Constraints in models and implementations of (VR) geo-info systems

Jildou Louwsma (1), Sisi Zlatanova (2), Ron van Lammeren (3) and Peter van Oosterom (4)

(1) Waterschap Roer en Overmaas, Parklaan 10, 6131 KG Sittard, The Netherlands (NL)
Phone: +31 46-4205700, Fax: +31 46-4205701, Email: j.lousma@overmaas.nl
(2) Delft University of Technology, Section GIS-technology, Jaffalaan 9, 2628 BX Delft, NL
Phone: +31 15-2782714, Fax +31 15 2782745, Email: s.zlatanova@otb.tudelft.nl
(3) Wageningen University, Centre for Geo-Information, PO. box 47, 6700 AA Wageningen, NL
Phone: +31 317-474206, Email: ron.vanlammeren@wur.nl
(4) Delft University of Technology, Section GIS-technology, Jaffalaan 9, 2628 BX Delft, NL
Phone: +31 15-2786950, Fax +31 15 2782745, Email: oosterom@otb.tudelft.nl

***Abstract**: Constraints are important elements of every modelling process but until now they have not received much attention in GIS. In GIS, constraints are conditions, which always have to be valid (true) within the model populated with real geographic object instances. In this paper we argue that constraints should be part of the object class definition, similar to other aspects of the object class definition: attributes, methods, and relationships (generalization/specialization, part/whole, and associations). This paper presents our view on modelling constraints, in which we distinguish three different stages: clarifying constraints applicable for the objects of interest, formally describing them using the unified modelling language/ object constraint language (UML/OCL) and implementing them in application tool or in the DBMS. In order to better understand the constraints and their use in GIS we propose a classification of the different types of constraints. The paper demonstrates the way UML (OCL) is used for modelling constraints in SALIX-2, a VR landscape modelling system. Depending on the type of constraints (and the application), some of them can be best implemented at (edit/simulation) front-end, others at DBMS (or perhaps throughout all subsystems), but this should all be derived from the same specification: the model including the formal constraints. This paper focuses on the implementation of the constraints using triggers (as assertions are not yet available) within the DBMS and the problems that have to be solved.*

Keywords: object constraints, object constraint language, unified modelling language, virtual reality, geo-information systems, landscape architecture, 3D objects

1. Introduction

In this paper we discuss the important, but often ignored role, of constraints when modelling geo-information and realizing their implementation in systems. Constraints are conditions, which always have to be fulfilled by the user of a system and by the data processing operators of the system. This implies that ‘educating’ either the user or the system helps to fulfil constraints. Indeed, the second option is the challenging one. Unfortunately, until today constraints have not received much attention in GIS and accordingly not in Virtual Reality (VR) based GIS. Especially the link with the VR-environment puts forward new questions considering the role and implementation of constraints.

Constraints can be defined and expressed in different ways, can be related to properties of the object itself and to relationships between the objects. One typical example of a constraint considering relationships between two objects is a ‘*Yucca* tree must never stand in the water’. A constraint ‘a tree must be always green’ reflects only properties of one object. Such constraints need a formal description and definition. In this paper we argue that the constraints should be part of the object class definition, similar to other aspects of the object class definition like attributes, methods, and relationships (generalization/specialization, part/whole, and associations). Finally, the constraints can be implemented at various levels; e.g. at application level, data exchange level, or at database level. Currently, in VR systems, many constraints are implemented as behaviour of objects (and thus maintained in the application). For example the constraint ‘two trees (objects) can not grow on the same location’ can be realised by collision detection, a well-known computer graphics technique. The variety of constraints however can vary significantly and may need more formal organisation. In this paper, we will investigate the possibilities to manage constraints in DBMS.

This paper elaborates on constraints with respect to the requirements of a VR system for landscape modelling (SALIX-2). Using this system a user can interactively introduce new objects (i.e. plant trees, bushes, etc.) in a 3D landscape. As it is also the case in reality, sometimes new objects (trees, bushes) have to be at a certain distance from each other (e.g. two trees have to be planted not closer that 3 m), or form other objects (e.g. a number of trees in a certain pattern together forming an higher level landscape architecture object), or are even not allowed to be planed in a given area (e.g. a tree on a road).

The paper is organised in six sections. Section 2 discusses the goal and current functionality of the SALIX system and motivates its extension towards maintenance of constraints. Section 3 and 4 elaborates on conceptual framework for classifying and defining constraints. Section 5 discusses the implementation of constraints in DBMS. The last section concludes on the results and outlines further research.

2. The SALIX System

Digital supported landscape architectural design has some intriguing challenges. From a digital perspective these challenges have to do with modelling the changes in time of the architectural primitives (mainly trees and shrubs) and modelling the relation between architectural objects, their architectural primitives and their spatial configuration in the layout of the ground plane. Nowadays we are confronted with commonly available virtual reality (VR) tools like VR - construction sets and VR-viewers, which do have the opportunities to experiment with a wide arrangement of design proposals on a geo-data based real-world representation. Such (VR) geo-information systems promise a three-dimensional laboratory to experiment with landscape architectural design proposals.

