Centre for Geo-Information

Thesis Report GIRS-2004-17

Constraints in geo-information models

Applied to geo-VR in landscape architecture

J.H. Louwsma

May 2004

```
INSERT INTO prcv_treesrd_point
*
ERROR at line 1:
ORA-20003: 3: The tree (x=104999.32, y=482677.4) is placed inside
or within a distance of 1 meter from 1 paving or soft paving surface(s).
A tree must always be placed >1m from paving or soft paving.
ORA-06512: at "ORAGIS02.PCK_SALIX", line 220
ORA-06512: at "ORAGIS02.AST_SALIX", line 64
ORA-04088: error during execution of trigger 'ORAGIS02.AST_SALIX'
```

WAGENINGEN UNIVERSITY
WAGENINGEN UR

TU Delft

Constraints in geo-information models

Applied to geo-VR in landscape architecture

Author

J.H. Louwsma
Reg.nr.
TU-Delft: G9530353
WUR: 800509-530-080

Supervisors Delft University of Technology

Prof. dr. ir. Peter van Oosterom
Dr. dipl.-ing. Sisi Zlatanova

Supervisor Wageningen University and Research centre

Dr. ir. Ron van Lammeren

May 2004
Wageningen, The Netherlands

Thesis for:
Delft University of Technology
Faculty of Civil Engineering and Geosciences
Department of Geodesy
Section GIS-technology

Thesis code number: GRS-80326
Wageningen University and Research Centre
Laboratory of Geo-Information Science and Remote Sensing
Thesis Report: GIRS-2004-17

Preface

This thesis is the final part of my study Geodesy at the Technical University in Delft. This research wasn't done in Delft but in Wageningen, because it's actually a research for the centre of geo information (CGI) of Wageningen University and Researchcentre (WUR). The CGI is more intersted in using GISs and in Delft the emphasis is more to the technology of GISs. The combination of these two point of views was very interesting and I enjoyed my stay in Wageningen (also because of the nice natural environment!).

When doing this thesis I had two supervisors; Sisi Zlatanova of the TUDelft and Ron van Lammeren of WUR. Both supported me a lot and gave me good feedback about my activities. Also Peter van Oosterom (TU Delft), John Stuiver (WUR), Theo Tijssen (TU Delft), Henk Kramer (WUR) and Arend Ligtenberg (WUR) helped me on some (or more) parts. I'd like to thank them all very much for their time and for sharing their expertise. And last but not least, I want to thank my friend Matthias Kruizinga for his great support.

Jildou Louwsma

Wageningen, May 2004.

Content

Abstract

Geo-VR applications can be created for a better communication about spatial data, especially for spatial planning purposes. The design process, the presentation and the interactions about spatial plans can be supported by these geo-VR applications.

The centre for geo-information (CGI) of Wageningen University and Research centre developed among others a simulation program for landscape architectural design in virtual reality; SALIX-2. To develop landscape architectural plans in SALIX-2, the growth of plantation objects (geo-objects) is simulated. Some improvements are possible for SALIX-2 to make it more realistic and one of the improvements concerns interaction of geo-objects. Plantation objects should not be allowed to be placed on locations that are not logical, such as in the water or on a road. This is still possible in SALIX-2 and this can be restricted to make the application more realistic. Therefore this research to the implementation of constraints in geo-VR was done. The objective of this research was to **define a way to specify and implement constraints in a geo information model**.

A number of constraints are specified for and implemented in SALIX-2 to serve as an example for implementing constraints in a geo-VR application. SALIX-2 is an application consisting of a VRML world for visualization, a Java applet for more interaction possibilities with buttons and scroll down menus and a DBMS for the storage of all data. Because SALIX-2 consists of a DBMS, Java and VRML, this research is also limited to geo-VR applications consisting of these components.

Many geo-VR applications are looked at but none of them had constraints to restrict the virtual world in a way that actions inside the application looked more like the real world actions. Therefore a general definition of a constraint was found (Molenaar, 1998) and converted to a definition of constraints for geo-VR applications. This definition is: a condition that must always be true for **objects** in a 3D model. An object description consist of attributes, behaviour and relations of that object. The different types of relations are:

- spatial topology;
- spatial metric;
- temporal;
- quantity;
- thematic.

With these types of relations also the different types of constraints are defined. When more than one constraint is defined, a check is necessary if all specified constraints do not conflict each other. In this research this check is only manually done in the conceptual phase.

Geo-VR applications consisting of VRML, Java and a DBMS have two different implementation approaches for constraints in the application. The implementation can be done in the digital visualization model (DVM) or in the digital landscape model (DLM). The question 'what is the best way to implement constraints' cannot be answered in general for all geo-VR applications, because each application is unique. To decide which implementation approach is most suitable for which application, some criteria have to be taken into account.

The possibilities of the DVM (Java and VRML) of the application for implementing constraints are very limited for VRML and good for Java. The only nodes in VRML that can make the virtual world more realistic are the collision detection and scripting. These nodes can add dynamic functionality to the virtual world, like starting a sound when an object is touched. The constraints that are defined in this research cannot be implemented using these VRML nodes.

The Java part of the application takes care of the more complex interaction possibilities in the VRML world and of the communication with the DBMS. Java is a programming language so there is a lot possible, including constraint implementation. However, for the constraint checking everything has to be designed and programmed from scratch and the constraints are not stored on a central place. Moreover, when new constraints are implemented, the program code has to be tested and debugged again.

In the DLM (DMBS) all constraint types can be implemented. Literature about DBMSs all mention the possibilities of implementing integrity constraints, however the possibilities of integrity constraints are

limited to rather simple constraints. One should be able to implement complex constraints by using general and base table constraints. However, general constraints only exist in theory (no mainstream DBMS has implemented them) and the base table constraints cannot contain subqueries in the check. The solution of implementing complex constraints in the DBMS means using database triggers and procedures. The advantages of using triggers with procedures are that:

- they are stored on a central place;
- it is possible to enable and disable database triggers separately;
- existing functions and operations of the DBMS can be used inside the triggers and procedures;
- software development environments often offer functionality to generate the code for triggers/procedures needed for the constraint implementation.

For the constraint implementation in SALIX-2c triggers are used to start the constraints checking and stored procedures and functions are invoked inside the triggers for the computations. Oracle is used as DBMS for this implementation and the syntax of Oracle database triggers is very clear. The original Java code is changed so Oracle could be used instead of MSAccess. Also some other changes were made to improve the application (like working with RD coordinates and give textual feedback about constraints).

Feedback about the constraints can best be done beforehand when the application is started and afterwards when some changes are made. A combination of visual and textual feedback is the best solution. Visual feedback is desirable, because a picture is more than thousands words and textual feedback is desirable because this can give more detailed information.

The combination of before, after, visual and textual feedback is however not easy to implement. VRML nodes must be generated to give visual feedback, for example a red area. The geometry of this area must actually be constructed in the DBMS and then a conversion to a VRML node with a certain appearance is necessary. It is difficult to create VRML nodes in Oracle. It is probably easier to generate such nodes in software with 3D extensions. Because the visual feedback was too complex, only textual feedback is implemented in SALIX-2c.

This research was only done for applications consisting of VRML, Java and a DBMS. This is rather limited and therefore a closer look to other geo-VR applications is desirable (e.g. applications with a gaming environments or geoVRML). They can have different system structures and implementation possibilities for constraints.

Furthermore DBMSs and their possibilities for implementing complex constraints can be investigated in more detail. Here database triggers and procedures are used, but there also exist development tools (like Oracle's Custom Development Method) and the Unified Modelling Language with Object Constraint Language. These tools/languages can be of help with modelling and implementing functionality to databases and can also offer possibilities for automatic code generation and easy constraint implementations.

1. Introduction

The overall approach of geographical information is rapidly expanding from only two-dimensional digital spatial data to 2½D and full 3D models. 3D models can among other things be used for spatial planning. The design process, the presentation and the interactions about spatial plans can be supported by 3D models, which especially improve the communication about spatial planning.

One of the institutes in the Netherlands who is active in spatial planning support with 3D models is the centre for geo-information (CGI) of Wageningen University and Research centre (WUR). They already finished a research on games for interactive spatial planning (Wachowicz et al., 2002) and also developed a simulation program for landscape architectural design in virtual reality (SALIX-2) (Lammeren et al., 2003).

SALIX-2 is a simulation program, which is meant for students of the study landscape architecture of the Wageningen University to develop landscape architectural plans. With SALIX-2, the idea to develop a virtual environment application of simulations of the growth of plantation objects (bushes and trees) in a park environment was fulfilled. This research was possible due to the program Virtual Green Environment (VGE) and the education-innovation of the ministry of Agriculture, nature and food quality (project STUWWAL) (Lammeren et al., 2003).

SALIX-2 is thus a virtual environment application of simulations of geo-objects (plantation objects) with a map of a park as basis. Therefore it can be seen as an interactive spatial planning application. Still a lot of improvements are possible for SALIX-2 to make it more realistic. Examples of possible improvements concern (Lammeren et al., 2003):

- *levels of detail:* representation of plants and trees is very abstract in SALIX-2 (more realistic representation of geo-objects);
- *constrained geo-objects:* plantation objects can be put on locations that are not logical, such as in the water or on a road (more realistic actions within the application).

This research concentrates on constrained geo-objects. More specifically, the **objective** of this research is:

Defining a way to specify and implement constraints in a geo-information model

SALIX-2 will serve as a case study to implement a representative selection of constraints. This implementation is a test of the suggested solution Some examples of constraints could be:

- Determine a minimum density for objects in an area;
- Some objects are not allowed to be placed on some surfaces (like trees on water);
- Determine a certain distance of the edge of a surface (for example: a tree can only be placed in a river if it lies within a distance of 1 meter of the bank).

