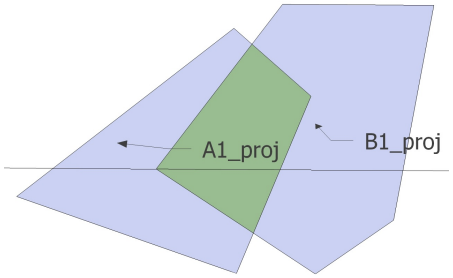
Given any valid (in Oracle) surface A and surface B, if they have any contact, function `SDO_AnyInteract()` returns True (step 0). (Although the ring test indicates that AnyInteract may see a disjoint case as interact, it does not miss interact cases. So if AnyInteract gives result FALSE then two surfaces indeed have no spatial contact.) Then these two objects are passed to the further check (see flowchart in figure 37).

- Step 1.
  Check if either surface is vertical.
  If yes, rotate them (the same rotation and same random angle for both) in 3D space until both are non-vertical.
  (See module 3D_Rotate in listing C.3 and line 97-102 in listing C.1).

- Step 2.
  Calculate the parameters of plane each surface belongs to and see if planes are parallel.
  If yes, then calculate the relation their projections in 2D (x,y) plane have (using SDO_GEOM.RELATE 2D available masks).
  If the relation indicates two interiors have intersection, be it EQUAL or CONTAINS or COVERS or COVEREDBY or INSIDE or OVERLAPBDYINTERSECT, then check also the orientation. For it can be two adjacent walls from different buildings (opposite orientations, two normal vectors point to opposite directions, which is fine), or one building surface placed in a wrong position (same orientation, two normal vectors point to the

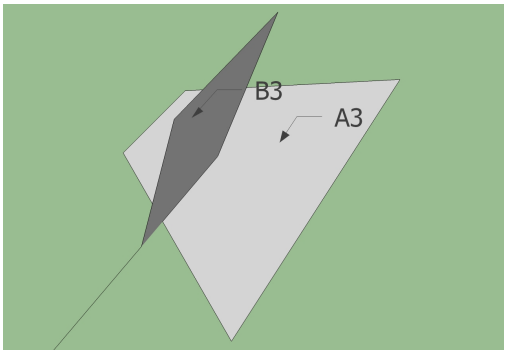


(a) In 3D, the line of intersection from two planes crosses both A1 and B1's interiors.

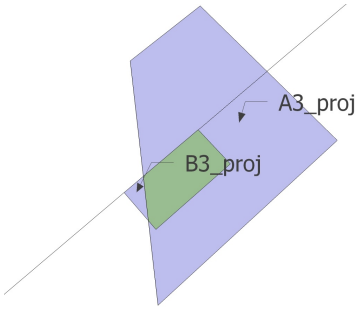


(b) In 2D, the projected intersection line crosses the shared geometry of both polygons. In this case A1_proj and B1_proj share a polygon (see the green hole).

Figure 34.: Intersecting polygons in 3D, their projections in 2D and the line of intersection. The hollow in 2D is the shared geometry.



(a) In 3D, the line of intersection crosses A3's interior and B3's boundary (so it is B3 Strong-Touches A3).



(b) In 2D, the shared geometry is also a polygon. But projection of the intersection line crosses only the boundary. And crossing occurs on A3_proj's interior and B3_proj's boundary.

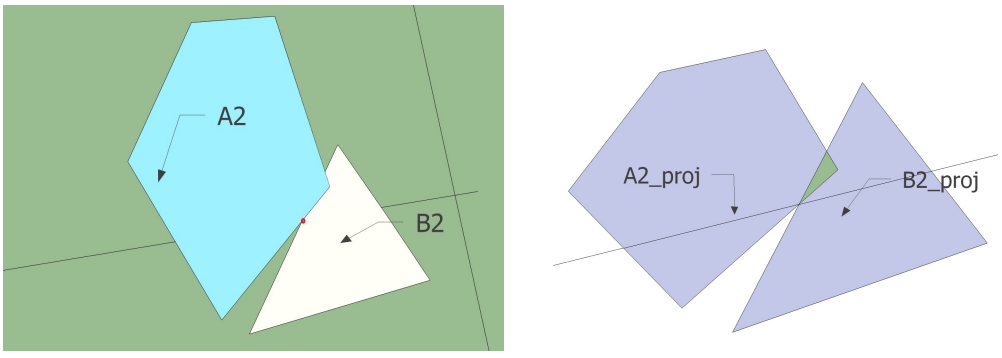
Figure 35.: Strong-touching polygons in 3D, their projections in 2D and the line of intersection.

same direction, which cannot be accepted). The former returns a string 'overlap-opposite', and the latter 'overlap-parallel'.

- Step 3.
  If two surfaces are not parallel or vertical, calculate the parameters of line of intersection where the planes meet.
  Then choose two points on this line to make up a 3D line. Since the next step is to check the relation between a 2D line and another 2D object (polygon or line or point). If the end point of the line would be intersected, there will be more topological relationships returned than the expected (cross or contain).
  To simplify the computation, the line is made to be like infinite to the surfaces. Thus the end points are selected from a broader extend, that is, double-sized bounding rectangle.

- Step 4.
  Project surface A and surface B, and line of intersection to 2D (x,y) plane. Then we get two polygons and a line. The polygons will share a common part, which can be polygon(s) or line(s) or point(s).
  Use Spatial function `SDO_Intersection` to retrieve the common parts (if two polygons overlap, it returns the overlapping region; if two polygons touch, it returns the shared line(s)/point(s)).
  Check relation the line has with the common part.
  If the shared geometry is on both interiors and is crossed by the line, then two surfaces in 3D intersect (see example figure 34(b)).
  If the line only goes through boundaries of both polygons two surfaces touch (example figure 36(b)).
  If the line disjoints the common part then it is disjoint (this is designed especially for the DISJOINT situations where AnyInteract() would return 'TRUE').

- Step 5.
  For the touch relation, use SDO_Intersection to obtain the shared part of intersection line



(a) In 3D, two touching polygons share a part of boundary (in this case it is a point), which is contained by the line of intersection.

(b) In 2D, the shared geometry (the overlapping triangle) only touches the line of intersection.

Figure 36.: Touching polygons in 3D, their projections in 2D and the line of intersection.

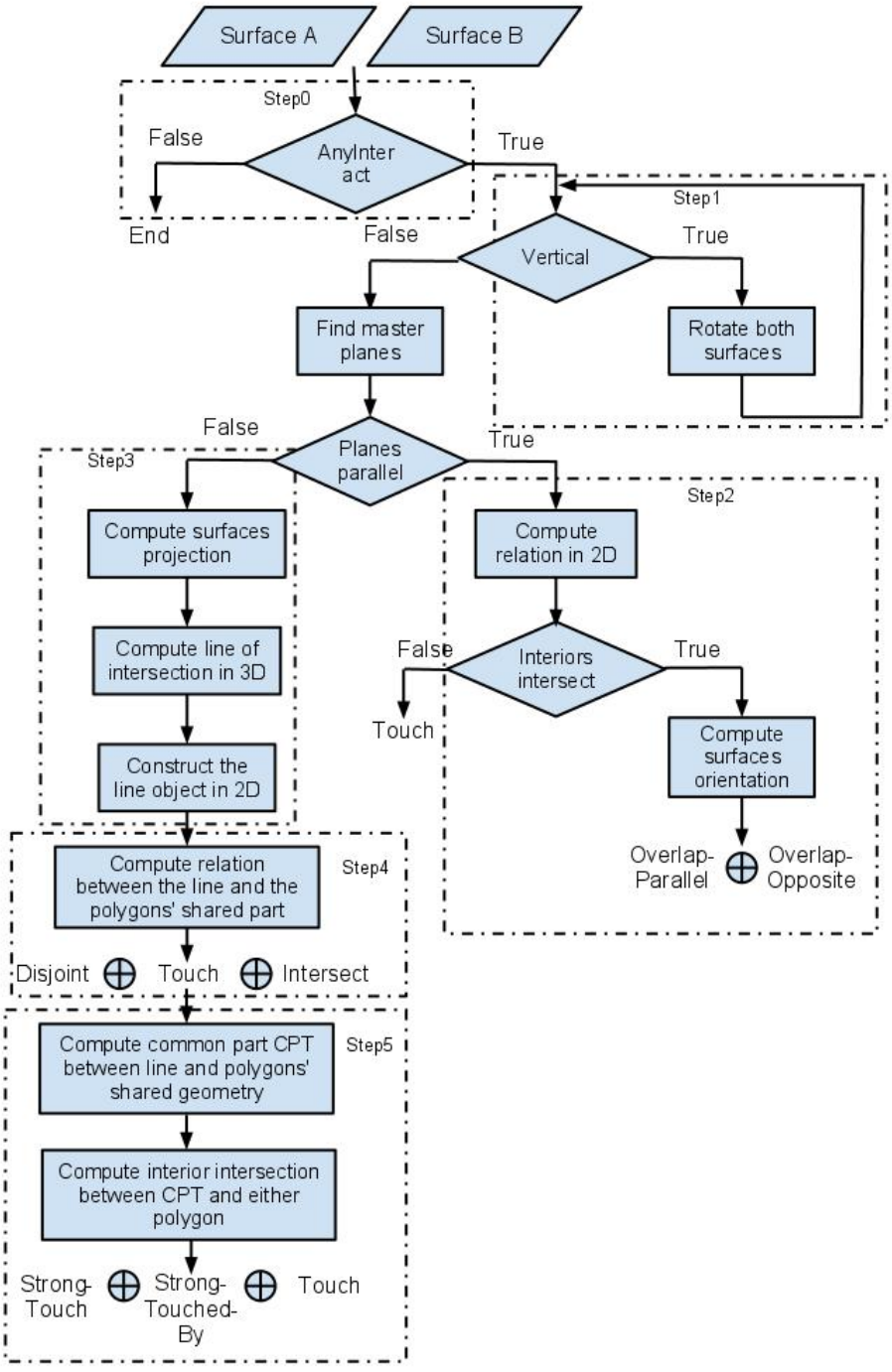


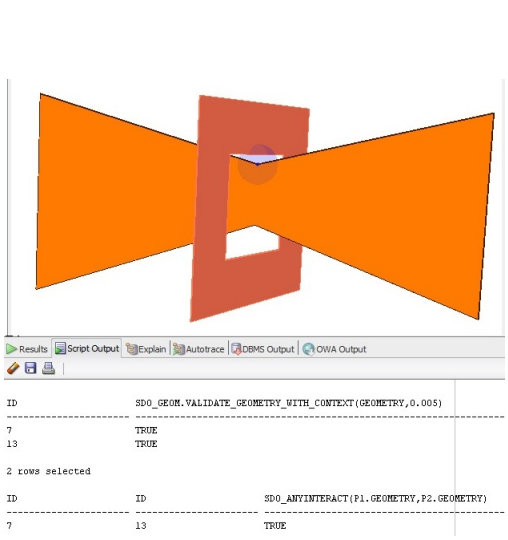
Figure 37.: Flowchart of the new spatial function 3D_SurfaceRelate.

and the shared geometry from two polygons.
If this shared geometry is inside either polygon's interior two surfaces strong-touch (see example figure 35(b)).
If it is interior of polygon A (projection of surface A), and boundary of polygon B, then the result returns 'A strong-touches B' or 'A strong-touched-by B', and vice versa.
If no intersection at all on either interior then it remains touch.

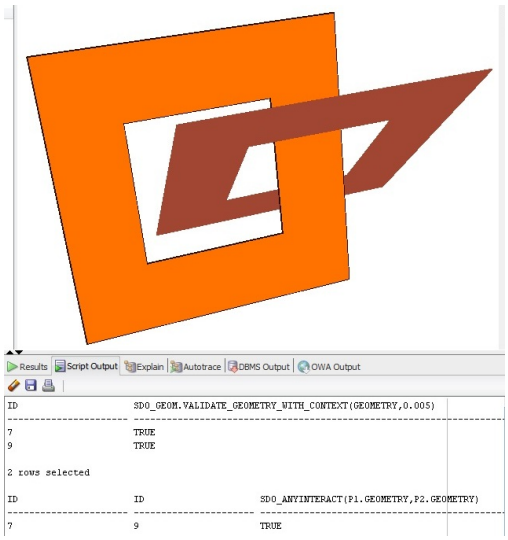
A group of 3D polygons are used to test the performance of 3D_SurfaceRelate, including special cases like concave/convex polygons, polygons with/without hole. In appendix B.3 the tests polygons are organised as:

- polygons that it returns 'intersect' are in figure group 44
- polygons that it returns 'strong-touch' are in figure group 45
- polygons that is returns 'touch' in figure group 46)

During the test SDO_AnyInteract appears to be problematic in two cases, both of which contain a (valid) 3D ring (3D polygon with a hole). In one test, the other object is a normal surface that goes through the hole without any contact (see figure 38(a)). In the other test, the other object is also a 3D ring, which chains this ring, but has no contact with the inner ring (see figure 38(b)). However, as is displayed in the figure, SDO_AnyInteract returns TRUE for both cases. This problem with ring object remains the same in SDO_GEOM.RELATE (mask='ANYINTERACT') which does not use spatial index as SDO_AnyInteract but rather actual geometry.



(a) A 3D polygon goes through a 3D ring and no touching occurs on the ring's inner boundary.



(b) Two 3D rings chain together and no touching occurs on the inner boundaries.

Figure 38.: Two disjoint tests (with ring) that AnyInteract returns 'TRUE'.

The other problem is found in function `SDO\_Inside` along the way. The expression SDO_INSIDE(geometry1, geometry2) evaluates to TRUE for when geometry1 has the INSIDE topological relationship w.r.t. geometry2, and FALSE otherwise. The geometry1 can be any Oracle geometry type but geometry2 must be solid. According to the definition of topological relationship in Oracle Spatial 11g2, INSIDE relation is the opposite of CONTAINS, which means 'The interior and boundary of one object is completely contained in the interior of the other object' [Ora, 2010b]. The example from document does not show anything 3D. The 3D line and solid tests against this operator works fine. But a strange situation happens on 3D point object. When a 3D point is exactly on the boundary of a solid (be it face, edge or vertex), this operator returns also 'TRUE'.

7.2 IMPLEMENTED CONSTRAINTS IN CCC DATABASE

Five constraints are implemented and each implementation is explained in detail in the following sections. They are implemented to show constraints in conceptual level from table II (appendix A.2) and OCL expressions in section 6.1 can be realised in database system. These implementations can be extended to include more city objects and attributes in the future.

- Building_Interrupt_Building: uses function 3D_SurfaceRelate and realises the constraint bb1 from table II in appendix A.2.
- Tree-building min distance exception: realises distance rule (bt2 in table II, appendix A.2) allowing exceptions (see section 3.1.2) for individual instances.
- Smooth temperature change: realises constraint (s1 in table II, appendix A.2) about maximum changing rate of sensor observation in time.
- Moving object restriction: checks moving pattern of mobile object (see constraint mb1 in table II, appendix A.2) and forbids 'jump' (proposed function from section 6.1).
- Contradicting constraints: three made-up contradicting rules proposed in section 6.2 are realised with triggers. It shows what the outcome of having contradicting rules in database can be.
- LODs consistency: guarantees the multi-scale geometries of the same object are consistent. A bug in Spatial 11g 3D function `SDO_AGGR_MBR` is found when realising this constraint.

All the constraints are coded as procedures within a PL/SQL package (see appendix C.2) and called by fired trigger. Notice the roles of row-level trigger and statement-level trigger in this implementation are different. The former one assigns values to global variables shared in the procedures. The reason to identify the modified row is that the geometries of surface in 3DCityDB are all stored in one table. Running a geometric constraint check, say, intersection of surface, every time through all the records would consume a lot of time.

A better way to have geometric intersection check is to select the objects that are close in space to the modified object. Then in these likely-intersected objects the check can be run and thus desires less time and resource. This requires a spatial query at run-time, which has to be called by trigger. However Oracle row-level trigger about database table has a mutating-table restriction. That is, the table at the time that a trigger statement is trying to modify cannot be queried/modified. A statement-level trigger does not have this restriction but the modified row is not identifiable in it. Therefore either type of trigger cannot manage to have row information and select query at the same time.

A workaround is to have a global variable specifying the modified row (or a particular attribute in the row) in the procedures, which is similar to default variable `:NEW` (and its attributes). Then use the row-level trigger to assign values from its `:NEW` to the variable in procedure (see triggers in listing C.7 from appendix C.3). And then in a statement-level trigger call the procedure that does spatial query according to this assigned row-like variable.

7.2.1 *Building_Interrupt_Building*

A general rule abstracted in section 6.1 shows a necessary restriction of city objects intersection. Almost all the existed city objects in CCC database are modelled by surface geometry, except

sensor which is by point geometry. Therefore the new function 3D_SurfaceRelate that examines the detailed relationship between a surface pair is of great use. With 5 object classes (building, tree, road, water, grass) the number of all binary relationship, either between instances from two classes or between two instances from the same class, is 5x5 = 25. If only the geometry of surface is necessary to determine object intersection it is better to have a general procedure that can be called in all situations.

However, giving the relationship returned by 3D_SurfaceRelate, the meaning of input surfaces is also needed. As was explained before, the same relationship, say 'INTERSECT', is considered differently between two trees and between two buildings. The former one can be accepted but the latter cannot. So each relationship string needs to be evaluated differently in each objects pair. Furthermore, the calculation with all 25 possible objects pairs may consume quite a few time. So it is better to code every objects pair separately.

In this sense, what the procedure that detects geometric interruption of objects should do first is to know the geometric relation with 3D_SurfaceRelate. Then a conditional statement interprets the relation with the meaning of surfaces. Then in relations that are considered as intersection for that particular pair of objects a run-time error is raised to warn user and rejects the data change.

This approach is implemented by an example procedure `building_interrupts_building` which checks if a modified building surface (updated or inserted surface geometry object) intersects any surface from other buildings (see also code in listing C.5, line 3-176). When the AFTER EACH ROW trigger passes the modified row into the package, the procedure selects surfaces from other buildings that have spatial interaction (using SDO_AnyInteract()) with this modified one. Then the surfaces are passed to function 3D_SurfaceRelate to calculate the exact spatial relation. For each relation, there is a counter and a list variable to store how many surfaces hold this relation with the modified record, and what their IDs are. The counters and ID lists will then make up the report (see table 2).