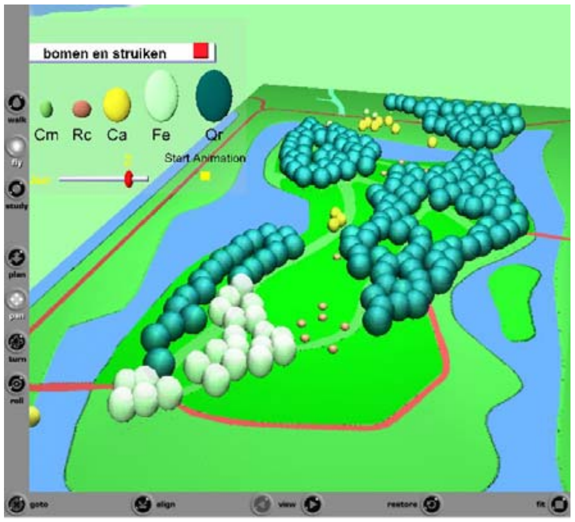

Figure 1: 3D scene of SALIX-2: an interactive landscape modelling system

Based on these opportunities and intentions the SALIX-2 system has been developed. SALIX-2 is a simulation program, which is intentionally meant for students of the study landscape architecture of Wageningen University, the Netherlands. On the other hand the system has been also developed for experimenting with VR-based geo-information systems. Especially the so-called seven factors of virtual reality (Wachowicz et.al., 2001) are experimented by the system. There are several reasons to explore the opportunities of SALIX-like applications in design studios to find how the seven factors will influence learning attitudes and results of design classes and studios. For example what effect will have the '1:1 scale' navigation, the simulated growing process, the alternating shift between planting and architectural objects and the quick transition between different views and projections have on the spatial thinking and reasoning of landscape architectural students.

Elements of VR-scene manipulation are object position changes of the relative position of objects e.g. making it possible to interact with a virtual object (Heim 1998) by which an object (or its attributes) in the scene can be deleted or added. With respect to SALIX-2 the underlying idea of the system is a virtual environment for simulations of the growth of plantation objects (bushes and trees). The students are able to plant interactively bushes and trees but (similarly to the real world) that have to be restricted from planning on particular areas (e.g. trees have not to be planted in the middle of the road). For that reason the system has to be provided with constraints related to the type of plantation objects and the type of geo-information objects.

In modelling, three different stages can be distinguished: clarifying constraints applicable for the objects of interest, formally describing them using modelling language and implementing them either in application tool or in the DBMS. The first stage is very much application-oriented. However, in order to better understand the constraint and their use, it is important to classify them, including their spatial/dimensional aspects (see Section 3). The specification of the constraints has to be easily accessible and intuitive for the user. In the second stage, constraints have to be included in the object model and if possible this model should be as formal as possible. Though UML class diagrams more or less constitute the 'default' approach when creating formal knowledge frameworks, the graphic diagram has limited semantic accuracy. A non-graphic language is provided within UML for the further modelling of semantics (knowledge frameworks) with the aid of the Object Constraint Language (OCL); see Section 4. Depending on the type of constraints (and the application), some of them can be best implemented at (edit/simulation) front-end, others at DBMS (or perhaps throughout all subsystems). This paper focuses on the constraints to be implemented within the DBMS. Since SQL92 the 'general constraints' (assertions) are part of this SQL standard and they could be used to implement the OCL constraints.. However, assertions are not supported in the currently available DBMSs and developers are referred to the use of triggers and procedures. The paper presents the implementation of constraints for SALIX-2 in DBMS using triggers (see Section 5). Once the system is 'educated' to operate with the constraints, the user should be informed what kind of constraints are available. For example, it can be a simple list with all the maintained constraints or a more sophisticated attempt-alert approach in the interactive edit environment.

.

3. Classification of constraints

Classification of the different types of (spatial) constraints will show up a complex taxonomy. In general, such a classification can be done considering different characteristics:

1. The number of involved objects/classes/instances. The constraints can be related to one instance (restrictions on attributes and relation between attribute values of a single instance) or to multiple instances of the same class, or to multiple instances from different classes;
2. The properties of objects and relationships between objects: metric (distance or angle between objects), topologic (neighbourhood or containment), temporal, thematic or mixed.
3. The dimension (2D or 3D or mixed time and space, that is, 4D).
4. The manner of expression: 'never may' (bush never may stand in water) or 'always must' (tree must always be planted in open soil);
5. The nature of the constraint can be 'physical impossible' (tree can not float in the air) or 'design objective' (bush should be south of tree).

Practically the manner of expression has value only for communicating the constraints between the users. Once the object and the constraints are formally defined the expressions 'never may' and 'always must' can be represented by one constraint, e.g. by the one that is more efficient from implementation point of view. For example, the constraint 'a tree never may be in water, or street or house' is equivalent to the constraint 'a tree always must be in garden, or park' under the assumption that all the possible objects on the ground are only five: water, street, house, garden and park. Apparently, the second expression is more efficient, because requires consideration of only two classes. The most interesting classification is the one based on the properties of the objects and the relationships between them. We distinguish between constraints:

- Related to properties or (attribute) state of objects: thematic, temporal and spatial.
- Based on (spatial) relationships between objects: directional, topological, metric, and quantitative. Examples of constraints based on spatially extended relations can be:
  1. *Direction*: The trees should be always south of paving polygons, so people can walk in the sunshine.

2. *Topology*: No trees and bushes inside water polygons; No trees and bushes inside paving polygons
3. *Metric:* No trees inside the water, except if < 1 meter from edge of water bushes > 1 meter from paving (so the leaves do not overlap with the paving)
4. *Quantity:* Maximum number of 10 plantation objects in a specified area in the centre of the park.

Note that constraints based on relationships can also involve non-spatial predicates such as temporal (e.g. the second object may only occur after first object) or thematic ones (e.g. a parcel must always be owned by a person). The formalisation in such constraints is closely related to the formalisation of the objects. Mechanisms for defining constraints can be the same as for describing relationships between objects.

Direction constraints are to be based on formalism for *directional* relations (Papadias et al. 1999). The directional relations are defined as the position of an object in comparison to another object, as the directions can be given in degrees in the range of [0°, 360°] or in verbal expressions (Northeast, North, Northwest, West, Southwest, South, Southeast, East). Actually the second approach is generalisation of the first one since each expression stands for an interval of degrees. Algorithms are also developed to assign the right direction to an object.