To reach the objective, the next **questions** should be answered:

1. Is there already been a research on this area?
2. What are constrained objects and what types of constraints exist (e.g. geometrical, temporal, thematic, topological)?
3. What is the current application structure of SALIX-2 (Virtual Reality Modelling Language (VRML), Java, External Authoring Interface (EAI), connection to the database and agent technology)?
4. Which 2D constraints can be implemented in SALIX-2 as a good example?
5. What is the best way to implement these constraints (storage in database or VRML environment, accessibility of constraints, interactivity to change the constraints). The plantation objects can be constrained but also some areas of the ground surface can be constrained.
6. How and when can the user of the application be informed about the constraints? (Show the user a list of defined constraints and a good and detailed feedback is necessary).

Within this research some restrictions are made. First of all only constraints for geo-VR applications are looked at, but these can also be applied to other geo-information models. Furthermore there are a lot of constraints one can think of, but within this research only constraints between point objects and polygon objects are considered. Furthermore SALIX-2 is mentioned very often in this report and every time it serves as an example to make the contents of this report more clear. Eventually some constraints are

implemented in SALIX-2 to serve as an example for an implementation in a geo-VR application. The application with constraints is called SALIX-2c (where c stands for constraint).

Chapter 1 is the introduction of this thesis. In this chapter the objective of the thesis and questions to reach this objective are formulated. In **chapter 2** some background information is given with respect to literature in the field of constraints in geo-VR applications. **Chapter 3** discusses the different relations between objects. All different types of relations and corresponding constraints are listed in this chapter. In **chapter 4** the constraints for geo-VR applications are applied to SALIX-2 and some example constraints for an implementation in SALIX-2 are formulated. The possible implementation approaches for constraints in geo-VR applications are described in **chapter 5**. A distinction is made between the implementation in the digital visualization model and the digital landscape model of the application. The implementation of some example constraints in SALIX-2 is a kind of test for the theoretical solutions. **Chapter 6** discusses the conceptual model of SALIX-2c. In **chapter 7** the implementation of constraints in SALIX-2c is discussed with all its remarks. **Chapter 8** gives the conclusions and recommendations for future research.

2. Background information

Geo-VR application can vary in many ways; in their application structure, in the used spatial data as well as in their final purposes. All geo-VR applications contain models that represent real world objects and can be seen as visualization models of the accompanying landscape models. Van der Schans introduced the WGDM model (Figure 2-1), which represents the relations between the real world (W), the graphical representation (G), the mental model (M) and the digital model (D) of this world (Schans, 1997). The idea behind this WGDM model is the capacity of humans to describe the real world in a model (MLM) and to visualize this model in the human's brain (MVM). Graphical visualizations of these MVM's could in history only have an analogues form, like maps, drawings and text. In the recent digital world also digital models exist. The digital landscape models (DLM) consist of digital data describing the landscape and the digital visualization model (DVM) is the visualization of this data. The structure of each geo-VR application (including constraints) can be related to the digital part (DLM/DVM) of the WGDM framework, which is described in more detail in the next section.

Within this research also a lot of terms are used. The first is a geo-VR application; within this research it has the meaning of an application concerning spatial data with 3D visualization and interaction possibilities.

The definition of constraints within such geo-VR applications can vary, but in this research the meaning of a constraint is a restriction of the interaction possibilities within the application to make these interaction possibilities more similar to the real world possibilities (for example no placement of trees on water surfaces are allowed).

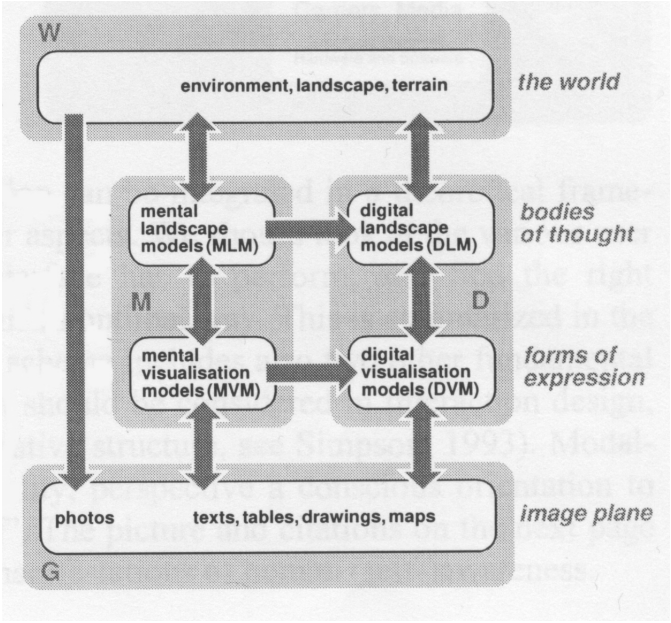

Figure 2-1: WGDM model
(from Schans, 1997)

After the DLM/DVM framework in section 2.1, some examples of existing geo-VR applications with their structure related to the DLM/DVM framework and with their constraints (if implemented) are described. SALIX-2 is an application where a database, geographic information, VRML and Java are combined and where interaction with the 3D model is possible. The VR examples given in this chapter are mostly selected because of the relation of the application structure to the structure of SALIX-2. In the last section the structure of SALIX-2 is described in more detail and is also related to the DLM/DVM framework.

2.1 Geo-data framework

The structure of a geo-VR application can be compared with the DLM/DVM framework illustrated in Figure 2-2. The VR applications have a database for storing the data that provide information about the landscape in real world (the DLM) and a VR representation for the visualization of this data (the DVM). All geo-VR applications contain:

- objects representing the world;
- functions of these objects and/or of the application.

The objects (and their functions) are stored in the database on the DLM side of the framework and are represented in the visualization of the application on the DVM side of the framework.

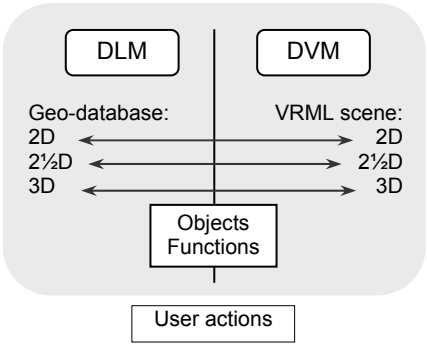

Figure 2-2: DLM/DVM framework for a geo-VR application

Each geo-VR application has its own structure of DLM and DVM and also the relations between the DLM and DVM are different per application. Users are able to interact with the application. These actions concern the objects and are actually not part of the DLM or DVM.

This research is about applications (with constraints) and tools to build/adapt the applications (which can also be used for implementing constraints). To build an application, base tools exist and some of them are even standardised, like C++, Java, SQL, UML, OCL, XML/GML, VRML and X3D. Also in the application domain tools exist to make the building process of an application (or model) easier, like 3D studio Max. And eventually the application itself exists. The system structures of Geo-VR applications discussed in this research vary, but the tools for making the applications are mostly: a GIS, database and VRML + Java for visualization over Internet or other VR environment.

A GIS is a computer-based information system that enables capture, modelling, manipulation, retrieval, analysis and presentation of geographically referenced data (Worboys, 1998). An existing commercial GIS or some other application concerning geographic information can be used for geo-VR applications.

A data set is a unified computer-based collection of data, shared by authorized users, with the capability for controlled definition, access, retrieval, manipulation and presentation of data within it. A Database management system (DBMS) is a software system that manages the data set. The DBMS handles data definition, manipulation and retrieval, transaction management, performance monitoring, back up and recovery. (Worboys, 1998) A data set for an application can be a DBMS or files stored on a server.

For visualization of the data, VRML can be used and additional Java programming make more (inter-)actions possible within the visualization. For the visualization of a VRML file an Internet browser with an additional software package is necessary. That's why VRML is a visualization language for distribution over the Internet. Also other VR environments can be used for the visualization (like geoVRML, X3D, 3D extensions in existing software packages or gaming environments).

The different approaches of interaction between the DLM and DVM of the example applications are discussed in the next section of this chapter. For each example the following aspects are discussed:

- Which tools are used for the application and how does the application structure look like? (GIS, DBMS, VRML + Java or other VR environment)
- Where is the data stored and how does this data flow through the application?
- In what way is the visualization of the data organized (usage of VRML, on the fly visualization, or other visualisation engine)?
- Are user (inter-)actions possible and if yes, are they constraint?

The last geo-VR application discussed in this chapter is Salix-2. Also for this application the above-mentioned aspects are described.

2.2 Examples of geo-VR applications

In this section some example applications are looked at in more detail to find out if already applications exist with some constraints in it, and if not to get an idea of the different system structures where constraints can be implemented in. There are many VR applications, but the examples given in this section give a good overview of the diversity of applications and of application structures consisting of a GIS or DBMS and visualization over the internet. The first application is a combination of GIS and VRML, the second of GIS, DBMS and another VR environment, the third application is a combination of a DB and VRML the fourth is an integration of a DB, GIS and VRML and the last one is SALIX-2 as combination of DB and VRML.

Integration of GIS and VRML

B. Huang et al. (2001) described the advantages of the combination of GIS and VRML (distribution via Internet) for setting up a platform for distributed spatial decision-making. In such an integration GIS provides quite rich spatial data, VRML helps to visualize the data with quite a realistic approach and the Internet facilitates information dissemination. A prototype toolkit, the GeoV&A (Visualization and analysis), is designed to serve as a testbed for this approach. Its implementation is on the basis of the desktop GIS ArcView (more info on URL 14) together with Internet techniques such as Java, Common Gateway Interface and HTML programming. The architecture of GeoV&A consists of a:

- java-based client on the web browser;
- web server with an ESRIMap extension in ArcIMS [URL 14];
- 3D V&A server, i.e. an application server using ArcView and its 3D extension via Avenue programming [URL 14].

It is obvious that this application has a client-server structure and that this structure is also the distinction to the DLM and the DVM. The server side of the application contains the data of the DLM and a VRML file is produced to send to the client as the DVM of this DLM.