When all interacted surfaces are examined, if there is any instance with relation 'Intersect' or 'Overlap_Parallel', then at least two surfaces are interrupting, which is seen as an error and data modification must be rejected. A predefined run-time error is raised. Then a (DBMS_Output) message is printed to tell which surface(s) intersects/overlaps this modified record and who they are (their IDs). After that, a user-defined error (Raise_Application_Error) is given to show there is something wrong in the data change. Raising of this error will automatically roll back the effect of change (see also raising of exception at the beginning of this chapter 2.3). Therefore the table remains the same as if nothing has happened.

The test of core function 3D_SurfaceRelate has been shown in appendix B.3. The test of procedure is carried out with two cubes, one of which is smaller than the other. It is allowed by constraint to have relation 'Strong-Touch' (reversed 'Strong-Touched-By') or 'Overlap_Opposite' or 'Touch'. These situations between two surfaces from different buildings will be considered as two buildings are adjacent. A report telling which surface holds what relation against this modified record will be given.

Table 2.: Messages produced by constraint relative to Building Intersection.

Error Message	Warning Message / Analysis Report
-20001, 'Spatial Constraint: The building surface(s) interrupts the other building(s). Please correct the intersecting and overlapping data detected in the diagnosis.'	The updated surface intersects/overlaps surfaces of ID = 124, 125, 137.
-	3 surfaces have contact with surface ID = 140. It strong-touches 2 surfaces(s) of ID = 126, 138. It is strong-touched by 1 surface(s) of ID = 139. It touches 1 surface(s) of ID = 129.
-	The target surface does not have any contact with the other building surfaces. But it is not clear if it really disjoints from them or it is inside another building without touching the faces. It is better to visualise the surfaces.

As you may notice, the relation between building solids is computed in surface level. One reason is that Oracle does not provide operators on solid relationships more than simple check, e.g. SDO_Inside. And the solid from CityGML is currently stored in Oracle as a collection of surface objects and linked by referential keys, rather than a single solid instance. The surface-level approach directly takes available data as input without any a-priori comprising.

A difficulty is found in unambiguously telling if the disjoint of surfaces really means disjoint between solids. For instance, after comparing every surface from one building to all the surfaces from the other, the relation turns out to be disjoint. Now it can be that one building is inside the other, and simply has no contact with the other's bounding faces. To tell this condition from the real disjoint, a possible solution is to form a solid (by composing the surfaces from one solid) for each building, and then check the relation between solids. This can easily become a huge computation due to the storage structure of a building solid in CityGML.

The default database schema from 3DCityDB [Kolbe et al., 2009] does not assign meaning of geometry in the geometry table, but indicates it in the other table and uses referential key to make the link. In practice, to reveal the meaning and the object a surface belongs to, two columns are added, CLASS_TABLE and CLASS_OBJECT_ID. The former shows the meaning of object. For example, if a geometry comes from a road then this column has value 'Road'. The latter column tells the ID of the assembly object in that semantics table. If a road face belongs to a record in table LANDUSE with $ID = 15$, then this column has value 15.

7.2.2 *Tree-building min distance exception*

From database point of view, given any situation a constraint is either satisfied or violated. There is no middle way or gray-area. To deal with uncommon and yet real situations which

make constraints more realistic and usable, especially parametric rules, a workaround is to move the instance explicitly marked as exception elsewhere. In practice, it rejects the change on the target table but stores the suspected cases elsewhere in database.

The exception about the distance rule between tree and building is implemented as an example. The rule states that a tree should be at least 2 meters away from a building. But tree with a shorter distance to a building does exist in reality. Hence when this rule is violated, the data should not be thoroughly rejected. But the specific tree instance should be added to exceptions.

What the implementation does is once a tree object being added is found to be a violation, it will not be committed directly to the table it means to go to. But rather, it will be moved by trigger to a special table that stores exceptional cases (see EXCEPTION block in listing C.6, line 252-277). The special table has all columns the original table has, plus two extra attributes. One is a flag with only values 'TRUE' or 'FALSE' (by default is FALSE), and the other the reason why this record is there (comes from message generated by DBMS_Output, e.g. 'this tree object is found too close to building'). The user can look into the instance himself in this table. If he the instance turns out to be an exception, he can change the flag from 'FALSE' into 'TRUE'. After that a trigger is fired, which inserts this instance into the original table (see trigger in listing C.10). And this instance just remains in the special table (for convenience of further check). A work flow of two triggers is shown in figure 39.

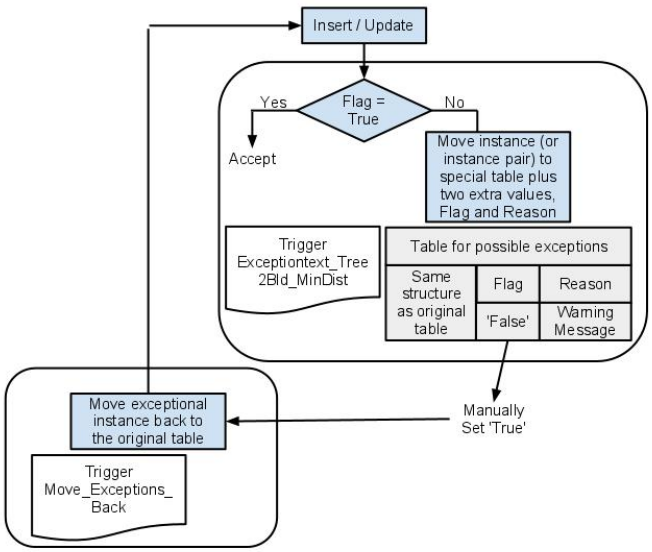


Figure 39.: Work flow of triggers for constraint exceptions.

The test data is a made-up cube as a building and a made-up surface as a tree instance, whose distance to one cube face is less than the min distance. At the beginning, to not let the exception

go into the original table, raise-application-error was used in the trigger. However, when a trigger causes an error, it will cancel all the transactions from DML statement (insert,update and delete), including the insertion to the special table. It is not possible to have error and any update/insertion happening at the same time. So an alternative is chosen, which deletes the record in the original table explicitly and then displays the message using DBMS_Output (see message content in table 3).

Table 3.: Messages produced by constraint exceptions relating to tree-bld min distance.

Error Message	**Warning Message / Analysis Report**
-	`The (part of) tree object with id = 23 is too close (dist < 2m) to building surface(s) of id = 45. The updated/inserted tree is suspected to be an error, and has been moved to table SURFACE_GEOMETRY_EXCEPTIONS. Please check the instance(s) yourself, and change the value in 'Accepted' into 'TRUE' if the instance is alright.`
-	`The exception instance has been inserted / updated in table SURFACE_GEOMETRY, and deleted from the exception lists. Free_Entry_Flag is set back to 'FALSE'.`

In principle, more properties can be added to the table of exceptional instances. For example, if the data is inspected by different users, adding properties as updater, date of change and reason why the instance is correct can help a better management.

7.2.3 *Smooth Temperature Change*

Restriction of temperature changing rate in time is the only realised temporal-thematic constraint. The parametric threshold is 'at most 3 Celsius degrees change in 1 minute'. This value is just set by experience of the temperature measurements in campus. It can be redefined by more scientific statistics. The realisation of this rule is only an example of implementation of temporal constraints on other sensor measurements. Test data is real temperature measurements. The error message is `-20006, 'Temporal Constraint: The modified temperature changes too fast! Max. rate is 3 Celsius degrees per minute.
Please check DBMS_OUTPUT message for detailed diagnosis.'`. This message can be also seen in Python IDLE (figure 42) when inserting a group of sensor observations from a text file.

7.2.4 *Moving Object Restriction*

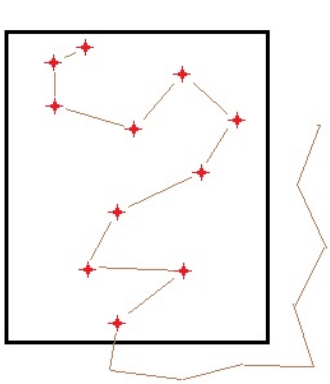
Temperature observations were captured by a person carrying temperature meter and positioning device. A presumed transaction is insertion of a set of points that is measured in one go. Usually in one go a carrier moves either completely outdoors/indoors or from outdoor into indoor. When sensor is close to the building boundary from outdoor, just a small degree of deviation in positioning would cause the location shifted to somewhere indoors (a 'jump'), and vice versa (see figure 40).

To tell if there is a jump, looking at an individual point does not give much clue. A series of points in time have to be considered in order to understand the moving pattern. An earlier attempt was made to construct a line object that is considered as moving path from the sensor points within one go. And then compare the topological relationship between this line and the building footprint. If somewhere in the middle of the line it crosses the footprint, it would then be considered as a jump.

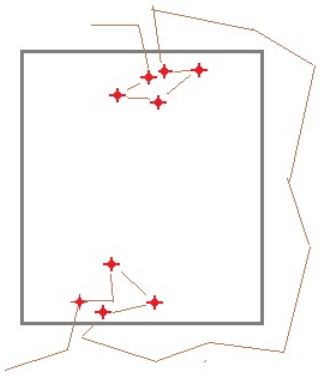
This approach is pretty dependent on the frequency of recording. In other word, when the time interval between two measurements is large, some important points (points of turning) in between maybe missing. For example, a carrier is moving along the boundary of building. When he makes a turn at the corner, the positioning device only records the points before and after the turning. In the construction, a line linking these two points may cross the building corner, which will confuse the constraint check. Another problem is with the line approach it is difficult to identify which points of the line cause the crossing, since points are sort of 'hidden' in the coordinate string. It is preferred to give user a detailed message about what is found out from the dataset by the constraint check. So simply saying 'the line object is suspected to cross building A' is not good enough.

Whilst weighting the pros and cons of the approach above, another solution in point level comes on the scene and becomes the final implementation. This implementation uses a window of three sequential points in time is used to 'scan' through all the sensor measurements. Given 3 sequential points, compare the relative location of each point with the building footprint. If the triplet relations are in pattern of 'inside, outside, inside' or 'outside, inside, outside', then the 2nd point is very likely to be a 'jump' case. The former one is seen as sensor jumping out of building, and the latter as jumping into building. After checking through points of the whole insertion, the jumping cases are recorded and delivered as a message to the user (see message content in table 4).

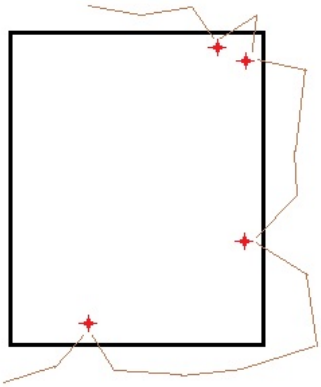
The test data is real observation carried by student. During the inspection of sensing data some points are found too far away from its precedent or following points. In other word, the speed of travelling between these points pairs looks too fast for human carrier (say, 40m in 10 seconds). Hence an extra check that limits the speed of mobile object is also implemented. The report is realised as an error message.



(a) On one of the moving paths, the sensor location moves from outside to inside the building.



(b) Another example of a moving path is when the sensor moves around and into a building through the entrances.



(c) An example of 'jump'. The carrier only hovers around the building but some locations appear to be indoors. Please note that the triplet where jumping point occurs in the 2nd position, follows a pattern of 'outside, inside(i.e. jump), outside'.

Figure 40.: Possible moving patterns and 'jumps' produced by mobile sensor(s). Red crosses represent indoor points.

Table 4.: Messages produced by the 'No_Jump' constraint.

Error Message	Warning Message / Analysis Report
`-20002, 'Tempo-Spatial Constraint: The speed of moving object is beyond its limit!`	-
-	`There are points possible to be JUMPING-in. They are with id = 79, 83, 90.`
-	`There are points possible to be JUMPING-out. They are with id = 45, 56, 69.`

There are several improvements that can be made upon this approach, depending on the carrier's moving pattern and how detailed the information of building is stored in database. Mobile sensing data in CCC database is carried by human and mainly on the ground. And the building model where the measurements took place around is a simple cuboid model without any interior structure, e.g. floors and stairs. So the topological calculation is done with the building footprint, which is actually a 2D polygon. And the sensing location, although stored with three dimensional coordinates, are also pushed down to ground level (2D) to simplify the computation. If the carrier goes upstairs and takes measurements from different floors, the 3rd dimension may need to be taken into account.

Also when possible jumping is detected, it could be a case that a carrier goes into (or comes out from) a building and then comes out (goes into). Thus the locations of entrance/exit of building becomes important to unambiguously tell if that really is a jumping or not. Information of the openings like doors are only available in LOD3 or above, which is not yet stored in this database. Because of this lacking information and the ambiguity, the current implementation does not give error when it sees a jump, but simply delivers a diagnosis.

7.2.5 *Contradicting Constraints*

The distance rules about tree-building, building-road and tree-road exemplified in section 6.2 are implemented by trigger. Each rule is checked by one trigger (see listing C.5 line 286-387) and produces an error message (see table 5). No warning messages are generated. The triggers about the same object, when defined in database upon the same spatial table, will run all in one go. When building and tree objects are already in the database, adding a road instance will fire the check of Building-Road and Tree-Road constraints. If nothing violates, the road instance will be accepted. Then it is impossible to modify any tree instance anymore.

Test data uses a cube representing building, a polygon (rectangle) as road and another polygon as tree. When inserting the tree polygon, which is within the max distance to the road polygon but beyond the max distance to the cube, error message from constraint Tree2House will pop up.

Table 5.: Messages produced by three contradicting distance constraints

Constraint	**Error Message**
Tree2House max. distance	`-20003, 'Spatial Constraint: Tree is too far (dist > 5m) from the house.'`
Tree2Road max. distance	`-20004, 'Spatial Constraint: Tree is too far (dist > 2m) from the road.'`
Road2House min. distance	`-20005, 'Spatial Constraint: Road is too close (dist < 10m) to the house.'`

7.2.6 *LODs Consistency of Single Object*

As was mentioned earlier, considering a single building as a solid object and its MBB a cube, the consistency check between 2 LODs becomes a comparison between two cubes. The centroid of cube gives location and magnitudes in x, y, z axises represent width, length and height, respectively. For a building object in CityGML is stored as a collection of surface instances, the MBB of building is actually a cube that enclose all its surfaces. As mentioned before, `SDO_AGGR_MBR`, which calculates a single bounding box enclosing specified (multiple) geometries, is a key operator on this realisation.

However, during tests function SDO_AGGR_MBR is found problematic. When giving more than one 3D polygon as input it returns an output with either incorrect SDO_Elem_Info_Array or incorrect coordinates. For the time being, LODs consistency constraint is not realised. If function/operator that aggregates 3D geometry becomes possible, it will not be difficult to carry out this example implementation.

8

CONSTRAINTS VISUALISATION

As was described earlier, there are two types of message, error message and DBMS_Output message. Once a constraint is violated, database will give an error message. This will always be delivered to the client. In contrast, DBMS_Output message is not guaranteed visible to all clients.

Although SQL Developer as a Oracle default data browser and analysis tool can display both messages, not everyone would manipulate with data using SQL Developer like a database developer or administrator would. More would modify data through clients like mobile devices (PDA, smart phone) that can capture observations in the field, or a CAD editor on a desktop (e.g. MicroStation) that enables visualisation and easy editing of geometries. The field data from mobile devices, outside automatic uploading from the chip, usually is manually uploaded by a piece of script (e.g. Python or Java programming language). A convenient and easily-recognised visualisation of database constraints in the front-end will largely expand their usability. In this research, Bentley Map and Python IDLE are tested to see how good the constraints are visualised.

Constraint from subsection 7.2.1 is tested here in Bentley Map. Imagine that a user loads campus model into Bentley Map to modify geometry of some building. He is not aware that one building intersects another building and thus he creates an unrealistic data. When he posts the modification back to database, he will see something similar to figure 41.

An advantage of having constraints in the central storage like database is that it is independent from platforms. No matter what way one is accessing the database through, as long as constraints detect something wrong the user can always be alerted.

The constraint to test in Python is 'smooth value change' from subsection 7.2.3. Imagine someone uploads a collection of temperature records. But somehow the records are found to contain two or more values, measured within 10 seconds, that have sharp difference (say 5 degrees). An error message regarding the unusual change thus is sent to Python and directly displayed in IDLE (see figure 42).

Since Oracle error message is not designed to contain lots of text and the raising of it by default rejects every real-time transaction, more abundant, detailed and specific information telling what really goes wrong or is likely to be wrong are given to user with DBMS_Output functions. However, DBMS_Output messages is not visible in Bentley Map's GUI, neither is it by default in Python IDLE (has to be configured manually). Although other clients are not tried, it is very likely that DBMS_Output message is invisible, or even is not recognised, to many of them. Hopefully Oracle database system will enable DBMS_Output messages to be visible to more clients as it has already done with the error message.