Topological constraints are to be constructed using frameworks for neighbourhoods (Egenhofer, 1989, Clementini et al., 1993) The 9-intersection topology model, recommended also in OGC/SFS standard (OGC, 1999) utilises the fundamental notions of general topology for topological primitives to investigate the interactions of spatial objects. The basic criterion to distinguish between different relations is the detection of empty and non-empty intersections between interior, exterior and boundary of objects. The number of detectable relations between two objects can be theoretically 512, but only a small part of them a possible in reality. Eight relations are given names, i.e. *disjoint*, *meet*, *contains*, *covers*, *inside*, *covered by*, *equal* and *overlap*. For example, if the boundaries of the two objects intersect but the interiors do not, then the conclusion is that the objects *meet*. The constraint 'no bush in a water' can be translated in 'no point-in-polygon' (assuming that a bush is represented as point and water with a polygon), which corresponds to topology relationship *inside*.

Distance constraints similarly to distance relations impose a constraint on a distance between objects. In can be expressed in linear meters, or by more approximate measures such as 'closer than', 'further than' or 'interval distance'.

Specifying a certain density of objects in a certain area is only implicitly related to spatial relationships. Knowing the distribution of objects on a certain area, the minimum distance between two objects can be computed and, eventually, the approach for distance constrains can be used. From a user point of view, however, a more intuitive approach will be specifying a number per given area. This constraint can be given as a minimum, exact or maximum number of objects related to an area surface (density) or not. Examples of density constraints can be a maximum number of houses in a residential area or the minimum number of trees in that area. Examples concerning exact number of objects can be: one tower, three benches and one statue must be placed in a 3D model. This exact number of certain objects can be seen as a special case of a density constraint, because it can be defined as an exact number of certain objects for the whole area (that is, the area in the 3D model).

Thematic information about objects can be found in the attributes (e.g. house, road, grass). Some objects of the same type have relations to objects of another type, e.g. all objects that are *houses* have a relationship with all objects that are *roads*. Real world thematic relations between objects can thus be used to formulate constraints to make the virtual world more look like the real world, e.g. houses must be placed near roads to be accessible, like in the real world.

Temporal constraints are to be specified on the basis of frameworks for describing temporal relationships between objects (Peuquet, 1995). Kwon et al. (1999) described the temporal relations between two time intervals. Given two time-intervals, there are seven distinct ways in which these time-intervals can be related (e.g. Before, Meets, Overlaps, Finishes, During, Starts, Equals). The relations can concern two time intervals or can be seen as relations between two objects with some time interval as existence time (with start and end time of existence as the boundaries of the time interval).

Spatial constraints can be associated with spatial properties of one object such as size or shape. An example of size constraint can be 'a tree should not become higher than 30m' and a constraint on shape could be 'a bush must be represented with a sphere'.

The last aspect to be discussed here is related to the dimension. In general all the different kinds of spatial relationship constraints can be specified for both 2D and 3D objects. Constraints can concern the 2D ground plane or the 3D objects (bushes and trees) that could be placed on the ground plane to create a spatial configuration. The rules concerning the ground plane find their origin in the policy for a certain area and in the fact that some destinations conflict with each other. The policy makers define per area destinations and write them down in plans. For example, some areas get an urban destination, some a rural destination and some parts are pointed out as agricultural areas. These destinations of the ground plane can easily be stored as an additional attribute of the separate polygons. However the defined restrictions by the policy makers could still conflict. E.g. a road can never lie in water (except when a bridge or tunnel is build), a forest never lies on a major road and all houses should be reachable by a path or road. On the other hand, such conflicting constraints could be a source for strategically spatial decision making too. The rules for 3D objects can even be more complex.

The system SALIX-2 system maintains currently three classes objects: *threes*, *bushes* and ground surface. The possible ground surfaces are *water*, *paving*, *soft_paving*, *grass* and *bridge*. The possible types of threes and bushes for planning are five (CorAve, CorMAs, FraxExc, QueRob RosCAn). Examples of rules for the position of objects in geo-VR environments can be; a tree (object a) must not overlap with water (object b) or a tree (object a) must be covered by a polygon with destination forest (object b). For these constraints it is logical to represent a tree as a circle (an extended object) and not as a point (centroid). Table 1 shows some examples of constraints for SALIX-2.

Table 1: Examples of constraints for SALIX-2

Type of relation	**Constraints formulated with forced relations between objects**
Direction	A bush always has to be placed south of a tree (1 value
Topology	Bushes always have to be disjoint or meet water (2 values) A bush always has to meet or be disjoint paved areas (also thematic constraint) (2 values)
Metric	Trees always have to be positioned > 1 meter from paving (1 value)
Temporal	An oak always grows for 70 years (1 value)
Quantity	There must always be at least 10 trees on the specified ground surface (1 value)
Thematic	A bush always has to meet or be disjoint paved areas (also topological constraint) (2 values)

Complex	The distance between trees inside water always is > 8 m AND the distance between the tree and the edge of the water always has to be < 0,5 meter AND the species must be a salix; Trees of type 1 always have to be placed west of trees of type 2 AND the distance between trees of type 1 and trees of type 2 must always be 7 meters (pattern).

4. Formal description of constraints

In the second stage, constraints have to be included the object model and if possible this model should be as formal as possible in order to be able to derive constraint implementations within the different subsystems (edit, store, exchange). Formal modelling is an essential part of every large project, but it is also very helpful in small and middle size projects. Using a formal model allows communicating ideas with other professionals as well as clear, unambiguous view on implementation strategies. Surveys have shown that large software projects have a high probability of failure if formal approaches are overlooked.

Here the Unified Modelling Language (UML) will be used being a standard for object-oriented modelling (OMG, 2002, Chapter 3). UML is graphic language, which gives a wide range of possibilities for representing objects and their static and dynamic relationships. In general the language can be used for modelling business processes, classes, objects and components, as well as for distribution and deployment modelling. UML consists of diagram elements (icons, 2D symbols, paths, strings), which can be used in nine different diagrams. The most appropriate diagram for describing static constrains is the class diagram. It provides formalism for describing objects/classes (with their attributes and behaviour) and relationships between the objects. Six relationships can be defined in the class diagram among which the most used ones: association, generalisation and aggregation.