The final 3D VRML model is created using ArcScene [URL 14]. Interaction with the VRML scene generated by the 3D V&A module is realized by the combination of Java scripting with VRML and the External Authoring Interface (EAI) method. EAI is an interface for the communication between Java and VRML and is described in more detail in section 2.4. The interaction possibilities of the user consist of querying the world by clicking on a button and specify the query. This query is send to the server, which generates an answer using the DLM and the answer is send back to the user to be visualized in the DVM.

The interaction possibilities only consist of querying the model. Direct manipulation of the VRML scene is not (yet) possible. Constraints as defined in this research (restricted interactions) are not implemented in this application.

Integration of GIS, DBMS and VR environment

E. Verbree et al. (1999) developed a 3D GIS&VR system (Karma VI) based on existing GIS and VR technology that uses three views to support design, development and presentation of large infrastructure plans. The three views (types of visualization) are: plan view, model view and world view. The plan view visualizes the data as a conventional map. The model view provides a 3D bird-eye's view on a partly symbolic and simplified 3D representation of the data. The world view gives a full immersive and photo-realistic 3D display. These views can be used simultaneously or intermittently and each provides interaction possibilities that are appropriate to that view. Through this interaction across views it is avoided that VR is used only as a presentation technique.

Karma is further developed and now it is called Key to virtual Inside (K2vI) (van Maren, 2003). K2vI is a VR interface on spatial data that supports visualisation, manipulation and editing of the spatial data from within a VR environment. The architecture of the multi-view approach of this system consists of an ArcSDE server [URL 14] as central GIS/DBMS, a WorldToolKit from Sense8 [URL 15] as central VR-system and the three views as user interface. Each view uses its own geometric model representation (the DVM) of the data stored in the GIS database (the DLM). With the views all using the same DLM (the same GIS data), consistency between the views is maintained. Using the multi-view approach, the design, decision-making and communication in the process of infrastructure design (or spatial planning) can be supported by an abstract map, a 3D scale model as well as by a very realistic 3D VR scene.

Manipulate your spatial data:

- Select by object, layer or polygon.
- Move and rotate the selected objects use the mouse or Spacemouse.
- Undo changes.
- Save the changes in the GIS database. Changes can be saved directly to the original data in SDE or ArcView (coming soon).
- Collision Detection. Before the actual saving of the changes, K2Vi carries out a collision detection in the K2Vi scene as well as in the GIS database.
- Delete individual or groups of GIS objects in the K2Vi scene.
- Delete individual or groups of GIS objects in GIS database.
- Create new 2D GIS objects from 3D CAD models (footprints).
- Save these new GIS objects to the GIS database.
- Scale and rotate CAD model to fit the GIS object boundaries (footprint).
- Save CAD model file location, position and scaling parameters with the associated GIS object.
- Save linked texture bitmaps parameters (filename, bitmap offset, UV tiling, angle, transparency, shading) with the associated GIS object.

Figure 2-3: manipulation possibilities within K2vI systems
(from [URL 12])

Within K2vI manipulation of the 3D model is possible. The list of possible manipulations can be seen in Figure 2-3. Also basic GIS functionalities (identify, 2D buffering by selecting objects and setting parameters, and 3D measuring by pointing and dragging a 3D line) are possible. Objects and interaction possibilities are not restricted in this application, so also this application gives no information about constraints in a geo-VR application. However, the interaction possibilities are great for this application!

Integration of a DBMS and VRML (extended with Java code)

Xiang LI et al. (2002) discussed a participatory comprehensive plan-making process based on virtual geographical environment. The system structure can be divided into three levels: the input level, the server level and the data level. The input and server level communicate through the Internet. In the input level, citizens can explore the virtual city, discuss and participate in the planning process (by leaving comments). This is the DVM of the application. The major functions of the server level are defining and maintaining virtual meetings, analysing and managing the meetings' result or the comments. Geo-referenced data, user information and participating results (comments) are saved and managed by the DBMS in the DLM of the application. VRML is used to create and maintain the models, Java is used to do the rest things and also here (like the first example) EAI is used for the communication between Java and VRML.

The structure of this application is relative simple. In this application it is not possible for citizens to adapt the virtual city but only discuss about it. The architecture of this application doesn't contain a GIS package so also analysing possibilities of the virtual world is limited. Because of the limited possibilities of interaction (discuss and leave comments), constraints are not implemented in this application.

Integration of a database, GIS and VRML

Andrew Lovett et al. (2002) described a visualization of sustainable agricultural landscapes. The case study area is situated on the boundary of Oxfordshire, Gloucestershire and Wiltshire. Drawing upon information obtained during ecological fieldwork and surveying farmers, as well as discussions with a range of stakeholder organisations, four scenarios for the future landscape of the study area were devised. For each future scenario a scenario map was created. Because 3D visualizations are helpful for getting a full sense of implications, the scenario maps were supplemented with 3D visualizations for key regions within the study area. These 3D visualizations were made using the Pavan virtual reality toolkit that operates within MapInfo [URL 13]. Comprehensive VRML authoring tools were available in the Pavan software. The main stages in the production of the VRML landscape models can be seen in Figure 2-4. In the stage 'add features to be modelled and set properties' trees and bushes were added to the model using the Pavan vegetation modelling tools. First the characteristics of the required feature were specified and subsequently the feature was positioned by a mouse click.

Editing the VRML model was not possible for the participants and stakeholders of the study area. Experience with GIS would be necessary as well as some patience to construct a new VRML model. The Pavan software is suitable for placing vegetation on specified locations in the 2D map and constructs a full 3D model of the entire environment afterwards. But adding, deleting or changing the position of vegetation is only possible in the 2D map and a new 3D model has to be constructed afterwards again.

The DLM is thus the 2D Arc/Info database (see Figure 2-4), which is a GIS package and not a DBMS. The DVM is the VRML model constructed after each adaptation in the DLM. This system structure is less suitable for interactive spatial planning, because experience is necessary to adapt the DLM.

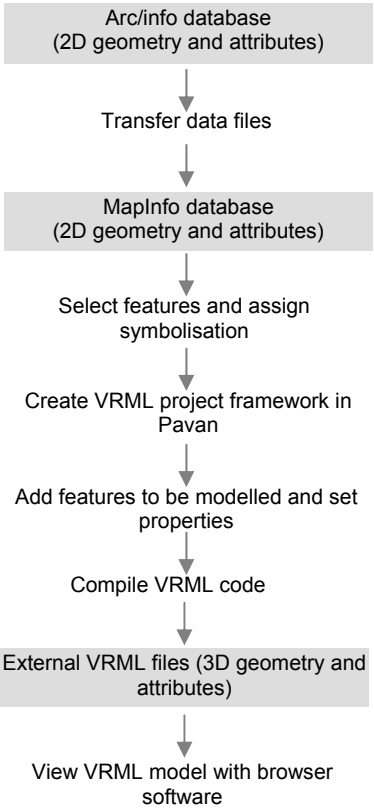

Figure 2-4: The main stages in the production of the VRML landscape models using Pavan. (from Andrew Lovett et al. 2002)

2.3 Structure SALIX-2 as integration of DBMS, VRML and Java

SALIX-2 is an application designed for students of landscape architecture to design a park environment with trees and bushes (plantation layout) and to visualize this design on different moments in time (trees and bushes can grow in SALIX-2). The user can load and make new plantation layouts. Within a plantation layout the user can also add, delete and replace trees and bushes by clicking buttons in the java panel in combination with mouse clicks in the VRML scene.

In SALIX-2 a DLM and a DVM can be indicated like all other applications described before. The DLM consist of a database (MSAccess) filled with tables. For each separate plantation layout a separate table exists in the DBMS filled with all plantation objects and their locations in that layout. A separate table exist for the plantation object types and their behaviour including the appearance in VRML. An image serves as ground surface for the study area. This image doesn't contain information about the geometry of the study area and is therefore not stored in the database.

The plantation layout with all plantation objects and the ground surface are visualized in a VRML browser, which is the DVM. Java is used to add some interaction buttons to the DVM for more interaction possibilities. The connection between VRML and Java is also here done by the EAI (see next section). Java connects the DVM and the DLM.

When loading a plantation layout, all objects are selected from the DBMS and are temporary stored as separate rows in the DBObjects vector. This vector contains all id's and x, y and z coordinates of all objects of the plantation layout. This vector is changed when the VRML model is changed (by the user). These changes are not immediately stored in the DBMS, but only when the plantation layout is saved. A plantation layout is saved after the 'save' button is clicked by the user. The only restriction that exists for manipulation possibilities of the 3D scene is that objects can only be placed on the ground plane image, not beside or floating over the ground plane.

2.4 External authoring interface

Java and VRML communicate with each other through EAI. The following text comes from [URL 3] and [URL 11] and starts with an example that illustrates what the EAI does.

> For instance, if Java wants to change the colour of a sphere in the VRML world, it would make a call to find the sphere by asking for it by name. Then it would make a call to find the colour field of the sphere. Then it would make a call to change the colour of that field. In VRML, you can name any node. That is the name the Java applet is asking for. Most nodes can sent events. When the Java applet finds the colour field and changes the colour of that field, the VRML world receives an event to change the colour, processes that event, and voila, the sphere changes colour. [URL 3]

> To receive notification when an eventOut is generated from the VRML scene, the Java applet must first implement the callback() method of the EventOutObserver. Next the advise() method of EventOut is passed to the EventOutObserver. Then whenever an event is generated for that eventOut the callback() method is executed and passed the value and timestamp of the event. This value is passed to the callback() method and can be used by the applet author to pass user defined data to the callback. [URL 11]

> Everything you can do within a script node (e.g. by using Java/VRMLScript) you can do with the EAI. However, in many situations it is better to use the Script Interface. Here are a few typical situations where the EAI is needed or should be used:
> - Control of interactive multimedia presentations involving more than VRML;
> - Visualization needing a 2D interface applet that controls the VRML scene in real time (e.g. users slides a button and something grows/shrinks/moves in the scene);
> - Custom VRML navigation;
> - Multi-user worlds with chat windows and other networking applications that need a user interface applet. [URL 3]

The previous text only describes the communication between VRML and the Java applets used for the more complex interaction possibilities of the application. Besides this communication also communication between Java and the DBMS is necessary. This can be done in different ways. For SALIX-2 java applets are used for this communication, but also an intelligent way of handling this communication can be done using agents. A short description of agents is given in Appendix C, but within this research the agent technology was not investigated..