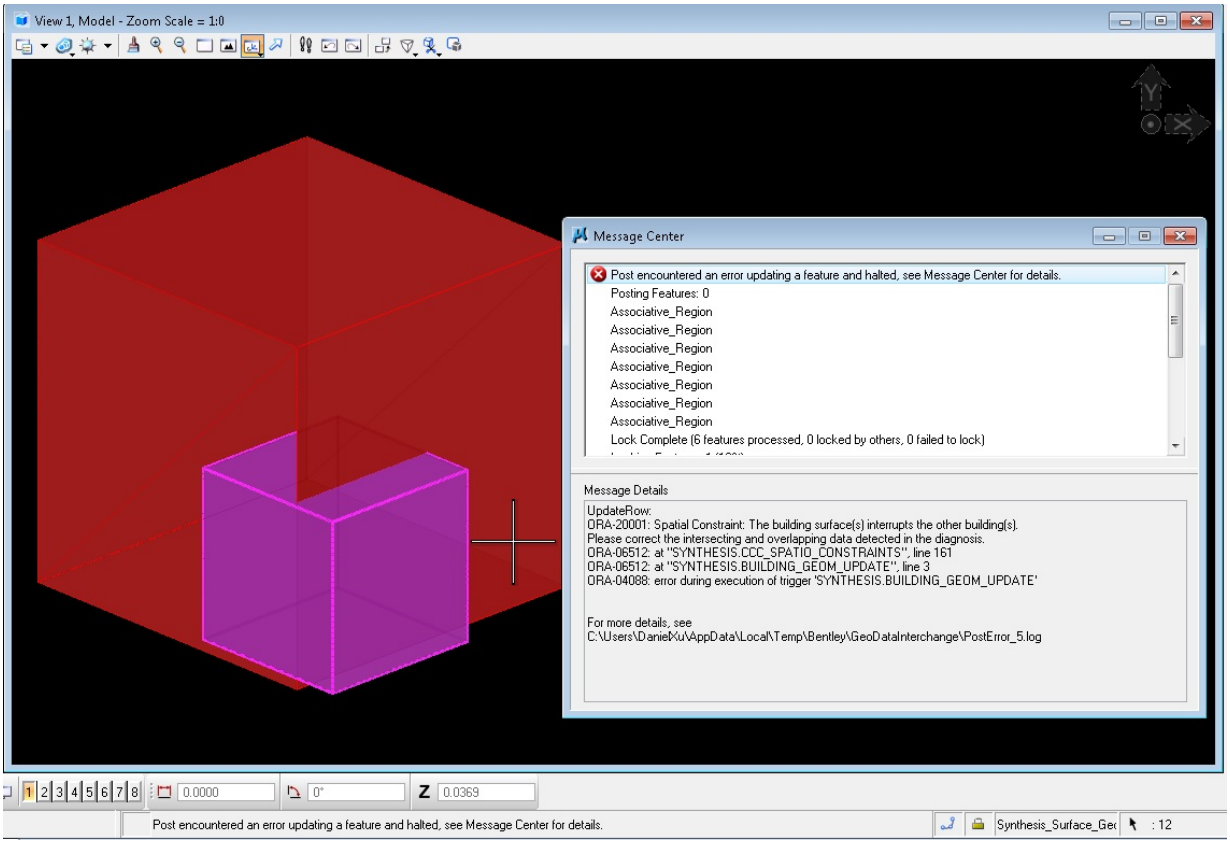


Figure 41.: The database trigger has detected that two buildings are intersecting. Oracle database then sends an error message to Bentley Map, and this message is then displayed at the Message Center which means it is easy to see on the GUI.

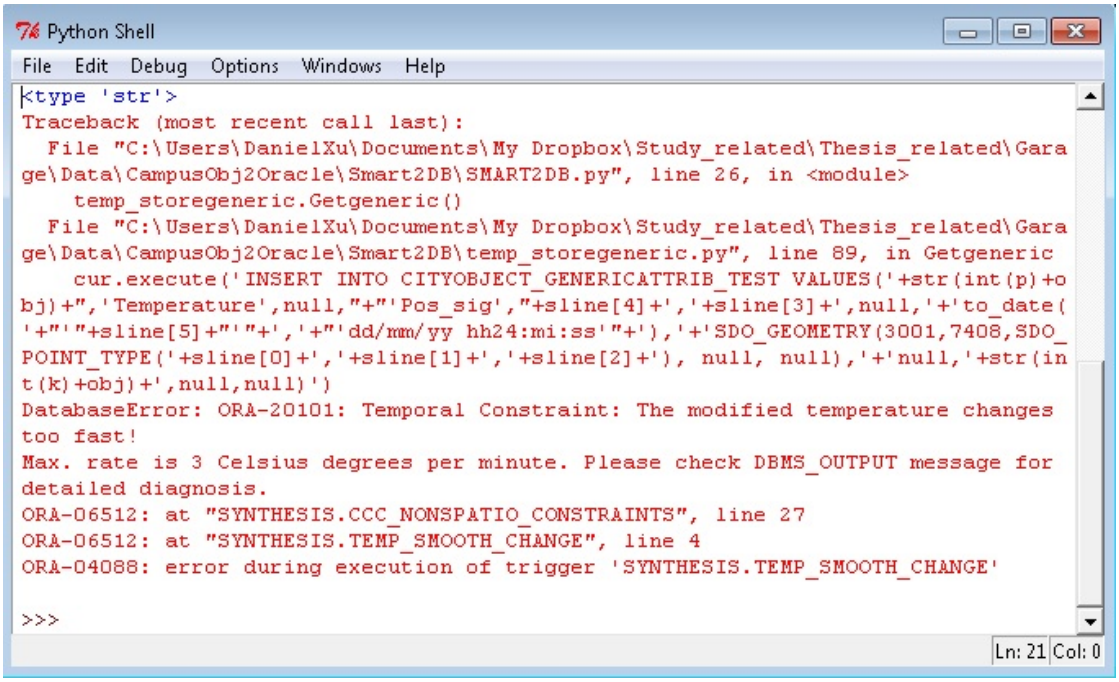


Figure 42.: The database trigger has detected that some temperature fluctuations have occurred too quickly within a short space of time. Then Oracle database sends an error message to Python, and this message is displayed directly on the IDLE interface.

9

CONCLUSION AND FUTURE WORK

The objective of the research carried out in this thesis is to design, conceptualise and implement 3D geographic constraints in a database. To achieve this goal, the Climate-City-Campus database which stores topographic city objects and attributes, is studied as a sample. Constraints are first discovered from this database and then expressed using natural language. Then objects in the statements are abstracted with geometric primitives and their spatial relationships with topological relationships. By doing so, spatial constraints are further specified into expressions with geometric primitives and topological terms. Following well-defined spatial types and operations as proposed in the ISO19107 standard, and with support using various tools, an attempt has been made to formalise these constraints using OCL. The pseudo OCL expressions are attached to the UML class diagram and thus can be considered a part of the model. Finally, some constraints are realised in the database by PL/SQL trigger mechanisms.

Results of this research, namely the answering of the research questions raised in the introduction (see chapter 1), are given in section 9.1. Some suggestions for researchers wanting to carry out further investigations into geo-constraints are included in section 9.2. Recommendations for improvements of standards and software are given in section 9.3.

9.1 RESULTS

During the study process, the research questions asked initially have been addressed, including current state-of-art geographic constraints modelling, their specification and formalisation, implementation and visualisation, conversion to executable code, ability/maturity of 3D functions in database. Other questions regarding a broader scope are also considered. Most of the initial research questions are answered:

1. Which 3D and sensor objects are needed?

All objects in the CCC database need constraints, namely building, tree, temperature sensor, road, grass, water and terrain objects. In the beginning, extra types of sensors (see section 4.2) were expected to be stored. Therefore considering that a sensor has some special properties that the rest of the campus objects do not have, e.g. mobility, high frequency of observation in a time interval (which means a constant change of value), various types that produce different parameters, a distinction meant to be made between other 3D objects (actually static campus objects) and sensor objects by this question.

As the research proceeded, no other sensor measurements appeared to be available, so the focus was shifted to 3D campus objects. And the existing temperature measurements made by mobile devices were studied in order to analyse moving patterns relating to mobile objects.

2. Which spatial, thematic, temporal and quantity constraints are necessary?

All the initially proposed constraints are shown in section 5.1.1 and tables in appendix A.2, of which 22 are spatial, 2 thematic, 1 temporal, 2 quantity. Some spatial constraints are then abstracted into a general constraint, e.g. intersection of objects and consistent LOD representations. The reduced number of constraints plus some new ones found during research (see table II in appendix A.2) are then expressed in OCL as given in section 6.1.

What constraints are necessary depends on what the purpose of having constraints is. This research aims to give a general methodology for designing and implementing 3D constraints so that the city model in a geographic database can be kept clean. Various research projects have been carried out in constraint categories e.g. spatial, thematic, temporal, quantity and a combination of them. So spatial constraints, esp. 3D, as a category that has not been studied in any great depth so far, become the main focus and also a refinement of current theory concerning spatial constraints in general (both 2D and 3D). Therefore most of the designed constraints belong to this class.

Examining answers relating this question will result in a better understanding of other different types of constraints:

- Higher-level/Abstract Rules Some rules such as intersection of city objects (explained in section 6.1) can be generalised to a higher level and then applied to more object classes so that they will not be duplicated in every class. And a refinement of these general rules can be made for a specific class/attribute/relationship.

- Parameter-Based Rules The parameter-based constraints constraining values of objects and their properties can be placed in a category by themselves. The same parameterised rule for one object class would vary from instance to instance, depending on the type (also explained in section 6.1). And the parameters can be defined by users to achieve a greater efficiency.

- Exceptional Rules The rules that accept exceptional instances can be in one class (see concept in section 3.1.2 and their implementation is referred to in section 7.2.2). Having this class makes it possible to include the unusual yet existing city object instances in a database system.

- Extra Check Rules To find out the contradicting constraints, extra check rules need to be explicitly formulated. They are like rules of rules and may not directly apply to a specific object/attribute/relationship in the model. Therefore they should form a different class to the others.

- Multi-scale geometry consistency rules When a data area is modelled with different scales, e.g. CityGML LODs and climate modelling, constraints that check the consistency between two different scales should also be taken into consideration and would form a class in itself.

3. What is the best way to formalise these constraints?

Overall OCL is the best formalisation tool (see section 6.1). It is used in the ISO standard and INSPIRE data specification and supported by various and growing numbers of model translation tools. Most of the constraints are spatial constraints. For non-spatial constraints, regarding categories of thematic, quantity and temporal, current OCL standard statement is

sufficient. For spatial constraints, the geometric/topological abstraction of real world objects makes it more specific and thereby paves the way to a more formal specification in OCL.

There are other reasons to bring OCL forward. Firstly, it is in principle a language for formally specifying object constraints, although the current standard does not support spatial types and functions yet. Secondly, there are well-defined spatial types and predicates presented in ISO19107 standard, so with this standard the OCL library can be expanded. Thirdly, OCL can be rewritten in First Order Logic or any other High Order Logic languages and then checked by a theorem prover to see if there are any contradicting constraints.

4. Which tools/software are the most suitable for this work?

As far as drawing UML and writing OCL is concerned, EA is a sufficient tool. When it comes to model translation from OCL to SQL, EA appears to be insufficient because it does not convert OCL expressions to DDL code (see also section 6.3).

Dresden OCL2SQL tool kit, which is embedded in the Eclipse system and can translate OCL constraints to SQL select commands, seems to bridge the gap of code conversion. There are three major issues to cope with (see section 6.3). The first is that the UML diagram from EA is not recognised in Eclipse by default. A middleware to recognise and use EA UML models in Eclipse, called `MDG Integration for Eclipse`, was supposed to cope with this first issue. But somehow in practice it did not function as was expected. So either another conversion tool should be found or the UML class diagram (or any other core model like Ecore or XML) should be made using the Eclipse system. The second issue is that the geometry types and topological names are not currently supported in OCL. This can be dealt with by extending OCL based on the ISO19107 standard. The third issue is that very few of them are recognised by databases at the moment.

5. What is the best way to implement constraints in a database?

Because immediate model translation is not available right now, the implementation was coded by hand with triggers in Oracle PL/SQL, which can combine the power of spatial functions and database mechanisms that are able to detect run-time error (see chapter 7).

To improve the performance of constraint checks, especially spatial constraints, spatial query should be used in triggers. Row level triggers, due to the mutating-table restriction, which does not allow query of table when it is being modified, only passes the row value to global variables in procedures. Statement level triggers (After-Statement) then call the procedures that use the assigned variables to do necessary query (see explanation about triggers in section 2.3 and implementation in section 7.2).

For 3D spatial computation, since the demand for spatial function capability was discovered, the current 3D geometric functions, e.g. SDO_AnyInteract, SDO_Inside, in Oracle Spatial appear to be insufficient (see section 7.1). Therefore a new function 3D_SurfaceRelate that detects more detailed topological relationships is built on top of the existing 2D geometric functions, plus some simple calculations from computational geometry. With the relationships detected from this function and the meaning of surface objects, it is possible to fulfill the rule 'one object in one location'.

6. Which database, PostGIS or Oracle, performs better in realising 3D constraints?

This question was not studied during the research. Instead the mission is handed to the development of a new function in Oracle.

7. Is it possible to have some visual feedback to report errors?

Yes this is possible by means of two types of messages generated in Oracle. One is user-defined run-time error messages, and the other is DBMS_Output messages. Error messages can automatically reject a data change when an error is raised. DBMS_Output messages can contain more detailed information and are also used. When some data modification of other tables is needed, it does not enforce rejection, which becomes a good alternative message carrier (see section 7.2.2).

A difference in terms of visualisation is that error messages are visible to all platforms but DBMS_Output messages are not. Clients such as Bentley Map and Python IDLE, both of which can connect to database and perform modifications, are tested and the error messages are visible on their GUIs (see section 8). In contrast, DBMS_Output is visible only within the Oracle developing tools (SQL Plus and SQL Developer). Bentley Map and Python IDLE have problems of seeing this type of messages without extra configuration.

The new questions confronted along the way are:

1. How to make sure there are no contradicting constraints?

The human mind is able to perform its analysis but this of course varies from individual to individual. If the number and diversity of constraints go beyond the capability of human reasoning, then machines have to be introduced to perform the finer checks.

A solution to this problem would be to rewrite constraints in First Order Logic (FOL), that is, to express every term in geographic constraints into FOL predicates and FOL functions, and then run them in a theorem prover that computes the counter-rules. If counter-rules are found, then one must look into them and find a solution for them, by either modifying the rules so that they do not conflict or deleting the troublemakers. However, it is very likely that FOL does not understand spatial types, e.g. 'point' and 'polygon', and predicates, e.g. 'distance()' and 'overlaps()'.

2. How to check that multi-geometries (LODs) are consistent for single objects?

A data specification that unambiguously addresses the expected change of different LODs and relationships between them is required at the first stage, as is discussed in the general concept of one of the INSPIRE documents [INSPIRE, 8 26]. Then the differences that are unexpected can be treated as inconsistencies. After that corresponding constraints can be realised in database systems to check them.

An attempt was made by abstracting a rule from current data specifications about LODs, regarding size differences. In principle, MBR of the single object can handle comparisons of these factors. But in practice, a core function SDO_AGGR_MBR from Spatial database, which calculates the enclosing bounding box of a collection of objects, is found to return incorrect geometry information. Therefore, this rule cannot be realised in the end.

9.2 FUTURE WORK

Suggestions for future work for researchers wishing to extend the work carried out in this thesis are:

1. 3D_SurfaceRelate can be extended to take over SDO_AnyInteract when dealing with binary surfaces relations. Now the first check is SDO_AnyInteract, when two polygons are found to be 'non-disjoint', the new function will take a secondary computation. But it still requires some improvement in detecting 'disjoint' cases. Hopefully in the future, given two random (Oracle valid) 3D polygons, this new function will be capable of telling everything.

2. Further study can be conducted into detecting contradicting constraints. Here the constraints types can be both spatial and non-spatial. So far there seems to be two ways, one is using OCL verification tools, and the other is using FOL and theorem provers. If someone would consider studying OCL verification tools, those that appear in OCL workshop proceedings [OCL, 2010b], e.g. HOL-OCL, UMLtoCSP are worth trying out. The second approach requires a great deal of input to enable FOL to understand geographic predicates and types.

3. It would be interesting to develop a city model constraints library, and then integrate it into CityGML data specifications. One aspect discovered initially in design constraints was that there are no data specifications stating what is normal/usual and what is abnormal/undesirable for a general 3D city database. So constraints have to be constructed by the researcher himself. Those that have already been proposed in table I - 'Normal situations' from appendix A.1, can be further extended, formulated and then added to the 3DCityDB schema.

4. The pseudo OCL expressions from section 6.1 need to be tested in conjunction with the UML diagrams so that translation to SQL code can be easier to achieve. Those expressions are the first attempt to formalise various 3D spatial constraints in OCL. They include new predicates and spatial types that are not currently supported by standard OCL.

5. It would be useful to extend EA, or Dresden OCL2SQL tool or any other OCL code generation tools to enable automatic model translation from OCL (esp. spatial constraints) to SQL. This requires mature Java programming since most of the MDA tools are developed in a Java environment. However, this can also be provided by software developers.

6. When pseudo OCL is tested and are translated to SQL, in order to implement those predicates, corresponding functions in database need to be developed. The existing geometric functions, either 2D or 3D, can be extended.

9.3 RECOMMENDATIONS

During this research, tools/software and standards have been studied and used. The experience gained in using them has led to the conclusion that some improvements could be made by manufacturers/developers and organisations.

Standards

- OCL: It would be highly productive to extend current OCL standards with 3D spatial types/operations and topological relationships. Some extensions are found in articles but are not yet in the standards and most of them are only in 2D. The ISO19107 standard or Oracle specifications are good references for the extension of OCL standards. It is important to bear in mind that Oracle and ISO standards are different when it comes to defining geometry, e.g. volume.
- UML: It would be an excellent idea (see section 6.1) to create a symbol for a constraint link which applies to the binary relation in the UML diagram.
- FOL: First order logic standard predicates and functions could be extended and adapted with geographic terms so that automatic or semi-automatic counter-rule reasoning will be more available. Considering that geometry is actually a series of ordinates (numeric values), and the spatial types such as point, line and polygon and topological relationships are comparable to predicates/propositions in FOL, this adaption is achievable.
- CityGML: It would be advisable to have geo-constraints incorporated into the UML model so that city modelling could be more concise and well-defined.

The CityGML data specifications could be more developed to enable the multi-geometries (LODs) consistency check.

Software

- Oracle
  1. A positive input would be to enable communication between DBMS_Output message and clients so that more abundant information about constraint check results, could be visible on different platforms. (Of course more input is also to be expected on the part of front-end clients.)

  2. More spatial functions relating to 3D and solids are desired from Oracle Spatial in order to meet the rising demands of dealing with 3D data.

  3. Likewise, an important aspect to address is the improvement of functions that are currently not working properly.

     a SDO_AnyInteract: problems with surface going through the inner ring (see problematic tests in figure 38 in section 7.1.

     b SDO_Inside: problems with points on the boundary (vertex, edge, face) of solids (also see section 7.1).

     c SDO_AGGR_MBR: problems with mismatching between SDO_Gtype and Elem_Info_Array (see section 7.2.6).

- Enterprise Architect
  1. A function that would make EA more usable would be to enable the OCL code generation in the DDL code generation. In this sense, constraints expressed in OCL can easily be converted to be a part of the database design.

  2. The current OCL validation is rather undeveloped in EA. As was explained in section 6.1, some OCL statements that are obviously incorrect in syntax can still be validated successfully.

- Bentley Map
  The DBMS_Output messages cannot be immediately seen in Bentley Map at the moment. As this type of message provides more details of what an error in the input data can be, and Bentley Map is commonly used as an editor for geometric objects in Spatial database, it would be valuable to visualise DBMS_Output messages on the GUI of Bentley Map.