Dynamics of objects (temporal constraints) can be modelled using one of the interaction diagrams, i.e. sequence or collaboration diagram. While the first one allows the explicit sequence of events to be described (with respect to time), the second one shows the interaction organised around the roles in the interaction and their relationships (thus no explicit in time). The collaboration diagram would applicable for designing interface software (e.g. communication between user-system when a user attempts to plant a tree on a forbidden area). Since the dynamic aspect is outside the scope of this paper, further details will not be discussed.

Despite the wide possibilities for formalizing object and processes, UML class diagrams are typically not refined enough to provide all the relevant aspects of constraints. Constraints, as shown above, are often initially described in natural language. Practice has shown that this always results in ambiguities. In order to write unambiguous constraints, a non-graphic language is provided within UML for the further modelling of semantics (knowledge frameworks), i.e. the Object Constraint Language (OCL) (OMG, 2002, Chapter 6). OCL is a pure expression language. When an OCL expression is evaluated, it simply returns a value. This means that the state of the system will never change because of the evaluation of an OCL expression.

The advantage of using OCL is that – as in the case of UML class diagrams – generic tools are available to support OCL (i.e. not GIS-specific). The context of an invariant is specified by the relevant class; e.g. the object class ‘parcel’ if the constraint were that ‘the area of a parcel is at least 5 m^2’. It is also possible within a constraint to use the association between two classes (e.g. every instance of the object class ‘parcel’ must have at least one owner, which could be depicted as an association with the class ‘person’).

OCL enables at least one to describe expressions and constraints on object-oriented models and other object modelling artefacts.OCL enables description of expressions and constraints (on models) as well as other object modelling artefacts. In this context, an *expression* is an indication or a specification of a value and a *constraint* is a restriction on one or more values of (part of) an object-oriented model or system.

In OCL expressions can be used in a number of places in a UML model to specify values and roles (e.g. an initial value of an attribute (or association end), a derivation rule of an attribute, the body of an operation) in a class diagram or to indicate instances, parameters or conditions in dynamic diagrams. Four general types of constraints can be defined in OCL:

- An *invariant* is a constraint that states a condition that must always be met by all instances of the class, type or interface. An invariant is described using an expression that evaluates to true if the invariant is met. Invariants must be true all the time.
- A *precondition* to an operation is a restriction that must be true at the moment that the operation is going to be executed. The obligations are specified by post-conditions.
- A *post-condition* to an operation is a restriction that must be true at the moment that the operation has just ended its execution.
- A *guard* is a constraint that must be true before a state transition fires.

Each OCL expression has a *context definition*, which specifies the model entity for which the OCL expression is defined. Usually this is a class, interface, data type or component. Sometimes the model entity is an operation or attribute, and rarely it is an instance. It is always a specific element of the model, usually defined in a UML diagram. This element is called the *context* of the expression. In OCL the standard data types are integer, real, string and boolean. The operations include the standard operations, like +, -, /, *, and, =, <>, but also the booleans *implies* and *if-then-else*-operators. With these operators, more complex constraints can be formulated. The formal representation of context is:

```
context Typename::operationName(param1 : Type1, ... ): ReturnType
pre : param1 > ...
post: result = ...
```

The context of an invariant is specified by the relevant class; e.g. ‘parcel’ if the constraint was that ‘the area of a parcel is at least 5 m2’. It is also possible within a constraint to use the association between two classes (e.g. ‘parcel’ must have at least one owner, which is an association with the class ‘person’). Below are two examples in UML/OCL syntax (keywords in bold print):

```
context Parcel inv minimalArea:
  self.area > 5

context Parcel inv hasOwner:
   self.Owner -> notEmpty()
```

The word ´self´ is used to specify the context and is left out in many cases, because it is obvious what is meant by the constraint without this word. The UML (OCL) models should be the foundation for the edit/simulation environments, the storage data models (further described in the DDLs of the DBMS) and the exchange data models. The eXtensible Markup Language (XML) can be used for the models containing the class descriptions at class level (XML schema document ‘xsd’) and for the data at object instance level (‘normal XML document with data 'xml'). XML documents also include the geometric aspect of objects (e.g. LandXML, GML, X3D). Figure 2 shows the UML class diagram with the objects and the constraints (depicted as associations) used for SALIX-2 (as described in Section 2).