2.5 Conclusion

Literature about the described geo-VR applications does not mention constraints in these applications. Also a definition of a constraint in a geo-VR context and the implementation possibilities are not mentioned. The different system structures of the described applications can serve as starting point for the discussion about the possible implementation approaches, but for the definition of constraints in geo-VR more general information technology (IT) literature must be used.

3. Constraints

In literature about geo-VR applications, hardly anything is said about constraints in such applications. Therefore it is necessary to find additional literature about constraints in general and different types of constraints. CAD/CAM literature mention constraints but this is a different field of research and therefore not taken into account here.

Constraints within this research are integrity constraints concerning actions and responses within the application model. They should (almost) be similar to the real world actions and responses. Also the ability to formulate constraints to get a 3D model answering predefined conditions (e.g. the area in the middle of the scene must stay empty) must be presented.

First of all a definition of the term constraint used in this research is necessary. A dictionary gives a definition of a constraint (in the context of programming and mathematics) as a relation, often equality or inequality relation, between the values of one or more variables (often two), for example x>3 is a constraint on x. In the field of geo-information Molenaar (1998) defines consistency or integrity constraints as conditions that must always be true for data items in a database. Adapting these definitions to make it suitable for this research, the definition can be formulated as:

a condition that must always be true for **objects** in a 3D model

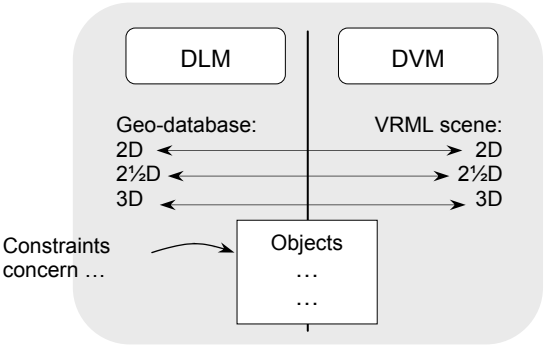

Figure 3-1: Constraints concern objects in the DLM/DVM framework.

In this definition the objects play an important role. A detailed description of objects is therefore desirable to define constraints for geo-VR applications. This description is given in the following section. Constraints can be formulated after the description of objects, this is done in section 3.2. Constraints for SALIX-2 are given in the next chapter.

3.1 Description of objects

Objects in the digital geo-information framework are objects stored in the DLM and visualized in the DVM. Constraints are conditions concerning objects, so constraints can also be implemented in the DLM or DVM, see also Figure 3-1. Objects can be described by their (Zlatanova, 2000):

- attributes;
- behaviour;
- relations.

The *attributes* used for the object description can be divided into thematic and geometric characteristics. The thematic characteristics specify the meaning, usage, etc. of objects in the real world. The geometric characteristics refer to position (and orientation), shape and size of objects in the real world. The attributes are mostly defined on instance level (attributes are different for each object).and not on class level (for all objects of the same type).

The *behaviour* represents the characteristics of objects and can concern:

- operations on geometry (e.g. the possibility of deleting, updating or adding that object)
- reactions of objects to events (e.g. reactions to a mouse click)
- reactions to interactions with other objects (e.g. defines what is going to happen when a car touches a building in the virtual world)
- degree of immersion (e.g. the ability to enter a building and explore the object in detail in the virtual world).

An example is the ability of a door to be open or to be closed. The behaviour is mostly on class level and not on instance level.

The *relations* of an object are the relaions to other objects. The relations can concern spatial information (spatial relations) and non-spatial information, like time (temporal relations), an amount

(quantity relations) and thematic attributes (thematic relations). Relations can be defined both on class level (e.g. all bushes never stand in the water), but also on instance level (the tree with id 21 must be in the water).

The detailed object description (attributes, behaviour and relations) can for example be applied on the SALIX-2 plantation objects. The attributes of the plantation objects are specified in the DLM tables (position, VRML description, etc.). The behaviour is specified in the DVM using Java and VRML (Touchsensors and delete/add/replace possibilities). The relations of the objects are missing for the plantation objects in SALIX-2.

Defining conditions that concern attributes, behaviour and relations of objects can constraint the geo-VR application. Constraints concerning the attributes or behaviour of an object are unary constraints (concern only one object). Constraints that concern relations of objects are binary (concern two objects) or set constraints (concern a set of objects).

SALIX-2 gave rise to this research to constraints in geo-VR applications and especially the (constrainted) relations between objects are of interest for this application. With relations between objects some patterns and complex constraints can be formulated (this in comparison to rather simple constraints concerning the attributes and behaviour of one object). Furthermore many different kinds of object relations exist. So a closer look to object relations is necessary. In this research the object relations are limited to binary relations. Conditions concerning the behaviour and attributes of objects are not further discussed in this research.

3.2 Object relations

Relations between objects A and B can be formulated as one relation but also as part of the description of both object A and object B. Look for example to the relation 'object A must be inside object B'. A part of object A's description is 'must be inside B' and a part of object B's description is 'must contain object A'. The *object relations* can therefore be derived from the *relation between objects* (which is formulated as one sentence).

The possible binary relations are described in this section and these relations are formulated as relations *between* objects (and not as separate object relations). The geometric or spatial relations are described first, followed by temporal, thematic and quantity relations. With these four relations many constraints can be formulated, which is done in section 3.3. In this section also object relations are derived from these relations between objects.

3.2.1 Spatial relations

The spatial relationships of objects specify the connections or interrelations between real objects in the geometric domain. Topology and metric relations are mentioned by Egenhofer (1989) and Zlatanova (2000) as different approaches of spatial relationships.

Topological relations – concern the neighbourhoods of objects and are invariant under topological transformations, such as translation, scaling and rotation. Examples are terms like *neighbour* and *disjoint*.

Metric relationships – exploit the existence of measurements, such as distances and directions. For instance, 'within 5 miles from the interstate highway I 95' describes a corridor based upon a specific distance. Both topological as metric relationships are described in more detail below.

Topological relations

A topology model in geo-information models use the topological primitives interior, exterior and boundary to find relations between two objects. The interior, exterior and boundary of an object A are denoted as A°, A^- and ∂A respectively. With this definition of objects, relations between two objects R(A,B) can be found by intersecting the topological primitives of the objects. The possible intersections are among others described by Egenhofer (1989) and Zlatanova (2000) and a shortened description is given below.

> Intersection of objects can be done with only the interior and boundaries of the objects, which lead to 4-intersection model, or with the interior, exterior and boundaries of objects, which leads to the 9-intersection model. The intersections of the 4-intersection model are: $\partial A \cap \partial B$, $A^\circ \cap B^\circ$, $\partial A \cap B^\circ$, $A^\circ \cap \partial B$ and the intersections of the 9-intersection model are: $A^\circ \cap B^\circ$, $\partial A \cap B^\circ$, $A^- \cap B^\circ$, $A^\circ \cap \partial B$, $\partial A \cap \partial B$, $A^- \cap \partial B$, $A^\circ \cap B^-$, $\partial A \cap B^-$, $A^- \cap B^-$. These intersections of topological primitives of two

objects can be empty (∅) or non-empty (≠∅). For example, if two objects have a common boundary, the intersection between the boundaries is non-empty, i.e. ∂A∩∂B ≠ ∅. So each pair of objects has it's own empty and non-empty intersections.

Topological relations can be constructed using references between objects and express the concepts of inclusion and neighbourhood between objects. M.J. Egenhofer distinguishes the following minimal set of topological relations among intervals in a one-dimensional space described by the intersection of boundaries and interiors of objects (4-intersection model):

- Disjoint, meets, overlap, inside, contains, covers, covered_by, equals

This set of topological relationships can also be found in table 3.1 and it can be generalized for objects of higher dimensions than only one-dimensional intervals.

Clementini et al. (1993) prove that only five separate topological relationships are needed to describe all possible relationships between any combination of two objects from the point, line and area types. These five relationships are touch, in, cross, overlap and disjoint. Van Oosterom et al. (1994) extend this set of relationships with the equal relationship and the definitions of the six relationships is slightly changed to cover also the 3D situation. The minimal set of topological relationships becomes:

- Disjoint, touch, overlap, in, cross, equal.

Table 3-1: minimal set of topological relationships. Relations are among intervals described by the intersection of boundaries (∂ ∩ ∂), interiors (° ∩ °), boundary with interior (∂ ∩ °), and interior with boundary (° ∩ ∂). (from Egenhofer, 1989)

(i1, i2)	∂ ∩ ∂	° ∩ °	∂ ∩ °	° ∩ ∂
disjoint	∅	∅	∅	∅
meet	≠ ∅	∅	∅	∅
overlap	∅	≠ ∅	≠ ∅	≠ ∅
inside	∅	≠ ∅	≠ ∅	∅
contains	∅	≠ ∅	∅	≠ ∅
covers	≠ ∅	≠ ∅	∅	≠ ∅
coveredBy	≠ ∅	≠ ∅	≠ ∅	∅
equal	≠ ∅	≠ ∅	∅	∅

The relationship names are slightly changed in comparison to the names Egenhofer uses. This is because these names have a reasonable intuitive meaning for users of spatial applications (Clementini et al., 1993). In the 9-intersection model many more combinations are possible than in the 4-intersection model, but these additional relations are hard to understand for the end users and is therefore not described here.