BIBLIOGRAPHY

[ben,] *Bentley Product Data Sheet*. Last access to http://www.bentley.com/en-us/products/bentley+datasheets/ on Aug20, 2011.

[Cit, 2008] (2008). *OpenGIS City Geography Markup Language (CityGML) Implementation Specification*. OGC, 1.0.0 edition. 232 pages.

[pro, 2009] (2009). *Prover9 Manual*. Last access http://www.cs.unm.edu/ mccune/mace4/manual/2009-11A/ on Aug 23, 2011.

[OCL, 2010a] (2010a). *Object Constraint Language*. OMGő, 2.2 edition. 238 pages.

[OCL, 2010b] (2010b). *OCL for formal modelling of topological constraints (OCL 2010)*.

[Ora, 2010a] (2010a). *Oracleő Database, PL/SQL Language Reference, 11g Release 2 (11.2)*. Oracleő, e17126-04 edition. 756 pages.

[Ora, 2010b] (2010b). *Oracleő Spatial Developer's Guide 11g Release 2 (11.2)*. Oracleő. 916 pages.

[hib, 2011a] (2011a). *Hibernate Getting Started Guide*. Hibernate. Last access to http://docs.jboss.org/hibernate/core/3.6/quickstart/en-US/html$_s ingle/onAug$23, 2011.

[hib, 2011b] (2011b). *Hibernate Spatial*, 1.1 edition. Last access to http://www.hibernatespatial.org/ on April 15, 2011.

[Alberto Belussi, 2004] Alberto Belussi, Mauro Negri, G. P. (2004). Geouml: A geographic conceptual model defined through specialisation of iso tc211 standards. In *10th EC GI & GIS Workshop, ESDI State of the Art*.

[Arens et al., 2005] Arens, C., Stoter, J., and van Oosterom, P. (2005). Modelling 3d spatial objects in a geo-dbms using a 3d primitive. *Computers & Geosciences*, 31(2):165–177.

[Borrebæk and Myrind, 2005] Borrebæk, M. and Myrind, G. (2005). Evaluation of the small subset of euroroads according to iso 19109 rules for application schema. In *ESDI Workshop on Conceptual Schema Languages and Tools*.

[Breunig et al.,] Breunig, M., Thomsen, A., Broscheit, B., Butwilowski, E., and Sander, U. Representation and analysis of topology in multi-representation databases. In *Proceedings of PIA*, volume 7, page 6.

[Casanova et al., 2000] Casanova, M., Wallet, T., and DŠHondt, M. (2000). Ensuring quality of geographic data with uml and ocl. In Evans, A., Kent, S., and (Eds.), B. S., editors, *UML'oo Proceedings of the 3rd international conference on The unified modeling language: advancing the standard*, pages 225–239, Heidelberg, Berlin. Springer.

[CityGML, 2010] CityGML (Last Modified on 31 May 2010). Citygml application domain extensions. In *CityGML-ADEs*. Retrieved from World Wide Web http://www.citygmlwiki.org/index.php/CityGML-ADEs in May 11.

[Cockcroft, 1997] Cockcroft, S. (1997). A taxonomy of spatial data integrity constraints. *GeoInformatica*, 1:4:327–343. Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

[Cockcroft, 2004] Cockcroft, S. (2004). The design and implementation of a repository for the management of spatial data integrity constraints. *GeoInformatica*, 8(1):49–69.

[Database, 2010] Database, O. (August 2010). Oracle database pl/sql packages and types reference 11g release 2 (11.2).

[Duboisset et al., 2005] Duboisset, M., Pinet, F., Kang, M.-A., and Schneider, M. (2005). Precise modeling and verification of topological integrity constraints in spatial databases: From an expressive power study to code generation principles. In et al. (Eds.), L. D., editor, *Conceptual Modeling Ű ER 2005*, pages 465–482, Klagenfurt, Austria. Springer.

[Egenhofer, 1989] Egenhofer, M. J. (1989). A formal definition of binary topological relationships. *LNCS*, 367:457 − 472.

[Egenhofer, 1992] Egenhofer, M. J. (1992). Categorizing binary topological relations between regions, lines and points in geographic databases. Technical report, University of Canifornia.

[Egenhofer, 1995] Egenhofer, M. J. (1995). Topological relations in 3-d. Technical report, National Center for Geographic Information and Analysis and Deparment of Spatial Information Science and Engineering Department of Computer Science university of Maine.

[Eliseo Clementini and van Oostero, 1993] Eliseo Clementini, P. D. F. and van Oostero, P. (1993). A small set of formal topological relationships suitable for end-user interaction. In *The 3rd International Symposium on Large Spatial Databases*.

[Elmasri and Navathe, 2003] Elmasri and Navathe (2003). *Fundamental of Database Systems*. Addison Wesley, 4 edition.

[Geomatics, 2010] Geomatics, M. (2010). Measure the climate, model the city - acquiring and storing temporal and spatial data for climate research. Technical report. GM2100 Synthesis Project, 112 pages.

[Ghawana et al., 2010] Ghawana, T., Consultants, I., and Zlatanova, S. (2010). Data consistency checks for building of 3d model: A case study of technical university, delft campus, the netherlands.

[Godse, 2009] Godse, A. (2009). *Computer Graphics*. Technical Publications.

[Gorte and Pfeifer, 2004] Gorte, B. and Pfeifer, N. (2004). 3d image processing to reconstruct trees from laser scans. In *Proceedings of the 10th annual conference of the Advanced School for Computing and Imaging (ASCI)*.

[GROGER and PLUMER, 2005] GROGER, G. and PLUMER, L. (2005). How to get 3d for the price of 2d - topology and consistency of 3d urban gis. *GeoInformatica*, 9:2:139–158.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. Technical report, KNOWLEDGE SYSTEMS LABORATORY, Computer Science Department, Stanford University.

[Hespanha et al., 2008] Hespanha, J., van Bennekom-Minnema, J., Van Oosterom, P., and Lemmen, C. (2008). The model driven architecture approach applied to the land administration domain model version 1.1-with focus on constraints specified in the object constraint language. In *fig working week*, page 19.

[INSPIRE, 8 26] INSPIRE (2009-08-26). D2.5: Generic conceptual model, version 3.2. EU.

[INSPIRE, 4 26a] INSPIRE (2010-04-26a). D2.8.i.5 inspire data specification on addresses Ű guidelines.

[INSPIRE, 4 26b] INSPIRE (2010-04-26b). D2.8.i.6 inspire data specification on cadastral parcels Ű guidelines.

[ISO, 2005] ISO (2005). Iso 19109:2005. In *Geographic information – Rules for application schema*.

[ISO, 2010] ISO (December 17 2010). Iso 19136:2007. In *Geographic information – Geography Markup Language (GML)*. ISO.

[ISO, 2008] ISO (September 17, 2008). Iso 19107:2003. In *Geographic information – Spatial schema*. ISO.

[Jeremy Avigad, 2007] Jeremy Avigad, Kevin Donnelly, D. G. P. R. (2007). A formally verified proof of the prime number theorem. *ACM Transactions on Computational Logic*, 9.

[Jun and Jie, 1998] Jun, C. and Jie, J. (1998). An event-based approach to spatio-temporal data modelling in land subdivision systems. *GeoInformatica*, 2:387Ű402.

[Kazar et al., 2008] Kazar, B. M., Kothuri, R., van Oosterom, P., and Ravada, S. (2008). *On valid and Invalid Three-Dimensional Geometries*, chapter 2, pages 19–46. Springer. Advances in 3D Geoinformation Systems, Lectures Notes in Geoinformation and Cartography.

[Kleppe et al., 2003] Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional.

[Kolar, 2004] Kolar, J. (2004). Representation of geographic terrain surface using global indexing. In *Geoinformatics*.

[Kolbe et al., 2009] Kolbe, P. D. T. H., König, G., Nagel, C., and Stadler, A. (2009). *3D-Geo-Database for CityGML*. Institute for Geodesy and Geoinformation Science, Technische Universität Berlin, 2.0.1 edition.

[Kwon et al., 1999] Kwon, Y.-M., Ferrari, E., and Bertino, E. (1999). Modeling spatio-temporal constraints for multimedia objects. *Data and Knowledge Engineering*, 30:217–238.

[Louwsma, 2004] Louwsma, J. (2004). Constraints in geo-information models - applied to geo-vr in landscape architecture. Master's thesis, TU Delft. GIRS-2004-17. 104 pages.

[Louwsma et al., 2006] Louwsma, J., Zlatanova, S., van Lammeren, R., and van oosterom, P. (2006). Specifying and implementing constraints in gisŮwith examples from a geo-virtual reality system. *GeoInformatica*, 10:531–550.

[Malik, 2000] Malik, S. (2000). *Dynamic Level of Detail Representation of Interactive 3D Worlds*. PhD thesis, Carleton University.

[Mas et al., 2005] Mas, S., Wang, F., and Reinhardt, W. (2005). Using ontologies for integrity constraint definition. In *Proceedings of the 4th International Symposium on Spatial Data QualityŠ05*, Beijing, China.

[Palagyi et al., 2003] Palagyi, K., Tschirren, J., and Sonka, M. (2003). Quantitative analysis of intrathoracic airway trees: Methods and validation. *LNCS*, 2732:222–233.

[Pinet et al., 2007] Pinet, F., Duboisset, M., and Soulignac, V. (2007). Using uml and ocl to maintain the consistency of spatial data in environmental information systems. *Environmental Modelling & Software*, 22.

[Pinet et al., 2004] Pinet, F., Kang, M.-A., and Vigier, F. (2004). Spatial constraint modelling with a gis extension of uml and ocl. In Wiil, U. K., editor, *Metainformatics*, pages 160–178, Salzburg, Austria. Springer.

[Raafat et al., 1994] Raafat, H., Yanga, Z., and Gauthler, D. (1994). Relational spatial topologies for historical geographical information. *International Journal of Geographical Information Systems*, 8:167–173.

[Ravada, 2008] Ravada, S. (2008). Oracle spatial 11g technical overview. Presentation. Accessed in Dec, 2010.

[Reeves et al., 2006] Reeves, T., Cornford, D., Konecny, M., and Ellis, J. (2006). Modeling geometric rules in object based models: An xml/gml approach. In *Progress in Spatial Data Handling*, 3, pages 133–148.

[Sheeren, 2005] Sheeren, D. (2005). Méthodologie dŠévaluation de la cohérence inter-représentations pour lŠintégration de bases de données spatiales. *These de doctorat informatique de lŠUniversité Pierre et Marie Curie (Paris VI), LIP6*.

[Sheeren et al., 2005] Sheeren, D., Mustière, S., and Zucker, J. (2005). Consistency assessment between multiple representations of geographical databases: a specification-based approach. *Developments in Spatial Data Handling*, pages 617–628.

[van Berlo and de Laat, 2010] van Berlo, L. and de Laat, R. (2010). Integration of bim and gis: The development of the citygml geobim extension. In *5th International 3D GeoInfo Conference*, Berlin, Germany. Springer.

[van Oosterom, 2006] van Oosterom, P. (2006). *Dynamic and Mobile GIS: Investigating Changes in Space and Time*, chapter 7. Constraints in Spatial Data Models, in a Dynamic Context., pages 104 - - 137. CRC Press.

[van Oosterom et al., 2002] van Oosterom, P., Masessen, B., and Quak, W. (2002). Generic query tool for spatio-temporal data. *International Journal of Geographical Information Science*, 16:713–748.

[W. Tegtmeier, 2003] W. Tegtmeier, S. Zlatanova, P. v. O. H. R. G. K. H. (2003). Information management in civil engineering infrastructural development: with focus on geological and geotechnical information. In *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Science*.

[Wachowicz et al., 2008] Wachowicz, S., Wang, F., and Reinhardt, W. (2008). *The European Information Society: Leading the Way with Geo-Information. LNGC*, chapter 2. Extending geographic data modeling by adopting constraint decision table to specify spatial integrity constraints, pages 435–454. Springer, Heidelberg, Berlin.

[Warmer and Kleppe, 2003] Warmer, J. and Kleppe, A. (2003). *The Object Constraint Language Second Edition: Getting Your Models Ready For MDA*. Addison Wesley Professional.

[Werder, 2009] Werder, S. (2009). Formalization of spatial constraints. In *12th AGILE International Conference on Geographic Information Science*, Leibniz Universität Hannover, Germany.

[Wilke et al.,] Wilke, C., Thiele, M., and Freitag, B. *Dresden OCL manual for Installation, Use and Development*. Faculty of Computer Science, Technische Universität Dresden. Last update on January 17, 2011.

[Zlatanova, 2000] Zlatanova, S. (2000). *3D GIS for Urban Development*. PhD thesis, The International Institute for Aerospace Survey and Earth Sciences (ITC) and The Institute for Computer Graphics and Vision (ICGV), Graz, Austria. 222 pages.

A

CONSTRAINT LIST

A.1 NORMAL SITUATIONS

Table I - Normal Situations

Object(s)	Proposed spatial characteristics
Building	structures of one building and are connected and all surfaces are closed;
Building - building	structures from two different buidings are separate or touched (adjacent building);
Building - tree	building and tree are separate or touched (tree leaves touch the wall);
Building - road	a road doesn't cross building (can go below or above building); building address is the same as the name of a nearby road;
Building - grass	building doesn't stand on the grass field; building roof/wall is covered with grass (roof greening, air garden);
Building - water	building doesn't float on water (except boat house); building stands on special structure like poles upon water surface;
Building - terrain	building foundation connects to the terrain or go underground;
Building - mobile sensor	mobile sensor doesn't suddenly jump into or out of building through wall;
Building - stationary sensor	The solar (panel) sensor should be installed outside the building;
Tree	tree with a certain species has limited size;
Tree - tree	trees are planted not too close;
Tree - road	tree doesn't stand on the road;
Tree - grass	-
Tree - water	certain species grow in water, others on the land;
Tree - terrain	tree trunk connects to the land or go underground;
Tree - mobile sensor	-
Tree - stationary sensor	-
Road	-
Road - road	-
Road - grass	road and grass are separate (grass doesn't stand on the road);
Road - water	road doesn't float on the water surface;
Road - terrain	road doesn't disconnect from the ground;
Road - mobile sensor	-
Road - stationary sensor	-
Grass	-
Grass - grass	different grassfields are separate and don't overlap;
Grass - water	grass and water are separate/adjacent and don't overlap;

Table I - Normal Situations (continued)	
Object(s)	Proposed spatial characteristics
Grass - terrain	grass lays on the ground or even on building roof;
Grass - mobile sensor	-
Grass - stationary sensor	-
Water	-
Water - water	-
Water - terrain	water surface lay on the terrain;
Water - mobile sensor	-
Water - stationary sensor	-
Land use	every piece of land has a landuse type;

A.2 CONSTRAINED SITUATIONS IN NATURAL LANGUAGE

Table II - Constrained Situations in Natural Language		
Spatial Constraints		
Object(s)	Do not accept	Unusual but accepted
Building	b11 - the building object is not closed; b12 - no hole on the wall to allow passing when there is a passage;	-
Building - building	bb1 - two buildings intersect or one building is inside the other;	bb2 - two buildings share a point;
Building - tree	bt1 - building and tree intersect or tree inside building;	bt2 - tree trunk too close to building;
Building - road	br11 - road intersects with building;	-
	br12 - address of building doesn't match name of the road nearby;	-
Building - grass	bg1 - building stands on the grass land or grass intersects building;	bg2 - grass attaches to building surface (e.g. roof/wall greening);
Building - water	bw1 - building footprint intersects with water;	bw2 - building floats upon water (except the boat houses);
Tree	t1 - tree exceeds its limited size;	-
Tree - tree	tt1 - trees are too close;	-
Tree - road	tr11 - tree stand on the road; tr12 - tree is too close to the road;	-
Tree - water	tw1 - tree of aquatic species is too far from water;	-
Mobile sensor - building	mb1 - mobile sensor jump into/out of building without going through opening first;	-
Stationary sensor - building	sb1 - solar sensor (panel) that should be installed outside the building becomes inside;	-
Road	-	-
Road - road	-	rr2 - road overlaps with road;
Road - grass	rg1 - road crosses grass;	-
Road - water	rw1 - road crosses water without any suporting structure (like bridge);	-
Grass	-	-
Grass - grass	-	gg2 - different grass fields overlap;
Grass - water	gw1 - grass placed on the water;	-
Grass - terrain	-	gt2 - grass doesn't connect to the terrain (could be attached to be building wall or roof);

Table II - Constrained Situations in Natural Language (continued)		
Object(s)	Do not accept	Unusual but accepted
Water	-	-
Water - water	-	-
Non-Spatial Constraints		
Building	b1 - A building should have its construction date earlier than its destruction date;	-
Tree	-	-
Road	r1 - A road should have its cnostruction date earlier than its destruction date;	-
Grass	-	-
Water	-	-
Sensor	s1 - Change of measurement value must not be too fast;	-
Landuse	l1 - A piece of land have different landuse types at the same time;	l2 - In the midst of a known area, a piece of land doesn't have any landuse type;
Terrain	n1 - An object doesn't have terrain intersection curve;	-
Special Issue - LOD	d1 - two LOD geometric representations must be consistent;	