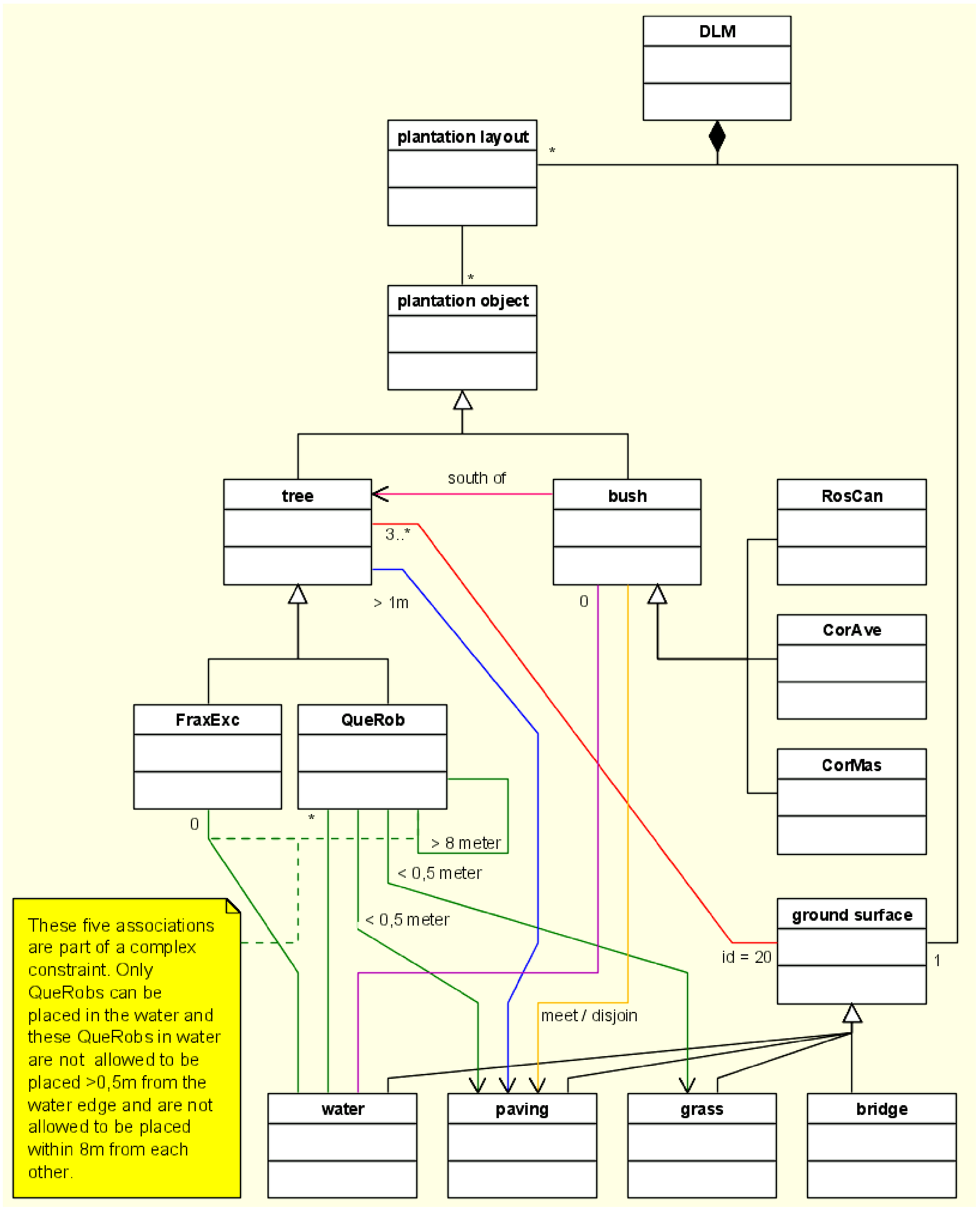
DLM
plantation layout
*
plantation object
*
tree
south of
bush
RosCan
3..*
> 1m
0
CorAve
FraxExc
QueRob
CorMas
0
*
> 8 meter
< 0,5 meter
< 0,5 meter
ground surface
id = 20
1
These five associations are part of a complex constraint. Only QueRobs can be placed in the water and these QueRobs in water are not allowed to be placed >0,5m from the water edge and are not allowed to be placed within 8m from each other.
meet / disjoin
water
paving
grass
bridge

Figure 2: UML class diagram representing the objects of interest in SALIX-2

5. Implementation of constraints in DBMS

As mentions before, the constraints can be implemented either with the application (front-end) or data exchange level or at database level. All approaches have advantages and disadvantages.

The apparent benefit of front-end implementation is the direct interaction with the user. If a user places a plantation object in the VRML scene, fast feedback of the validity of this placement can be realised if the constraints are also maintained in the visual environment (e.g. the VR component of the SALIX-2 system). However, the possibilities of changing the constraints within the VR-environment are limited, because the constraints are 'hard-coded' in the VR application code (and not managed centrally). In future development environments it should be possible to automatically generate the part of the VR program application code that implements the constraints (as specified in UML/OCL).

Database implementation offers better management of constraints. If the constraints are stored in a database, they are stored in a central place, easily accessible and therefore easily adaptable. However, the VR application only connects to the database when saving or loading a planting plan (such as with SALIX-2), there is not a connection during the interactive creation (editing) of the plantation plan. So the user gets only feedback when the plantation plan is saved, not when the plantation object is placed.

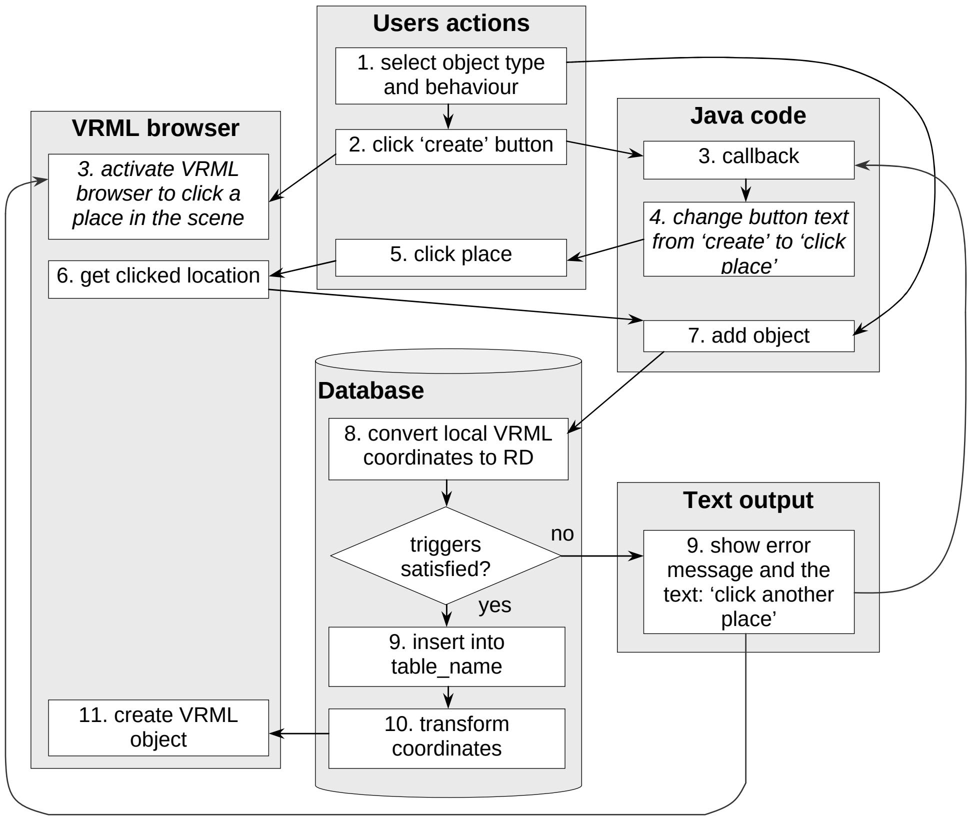
Users actions
1. select object type and behaviour
2. click 'create' button
5. click place
VRML browser
3. activate VRML browser to click a place in the scene
6. get clicked location
11. create VRML object
Java code
3. callback
4. change button text from 'create' to 'click place'
7. add object
Database
8. convert local VRML coordinates to RD
triggers satisfied?
no
yes
9. insert into table_name
10. transform coordinates
Text output
9. show error message and the text: 'click another place'