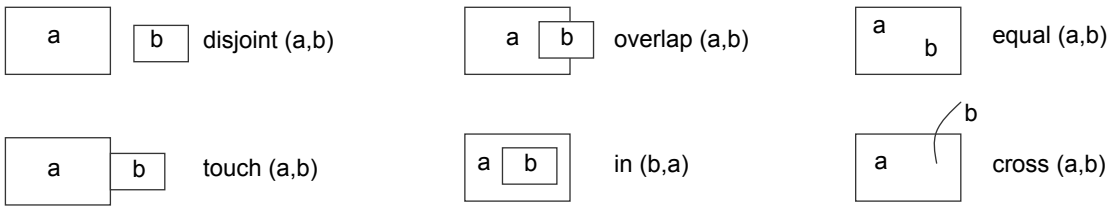

Figure 3-2: topological relations between two objects a and b

Examples of the six topological relationships of the 4-intersection model are given for two objects. Not all relations can be applied to all object types (point P, line L, area A). The *disjoint* relationship can be applied to every situation. *Touch* relationships can be applied to A/A, L/L, L/A, P/A and P/L situations. The *overlap* relationship can be applied to A./A and L/L. *In* relationships can be applied to every situation. *Cross* relationships can be applied to L/L and L/A. The *equal* relationship can also be applied to every situation. The definitions of five relationships between two area objects are given according to M.J. Egenhofer (1989):

1. If all four intersections among all object parts are empty, then the two objects are **disjoint**.
2. If the intersection among the boundaries is not empty, whereas all other 3 intersections are empty, then the two objects **touch**.
3. Two objects **overlap** if they have common interior and the boundaries have common parts with the opposite interior.
4. An object A is **in** another object B if (1) A and B share their interiors, but not their boundaries, (2) A has boundaries which are interior of B, and (3) none of B's boundaries coincides with any of A's interior.

5. Two objects are **equal** if both intersections of boundary and interior are not empty while the two boundary-interior intersections are empty.

The cross relationship can only be applied to line/line and line/area situations:

6. Two objects **cross** if they have common interior and the intersection of the boundaries is empty.

The graphical representation of these relations can be seen in Figure 3-2.

The mentioned topological relations can be applied in 2D and also for 3D topology object relations. This is for example shown in the 3D-GEO++. 3D-GEO++ is a geographic front-end that can be used in addition to a DBMS and is described in Van Oosterom et al. (1994).

Metric relations

Metric relations exploit an existence of measurement, like directions and distances. These relations can't be defined without a reference system with a zero and scale. So a ratio scale is necessary, because this is the only scale with a zero (the other scales are nominal, ordinal, binary and ratio).

For the *directional* relations an origin and an azimuth are necessary. The directional relations are defined as the position of an object in comparison to another object. For point objects it is clear what the direction between two point objects is. The easiest way for non-point objects is to formulate the relation for centroids of objects. Directions can be given in degrees in the range of [0°, 360°], but can also be distinguished in (Papadias et al., 1999):

- Northeast, North, Northwest, West, Southwest, South, Southeast, East

Each of these eight directions (see also Figure 3-3) stands for an interval of degrees. However, the interval boundaries can only be seen as fuzzy numbers and not as exact numbers. E.g. an object is almost never *exactly* east of another object, but can be positioned east, east-northeast or east-southeast of that object. Algorithms are developed to assign the right direction to an object, which isn't exactly positioned according to these eight directions. However, it is not necessary for this research to apply such algorithms, because the plantation objects in SALIX-2 are point objects. So here the subdivision of these eight directions is good enough.

NW	N	NE
W	●	E
SW	S	SE

Figure 3-3: possible directions for directional constraints

Distance relations specify a distance between objects and therefore also a ratio scale is necessary. This distance should be respected between objects and can be:

- a closer than, a farther than or an interval distance

All these distances can be computed by buffer operations in GIS's. These buffers can be computed around points, lines and/or polygons (in 2D plane). The interval distance can also be used to specify an exact distance between objects by making the interval infinite small (the boundaries are the same at both ends and this number specifies the exact distance). In GIS's other distance operations (besides the buffer operation) exist for computing these distances.

3.2.2 Temporal relations

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. These relations (known as Allen's relations) are:

- Before, Meets, Overlaps, Finishes, During, Starts, Equals

Figure 3-4 illustrates the temporal relations supported by this model for intervals a and b. Finished-by and during-by are the inverse relations of finished and during.

The relations mentioned above concern two time intervals (bi-temporal), but can also 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

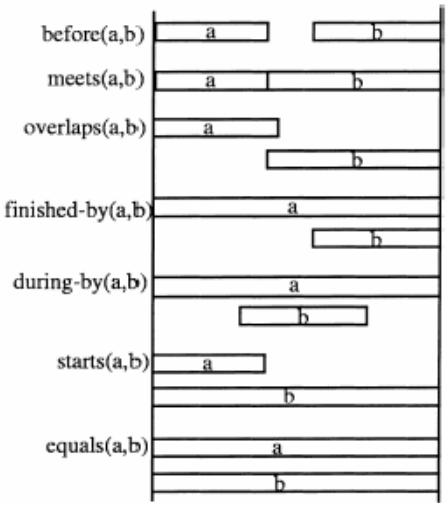

Figure 3-4: temporal relations between two time intervals
(from Kwon et al., 1999)

interval). These relations can be used to define temporal constraints. Temporal constraints are the same for 2D and 3D, because they are not spatial. E.g. the time doesn't change when a representation is changed from 2D to 3D.

Geo-VR applications can represent a static situation, but more often dynamic representations are created. Intelligent objects are placed in the environment that changes in time. Also SALIX-2 has plantation objects that grow over the years (this was the initial idea for developing the application!). An example of a temporal constraint can be that trees of type 1 die earlier than trees of type 2 or that some trees are planted later or earlier than other trees. So in geo-VR environments, the order of events or the order of existence of objects can be defined using temporal relations.

3.2.3 Quantity relations

Specifying a certain density of objects in a certain area is not based on a spatial or temporal relation, but on a quantity of object, therefore here the name 'quantity relation' is used. This relation can be divided into:

- 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: 'there is only one tower in the model', 'there are three benches in the model' or 'one statue must be placed in the 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).

3.2.4 Thematic relations

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 and object attributes can thus be used to formulate constraints to make the virtual world more look like the real world, e.g. 'every house has an address', 'the name of the road (where the house is geometrically connected to) is part of the address of the house' and 'every house has an owner'.

3.2.5 Conclusion object relations

The relations described in this section should all be binary relations. This is true for the spatial and temporal relations, so not all possible relations are described here. To give a complete list of possible relations, these relations have to be extended to relations also concerning a set of objects. Examples of set relations can be

- 'the whole domain must be covered by parcels and the parcels do not overlap' (spatial);
- 'the average age of trees in the model is less than 20 years' (temporal);
- 'object b exists after object a and before object c' (temporal).

The quantity relations described already concern a set of objects (e.g. a minimum number of 5 trees must be placed on a field of grass) and are not limited to binary relations. The thematic relations can be unary (e.g. the house is red), binary (e.g. the house has an owner, where owner is also a specific object in the form of a person) or can be a set relation (all houses have an address). A complete list of all unary, binary and set object relations can be defined in future research.

3.3 Typology of constraints

The objects and in detail the relations between two objects are discussed in the previous sections and the constraints can be formulated using these relations between objects. To get a list of different types of constraints, some distinctions can be made to categorize them. First of all the different kinds of relations can be used to categorize the constraints. Secondly a distinction can be made in the formulation of a constraint. A constraint can be formulated in a forced or a restricted way, while the meaning stays the same. Furthermore a distinction can be made in constraints only using one condition, the so-called simple constraints, or constraints that are a composition of two or more simple constraints, the so-called complex constraints. The distinctions concerning the restricted or forced formulation and the simple and complex constraints are discussed below.

3.3.1 Forced and restricted relations

Constraints can be formulated in the forced way (always have to) and in the restricted way (never to do). The different formulation of the same constraint can in some cases lead to a complicated formulation. For example 'grass always has to be green' has the same meaning as 'grass can never be red, orange, yellow, blue, purple, black, white etc. The last (restricted) formulation is much more complicated and it's more likely that a part of the constraint is forgotten (e.g. orange), which can lead to a very different meaning of the constraint.

Table 3-2 gives examples of constraints and the difference between the forced and restricted formulation becomes clear from the examples 1, 3, 4 and 5. In the formulation of these examples all possibilities have to be mentioned, either in the restricted or the forced formulation. If not all possibilities are mentioned the formulation is not according to the initial meaning of the user. For example, in the restricted formulation of example 4 an infinite number of values must be given, if e.g. the number 4 is omitted in the formulation of this constraint, the number of trees is permitted to be 3 or 4, instead of only 3. This problem doesn't appear if the corresponding positive constraint definition is used, because then only one value has to be specified.

It becomes clear from the given examples in Table 3-2, that there are formulations where:
- only one value is specified;
- many (but limited) values are specified;
- infinite number of values are specified.

There is only a difference between the forced or restricted formulation for examples where n or ∞ values are mentioned. When a minimum/maximum defined time or distance (e.g. example 2) must be specified, it doesn't matter if the constraint is formulated in the forced or the restricted way. This distance and time constraint can be seen as a 'threshold' constraint with only one value and for this threshold relation. This is also true for other relationship constraints concerning only one value.

For all constraints the best formulation must be used. This is the formulation with a minimum number of values. If one really wants to use a forced or a restricted formulation for a constraint, the formulations with n or ∞ values can be rewritten. This can be done using negations (by adding the word 'NOT') to the constraints formulated in the other way. After all, 'always not' is the same as 'never' and 'never not' is the same as 'always'. Look for example to the spatial topology constraints in Table 3-2. The restricted formulation of this example is the best formulation ('houses are never inside water'). If one wants to formulate this constraint in a forced way, this can (besides the example 'houses always have to disjoint, touch or overlap water objects') be done using a negation. The forced constraint will become: 'houses are always NOT inside water'. Also in this formulation the number of values necessary for the constraint formulation is limited.