A.3 SITUATIONS THAT CANNOT BE ACCEPTED

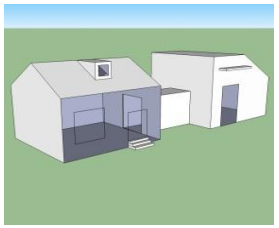
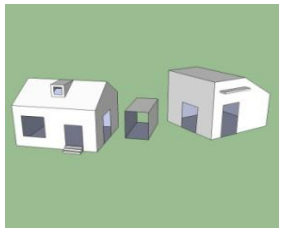
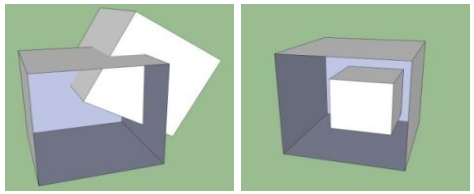
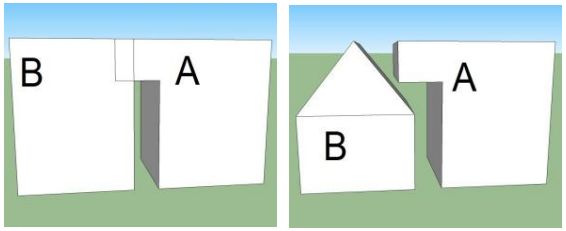
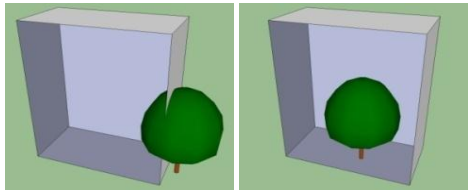
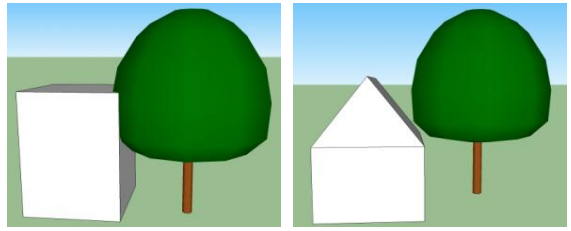
Table III - Situations that cannot be accepted			
Object(s)	Natural Language	Geometric and topological Abstraction	Sensitive LOD
Building	b11 - the building object is not closed; b12 - no hole on the wall to allow passing when there is a passage; 	b11 - solid (3D): solid A is not closed (check with Oracle validate_geometry function); b11 - surfaces (3D): surfaces don't enclose a space; b12 - surfaces (3D): the wall surface doesn't have holes;	b11 - LOD1; b12 - LOD4; 
Building - building	bb1 - two buildings intersect or one building is inside the other; 	solid - solid (3D): solid A contains or covers or inside or intersects solid B (defined by 3D 9 intersection model); surface - solid (3D): surface A intersects solid B; surface - surface (3D): surface A intersects surface B;	first check LOD1, if so then further check LOD2; 
Building - tree	bt1 - building and tree intersect or tree inside building; 	solid - solid (3D): solid A contains or covers or inside or intersects solid B (defined by 3D 9 intersection model); surface - solid (3D): surface A intersects solid B;	same as building - building; 

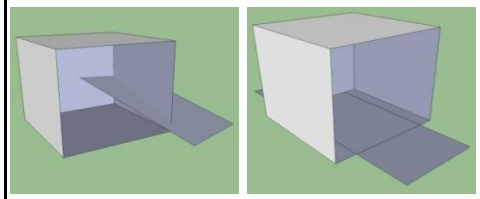
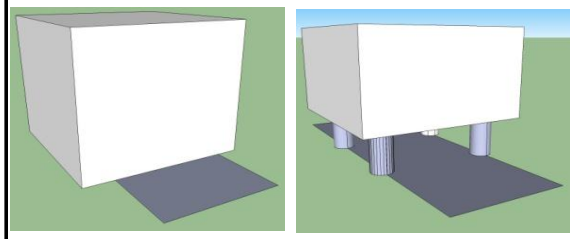
Table III - Situations that cannot be accepted (continued)			
Object(s)	Natural Language	Geometric and topological Abstraction	Sensitive LOD
Building - road	br11 - road intersects with building; 	surface - solid (3D): surface A strong-meets or intersects solid B; surface - surface (3D): two surfaces intersect;	first check LOD1, if so then further check LOD2; 
	br12 - address of building doesn't match name of the road nearby;	semantic-topo relation (distance calculation + string matching); (2D)	-
Building - grass	bg1 - building stands on the grass land or grass intersects building;	same as building - road (br11) relation;	first check LOD1, if so then further check LOD2;
Building - water	bw1 - building footprint intersects with water;	same as building - road (br11) relation;	first check LOD1, if so then further check LOD2;
Building - mobile sensor	mb1 - mobile sensor jump into/out of building without going through opening first;	algorithm not clear yet;	LOD1
Building - stationary sensor	sb1 - solar sensor (panel) that should be installed outside the building becomes inside;	surface - solid(3D): surface A inside solid B; surface - surface(3D): Giving that a surface B is from outside wall or roof of building, surface A (solar panel) meets or strong-meets surface B;	LOD3 or higher;
Tree	t1 - tree exceeds its limited size;	attributes (height, crown diamter or trunk diameter) exceeds a certain value;	
Tree - tree	tt1 - trees are too close;	point - point (2D): distance between point A and point B is smaller than value n;	

Table III - Situations that cannot be accepted (continued)			
Object(s)	Natural Language	Geometric and topological Abstraction	Sensitive LOD
Tree - road	tr11 - tree stand on the road; tr12 - tree is too close to the road;	point - surface (3D): point A is inside surface B; distance from point A to surface B smaller than value n;	
Tree - water	tw1 - tree is in the water, except the aquatic species;	solid - surface (3D): solid A strong-meets or intersects surface B; point - surface (3D): point A inside surface B;	
Tree - stationary sensor	-	-	
Road	-	-	
Road - road	-	-	
Road - grass	rg1 - road crosses grass;	surface - surface (3D): surface A intersects or strong-meets surface B;	
Road - water	rw1 - road crosses water without any suporting structure (like bridge);	surface - surface (3D): surface A intersects or strong-meets surface B;	
Road - mobile sensor	-	-	
Road - stationary sensor	-	-	
Grass	-	-	
Grass - grass	-	-	
Grass - water	gw1 - grass placed on the water;	surface - surface (2D): surface A overlaps or contains/in surface B;	
Grass - mobile sensor	-	-	
Grass - stationary sensor	-	-	

Table III - Situations that cannot be accepted (continued)			
Object(s)	Natural Language	Geometric and topological Abstraction	Sensitive LOD
Water	-	-	
Water - water	-	-	
Water - mobile sensor	-	-	
Water - stationary sensor	-	-	
Landuse	l1 - A piece of land have different landuse types at the same time;	time factor is involved, difficult to describe merely by geometry;	
Terrain	n1 - An object doesn't have a corresponding terrain intersection curve;	-	

A.4 SITUATIONS THAT ARE UNUSUAL BUT ACCEPTABLE

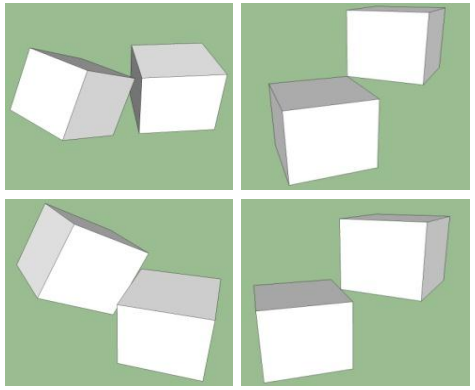
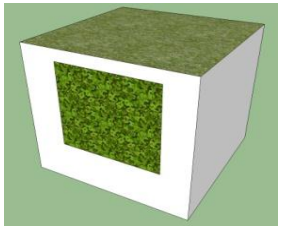
Table IV - Situations that are unusual but acceptable			
Object(s)	Natural Language	Geometric and topological Abstraction	Sensitive LOD
Building	-	-	-
Building - building	bb2 - two buildings share a point; 	solid - solid: (3D) solid A meets solid B; surface - solid (3D): surface A meets solid B on a point; surface - surface (3D): surface A meets surface B on a point;	LOD1
Building - tree	bt2 - tree trunk too close to building;	(root) point - solid: (3D) distance between point A and solid B is less than n; point - surface: (3D) distance from point A to surface B is less than n;	LOD1
Building - road	-	-	-
Building - grass	bg2 - grass attaches to building surface (e.g. roof/wall greening); 	surface - solid: (3D) surface A and solid B strong-meet; surface - surface: (3D) Giving that surface A is a side wall/roof of building, surface A contains or is equal to surface B;	LOD1

Table IV - Situations that are unusual but acceptable (continued)			
Object(s)	Natural Language	Geometric and topological Abstraction	Sensitive LOD
Building - water	bw2 - building floats upon water (except the boat houses);	surface - solid: (3D) surface A strong-meets solid B; surface - surface: (3D) Giving that surface B is building's bottom face, water surface A contains surface B;	LOD1
Mobile sensor - building	-	-	-
Stationary sensor - building	-	-	-
Tree	-	-	
Tree - tree	-	-	
Tree - road	-	-	
Tree - water	-	-	
Tree - mobile sensor	-	-	
Tree - stationary sensor	-	-	
Road	-	-	
Road - road	rr2 - road overlaps with road;	surface - surface (3D): surface A intersects surface B;	
Road - grass	-	-	
Road - water	-	-	
Road - mobile sensor	-	-	
Road - stationary sensor	-	-	

Table IV - Situations that are unusual but acceptable (continued)			
Object(s)	Natural Language	Geometric and topological Abstraction	Sensitive LOD
Grass	-	-	
Grass - grass	gg2 - different grass fields overlap;	surface - surface (2D): surface A overlaps surface B;	
Grass - water	-	-	
Grass - terrain	gt2 - grass doesn't have terrain intersection curve (can be attached to be building wall or roof);	surface - surface (3D): surface A doesn't match surface B;	
Grass - mobile sensor	-	-	
Grass - stationary sensor	-	-	
Water	-	-	
Water - water	-	-	
Water - terrain	-	-	
Water - mobile sensor	-	-	
Water - stationary sensor	-	-	
Landuse	l2 - In the midst of a known area, a piece of land doesn't have any landuse type;	surface (2D): algorithm not clear yet;	

B

ORACLE

B.1 GEOMETRY FUNCTIONS

Table 6 is a complete list of all functions/operators/subprogrammes (2D and 3D) that are studied within this research. For more detailed information, please check Oracle Spatial Developer's Guide 11g Release 2 [Ora, 2010b].

Table 6.: Geometry functions in Oracle Spatial that are studied within this research

Function Name	**Short Description**
SDO_ANYINTERACT (2D and 3D)	Checks if any geometries in a table have the ANYINTERACT topological relationship with a specified geometry.
SDO_FILTER (2D and 3D)	Uses the spatial index to identify either the set of spatial objects that are likely to interact spatially with a given object.
SDO_INSIDE (3D only)	Checks if any geometries in a table have the INSIDE topological relationship with a specified geometry.
SDO_NN (2D and 3D)	Uses the spatial index to identify the nearest neighbors for a geometry.
SDO_WITHIN_DISTANCE (2D and 3D)	Uses the spatial index to identify the set of spatial objects that are within some specified distance of a given object.
SDO_AGGR_MBR (2D and 3D)	Returns the minimum bounding rectangle (MBR) of the specified geometries, that is, a single rectangle that minimally encloses the geometries.
SDO.RELATE (2D and 3D)	Uses the spatial index to identify either the spatial objects that have a particular spatial interaction with a given object. It support 3D only with 'ANYINTERACT' and 'INSIDE' masks.
SDO_GEOM.RELATE (2D and 3D)	Examines two geometry objects to determine their spatial relationship. Similar to SDO_Relate but supports 3D only with 'ANYINTERACT' mask.
SDO_GEOM.SDO_AREA (2D and 3D)	Returns the area of a 2D or 3D polygon.

Table 6.: (Previous table continued)

SDO_GEOM.SDO_DISTANCE (2D and 3D)	Computes the distance between two geometry objects.
SDO_GEOM.SDO_LENGTH (2D and 3D)	Returns the length or perimeter of a geometry object.
SDO_GEOM.SDO_MAX_MBR_ORDINATE (2D and 3D)	Returns the maximum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_MBR (2D and 3D)	Returns the minimum bounding rectangle of a geometry object, that is, a single rectangle that minimally encloses the geometry.
SDO_GEOM.SDO_MIN_MBR_ORDINATE (2D and 3D)	Returns the minimum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_VOLUME (3D only)	Returns the volume of a three-dimensional solid.
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT (2D and 3D)	Performs a consistency check for valid geometry types and returns context information if the geometry is invalid.
SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT (2D and 3D)	Examines a geometry column to determine if the stored geometries follow the defined rules for geometry objects, and returns context information about any invalid geometries.
SDO_GEOM.WITHIN_DISTANCE (2D and 3D)	Determines if two spatial objects are within some specified distance from each other.
SDO_GEOM.SDO_INTERSECTION (2D only)	Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.

B.2 ORACLE TOPOLOGICAL NAMES

Some of the topological terms originated from Max Egenhofer 9I model are realised in Oracle Spatial 11g2. They are possible results from operator SDO_Relate. Spatial uses the following names (see also figure 43):

- DISJOINT: The boundaries and interiors do not intersect.
- TOUCH: The boundaries intersect but the interiors do not intersect.
- OVERLAPBDYDISJOINT: The interior of one object intersects the boundary and interior of the other object, but the two boundaries do not intersect. This relationship occurs, for example, when a line originates outside a polygon and ends inside that polygon.
- OVERLAPBDYINTERSECT: The boundaries and interiors of the two objects intersect.
- EQUAL: The two objects have the same boundary and interior.
- CONTAINS: The interior and boundary of one object is completely contained in the interior of the other object.
- COVERS: The interior of one object is completely contained in the interior or the boundary of the other object and their boundaries intersect.
- INSIDE: The opposite of CONTAINS. A INSIDE B implies B CONTAINS A.
- COVEREDBY: The opposite of COVERS. A COVEREDBY B implies B COVERS A.
- ON: The interior and boundary of one object is on the boundary of the other object (and the second object covers the first object). This relationship occurs, for example, when a line is on the boundary of a polygon.
- ANYINTERACT: The objects are non-disjoint.

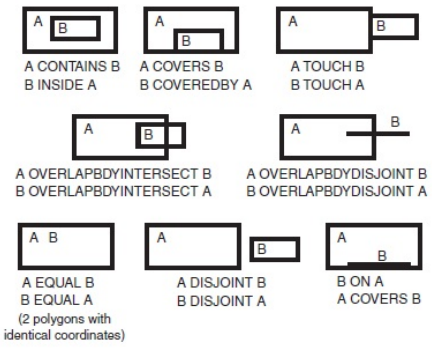
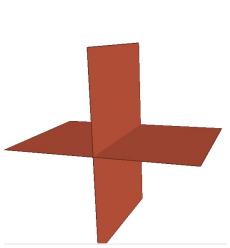
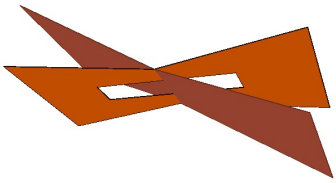


Figure 43.: Topological relations that Spatial realises.

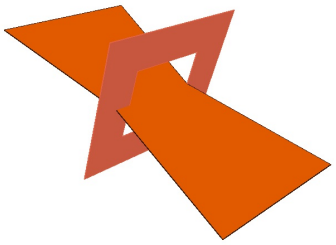
B.3 TEST POLYGONS FOR 3D_SURFACERELATE



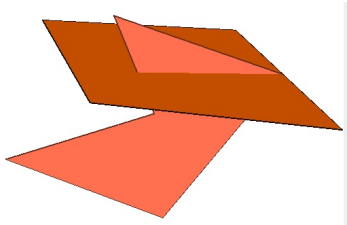
(a) Two perpendicular polygons intersect.



(b) Intersection of a polygon with hole and a polygon without hole.

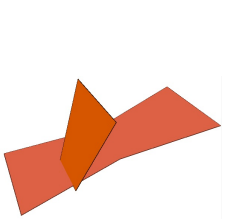


(c) A polygon goes through a hole of the other and intersects from the inner ring.

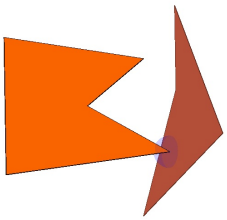


(d) A concave polygon intersects a convex polygon.

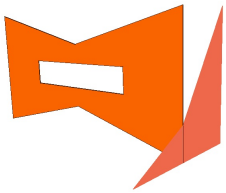
Figure 44.: Tested polygons that all return 'INTERSECT'.



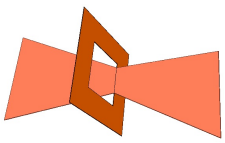
(a) A convex polygon strong-touches a concave surface.



(b) One concave surface strong-touches the other by a point.

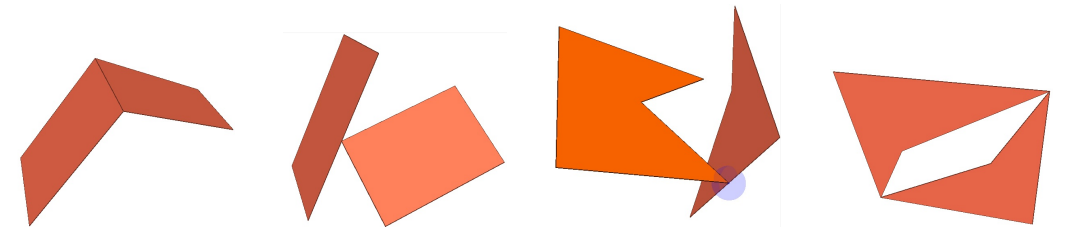


(c) A polygon with hole strong-touches a concave polygon with its outer boundary.



(d) A ring strong-touches a concave polygon with its inner boundary.

Figure 45.: Tested polygons that all return 'STRONG-TOUCH'.

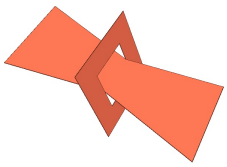


(a) Two regular polygons share an edge.

(b) A convex polygon touches the other at one point (with its vertex).

(c) A concave polygon touches the other polygon at one point.

(d) Two concave polygons touch at two points.



(e) A polygon goes through a ring and touches the inner boundary at two points.

Figure 46.: Tested polygons that all return 'TOUCH'.

C

CODE

Constraints in this research are all coded as procedures within database packages and called by triggers. Functions (e.g. to calculate surface relation) are not included in a particular package so can be called within the whole database. Codes of functions and procedures are given below.