Figure 3: Flowchart showing the interaction with the system SALIX-2 (with triggers)

A combination of these two can benefit from both pros, i.e. storing the constraints in a database on a central location and encapsulating this information in the application code. Using this approach the feedback of the system will be significantly improved. However, the constraints are stored twice, which may violate the consistency. Therefore it is very important that the implementation of the constraints in the VR-environment is automatically derived and consistency is guaranteed.

5.1 Assertions in SQL92

The following text will discuss the possibilities for implementation at DBMS level. There are two possible ways for implementing constraints: by assertions and by triggers. The syntax of an assertion (‘general constraint’) is

```
            CREATE ASSERTION <assertion_name> CHECK <constraint_body>
```

The syntax of an assertion is quite simple and straightforward. When an attempt is made to commit the changes in a database, after a set of updates, insets, and deletes, the assertion is checked and if the expression evaluates ‘true’ than the commit succeeds, otherwise it fails and the database remains in the old state (for which the expression was also ‘true’). One could easily imagine automatic generation of these assertions from the UML/OCL invariants in a way similar to which database table definitions (DDL) can be derived from UML class diagrams. Assertions are supported in the standard SQL92 (Date, 1997), but unfortunately they are not yet implemented in any of mainstream DBMSs (Oracle, DB2, Ingres, Informix, PostgreSQL, MySQL). We will now formulate a number of example constraints as assertions. The tables of SALIX-2 that are used in the assertions are: prcv_treesrd_point (plantation objects of type trees and bushes) and prcv_gvkrd_poly (ground surface with description water, paving, soft_paving, grass, and bridge):

Bushes never lie inside water:

```
  create assertion constraint_1 check (not exists (
  select * from prcv_treesrd_point t, prcv_gvkrd_poly g
   where t.treetype in (‘CorAve’, ‘CorMas’, ‘RosCan’)
     AND g.descript = ‘water’
     AND sdo_relate (g.geom., t.geom., ‘mask=inside, querytype=window’) =’TRUE’))
```

A bush always has to be placed directly south of a tree:

(For this assertion a geometry that represents the restricted area is necessary, this geometry must first be created and this can be done with a function. Here the name ‘fu_restricted_area’ is used and the inputs of the function are two angles that represent the direction and a maximum distance to restrict the search area. The function returns a geometry which has it’s basis in the location of the involved bush and can be used in the assertion.)

```
  create assertion constraint_2 check (exists (
  SELECT * FROM prcv_treesrd_point t, prcv_treesrd_point b
   WHERE t.treetype IN ('FraxExc', 'QueRob')
     AND b.treetype IN (‘CorAve’, ‘CorMas’, ‘RosCan’)
     AND sdo_relate (t.geom,
         fu_restricted_area(first_angle, second_angle, distance, b.geom),
         'mask=ANYINTERACT, querytype=window')='TRUE'))
```

Trees always have to be positioned > 1 meter from paving:

```
  create assertion constraint_3 check (not exists (
  SELECT * FROM prcv_treesrd_point t, prcv_gvkrd_poly g
   WHERE t.treetype IN (‘FraxExc’, ‘QueRob’)
     AND g.descript IN (‘paving’, ‘soft_paving’)
     AND sdo_within_distance (g.geom., t.geom., ‘distance=1’) =’TRUE’))
```

There must always be at least 3 trees on a specified ground surface
(for this constraint the grass polygon with id 20 is used).

```
  create assertion constraint_4 check ( (
  SELECT count(t.treeid) FROM prcv_treesrd_point t, prcv_gvkrd_poly g
   WHERE t.treetype IN (‘FraxExc’, ‘QueRob’)
     AND g.id=20
     AND sdo_relate(t.geom, g.geom, ‘mask=ANYINTERACT, querytype=window’)=’TRUE’) >=3)
```