Table 3-2: examples of simple relationship constraints formulated in the forced and restricted way.
The examples are followed by the number of values that are used in that example, where n stands for a limited number of values and ∞ stands for an infinite number of values.

Type of relation	Constraints based on forced relations (always have to)	Constraints based on restricted relations (never to do)
1. spatial topology	Houses always have to disjoint, touch or overlap water objects. (n values)	Houses are never inside water (1 value)
2. spatial metric	Houses always lie more than 5 meters away from other houses (1 value)	Houses never lie within a distance of 5 meters of other houses (1 value)
3. temporal	The building process of house a is always before the building process of house b. (1 value)	The building process of house a never meets, overlaps, finished_by, during_by, starts or equals the building process of house b (n values)
4. quantity	There must always be three trees around a house (1 value)	There must never stand 0, 1, 2, 4, 5, 6 etc. trees around a house (∞ values)
5. thematic	Houses always have to be placed on cadastral parcels (1 value)	Houses must never be placed on roads, in water, in forest etc. (n values)

3.3.2 Simple and complex relations

Besides a different formulation, constraints can be divided in simple and complex constraints. All examples in table 3-2 are simple constraints. These constraints are based on one condition. In this

condition it is however possible to define more relations, e.g. the combination of disjoint, meet and overlap can be used in one simple constraint (see spatial topology relation in table 3-2). However, these relations are of the same type. Simple constraints can thus be categorized in spatial, temporal, quantity or thematic constraints.

Complex constraints are combinations of simple constraints, e.g. the distance between houses always is more than 5 meter AND there must always be three trees around each house. The word AND is a keyword for complex constraints. Complex constraints cannot be categorized in spatial, temporal, quantity or thematic constraints because combinations of these types can exist (there are 4x4=16 possibilities when a combination of only two simple constraints is used). Also the restricted and forced formulation can be used several times in one complex constraint. This means that all complex constraints cannot be categorized; they are just complex constraints.

In some cases it is desirable to use patterns for the design process. Complex constraints have the ability to formulate patterns, e.g. by specifying distances and directions between objects. An example of a pattern constraint could be: 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. A table with all possible examples of complex constraints, like table 3-2 for the simple constraints, would become too extensive and isn't added to this report.

3.3.3 Constraints as object relations

The different types of constraints are formulated as 'relations between objects' and not as 'object relations'. However, the constraints can be written as part of the object descriptions (besides the attributes and behaviour). Doing this, a check can take place if the constraints are not conflicting. Table 3-3 shows a so-called (that is, in this report) cross relation table, with some object relations in the object description. The implemented example constraints between object instances are:

1. Object A is always inside object D;
2. Object B is never > 3 meter from object D AND
   Object B always meets object E;
3. Object D always contains object A;
4. Object E is always < 5 meters from object A AND
   Object E is always > 4 meter from object B.

The object relations are implemented in the triangle belonging to that object. That is, object A's description contains the relation 'always inside object D' that is implemented in the upper triangle. Object A's description does not contain the relation 'is always <5m from object E' (example 3). This relation is implemented in the lower triangle as description of object E. If all constraints were implemented in all object descriptions the table would become too full.

Table 3-3: implementation of some example constraints in a cross relation table as object relations.
The object instances are described by their attributes, behaviour (not of interest for this research) and their relations (to other objects).

		Object A (attributes)	Object A (behaviour)	Object B (attributes)	Object B (behaviour)	Object C …
Object D	(attributes)		always inside		never > 3 m	…
	(behaviour)	always contain		-		
Object E	(attributes)		-		always meets	…
	(behaviour)	always < 5 m		always > 4 m		
Object F …		…		…		…

A check can take place if the constraints do not conflict each other when all constraints are implemented as part of the object descriptions. E.g. the relation between object A and object D is valid for both the description of object A as for the description of object D ('object A always inside object D' doesn't conflict 'object D always contain object A'). But the relations of object B and object E conflict with each

other (object B always meets object E AND object E always > 4 m from object B) so they can't be both valid.

If many (complex) constraints are formulated, the change of invalid constructions becomes greater, but on the other hand also some valid patterns can be created. However, this cross relation table is not sufficient to guarantee that a set of constraints do not conflict each other. There are combinations one can think of that do conflict, but this conflict is not detected from the cross relation table. For example the combination of 'object A is always inside object B', 'object B is always inside object C' and object C is always inside object A' are conflicting, but this conflict is not detected in the cross relation table.

The moment of the consistency check must be before the constraints are implemented and when constraints are changed. It is preferred that this check should take place automatically. However, even existing tools to implement integrity rules, e.g. topology rules in ArcSDE (see section 5.2.5 for more details about ArcSDE) haven't a consistency check for all defined rules. Neither beforehand nor after the list of rules is modified. So it's outside the scope of this thesis to implement an automatic consistency check in the application. For this research only a manual check beforehand is done. The actual implementation of the constraints is done with constraints between objects.

4. Constraints in SALIX-2

In this chapter example constraints are defined for SALIX-2. Before defining suitable constaints it is necessary to look to the object model of SALIX-2 (including the table structure in the DBMS). After all, the objects are the central part of a constraint. The object model is described in section 4.1. In section 4.2 the example constraints are formulated. A selection of these constraints for implementing in SALIX-2 is also made in this section. Object relations can be derived from these selected constraints and can be filled in in the cross relation table. This is done in section 4.3.

4.1 Object model of SALIX-2

The objects that are presented in the plantation layouts of SALIX-2 can describe the object model of SALIX-2. Plantation layouts are filled with plantation objects, which can be divided into trees and bushes and are represented as point objects. Each plantation layout is stored as a separate table in the DBMS. Table 4-1 is an example of a plantation layout table (only containing 5 plantation objects). All plantation objects (both trees and bushes) are stored in this plantation layout table. Each row is filled with the attributes of one plantation object, which can be different for all plantation objects. The number of different plantation objects in SALIX-2 is limited to five. The attributes that are the same for all objects of the same type are stored in the object type table (see Table 4-2).

Table 4-1: Example of a plantation layout table
This plantation layout only consists of 5 plantation objects. The attributes of the objects are the type, the location (X, Y, Z) and the age. These attributes can be different for all objects.

TreeID	TreeType	TreePosX	TreePosY	TreePosZ	TreeAge
0	CorMas	0	0	0	20
1	CorAve	5	0	5	20
2	RosCan	10	0	10	20
3	QueRob	15	0	15	12
4	FraxExc	20	0	20	15

Table 4-2: The object type table in SALIX-2.
This table is filled with all available different object types. The attributes of the different object types are the same for all objects of that type (this in comparison to the attributes stored in the plantation layout table).

TreeTypeID	TreeType	Kind	Definition	ProtoFile	LatinName
1	CorAve	Bush	EXTERNPROTO CorAve […]	corave3_p.wrl	Corylus avellana
2	CorMas	Bush	EXTERNPROTO CorMas […]	cormas3_p.wrl	Cornus mas
3	FraxExc	Tree	EXTERNPROTO FraxExc […]	fraxexc7_p.wrl	Fraxinus excelsior
4	QueRob	Tree	EXTERNPROTO FraxExc […]	querob3_p.wrl	Quercus robur
5	RosCan	Bush	EXTERNPROTO FraxExc […]	roscan3_p.wrl	Rosa canina

Besides the plantation objects also the ground surface is presented in the plantation layout. This ground surface image can be divided into grass, paving, water and bridge areas (polygons). These areas all have their unique colour and can be seen as the ground surface objects.

4.2 Example constraints for implemting in SALIX-2

The different kind of relations and the objects of SALIX-2 can be used for the constraint definition. Many constraints can be formulated, however, only one example per constraint type is enough to get an idea of the possible constraints for SALIX-2. The example constraints can be found in Table 4-3. A distinction is made between the forced and restricted formulations.

For the implementation in SALIX-2, all mentioned constraints in Table 4-3 can be used. The intention of the implementation is a kind of benchmark for one implementation approach for constraints in an application consisting of a GIS/DBMS and VRML. The selected constraints must be a representative set of constraints and therefore it is preferable to implement an example of each constraint type. However, a number of issues concerning the objects must be considered before implementing different types of constraints.

The first issue concerns the objects of SALIX-2. The plantation objects are stored in the DLM as point objects and are represented in the DVM as 3D objects. Topological relations to point objects are limited to 'inside', 'touch' and 'disjoint'. Furthermore, the ground surface is only stored in the DVM as an image. For the constraint checking it is desirable to use the objects from the DLM. So first of all, the ground surface has to be converted from an image to a geodataset. The objects in this geodataset are the surfaces of the same type, e.g. a grass surface becomes a grass object with a certain id. The conversion process is described in Appendix D. The geodataset of the ground surface stored in the DLM.

The second issue concerns the types of relations. The temporal constraints are not selected for the implementation. This is because there's not much change over time in SALIX-2. Only the event of growing plantation objects exist as temporal aspect.

Table 4-3: example constraints for SALIX-2, formulated in a forced and restricted way.

Relation	Constraints formulated in the **forced** way	Constraints formulated in the **restricted** way
Spatial topology *	1. bushes always have to disjoin or touch water (2 values) 2. A bush always has to touch or disjoint paved areas (also thematic constraint) (2 values)	1. bushes never lie inside water (1 value) 2. A bush never is inside paved areas (6 values)
Spatial metric	Directional constraints: 3. A bush always has to be placed south of a tree (1 value) Distance constraints: 4. Trees always have to be positioned > 1 meter from paving (1 value)	Directional constraints 3. Bushes are never placed northeast, east, southeast, south, southwest, west or northwest of a tree (7 values) Distance constraints: 4. Trees never be located < 1 meter from paving (1 value)
Temporal	5. An oak always grows for 70 years (1 value)	5. An oak never not grows for 70 years (1 value)
Quantity	6. There must always be at least 10 trees on the specified ground surface (1 value)	6. A specified ground surface can never have < 10 trees on it (1 value)
Thematic	7. A tree is always of the type QueRob or FraxExc (2 values) 8. A bush always has to touch or disjoint paved areas (also topological constraint) (2 values)	7. A tree is never not of the type QueRob or FraxExc (2 values) 8. A bush never is inside paved areas (also topological constraint) (2 values)
Complex	9. The distance between two 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 QueRob; 10. 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).	