C.1 FUNCTIONS

3D_SurfaceRelate is to calculate relation between 2 3D polygon objects (see listing C.1). It includes several modules: '3D_PlaneParameters' (listing C.2) calculates parameters of a 3D plane, '3D_Rotate' (listing C.3) rotate a 3D object about a user-defined axis, '3D_MBR2NormalGeom' (listing C.4) converts a rectangle represented by 2 points into a usual representation with all points. This function returns string 'Disjoint' or 'Touch' or 'Overlap_Parallel' or 'Overlap_Opposite' or 'Intersect'

Listing C.1: Self-developed function 3D_SurfaceRelate

```
CREATE or REPLACE FUNCTION 3D_SurfaceRelate(

-- Geometries of the surfaces
geom1 SDO_Geometry,
geom2 SDO_Geometry)
RETURN VARCHAR2

AS
  surf_a SDO_Geometry; -- Geometry of surface A.
  surf_b SDO_Geometry; -- Geometry of surface B.

  -- Geometry of projection of the intersection line.
  proj_line SDO_Geometry;

  -- Geometry of the interacting part of two surfaces.
  common_surf_geom SDO_Geometry;

  -- Geometry of the common touching part;
  common_touch_geom SDO_Geometry;

  cos_theta NUMBER(20, 10);
  relation VARCHAR2(50); -- The relation between two input surfaces

  -- Geometry of the minimum bounding rectangle of two surfaces.
  union_mbr SDO_Geometry;
```

```
27    -- Coordinates of the lower left corner from union mbr.
      xll_union NUMBER(20, 10);  yll_union NUMBER(20, 10);

      -- Coordinates of the upper right corner from union mbr.
      xur_union NUMBER(20, 10);  yur_union NUMBER(20, 10);
32
      -- Geometry of the new minimum bounding rectangle
      -- (double size) that CONTAINS both objects.
      doublesize_mbr SDO_Geometry;

37    -- Coordinates of the lower and upper corner of the new MBR.
      xll_big NUMBER(20, 10);  yll_big NUMBER(20, 10);
      xur_big NUMBER(20, 10);  yur_big NUMBER(20, 10);

      -- ordinates of points that are on the projection
42    -- of intersection line
      pt1_x NUMBER(20, 10);  pt1_y NUMBER(20, 10);
      pt2_x NUMBER(20, 10);  pt2_y NUMBER(20, 10);

      -- For one surface
47    -- Ordinates of the 1st point.
      xa1 NUMBER(20, 10);  ya1 NUMBER(20, 10);  za1 NUMBER(20, 10);

      -- Ordinates of the 2nd point.
      xa2 NUMBER(20, 10);  ya2 NUMBER(20, 10);  za2 NUMBER(20, 10);
52
      -- Ordinates of the 3rd point.
      xa3 NUMBER(20, 10);  ya3 NUMBER(20, 10);  za3 NUMBER(20, 10);

      -- For the other surface
57    -- Ordinates of the 1st point.
      xb1 NUMBER(20, 10);  yb1 NUMBER(20, 10);  zb1 NUMBER(20, 10);

      -- Ordinates of the 2nd point.
      xb2 NUMBER(20, 10);  yb2 NUMBER(20, 10);  zb2 NUMBER(20, 10);
62
      -- Ordinates of the 3rd point.
      xb3 NUMBER(20, 10);  yb3 NUMBER(20, 10);  zb3 NUMBER(20, 10);

      -- The variables from plane equation A x + By + Cz + D = 0.
67    aa1 NUMBER(20, 10);
      bb1 NUMBER(20, 10);
      cc1 NUMBER(20, 10);
      dd1 NUMBER(20, 10);

72    aa2 NUMBER(20, 10);
      bb2 NUMBER(20, 10);
      cc2 NUMBER(20, 10);
      dd2 NUMBER(20, 10);

77    -- A small number to ensure non-zero value in the divisor
      divisor NUMBER(20, 10);

    BEGIN

82    -- Assign default values to surf_a and surf_b
      surf_a := geom1;
      surf_b := geom2;
```

```
-- Convert the MBR to a normal rectangle geometry
IF geom1.sdo_elem_info(3) = 3 AND geom1.sdo_ordinates.COUNT = 6
THEN
  surf_a := 3D_mbr2normalgeom(geom1);
END IF;
IF geom2.sdo_elem_info(3) = 3 AND geom2.sdo_ordinates.COUNT = 6
THEN
  surf_b := 3D_mbr2normalgeom(geom2);
END IF;

-- First rotate the surfaces if either is vertical
WHILE (sdo_geom.sdo_area(SDO_UTIL.REMOVE_DUPLICATE_VERTICES(sdo_cs.make_2d(surf_a),
    0.005), 0.005) = 0 OR sdo_geom.sdo_area(SDO_UTIL.REMOVE_DUPLICATE_VERTICES(
    sdo_cs.make_2d(surf_b), 0.005), 0.005) = 0) LOOP
  surf_a := 3D_Rotate(surf_a);
  surf_b := 3D_Rotate(surf_b);
END LOOP;

/* ----------------------------------------
The part BELOW can be coded in a function, whose input
is a planar surface geometry, and output is the plane
parameters this surface belongs to.
----------------------------------------*/

-- Ordinates of 3 points on the 1st plane
xa1 := surf_a.sdo_ordinates(1);
ya1 := surf_a.sdo_ordinates(2);
za1 := surf_a.sdo_ordinates(3);

xa2 := surf_a.sdo_ordinates(4);
ya2 := surf_a.sdo_ordinates(5);
za2 := surf_a.sdo_ordinates(6);

xa3 := surf_a.sdo_ordinates(7);
ya3 := surf_a.sdo_ordinates(8);
za3 := surf_a.sdo_ordinates(9);

-- Parameters of the 1st plane
aa1 := 3D_planeparameters(surf_a)(1);
bb1 := 3D_planeparameters(surf_a)(2);
cc1 := 3D_planeparameters(surf_a)(3);
dd1 := 3D_planeparameters(surf_a)(4);

---------- Ordinates and parameters of the 1st plane Finished ----------

-- Ordinates of 3 points on the 2nd plane
xb1 := surf_b.sdo_ordinates(1);
yb1 := surf_b.sdo_ordinates(2);
zb1 := surf_b.sdo_ordinates(3);

xb2 := surf_b.sdo_ordinates(4);
yb2 := surf_b.sdo_ordinates(5);
zb2 := surf_b.sdo_ordinates(6);

xb3 := surf_b.sdo_ordinates(7);
yb3 := surf_b.sdo_ordinates(8);
zb3 := surf_b.sdo_ordinates(9);

-- Parameters of the 2nd plane
```

```
  aa2 := 3D_planeparameters(surf_b)(1);
  bb2 := 3D_planeparameters(surf_b)(2);
  cc2 := 3D_planeparameters(surf_b)(3);
  dd2 := 3D_planeparameters(surf_b)(4);
147
  ------ Ordinates and parameters of the 1st plane Finished ----------

  -- Make sure the divisor won't be ZERO.
  IF aa1 = 0 THEN
    aa1 := 0.00001;
152 ELSIF bb1 = 0 THEN
    bb1 := 0.00001;
  ELSIF cc1 = 0 THEN
    cc1 := 0.00001;
157 ELSIF aa2 = 0 THEN
    aa2 := 0.00001;
  ELSIF bb2 = 0 THEN
    bb2 := 0.00001;
  ELSIF cc2 = 0 THEN
162   cc2 := 0.00001;
  END IF;

  /*-----------------------------------------
  The part ABOVE can be coded in a function, whose input
167 is a planar surface geometry, and output is the plane
  parameters this surface belongs to.
  -----------------------------------------*/

  -- Check if the two planes are coplanar by calculating
172 -- the angle in between.
  cos_theta := (aa1*aa2 + bb1*bb2 + cc1*cc2) /
    (sqrt(aa1*aa1 + bb1*bb1 + cc1*cc1)
    * sqrt(aa2*aa2 + bb2*bb2 + cc2*cc2));

177 -- If parallel then cos(theta) = 1, and the relation in 2D
  -- represents the actual 3D relation;
  IF abs(cos_theta - 1) <= 0.002 THEN
    relation := SDO_GEOM.Relate(SDO_CS.Make_2D(surf_a),
      'determine', SDO_CS.Make_2D(surf_b), 0.005);
182
    -- If they overlap then the orientations of two surfaces
    -- are the same, and one one solid the surface comes from
    -- is inside the other.
    IF relation = 'OVERLAPBDYINTERSECT' OR relation = 'EQUAL'
187   OR relation = 'INSIDE' OR relation = 'CONTAINS'
      OR relation = 'COVERS' OR relation = 'COVEREDBY' THEN
      relation := 'Overlap_Parallel';
    END IF;

192 -- If opposite then cos(theta) = -1, and one solid the surface
  -- comes from is adjacent to the other, e.g. sharing a common face.
  ELSIF abs(cos_theta + 1) <= 0.002 THEN
    relation := SDO_GEOM.Relate(SDO_CS.Make_2D(surf_a), 'determine',
      SDO_CS.Make_2D(surf_b), 0.005);
197
    IF relation = 'OVERLAPBDYINTERSECT' OR relation = 'EQUAL'
      OR relation = 'INSIDE' OR relation = 'CONTAINS'
      OR relation = 'COVERS' OR relation = 'COVEREDBY' THEN
      relation := 'Overlap_Opposite';
```

```
202     END IF;

    -- IF two planes are not coplanar
    ELSE
      union_mbr := SDO_GEOM.SDO_MBR(SDO_GEOM.SDO_Union
207       (sdo_cs.make_2d(surf_a), sdo_cs.make_2d(surf_b), 0.005));

      -- Assign the values of ordinates from MBR
      xll_union := union_mbr.sdo_ordinates(1);
      yll_union := union_mbr.sdo_ordinates(2);
212   xur_union := union_mbr.sdo_ordinates(3);
      yur_union := union_mbr.sdo_ordinates(4);

      -- Form the corner coodinates of the bigger MBR, which will
      -- surely CONTAIN the objects.
217   xll_big := xll_union - abs((xur_union - xll_union) / 2);
      yll_big := yll_union - abs((yur_union - yll_union) / 2);
      xur_big := xur_union + abs((xur_union - xll_union) / 2);
      yur_big := yur_union + abs((yur_union - yll_union) / 2);

222   -- Find out the ordinates of two points that're on the line
      pt1_x := xll_big;
      pt2_x := xur_big;

      divisor := bb1 - cc1*bb2/cc2;
227   -- Make sure the divisor won't be zero.
      IF divisor = 0 THEN
        divisor := divisor + 0.00001;
      END IF;

232   -- Compose a line object for projection
      pt1_y := (cc1*dd2/cc2 - dd1 - (aa1 - cc1*aa2/cc2)*pt1_x) / divisor;
      pt2_y := (cc1*dd2/cc2 - dd1 - (aa1 - cc1*aa2/cc2)*pt2_x) / divisor;

      proj_line := MDSYS.SDO_GEOMETRY(2002, null, null,
237     MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1),
        MDSYS.SDO_ORDINATE_ARRAY(pt1_x,pt1_y, pt2_x,pt2_y));

      common_surf_geom := SDO_GEOM.SDO_Intersection(SDO_CS.Make_2D(surf_a),
        SDO_CS.Make_2D(surf_b), 0.005);
242
      -- Here is the essential distinction amongst different relations
      IF SDO_GEOM.Relate(proj_line, 'determine', common_surf_geom, 0.005)
        = 'OVERLAPBDYDISJOINT' THEN
        relation := 'Intersect';
247   ELSIF SDO_GEOM.Relate(proj_line, 'determine', common_surf_geom, 0.005)
        = 'DISJOINT' THEN
        relation := 'Disjoint';
      ELSE
        relation := 'Touch';
252   END IF;

      -- Retrieve the shared part from the intersection of
      -- two polygons and the projected line.
      common_touch_geom := SDO_Geom.SDO_Intersection(proj_line,
257     common_surf_geom, 0.005);

      -- Distinguish Strong-touch and Touch
      -- If the common geometry is line
```

```
    IF relation = 'Touch' THEN
      -- When the shared part is line object
      IF common_touch_geom.sdo_gtype = 2002 THEN
        IF SDO_GEOM.relate(common_touch_geom, 'determine',
          SDO_CS.Make_2D(surf_a), 0.005) != 'ON' THEN
          relation := 'Strong-Touched-By';
        ELSIF SDO_GEOM.relate(common_touch_geom, 'determine',
          SDO_CS.Make_2D(surf_b), 0.005) != 'ON' THEN
          relation := 'Strong-Touch';
        END IF;
      END IF;

      -- When the shared part is point object
      IF common_touch_geom.sdo_gtype = 2001 THEN
        IF SDO_GEOM.relate(common_touch_geom, 'determine',
          SDO_CS.Make_2D(surf_a), 0.005) != 'TOUCH' THEN
          relation := 'Strong-Touched-By';
        ELSIF SDO_GEOM.relate(common_touch_geom, 'determine',
          SDO_CS.Make_2D(surf_b), 0.005) != 'TOUCH' THEN
          relation := 'Strong-Touch';
        END IF;
      END IF;
    END IF; -- If the relation is Touch
  END IF; --Check if two surfaces are parallel

  RETURN(relation);
END 3D_SurfaceRelate;
```

Listing C.2: Module function 3D_PlaneParameters

```
CREATE or REPLACE FUNCTION 3D_planeparameters(

/* Given a planar surface geometry as input, this function
   returns the parameters of plane this surface belongs to. */

  geom MDSYS.SDO_Geometry)
  RETURN plane_para_list

AS
  -- Declare the variables for one surface
  -- Ordinates of the 1st point.
  x1 NUMBER(20, 10);
  y1 NUMBER(20, 10);
  z1 NUMBER(20, 10);

  -- Ordinates of the 2nd point.
  x2 NUMBER(20, 10);
  y2 NUMBER(20, 10);
  z2 NUMBER(20, 10);

  -- Ordinates of the 3rd point.
  x3 NUMBER(20, 10);
  y3 NUMBER(20, 10);
  z3 NUMBER(20, 10);

  -- The variables from plane equation A x + By + Cz + D = 0.
  aa NUMBER(20, 10);
  bb NUMBER(20, 10);
```

```
  cc NUMBER(20, 10);
  dd NUMBER(20, 10);

  surf SDO_Geometry;

  -- A variable of plane_para_list for resuling a,b,c,d
  plane_para plane_para_list := plane_para_list(0,0,0,0);

BEGIN

  -- Check if it's a MBR geometry, if yes it's to be converted
  -- to a usual geometric representation (all vertices instead
  -- of 2 corner points)
  IF geom.sdo_elem_info(3) = 3 AND geom.sdo_ordinates.COUNT = 6
  THEN
    surf := 3D_mbr2normalgeom(geom);
  ELSE
    surf := geom;
  END IF;

  -- Begin the calculations
  x1 := surf.sdo_ordinates(1);
  y1 := surf.sdo_ordinates(2);
  z1 := surf.sdo_ordinates(3);

  x2 := surf.sdo_ordinates(4);
  y2 := surf.sdo_ordinates(5);
  z2 := surf.sdo_ordinates(6);

  x3 := surf.sdo_ordinates(7);
  y3 := surf.sdo_ordinates(8);
  z3 := surf.sdo_ordinates(9);

  plane_para(1) := y1*(z2 - z3) + y2*(z3 - z1) + y3*(z1 - z2);
  plane_para(2) := z1*(x2 - x3) + z2*(x3 - x1) + z3*(x1 - x2);
  plane_para(3) := x1*(y2 - y3) + x2*(y3 - y1) + x3*(y1 - y2);
  plane_para(4) := -(x1*(y2*z3 - y3*z2) + x2*(y3*z1 - y1*z3)
    + x3*(y1*z2 - y2*z1));

  RETURN(plane_para);

END 3D_planeparameters;
```

Listing C.3: Module function 3D_Rotate

```
CREATE or REPLACE FUNCTION 3D_Rotate(
  /* Rotate a given 3D object about a user-defined axis. */

  -- Geometry of the object that's gonna be rotated in 3D space.
  geom SDO_GEOMETRY,

  -- The direction vector, which is parallel to the rotating axis
  axis_vector dir_vector DEFAULT dir_vector(1, 1, 1),

  -- The point that the vector goes through
  start_point collinear_point DEFAULT collinear_point(0, 0, 0),

  -- The angle the geometry is rotated by, counter-clockwise
  angle NUMBER DEFAULT 50.0)
```

```
  RETURN SDO_GEOMETRY

AS
  -- Declare variables
  new_geom SDO_Geometry;
  normal_geom SDO_Geometry;

  -- The parameters of direction vector.
  u NUMBER; v NUMBER; w NUMBER;

  -- The point to be rotated
  x NUMBER(20, 5);  y NUMBER(20, 5);  z NUMBER(20, 5);

  -- The point that the vector goes through
  aa NUMBER; bb NUMBER; cc NUMBER;

BEGIN

  -- Convert the MBR to normal surface geometry
  IF geom.sdo_elem_info(3) = 3 AND geom.sdo_ordinates.COUNT = 6 THEN
    normal_geom := 3D_mbr2normalgeom(geom);
    new_geom := normal_geom;
  ELSE
    -- The initial value of the new geometry is the same as the old.
    normal_geom := geom;
    new_geom := normal_geom;
  END IF;

  u := axis_vector(1);
  v := axis_vector(2);
  w := axis_vector(3);

  aa := start_point(1);
  bb := start_point(2);
  cc := start_point(3);

  FOR i IN 1..normal_geom.sdo_ordinates.COUNT LOOP
    IF mod(i, 3) = 1 THEN
      -- The cursor is at the x ordinate.
      x := normal_geom.sdo_ordinates(i);
      y := normal_geom.sdo_ordinates(i + 1);
      z := normal_geom.sdo_ordinates(i + 2);

      -- Rotate x value.
      new_geom.sdo_ordinates(i) := (aa*(v*v + w*w) - u*(bb*v
        + cc*w - u*x - v*y - w*z))*(1 - cos(angle)) +
        x*cos(angle) + (-cc*v + bb*w - w*y + v*z)*sin(angle);
      -- Rotate y value.
      new_geom.sdo_ordinates(i + 1) := (bb*(u*u + w*w) -
        v*(aa*u + cc*w - u*x - v*y - w*z))*(1 - cos(angle)) +
        y*cos(angle) + (cc*u - aa*w + w*x - u*z)*sin(angle);
      -- Rotate z value.
      new_geom.sdo_ordinates(i + 2) := (cc*(u*u + v*v) -
        w*(aa*u + bb*v - u*x - v*y - w*z))*(1 - cos(angle)) +
        z*cos(angle) + (-bb*u + aa*v - v*x + u*y)*sin(angle);
    END IF;
  END LOOP;

  RETURN(new_geom);
```

```
74 END 3D_Rotate;
```