5.2 Triggers and procedures

The assertions look nice and are relatively easy to specify, but as mentioned above, not supported in the mainstream DBMSs. Therefore, currently triggers offer the only practical way to implement constraints. Triggers can be seen as small programs that check certain conditions and give an alert with respect to the condition. Using triggers one can basically achieve the same effect as by defining assertions. However, one has to develop the code for the triggers (and the used procedures) oneself. This is more labour intensive compared to specifying the more highlevel assertion, but also gives additional functionality. For example, database triggers can be before triggers or after triggers. Triggers can be row triggers or statement triggers. The order of execution triggers is of critical importance for DBMS. When there are two or more triggers of the same type the order of execution of these triggers is arbitrarily. In case of constraints it doesn’t matter in which order they are checked, because they do not influence each other. The syntax of creating the trigger is:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
BEFORE | AFTER
INSERT OR UPDATE [<column(s)>] OR DELETE ON <table_name>
[FOR EACH ROW [WHEN (condition)]]
<trigger_body>
```

In order to avoid the ‘low level’ hand-coding of constraints (or business rules), Oracle provides a development tool called Custom Development Method (CDM) for automatically generating this code for the DBMS (Muller, 2000, Boyd, 2000). The CDM RuleFrame is the business rules implementation framework of CDM. The rules consist of three parts (Muller, 2000):

- a function that indicates when the rule should be validated;
- a function that performs the actual validation, when the previous function indicates the need;
- a handling procedure, that manages the communication with the outside world.

CDM RuleFrame does not check business rules at the moment the user performs insert, update or delete statements. Rather, CDM RuleFrame stacks the rules that have to be enforced and checks them only at the moment of commit. This stack of the rules and the eventual business rule enforcement is done in the Transaction Management component.

5.3 Our implementation of constraints

For the time being, we decided to convert and hand code our constraints in SALIX-2 and use the Oracle before and after triggers. One before each row trigger is sufficient to implement the constraints at once. First the temporary table is updated followed by the constraint checking. For each constraint checking a stored procedure is used. The disadvantage of using only one trigger for the constraint implementation is that the user does not have the possibility to enable or disable separate constraints (as database triggers). This implies that the interaction possibilities are currently rather limited in our simple prototype implementation, but using only one trigger is less complex than using more.

The constraint checking is organized as follows: For each insert or update statement of one object in the configuration table (with the planted trees and bushes), a before each row trigger is invoked. This trigger first inserts all new attribute values of the involved object in some package variables. These attributes are easily accessible and can be used in all procedures and functions. All procedures and functions are stored in one package (named ‘salix’). A check takes place if the involved object is a bush and then all constraints concerning bushes must be checked (using functions and procedures). If the involved object is a tree, the constraints concerning bushes are out of interest and all constraints concerning trees must be checked (also using functions and procedures). In addition, all constraints that cannot have the involved object as starting point, these are checked in the after statement trigger (for example in our case the quantity constraint concerns a specific ground surface polygon).

A partial example (‘bush must not stand in the water’) of an implemented constraint in a DBMS using triggers and procedures is described in box 1. Within the front-end application a new object is created and an insert statement is sent to the DBMS. The statement is checked for constraints and feedback to the user is given through DBMS outputs. This output can be used by the front-end side. The constraint checking is done using triggers, packages, procedures and functions; all standard functionality in DBMS’s. Within the procedures also more complex (spatial) constraints can be checked using PL/SQL; e.g. based on the functionality of Oracle spatial.

Box 1: Example PL/SQL code for triggers and procedures to implement constraints

```
CREATE OR REPLACE TRIGGER brt_all_salix
BEFORE INSERT ON prcv_treesrd_point
FOR EACH ROW
BEGIN
   -- The treeid and the treetype of the involved object in the insert
   -- statement are saved in a package.
   -- These attributes can be used in the triggers, procedures and
   -- functions. For the constraint checking in SALIX-2 they are
   -- used to determine if the object is a tree or a bush and
   -- only the corresponding constraints are checked
   pck_salix.treeid_io := :new.treeid;
   pck_salix.treetype_io := :new.treetype;
END;

CREATE OR REPLACE TRIGGER ast_salix
AFTER INSERT ON prcv_treesrd_point
BEGIN
   -- constraint 1: a bush must never be placed inside water the existing
   -- table with all objects has no bushes inside the water, so only
   -- after each update or insert a check has to take place if
   -- the new location of a bush is inside water. This can be
   -- an after statement trigger.
   IF pck_salix.treetype_io IN ('CorMas', 'RosCan', 'CorAve') THEN
      -- the object is a bush, so all constraints concerning
      -- bushes must be run.
      DBMS_OUTPUT.PUT_LINE('the involved object is a bush');
```

```
            pck_salix.pr_topology_c1;


         ELSE raise_application_error(-20099,'the object is a tree,


           no constraints have to be checked.');
         END IF;
      END;

      CREATE OR REPLACE PACKAGE pck_salix
      IS
         treeid_io NUMBER;
         treetype_io varchar2(15);
         -- procedure to check constraint 1:
         -- a bush can never be placed inside water
         procedure pr_topology_c1;
      END pck_salix;

      CREATE OR REPLACE PACKAGE BODY pck_salix AS
      ------
      -- procedure pr_topology_c1: a bush can never be placed inside water
      ------
      PROCEDURE pr_topology_c1
      IS
         description varchar2(15);
         xrd_io number;
         yrd_io number;
         bush_in_water EXCEPTION;
      BEGIN
         SELECT g.descript, t.geom.sdo_point.x, t.geom.sdo_point.y
         INTO description, xrd_io, yrd_io
         FROM prcv_gvkrd_poly g, prcv_treesrd_point t
         WHERE t.treeid = pck_salix.treeid_io
         AND sdo_relate(g.geom, t.geom, 'mask=anyinteract, querytype=window')
            ='TRUE'
         GROUP BY g.descript, t.geom.sdo_point.x, t.geom.sdo_point.y;

         IF description = 'water' THEN
            raise bush_in_water;
         ELSE DBMS_OUTPUT.PUT_LINE('1: the bush is not placed in water');
         END IF;