* note: bushes and trees are point objects. The topological relations of points are limited to inside, touch or disjoint.

The selected set of constraints to implement in SALIX-2 is listed in Table 4-4. This is a selection of the example constraints considering the above-mentioned issues. Also the best formulation is used (forced or restricted).

Table 4-4: Example constraints to implement in SALIX-2

Type of relation	Constraints to implement in SALIX
Spatial topology	1. Bushes never lie inside water
Spatial metric	2. A bush always has to be placed south of a tree
	3. Trees always have to be positioned > 1 meter from paving
Quantity	4. There must always be at least 10 trees on the specified ground surface
Thematic	5. A bush always has to meet or disjoint paved areas
Complex	6. The distance between two 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 QueRob

4.3 Constraints as object relations in SALIX-2

The constraints mentioned in Table 4-4 can be inserted in the cross relation table to check if the constraints do not conflict each other. The objects involved in the constraint definition of Table 4-4 do not always concern object instances (specific objects), but mostly object classes (e.g. all trees, all water areas). If we want to implement these constraints in a cross relation table, it is better to take an object class as column or row instead of an object instance.

The selected constraint in Table 4-4 do not concern two or more ground surface objects, so these objects can be put in the cross relation table once (either in separate rows or separate columns). The plantation objects (trees and bushes) however, can have relations between themselves and with ground surface objects. So the plantation objects have to be placed in the table in a way that relations to ground surfaces and other plantation objects are possible. Table 4-5 can now be created and all selected example constraints are implemented as object relations.

Table 4-5: cross relation check for example constraints to be implemented in SALIX-2

	Tree	Bush
Tree	Always > 8 m if both are inside water (always > 8 m if both are inside water)	Always has to be south of ---
Bush	--- (Always has to be south of)	--- ---
Water	If inside water, always < 0,5 from edge ---	Never inside ---
Paving (including soft_paving)	Always > 1 m ---	Always meet or disjoint ---
Specified ground surface	--- Always has at least 10 inside	--- ---

The object relations in Table 4-5 do not conflict each other, so it looks like the set of constraints can be implemented without consistentcy problems. But again, this cross relation table is not sufficient to guarantee that there are no conflicts between the constraints (as is described in 3.3.3).

After this (first sight) check the constraints have to be implemented in the application. When and how the constraints should be checked can vary per application, because each application has a different structure. For each implementation it is desirable to look closer to the application structure and the flowcharts of actions within the application. With this the implementation story for constraints in SALIX-2 (or another comparable geo-VR environment) is started. A check beforehand is necessary to find out if the constraints are valid for the current 3D model. After that all changes made in the 3D model of a geo-VR application have to be checked. So a constraint checking should follow on each modification in the 3D model.

The possible position for the constraints checking in SALIX-2 can be seen in Figure 4-1. The plantation layout can be edited and the changed plantation layout can only be saved after the changes are checked for validity. Then the plantation layout can be edited again. Note that in this figure the check whether the constraints are valid for the existing plantation layouts is not visualized. For SALIX-2 this beforehand check is only done manually for one plantation layout (used for the example implementation). All other existing layouts have to be constructed again (or also checked manually) if one wants to know if these layouts also fulfull the constraints.

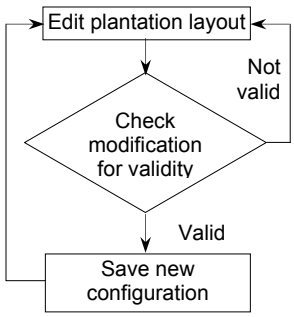

Figure 4-1: The position of the constraint checking in SALIX-2.

With the abstract structure of Figure 4-1 the constraint checking (check for validity) is still a black box. The discussion how the constraint checking should take place is therefore started in the next chapter.

5. Approaches for implementing constraints in a geo-VR environment

After defining the different kinds of constraints, one can think about the implementation of the constraints in an application. The most suitable implementation approach can vary per application, because each application has it's own structure. This research aims to find a way to implement constraints in a geo-VR application consisting of a DLM and DVM. These two components can also be used to distinguish two implementation approaches:

1. implementing constraints in the DLM;
2. implementing constraints in the DVM.

The DVM can be compared to the client side of a client-server application. It is the VRML code (and possible additional Java programming) that is distributed over the Internet, while the DLM can be compared to the server side of a client-server application. This server side consists of the DBMS of the application. This can also be seen in Figure 5-1.

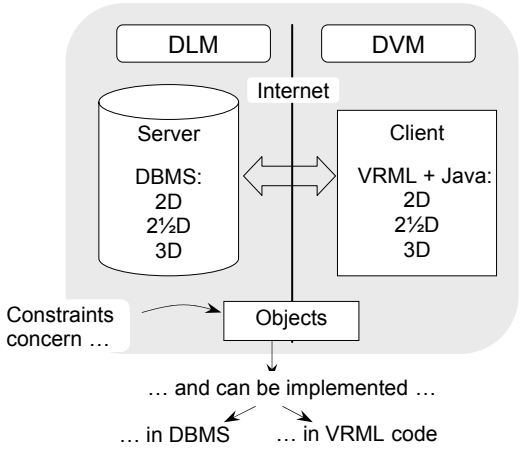

Figure 5-1: Implementation possibilities of constraints in the DLM/DVM framework. The constraints can be implemented in the visual environment. This is the right side of the framework and can be compared with the client side of a client-server application with distribution over the internet. The constraints can also be implemented in the database. This is the left side of the framework and can be compared with the server side of a client-server application..

If constraints are implemented in the **DVM**, there is no connection to the database necessary for the constraint checking. This means that the constraint checking could be rather quickly. However, when implementing the constraints in the application code on the DVM side, the constraints are not stored on a central place. The application structure becomes less orderly and it is not easy to change the constraints.

If constraints are stored in the **DLM**, they are stored in a central place. However, for each constraint checking a connection to the database is necessary and some VR applications do not connect to the database after each modification made by the user (e.g. there's only a connection to the database in SALIX-2 when saving or loading a plantation layout). Besides the option of connecting to the database after each modification, one can also think of a 'save' button in the graphical user interface (GUI). With this button the user can decide when he/she wants to save the changes and also when to check the modifications for validity. Feedback to the user about the validity of the modifications is only given after the connetion to the database is made. Fast feedback is thus not guaranteed.

Besides the discussion how to implement constraints also a discussion about the interactivity of constraint definitions is necessary. For some constraints it can be desirable that they are editable by the user or that it is possible to let the users define constraints by themselves (e.g. to implement some policy rules in an application). On the other hand it can be desirable that some constraints are not editable (e.g. 'natural' constraints, like 'a tree never stands upside down'). This editable or non-editable classification of constraints can for example be implemented as extra attribute per constraint. This discussion is important for the final decision about how to implement constraints and how to structure the application. In this and the next chapters this subject is taken into account when discussing the implementation possibilities.

The implementation options (DVM or DLM) for implementing constraints in a geo-VR application have advantages and disadvantages. For both implementation approaches a more detailed exploration is made. The next section describes the implementation approach for the DVM followed by the section describing the DLM possibilities. Within these sections also the editable/non-editable distinction of constraints is taken into account. In section 5.3 a consideration follows about using freeware or commercial software.

This can be very dependent on the final purposes of the application. Eventually a list of criteria is given in section 5.4 for deciding which implementation approach is best for which geo-VR application.

5.1 Implementing constraints in the DVM

Implementing constraints in the DVM means implementing constraints in the VRML code of the application (and/or in the possibly additional Java code). To know the possibilities within VRML that could be of help for the constraints checking, an introduction in VRML is necessary. This introduction is followed by a more detailed explanation of the different possibilities for the constraints checking. Furthermore it is useful to look to the state diagrams in SALIX-2c when changing the plantation layout (by placing or dragging a plantation object). These changes can be the starting point of the constraitns checking in SALIX-2c and can give an idea of the implementation possibilities. These implementation possibilities are maybe also applicable for other geo-VR applications.

In the next subsection state diagrams are given, that represent one of the possible moments of the constraint checking in SALIX-2c. They are followed by a general introduction in VRML. The collision node and touch sensor node of VRML are described after the general introduction in VRML. With the (rather complex) script node there is even more possible, therefore a description of this node with its possibilities can be found in subsection 5.1.5.

5.1.1 Possible moment of constraint checking in the DVM of SALIX-2c

In this section state diagrams are made with an indication of the place of constraint checking in the DVM and can be seen in Figure 5-2 and Figure 5-3. The constraint checking can be done in the VRML (Figure 5-2) or Java part (Figure 5-3) of the application. In these figures also a box is present with the text 'give feedback…'. More information about the feedback can be found in section 6.5.

During the actions of dragging or adding an object in the VRML scene of SALIX-2, no connection to the database is made. The text 'check for constraints' in Figure 5-2 and Figure 5-3 can be approached as a black box, which has to be filled in (by own hand written code). A connection to a DBMS is necessary in this 'black box' if constraints are stored in a table in the DBMS. This is not necessary if VRML (or Java) offers the opportunity of constraint checking in the DVM and when constraints are also stored in this side of the application. The possibilities VRML offers for constraint checking are investigated in this section.