Listing C.4: Module function 3D_MBR2NormalGeom

```
1  CREATE or REPLACE FUNCTION 3D_MBR2NormalGeom(

     /* Convert a MBR geometry to a usual surface geometry. */

     mbr_geom SDO_Geometry)
6    RETURN SDO_Geometry

   AS
     normal_geom SDO_Geometry;
     x1 NUMBER(20,10) DEFAULT 0;
11   y1 NUMBER(20,10) DEFAULT 0;
     z1 NUMBER(20,10) DEFAULT 0;

     x2 NUMBER(20,10) DEFAULT 0;
     y2 NUMBER(20,10) DEFAULT 0;
16   z2 NUMBER(20,10) DEFAULT 0;

     x3 NUMBER(20,10) DEFAULT 0;
     y3 NUMBER(20,10) DEFAULT 0;
     z3 NUMBER(20,10) DEFAULT 0;
21
     x4 NUMBER(20,10) DEFAULT 0;
     y4 NUMBER(20,10) DEFAULT 0;
     z4 NUMBER(20,10) DEFAULT 0;
   BEGIN
26
     -- When the surface is a MBR, it only has 2 points
     -- and is parallel to xy or yz or xz plane.
     IF mbr_geom.sdo_elem_info(3) = 3 AND
       mbr_geom.sdo_ordinates.COUNT = 6
31   THEN
       normal_geom := mbr_geom;

       x1 := mbr_geom.sdo_ordinates(1);
       y1 := mbr_geom.sdo_ordinates(2);
36     z1 := mbr_geom.sdo_ordinates(3);

       x3 := mbr_geom.sdo_ordinates(4);
       y3 := mbr_geom.sdo_ordinates(5);
       z3 := mbr_geom.sdo_ordinates(6);
41
       IF x1 = x3 THEN
         x2 := x1;
         y2 := y3;
         z2 := z1;
46       x4 := x1;
         y4 := y1;
         z4 := z3;
       ELSIF y1 = y3 THEN
         x2 := x3;
51       y2 := y1;
         z2 := z1;
         x4 := x1;
         y4 := y1;
         z4 := z3;
```

```
    ELSIF z1 = z3 THEN
      x2 := x3;
      y2 := y1;
      z2 := z1;
      x4 := x1;
      y4 := y3;
      z4 := z1;
    END IF;

    normal_geom.sdo_ordinates.EXTEND(9);

    normal_geom.sdo_ordinates(4) := x2;
    normal_geom.sdo_ordinates(5) := y2;
    normal_geom.sdo_ordinates(6) := z2;

    normal_geom.sdo_ordinates(7) := x3;
    normal_geom.sdo_ordinates(8) := y3;
    normal_geom.sdo_ordinates(9) := z3;

    normal_geom.sdo_ordinates(10) := x4;
    normal_geom.sdo_ordinates(11) := y4;
    normal_geom.sdo_ordinates(12) := z4;

    normal_geom.sdo_ordinates(13) := x1;
    normal_geom.sdo_ordinates(14) := y1;
    normal_geom.sdo_ordinates(15) := z1;

    normal_geom.sdo_elem_info(3) := 1;

    RETURN(normal_geom);

  ELSE
    dbms_output.put_line('The input geometry is NOT 3D
      or a minimum bounding rectangle!');
    RETURN(mbr_geom);
  END IF;

END 3D_MBR2NormalGeom;
```

C.2 PROCEDURES

Listing C.5: Package of all constraints

```
CREATE or REPLACE PACKAGE 3D_CCC_CONSTRAINTS
IS
  /* The codes saved here can be used in the triggers, procedures
    and functions. */

  -- A global variable to store the row that's being modified
  -- at table SURFACE_GEOMETRY
  surfrecord_io surface_geometry%rowtype;

  -- A global variable to store the IDs of rows being modified
  -- at table CITYOBJECT_GENERICATTRIB.
  mobile_senspts_ids ids_list := ids_list();
```

```
  -- If an exception candidate in the table of exceptions is set 'TRUE'
  -- in the flag 'Accepted', then free_entry_flag is set 'TRUE' also.
  -- Then pass the dataset without the same check that would again
  -- move it to the table of exceptions.
  free_entry_flag CHAR(5) DEFAULT 'FALSE';

  -- Intersection of city object, taking building as an example.
  PROCEDURE 3D_building_interrupt_building;

  -- Exception for 'should' cases, taking tree-building min distance
  -- as an example.
  PROCEDURE 3D_tree_mindist2_building;

  /* 3 make-up rules about distance amongst tree-building-road
  that are conflicting. */
  -- Tree should be < 5m from the house
  PROCEDURE 3D_tree2house_maxdist;

  -- Road should be > 10m from the house
  PROCEDURE 3D_road2house_mindist;

  -- Tree should be < 2m from the road
  PROCEDURE 3D_tree2road_maxdist;
  /* 3 make-up rules about distance amongst tree-building-road
  that are conflicting. */

  -- Detect JUMP points in sensing route of temperature measurements.
  -- An example to study the spatial characteristic of mobile city object.
  PROCEDURE 3D_mobilesensor_building;

  -- Check if a modified temperature exceeds the max. changing rate.
  -- An example constraint for climatic measurements
  PROCEDURE smooth_temp_change;

END CCC_SPATIO_CONSTRAINTS;
```

Package Body

Listing C.6: All procedures that code constraints

```
CREATE or REPLACE PACKAGE BODY 3D_CCC_SPATIO_CONSTRAINTS AS
  -- An example of spatial intersection of city objects
  PROCEDURE 3D_building_interrupt_building
  AS

    relation VARCHAR2(50) DEFAULT 'Disjoint';

    exclude_inside_container SDO_Geometry;

    -- Check how many MBR contains the updated surface
    -- this is to make sure the surface is not inside
    -- the other building
    inside_mbr_ids ids_list := ids_list();

    -- The id of other building surfaces that interact with
    -- the updated surface
    interact_ids ids_list := ids_list();

    coords geoms_list := geoms_list();
```

```
    -- The amount of surface relations in each category
22  surfids_intersect ids_list := ids_list();
    intersect_count INTEGER DEFAULT 0;

    surfids_overlap_parallel ids_list := ids_list();
    parallel_count INTEGER DEFAULT 0;
27
    surfids_overlap_opposite ids_list := ids_list();
    opposite_count INTEGER DEFAULT 0;

    surfids_strong_touch ids_list := ids_list();
32  strong_touch_count INTEGER DEFAULT 0;

    surfids_strong_touch_by ids_list := ids_list();
    strong_touch_by_count INTEGER DEFAULT 0;

37  surfids_touch ids_list := ids_list();
    touch_count INTEGER DEFAULT 0;

    buildings_interrupt EXCEPTION;

42  BEGIN

    IF surfrecord_io.class_name = 'BUILDING' THEN
      dbms_output.put_line('The changed ID is ' ||
        surfrecord_io.id || '.');
47
      /* Select the geometries of building that
      interact with this updated surface. */
      SELECT geom1.geometry, geom1.id
        BULK COLLECT INTO coords, interact_ids
52    FROM surface_geometry geom1, surface_geometry geom2
      WHERE geom2.id = surfrecord_io.id
        -- make sure geom2 is a different building
        AND geom1.root_id <> geom2.root_id
        AND geom1.class_name = 'BUILDING'
57      AND SDO_Anyinteract(geom1.geometry, geom2.geometry) = 'TRUE';

      -- Detect the relation
      IF coords.COUNT > 0 THEN
        FOR i IN 1..coords.COUNT
62      LOOP
          -- Compare a surface from bld_root_id with every
          -- surface from building2
          relation := 3D_SurfaceRelate(surfrecord_io.geometry, coords(i));

67        IF relation = 'Intersect' THEN
            intersect_count := intersect_count + 1;
            surfids_intersect.EXTEND;
            surfids_intersect(intersect_count) := interact_ids(i);
          ELSIF relation = 'Overlap_Parallel' THEN
72          parallel_count := parallel_count + 1;
            surfids_overlap_parallel.EXTEND;
            surfids_overlap_parallel(parallel_count) := interact_ids(i);
          ELSIF relation = 'Strong-Touch' THEN
            strong_touch_count := strong_touch_count + 1;
77          surfids_strong_touch.EXTEND;
            surfids_strong_touch(strong_touch_count) := interact_ids(i);
```

```
      ELSIF relation = 'Strong-Touched-By' THEN
        strong_touch_by_count := strong_touch_by_count + 1;
        surfids_strong_touch_by.EXTEND;
        surfids_strong_touch_by(strong_touch_by_count) := interact_ids(i);
      ELSIF relation = 'Overlap_Opposite' THEN
        opposite_count := opposite_count + 1;
        surfids_overlap_opposite.EXTEND;
        surfids_overlap_opposite(opposite_count) := interact_ids(i);
      ELSIF relation = 'Touch' THEN
        touch_count := touch_count + 1;
        surfids_touch.EXTEND;
        surfids_touch(touch_count) := interact_ids(i);
      END IF;
    END LOOP;

    -- Two buildings have surfaces intersected or parallel
    IF surfids_overlap_parallel.COUNT > 0 OR
      surfids_intersect.COUNT > 0 THEN
      RAISE buildings_interrupt;
    ELSE
      dbms_output.put_line(coords.count
        || ' surfaces have contact with surface ID='
        || surfrecord_io.id || '.');

      -- Display the strong-touch details
      IF strong_touch_count > 0 THEN
        dbms_output.put_line('It strong-touches '
          || strong_touch_count || ' surface(s)');
        dbms_output.put('which are of ID=');
        FOR i IN 1..surfids_strong_touch.COUNT LOOP
          dbms_output.put(surfids_strong_touch(i) || ' ');
        END LOOP;
      END IF;

      -- Display the strong-touched-by details
      IF strong_touch_by_count > 0 THEN
        dbms_output.put_line('It is strong-touched_by ' ||
        strong_touch_by_count || ' surface(s)');
        dbms_output.put('which are of ID=');
        FOR i IN 1..surfids_strong_touch_by.COUNT LOOP
          dbms_output.put(surfids_strong_touch_by(i) || ' ');
        END LOOP;
      END IF;

      -- Display the touch details
      IF touch_count > 0 THEN
        dbms_output.put_line('It touches ' || touch_count ||
          ' surface(s)');
        dbms_output.put('which are of ID=');
        FOR i IN 1..surfids_touch.COUNT LOOP
          dbms_output.put(surfids_touch(i) || ' ');
        END LOOP;
      END IF;

      dbms_output.put_line(' ');
    END IF;

  /* When no surfaces are found interacting, it's necessary
  -- to check if two buildings are really disjoint or
```

```
      -- one is actually inside the other.
      -- BUT HOW? */
      ELSE
        /* Select two sets of surfaces and construct
        an aggregational MBR */
        dbms_output.put_line('The target surface does not have
          any contact with the other building surfaces.  But it is not
          clear if it really disjoints from them or it is inside another
          building without touching the faces.  It is better to visualise
          the surfaces.');
      END IF;
    END IF;

  EXCEPTION
    WHEN buildings_interrupt THEN
      -- Show the number of intersections
      IF intersect_count > 0 THEN
        FOR i IN 1..surfids_intersect.COUNT
        LOOP
          dbms_output.put_line('The updated surface intersects
            surfaces of ID=' || surfids_intersect(i) || ' .');
        END LOOP;
      END IF;

      -- Show the number of overlappings (means this surface
      -- touches the other building from inside.
      IF parallel_count > 0 THEN
        FOR i IN 1..surfids_overlap_parallel.COUNT
        LOOP
          dbms_output.put_line('The updated surface overlaps surfaces
            of ID=' || surfids_overlap_parallel(i) || ' .');
        END LOOP;
      END IF;
      raise_application_error(-20001, 'Spatial Constraint:
The building surface(s) interrupts the other building(s).
Please correct the intersecting and overlapping data
detected in the diagnosis.');

  END 3D_building_interrupt_building;

  -- Exception for 'should' cases
  PROCEDURE 3D_tree_mindist2_building
  AS
    should_tree_ids ids_list := ids_list();

    -- The whole :New row of surface_geometry
    treegeom_record surface_geometry%rowtype;

    -- The list of buildings that are too close (< 0.5) to
    -- this updated/inserted tree surface
    bldsurf_tooclose_ids ids_list := ids_list();

    -- Have the message texts stored in column 'Reason', which shows
    -- why the instance is likely to be an exception
    buffer VARCHAR2(1000);
    status INTEGER;

    -- The number of columns in surface_geometry table
    surf_geom_column_count INTEGER;
```

```
197   -- and the clone table for exceptions.
      surf_geom_exc_column_count INTEGER;

      tree_tooclose2bld EXCEPTION;

202   BEGIN

      IF surfrecord_io.class_name = 'TREE' THEN
        /* Select the building surfaces that are within min distance
          to this (part of) tree. */
207     SELECT bld1.id BULK COLLECT INTO bldsurf_tooclose_ids
        FROM surface_geometry bld1
        WHERE bld1.class_name = 'BUILDING'
          AND SDO_Within_Distance(bld1.geometry, surfrecord_io.geometry,
            'distance=2m') = 'TRUE';
212
        SELECT COUNT(*) INTO surf_geom_column_count
        FROM user_tab_columns
        WHERE table_name = 'SURFACE_GEOMETRY';

217     SELECT COUNT(*) INTO surf_geom_exc_column_count
        FROM user_tab_columns
        WHERE table_name = 'SURFACE_GEOMETRY_EXCEPTIONS';

        -- It's forbidden to have tree closer to a building than 2 meters.
222     IF bldsurf_tooclose_ids.COUNT > 0 THEN
          -- Check whether it's from surface_geometry table or from
          -- exceptions table.

          -- Assume that this is the 1st attempt to insert/update, Reject!
227       dbms_output.put('The (part of) tree object with id=' ||
          surfrecord_io.id || ' is too close (dist < 2m) to
          building surface(s) of id=');

          FOR i IN 1..bldsurf_tooclose_ids.COUNT LOOP
232         dbms_output.put(bldsurf_tooclose_ids(i) || ',');
          END LOOP;
          dbms_output.put_line(' .');

          -- Get the message text into variable as the value
237       -- for column 'Reason'
          dbms_output.get_line(buffer, status);

          RAISE tree_tooclose2bld;

242     ELSE
          dbms_output.put_line('The (part of) tree object with id='
            || surfrecord_io.id || ' is placed in a proper distance
            (> 2m) from building(s).');
        END IF;
247   END IF; -- If the changed object is from tree class.

      EXCEPTION
        WHEN tree_tooclose2bld THEN
          -- The mutating-table restriction doesn't apply to EXCEPTION?
252       DELETE FROM surface_geometry
          WHERE id = surfrecord_io.id;

          -- Strange that the rowtype doesn't work?!
```

```
        INSERT INTO surface_geometry_exceptions VALUES (
257       surfrecord_io.id,
          surfrecord_io.gmlid,
          surfrecord_io.gmlid_codespace,
          surfrecord_io.parent_id,
          surfrecord_io.root_id,
262       surfrecord_io.is_solid,
          surfrecord_io.is_composite,
          surfrecord_io.is_triangulated,
          surfrecord_io.is_xlink,
          surfrecord_io.is_reverse,
267       surfrecord_io.geometry,
          surfrecord_io.class_name,
          surfrecord_io.class_object_id,
          'FALSE',
          buffer);
272     /* In trigger the Insertion/Update doesn't need to COMMIT. */

        /* Actually all the transactions (change of data) is canceled
          by raising application error.  Which means the insertion to
          the clone table can't be done! */
277     dbms_output.put_line('The updated/inserted tree is suspected to be
          an error, and has been moved to table SURFACE_GEOMETRY_EXCEPTIONS.
          Please check the instance(s) yourself, and change the value in
          'Accepted' into 'TRUE' if the instance is alright.');

282 END 3D_tree_mindist2_building;

    /* Make-up rules that are conflicting */
    -- Tree should be < 5m from the house
    PROCEDURE 3D_tree2house_maxdist
287 AS
      bldsurf_within5m_ids ids_list := ids_list();
      tree_toofar_from_bld EXCEPTION;

    BEGIN
292
      IF surfrecord_io.class_name = 'TREE' THEN
        -- Select the building surfaces that are within 5m
        -- from the inserted/updated tree object.
        SELECT s1.id BULK COLLECT
297       INTO bldsurf_within5m_ids
        FROM surface_geometry s1
        WHERE s1.class_object_id = 602 AND s1.class_name = 'BUILDING'
          AND SDO_Within_Distance(s1.geometry, surfrecord_io.geometry,
          'distance=5m') = 'TRUE';
302
        -- When there is no building within 5 meters
        IF bldsurf_within5m_ids.COUNT = 0 THEN
          RAISE tree_toofar_from_bld;
        ELSE
307       dbms_output.put_line('The tree is within 5m from the house,
            which is good!');
        END IF;
      END IF;

312 EXCEPTION
      WHEN tree_toofar_from_bld THEN
        RAISE_APPLICATION_ERROR(-20003, 'Spatial Constraint: Tree is
```

```
    too far (dist > 5m) from the house.');

END 3D_tree2house_maxdist;

-- Road should be > 10m from the house
PROCEDURE 3D_road2house_mindist
AS
  bldsurf_within10m_ids ids_list := ids_list();
  road_tooclose_to_bld EXCEPTION;
BEGIN
  NULL;
  IF surfrecord_io.class_name = 'ROAD' THEN
    -- Select the building surfaces that are within 5m from
    -- the inserted/updated tree object.
    SELECT s1.id BULK COLLECT
      INTO bldsurf_within10m_ids
    FROM surface_geometry s1
    WHERE s1.class_object_id = 602 AND s1.class_name = 'BUILDING'
      AND SDO_Within_Distance(s1.geometry, surfrecord_io.geometry,
      'distance=10m') = 'TRUE';

    -- When there is a building within 10 meters
    IF bldsurf_within10m_ids.COUNT <> 0 THEN
      RAISE road_tooclose_to_bld;
    ELSE
      dbms_output.put_line('The road is further than 10m from
        the house, which is good!');
    END IF;

  END IF;

EXCEPTION
  WHEN road_tooclose_to_bld THEN
    RAISE_APPLICATION_ERROR(-20003, 'Spatial Constraint: Road
    is too close (dist < 10m) to the house.');

END 3D_road2house_mindist;

-- Tree should be < 2m from the road
PROCEDURE 3D_tree2road_maxdist
AS
  roadsurf_within2m_ids ids_list := ids_list();
  tree_toofar_from_road EXCEPTION;
BEGIN
  NULL;
  IF surfrecord_io.class_name = 'TREE' THEN
    -- Select the building surfaces that are within 5m from
    -- the inserted/updated tree object.
    SELECT s1.id BULK COLLECT
      INTO roadsurf_within2m_ids
    FROM surface_geometry s1
    WHERE s1.class_object_id = 9001
      AND s1.class_name = 'ROAD'
      AND SDO_Within_Distance(s1.geometry, surfrecord_io.geometry,
      'distance=2m') = 'TRUE';

    -- When there is no road within 2 meters from the tree
    IF roadsurf_within2m_ids.COUNT = 0 THEN
      RAISE tree_toofar_from_road;
```

```
      ELSE
        dbms_output.put_line('The tree is closer to the road
          than 2m, which is good!');
      END IF;

    END IF;

  EXCEPTION
    WHEN tree_toofar_from_road THEN
      RAISE_APPLICATION_ERROR(-20003, 'Spatial Constraint: Tree
      is too far (dist > 2m) from the road.');

  END 3D_tree2road_maxdist;
  /* Make-up rules that are conflicting */

  -- Detect the sensing route of mobile sensor, temporal+spatial const.
  PROCEDURE 3D_mobilesensor_building
  AS
    points_list geoms_list := geoms_list();
    pt1 SDO_Geometry;
    pt2 SDO_Geometry;
    pt3 SDO_Geometry;

    -- The footprint geometry of referenced building
    ref_bld_footprint SDO_Geometry;

    insidebld_ids ids_list := ids_list();
    outsidebld_ids ids_list := ids_list();
    jumpin_ids ids_list := ids_list();
    jumpout_ids ids_list := ids_list();

    speeds ids_list := ids_list();
    starting_id INTEGER;
    end_id INTEGER;

    beyond_speed_limit EXCEPTION;

  BEGIN

    starting_id := mobile_senspts_ids(1);
    end_id := mobile_senspts_ids(mobile_senspts_ids.LAST);

    SELECT sdo_geom.sdo_distance(c1.geomval, c2.geomval, 0.00005)/
      ((c1.dateval - c2.dateval)*86400 + 0.0001) -- Speed in dist/second
      BULK COLLECT INTO speeds
    FROM cityobject_genericattrib c1, cityobject_genericattrib c2
    WHERE c1.id <= end_id - 1 AND c1.id >= starting_id
      AND c2.id <= end_id AND c2.id >= starting_id + 1
      AND c1.id = c2.id - 1;

    SELECT geomval BULK COLLECT INTO points_list
    FROM cityobject_genericattrib
    WHERE id <= end_id AND id >= starting_id;

    FOR i IN 1..speeds.COUNT LOOP
      IF speeds(i) >= 3 THEN
        RAISE beyond_speed_limit;
      END IF;
    END LOOP;
```

```
                    Jump Check
SELECT pts.id BULK COLLECT INTO outsidebld_ids
FROM cityobject_genericattrib_test pts, surface_geometry surf
WHERE pts.id <= end_id AND pts.id >= starting_id
  AND surf.id = 60056
  AND sdo_geom.relate(sdo_cs.make_2d(pts.geomval), 'determine',
      sdo_cs.make_2d(surf.geometry), 0.00005) = 'DISJOINT';

SELECT pts.id BULK COLLECT INTO insidebld_ids
FROM cityobject_genericattrib_test pts, surface_geometry surf
WHERE pts.id <= 131 AND pts.id >= 119
  AND surf.id = 60056
  AND sdo_geom.relate(sdo_cs.make_2d(pts.geomval), 'determine',
      sdo_cs.make_2d(surf.geometry), 0.00005) <> 'DISJOINT';

-- OTB is the reference building in this case, with ID = 60056
SELECT SDO_CS.make_2d(geometry) INTO ref_bld_footprint
FROM surface_geometry
WHERE id = 60056;

-- If the inserted points are all inside or outside
-- the building then neglect the check.
IF outsidebld_ids.COUNT = mobile_senspts_ids.COUNT THEN
  dbms_output.put_line('All inserted measurements are
     captured outdoors.');
ELSIF insidebld_ids.COUNT = mobile_senspts_ids.COUNT THEN
   dbms_output.put_line('All inserted measurements are
     captured indoors.');
ELSE
  -- Not all are outdoor or indoor. Then it needs a
  further check of possible JUMP points.
  /* Let user specify Indoor or Outdoor? */
  FOR i IN 1..(points_list.COUNT - 2) LOOP
    -- Compare the geometries in 2D (x,y) plane.
    pt1 := SDO_CS.make_2d(points_list(i));
    pt2 := SDO_CS.make_2d(points_list(i + 1));
    pt3 := SDO_CS.make_2d(points_list(i + 2));

    -- If pt2 is indoor by its two neighbours are outdoor
    -- Then pt2 is suspected to be a 'jump-in' point.
    IF SDO_Geom.relate(pt1, 'determine', ref_bld_footprint,
        0.00005) = 'DISJOINT'
      AND SDO_Geom.relate(pt2, 'determine', ref_bld_footprint,
        0.00005) <> 'DISJOINT'
      AND SDO_Geom.relate(pt3, 'determine', ref_bld_footprint,
        0.00005) = 'DISJOINT' THEN
      jumpin_ids.EXTEND;
      jumpin_ids(jumpin_ids.LAST) := mobile_senspts_ids(i + 1);
    END IF;

    IF SDO_Geom.relate(pt1, 'determine', ref_bld_footprint,
        0.00005) <> 'DISJOINT'
      AND SDO_Geom.relate(pt2, 'determine', ref_bld_footprint,
        0.00005) = 'DISJOINT'
      AND SDO_Geom.relate(pt3, 'determine', ref_bld_footprint,
        0.00005) <> 'DISJOINT' THEN
      jumpout_ids.EXTEND;
      jumpout_ids(jumpout_ids.LAST) := mobile_senspts_ids(i + 1);
```

```
492         END IF;
          END LOOP;

          IF jumpin_ids.COUNT > 0 THEN
            dbms_output.put('There are points possible to be JUMPING-in.
              They are with id = ');
497         FOR i IN 1..jumpin_ids.COUNT LOOP
              dbms_output.put(jumpin_ids(i) || ', ');
            END LOOP;
            dbms_output.put_line(' ');
502       END IF;

          IF jumpout_ids.COUNT > 0 THEN
            dbms_output.put('There are points possible to be JUMPING-out.
              They are with id = ');
507         FOR i IN 1..jumpout_ids.COUNT LOOP
              dbms_output.put(jumpout_ids(i) || ', ');
            END LOOP;
            dbms_output.put_line(' ');
          END IF;
512
      END IF;

    EXCEPTION
      WHEN beyond_speed_limit THEN
517     raise_application_error(-20005, 'Tempo-Spatial Constraint:
The speed of moving object is beyond its limit!');
    END 3D_mobilesensor_building;

    -- Prevent temperature measurement from changing too fast.
522 PROCEDURE smooth_temp_change
    AS
      quickchange_ids ids_list := ids_list();
      valchangetooquick EXCEPTION;
    BEGIN
527   SELECT temp.id BULK COLLECT
        INTO quickchange_ids
      FROM cityobject_genericattrib_test temp
      WHERE temp.id <> sensor_io.id
        AND abs((sensor_io.intval - temp.intval) / ((sensor_io.dateval
532       - temp.dateval)*86400 + 0.0001)) > (3/60);

      IF quickchange_ids.COUNT > 0 THEN
        RAISE valchangetooquick;
      END IF;
537
    EXCEPTION
      WHEN valchangetooquick THEN
        dbms_output.put('The modified temperature value changes
          too fast w.r.t. records of id= ');
542     FOR i IN 1..quickchange_ids.COUNT LOOP
          dbms_output.put(quickchange_ids(i) || ', ');
        END LOOP;
        dbms_output.put_line('.');

547     RAISE_APPLICATION_ERROR(-20101,'Temporal Constraint: The modified
temperature changes too fast!  Max. rate is 3 Celsius degrees per minute. Please check
    DBMS_OUTPUT message for detailed diagnosis.');
    END smooth_temp_change;
```

```
END CCC_SPATIO_CONSTRAINTS;
```