      EXCEPTION
         WHEN bush_in_water THEN
            raise_application_error (-20001,
            '1: The bush (x='||to_char(xrd_io)||', y='||to_char(yrd_io)||') is
                placed inside water, but a bush may never be placed in water.
                Place the bush on another location.')
      END pr_topology_cl;
      END;
```

6. Conclusions

6.1 Obtained results

Constraints did not get a lot of attention in VR GIS, despite the serious need for it, especially for constraints between objects. In this paper we proposed a classification of the constraint types relevant in a (VR) GIS based on five aspects: 1. number of involved objects/classes; 2. properties of objects and/or relationships between objects: metric, topological, temporal, quantity, thematic or mixed; 3. dimension (2D or 3D or mixed time and space, that is, 4D); 4. manner of expression 'never may' or 'always must'; and 5. nature of the constraint 'physical impossible' or 'design objective'. We analyzed and defined a number of different types of constraints for our SALIX-2 system, a VR GIS/simulation for landscape architecture.

The formalisation of (spatial) constraints must be specified in UML/OCL. From this single specification, implementation of different parts of the system should be automatically derived (in the future as current development environments are not yet capable to support this). The implementation should be realized in the front-end (to allow users to get direct feedback during editing), at database site (to make sure that only valid data is stored and accepted) and during the communication or data exchange in case of loosely coupled clients and servers (to make sure the client is aware of the constraints). Database assertions do seam quite close to the UML/OCL invariants and it is therefore rather disappointing that these assertions are not yet available in mainstream DBMSs. Currently the implementation of constraints has to be realized by triggers (and procedures). In this way we implemented with success the different types of constraints for our SALIX-2 database.

6.2 Further investigations

The first step should be the development of a good checking mechanism whether the specified constraints (as part of the object descriptions) conflict each other or not. In this research only a check beforehand (in the conceptual phase) is done. When users can change the constraint definitions when the application is already created, also a conflict check should take place when constraints are changed. This aspect is also of interest for existing software. For example topology constraints can be implemented in the ESRI software (geodatabase), but there is no check whether the constraints conflict each other or not (not beforehand and not while running the software). This can result in an infinite loop.

For real interactive applications a user must be able to change, delete or make new constraints, therefore finding a possibility to make interactively constraint definitions is desirable. This is closely related to modelling. Look for example to the UML class diagram with all relevant object classes and their (restricted) relationships.

A visual feedback should be given by the user interface during editing; e.g. red or green areas during an insert. These areas should be derived from constraints and the instance geometry in a DBMS based on spatial functionally: buffers, overlays, etc. An investigation should take place if these areas can be created inside the DBMS (and presented as view) or with specific GIS software.

This research can be extended to 2½D or 3D referenced objects. This extension concerns the objects of interest and their (interrelated) constraints. The objects of interest are currently limited to point objects (plantation objects) and polygon objects (ground surfaces), but could be extended to volume (polyhedron) objects. However data types and operations for 2½D and 3D geometry in DBMS are indispensable and conditional for implementing 2½D and 3D constraints concerning 2½D or 3D objects.

Acknowledgements

The research described in this paper was part of the Geodetic Engineering, Delft University of Technology. The MSc-thesis study was performed by the first author at the Centre for Geo-Information, Wageningen University and Research centre (Louwsma, 2004). This publication is in part (that of Sisi Zlatanova and Peter van Oosterom) the result of the research program 'Sustainable Urban Areas' (SUA) carried out by Delft University of Technology and partly (by Ron van Lammeren) the research theme ‘Geo-visualisation and communication’ of CGI, Wageningen University.

References

Boyd, Lauri L. (2000), CDM RuleFrame – the business rule implementation framework that saves you work. Oracle Corporation, iDevelopment Center of Excellence. From: www.odtug.com

Clementini, E., P. Di Felice, P.van Oosterom (1993), A small set of formal topological relationships suitable for end-user interaction. In *SSD’93: the third international symposium on large spatial databases*, Singapore, (LNCS nr. 692), pp. 277-295. Berlin. Springer-Verlag.

Date, C.J. and Hugh Darwen (1997), A guide to the SQL standard, 4th edition. Addison-Wesley. ISBN 0201964260. Chapter 14 (p197-218).

Egenhofer, M. J. (1989), A formal definition of binary topological relationships, in: *Proceedings of 3th International conference on foundation of data organisation and algorithms*, pp. 457-472

ESRI (2002), *Working with the geomdatabase: powerful multiuser editing and sophisticated data integrity*. An ESRI white paper, February 2002.

Heim, M. (1998). Virtual realism. New York, New York:Oxford.

J.H. Louwsma, J.H. (2004), Constraints in geo-information models. Applied to geo-VR in landscape architecture. MSc-thesis Geodetic Engineering, Delft University of Technology.

Kwon Y-M, Ferrari E., Bertino E. (1999), *Modelling spatio-temporal constraints for multimedia objects*. In Data & Knowledge engineering 30, 217-238.

Lammeren, R. van et.al. (2002), Virtual Reality in the landscape design procees. In: *Landscape planning in the era of globalisation* / D. Ogrin, I. Marusic & T. Simanic. - [S.l.] : [s.n.], 2002 - p. 158 - 165.

OGC (1999) Open GIS Consortium, Inc., OpenGIS Simple Features Specification For SQL, Revision 1.1, OpenGIS Project Document 99-049.

OMG (2002), Object Management Group, Unified Modeling Language Specification (Action Semantics), UML 1.4 with action semantics, January 2002.

Oracle (2002), Oracle 9*i* JDBC Developer’s Guide and Reference, release 2 (9.2) March 2002, Part No. A96654-01. (Chapter 3 – basic features, chapter 7 - Accessing and Manipulating Oracle Data).

Papadias, D. and Y. Theodoridis (1997), Spatial relations, minimum bounding rectangles and spatial data structures, in: *International journal of GIS*, Vol. 11, No.2, pp. 111-138

Peuquet, D.J. (1995), It's about time: a conceptual framework for the representation of temporal dynamics in GIS, in: *Temporal data in Geographic Information Systems*, W. Kuhn and P. Haunold, pp. 149-170

Muller, S. (2000), CDM RuleFrame Overview: 6 reasons to get framed! Oracle Corporation, iDevelopment Center of Excellence. (http://otn.oracle.com/consulting/idelivery/cdma/pdf/rf6reasons.pdf)

Wachowicz , M., Bulens, J.D., Kramer, H., Lammeren, R.J.A., Ligtenberg, A., Rip, F. (2002), GeoVR construction and use: the seven factors. In: *proceedings of the 5th Agile conference on geographic information science* by Ruiz, M., Gould, M., Ramon, Palma, pp.417-422