C.3 TRIGGERS

After row triggers, which are used to pass the row level value to procedural variables for constraints check.

Listing C.7: Row-level triggers to assign row values

```
CREATE or REPLACE TRIGGER GET_SURF_INFO_ALL
AFTER INSERT OR UPDATE ON surface_geometry
FOR EACH ROW

BEGIN
  IF :new.geometry IS NOT NULL THEN
    -- Assign the new row to rowtype variable column by column
    ccc_spatio_constraints.surfrecord_io.id := :new.id;
    ccc_spatio_constraints.surfrecord_io.gmlid := :new.gmlid;
    ccc_spatio_constraints.surfrecord_io.gmlid_codespace := :new.gmlid_codespace;
    ccc_spatio_constraints.surfrecord_io.parent_id := :new.parent_id;
    ccc_spatio_constraints.surfrecord_io.root_id := :new.root_id;
    ccc_spatio_constraints.surfrecord_io.is_solid := :new.is_solid;
    ccc_spatio_constraints.surfrecord_io.is_composite := :new.is_composite;
    ccc_spatio_constraints.surfrecord_io.is_triangulated := :new.is_triangulated;
    ccc_spatio_constraints.surfrecord_io.is_xlink := :new.is_xlink;
    ccc_spatio_constraints.surfrecord_io.is_reverse := :new.is_reverse;
    ccc_spatio_constraints.surfrecord_io.geometry := :new.geometry;
    ccc_spatio_constraints.surfrecord_io.class_name := :new.class_name;
    ccc_spatio_constraints.surfrecord_io.class_object_id := :new.class_object_id;
  END IF;

END;


CREATE or REPLACE TRIGGER GET_SENSOR_INFO_ALL
AFTER UPDATE OR INSERT OF dateval, intval
ON cityobject_genericattrib_test
FOR EACH ROW

BEGIN
  -- Assign the new row to global variable column by column
  ccc_nonspatio_constraints.sensor_io.id := :new.id;
  ccc_nonspatio_constraints.sensor_io.attrname := :new.attrname;
  ccc_nonspatio_constraints.sensor_io.datatype := :new.datatype;
  ccc_nonspatio_constraints.sensor_io.strval := :new.strval;
  ccc_nonspatio_constraints.sensor_io.intval := :new.intval;
  ccc_nonspatio_constraints.sensor_io.realval := :new.realval;
  ccc_nonspatio_constraints.sensor_io.urival := :new.urival;
  ccc_nonspatio_constraints.sensor_io.dateval := :new.dateval;
  ccc_nonspatio_constraints.sensor_io.geomval := :new.geomval;
  ccc_nonspatio_constraints.sensor_io.blobval := :new.blobval;
  ccc_nonspatio_constraints.sensor_io.cityobject_id := :new.cityobject_id;
  ccc_nonspatio_constraints.sensor_io.surface_geometry_id := :new.surface_geometry_id;
--  dbms_output.put_line('This trigger to get newly inserted row is fired!');
END;
```

```
CREATE or REPLACE TRIGGER GET_SENSOR_PTS_INFO
AFTER INSERT ON CITYOBJECT_GENERICATTRIB_TEST
FOR EACH ROW
BEGIN
  ccc_spatio_constraints.mobile_senspts_ids.EXTEND;
  ccc_spatio_constraints.mobile_senspts_ids(ccc_spatio_constraints.mobile_senspts_ids.
      LAST) := :new.id;
END
```

After statement triggers.

Listing C.8: Building intersection detection

```
CREATE or REPLACE TRIGGER BUILDING_GEOM_UPDATE
AFTER UPDATE OF GEOMETRY
ON SURFACE_GEOMETRY

BEGIN
  -- Check if there is any interruption between buildings
  ccc_spatio_constraints.3D_building_interrupt_building;
END;
```

Listing C.9: Move exceptions-like instance to special table

```
CREATE or REPLACE TRIGGER "EXCEPTIONTEST_TREE2BLD_MINDIST"
AFTER INSERT OR UPDATE
ON SURFACE_GEOMETRY

BEGIN
  IF ccc_spatio_constraints.free_entry_flag = 'TRUE' THEN
    /*
    -- The modifying of the table that the trigger is working on
    -- may be impossible due to mutating-table restriction.
--    DELETE FROM surface_geometry_exceptions
--    WHERE id = ccc_spatio_constraints.surfid_io;
    */

    ccc_spatio_constraints.free_entry_flag := 'FALSE';

    dbms_output.put_line('The exception instance has been inserted/updated
in table SURFACE_GEOMETRY, and deleted from the exception lists.
Free_Entry_Flag is set back to 'FALSE'.');

  ELSE
    ccc_spatio_constraints.3D_tree_mindist2_building;
  END IF;
END;
```

Listing C.10: Insert the verified exceptional instance back to the table it aims to go to

```
CREATE or REPLACE TRIGGER MOVE_EXCEPTIONS_BACK
AFTER UPDATE OF ACCEPTED
ON SURFACE_GEOMETRY_EXCEPTIONS
FOR EACH ROW
BEGIN
  IF :new.accepted = 'TRUE' THEN
    ccc_spatio_constraints.free_entry_flag := 'TRUE';
```

```
    INSERT INTO surface_geometry VALUES
    (
      :new.id,
      :new.gmlid,
      :new.gmlid_codespace,
      :new.parent_id,
      :new.root_id,
      :new.is_solid,
      :new.is_composite,
      :new.is_triangulated,
      :new.is_xlink,
      :new.is_reverse,
      :new.geometry,
      :new.class_name,
      :new.class_object_id
    );

      /* The TRUE exception should stay for a further check from other trigger.*/
  END IF;
END;
```

Listing C.11: Trigger to call the check about jumping points in mobile sensors.

```
CREATE or REPLACE TRIGGER CHECK_JUMPING_SENSORPTS
AFTER INSERT
ON CITYOBJECT_GENERICATTRIB_TEST
BEGIN
  ccc_spatio_constraints.3D_mobilesensor_building;
END;
```

Listing C.12: Three made-up constraints that contradict and never can be satisfied.

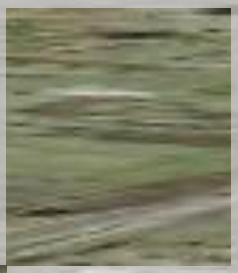
```


CREATE or REPLACE TRIGGER 3D_tree2house_maxdist
AFTER INSERT OR UPDATE OF GEOMETRY
ON SURFACE_GEOMETRY
BEGIN
  dbms_output.put_line('The 3D_tree2house_maxdist trigger fires!');
  ccc_spatio_constraints.3D_tree2house_maxdist;
END;

CREATE or REPLACE TRIGGER
"3D_road2house_mindist"
AFTER INSERT OR UPDATE OF GEOMETRY
ON SURFACE_GEOMETRY
BEGIN
  dbms_output.put_line('The 3D_road2house_mindist trigger fires!');
  ccc_spatio_constraints.3D_road2house_mindist;
END;

CREATE or REPLACE TRIGGER 3D_tree2road_maxdist
AFTER INSERT OR UPDATE OF GEOMETRY
ON SURFACE_GEOMETRY
BEGIN
  dbms_output.put_line('The 3D_tree2road_maxdist trigger fires!');
  ccc_spatio_constraints.3D_tree2road_maxdist;
END;
```

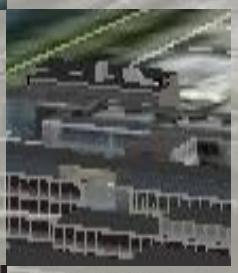
COLOPHON

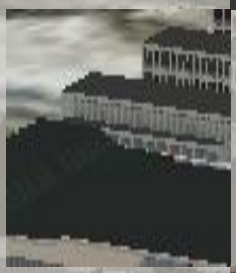
This thesis was typeset with LaTeX on Windows 7 using TeXnicCenter as the front-end, and BibTeX for bibliography management.

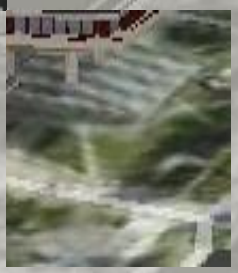
The typographic style was inspired by Robert Bringhurst's genius as presented in *The Elements of Typographic Style*. This style was implemented by André Miede and is now accessible via CTAN (`classicthesis`).

The UML diagrams were generated by *Enterprise Architect*. Figures showing 3D models were created in *Google SketchUp*. And the code listings were typeset to adjust to the coding style in *Oracle SQLDeveloper* environment.









Supervised by Prof. dr. Peter van Oosterom and Dr. Sisi Zlatanova.

Department of GIS Technology
OTB Research Institute for the Built Environment

Delft University of Technology
Jaffalaan 9
2628BX Delft
The Netherlands

Abstract

Nowadays the field of geo-information is undergoing major changes, and the transition from 2D to 3D is having a major influence. A significant amount of 3D datasets are stored in the database. Experts are aware that new quality control mechanisms need to be built into the database systems in order to secure and guarantee high-performing data.

Constraints are effective in providing solutions needed to avoid errors and enable maintenance of data quality. Whereas constraints for 2D geographic datasets have already been the subject of several research projects, studies into 3D geo-data constraints are largely unexplored. This thesis research discovers a new approach to model, conceptualise and implement 3D geo-constraints which can function in the database. At the outset, constraints can be formulated using natural language. As natural language is subjective and varies between individuals, expressions can be ambiguous and can easily cause confusion. So spatial constraints are abstracted using geometry that depicts the exact shape, and also topology that reveals the spatial relationship between geometries. This step makes the meaning of a constraint clearer to others. Furthermore, using standardised UML diagrams and OCL expressions, geo-constraints can be formalised to an extent that not only humans, but also machines can understand them. With model-driven architecture supported by various software, OCL expressions can be automatically converted to other models/executable codes (e.g. PL/SQL) just by a few clicks. And with small modifications, database triggers can be formulated to carry out constraints check.

A database including various topographic objects (e.g. buildings, trees, roads, grass, water-bodies and terrains) is used as a study case to apply the discovered approach. During this research, a first attempt to formulate 3D geo-constraints in OCL has been made. These expressions can be tested and translated to other models/implementations when the OCL standard is extended with spatial types and operations.

In the implementation stage, the current 3D functions in Oracle Spatial database are found to be insufficient. A new 3D function using existing 2D functions - plus additional code relating to computational geometry - has been developed by the author to bridge the gap. Based upon this function, a large group of spatial constraints which apply to objects in 3D space can be checked.

Bentley Map and Python IDLE are used to test the performance of constraints as well as the visualisation of warning messages to clients. Database error messages are immediately displayed on the front-ends when a modification that does not satisfy a constraint is attempted to commit to the database.

During the case study, new classes of constraints are also discovered. They are higher-level constraints, parameterised constraints, constraints allowing exceptional instances, extra-check rules to detect conflicting constraints and constraints relating to multi-scale representations.