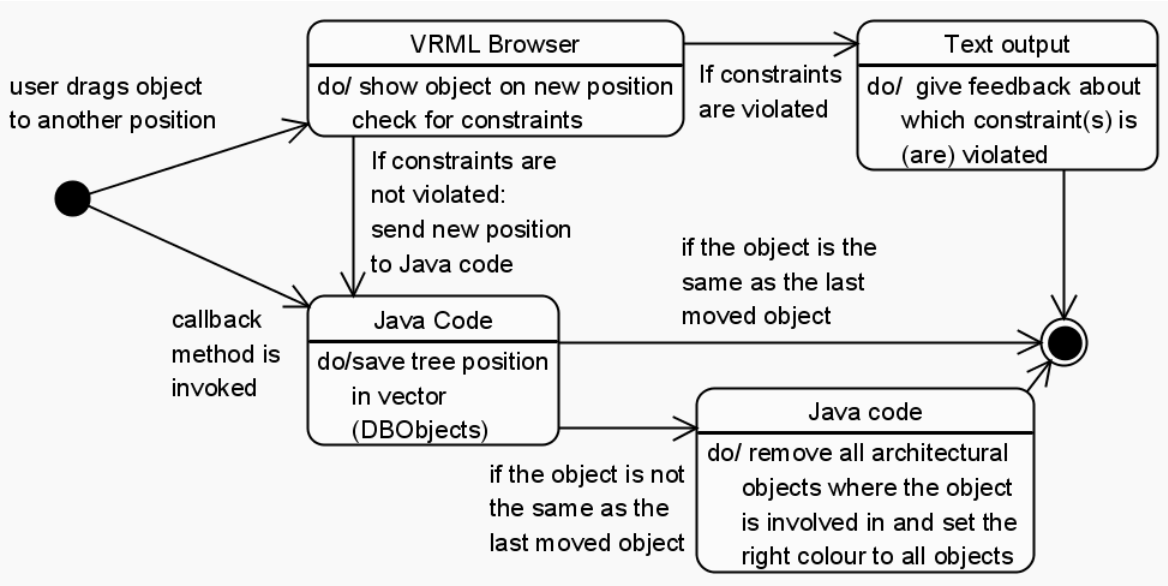

Figure 5-2: possible postion of constraint checking in DVM by dragging an object in SALIX-2c.
This state diagram contains an indication of the position of the 'constraint check' in the VRML Browser of the DVM. When the constraints are not violated, the following actions are exactly according to the acctions in SALIX-2. When the constraints are violated feedback must be given to the user and the changes should not be saved. A connection to the database is not necessary with this constraint implementation approach.

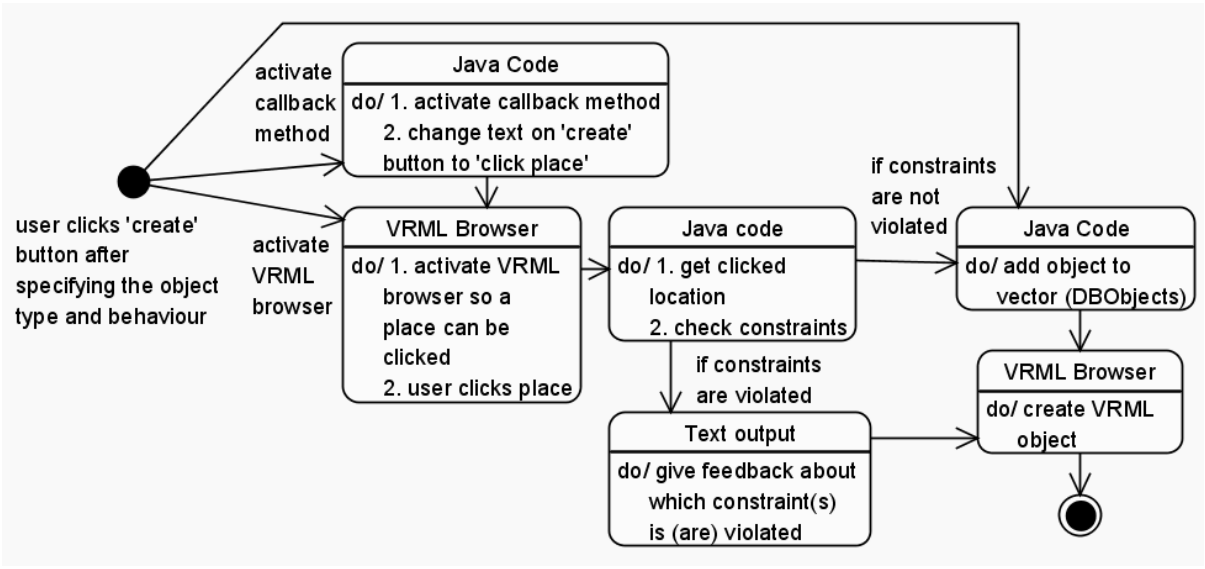

Figure 5-3: possible postion of constraint checking in DVM by adding an object in SALIX-2c.
This state diagram contains an indication of the position of the 'constraint check' in the Java code of the DVM. When the constraints are not violated, the following actions are exactly according to the acctions in SALIX-2. When the constraints are violated feedback must be given to the user and the changes should not be saved. A connection to the database is not necessary with this constraint implementation approach.

5.1.2 Introduction VRML

VRML allows one to describe 3D objects and combine them into scenes or worlds. One can use VRML to create interactive simulations that incorporate animation, motion physics, and real-time, multi-user participation. Virtual landscapes that are created using VRML can be distributed using the World Wide Web, displayed on another user's computer screen, and explored interactively by remote users. (Hartman et al., 1996).

VRML is not a programming language, like C or Java, but a modelling language, which means you use it to describe 3D scenes. With a VRML file, one can view a scene from an infinite number of viewpoints, this in comparison to some photo's of the same area which only represent the area of one single viewpoint. The browser has navigation tools that make travelling through the scenes possible, taking as many different paths as the user desires. (Hartman et al., 1996).

VRML is also the basis for other visualization languages, like GeoVRML and X3D. These are both very close related to VRML. GeoVRML is a specification for representing various geographic data using VRML [URL 6]. GeoVRML is a set of Nodes implemented as VRML PROTOs [URL 7]. An example of such a GeoVRML node is the GeoCoordinate. In this node one can build geometry using geographic coordinates. The syntax of a GeoCoordinate node is [URL 6]:

```
EXTERNPROTO GeoCoordinate [
   field  SFNode    geoOrigin    # NULL
   field  MFString  geoSystem    # [ "GD", "WE" ]
   field  MFString  point        # []
] [ "urn:web3d.org:vrml97:node:GeoCoordinate"
    "file:///C:/Program%20Files/GeoVRML/1.1/protos/GeoCoordinate.wrl"
    "http://www.geovrml.org/1.1/protos/GeoCoordinate.wrl" ]
```

Furthermore the development of VRML has stopped since the Web3D Consortium started to work on a XML version of VRML, in order to integrate with other web technologies and tools. This successor of VRML was X3D (eXtensible 3D). The specifications of X3D have only recently become available (May, 2003) (Vries et al., 2003). Both GeoVRML as X3D are not further discussed in this report, but can be very interesting for future research.

The VRML world is composed of nodes, which are objects or groups of objects in the scene. The nodes may have routes between them, above and beyond the scene graph hierarchy, which define the possible interactions of one node with another. Nodes have fields, which define what actual values the node has, like the geometry or colour of the node [URL 2].

For the approach of implementing constraints only in the VRML code, the touch sensors or the collision detection in VRML seem to be the best entries for implementing spatial constraints. The collision node can detect collisions between the 'user' and other objects and the touch sensor node can detect whether the pointing device touches the touch sensor node or not. So if a user for example places a tree on a water surface, the collision detection or the touch sensor can detect a touching between objects. But with only detecting a touching between objects no constraint checking can be done, because the information is not sufficient enough for determining which objects are involved and if this touching is valid or not. If all this necessary information can be extracted with the collision detection or the touch sensor node, is investigated in the next subsections.

5.1.3 Collision detection in VRML

To make a realistic virtual environment, it is necessary to define certain objects as solid and make sure that one can't walk through these objects. Collision detection is the method to realise this and can be used to make sure that a person stays on the ground in a virtual environment or that this person can't walk through walls.

```
Collision {
  eventIn       MFNode    addChildren
  eventIn       MFNode    removeChildren
  exposedField  MFNode    children       []
  exposedField  SFBool    collide        TRUE
  field         SFVec3f   bboxCenter     0 0 0
  field         SFVec3f   bboxSize       -1 -1 -1
  field         SFNode    proxy          NULL
  eventOut      SFTime    collideTime
}
```

Figure 5-4: syntax of the Collision node in VRML. The only EventOut is the collide Time

For a proper constraint checking, geometric information about objects touching surfaces or other objects is necessary as well as information about which objects and surfaces are involved. A tree can for example be placed on grass, but can't be placed on water or on a road.

Collision detection between objects is easy to implement in VRML, because a collision node is a default node of the VRML language. In Figure 5-4 the syntax of the collision node is shown. The following text comes from [URL 1] and gives a good description of the collision node and its properties.

> The Collision node is a grouping node that specifies the collision detection properties for its children (and their descendants), specifies surrogate objects that replace its children during collision detection, and sends events **signaling that a collision has occurred between the avatar and the Collision node's geometry or surrogate** (where the avatar is the representation of the person who walks through the scene). By default, all geometric nodes in the scene are collidable with the viewer except IndexedLineSet, PointSet, and Text. Browsers shall detect geometric collisions between the avatar and the scene's geometry and prevent the avatar from 'entering' the geometry.
>
> The Collision node's *collide* field enables and disables collision detection. If *collide* is set to FALSE, the children and all descendants of the Collision node shall not be checked for collision, even though they are drawn. This includes any descendent Collision nodes that have *collide* set to TRUE (i.e., setting *collide* to FALSE turns collision off for every node below it).
>
> Collision nodes with the *collide* field set to TRUE detect the nearest collision with their descendent geometry (or proxies). When the nearest collision is detected, the collided Collision node sends the time of the collision through its *collideTime* eventOut. If a Collision node contains a child, descendant, or proxy that is a Collision node, and both Collision nodes detect that a collision has occurred, both send a *collideTime* event at the same time. A *collideTime* event shall be generated if the avatar is colliding with collidable geometry when the Collision node is read from a VRML file or inserted into the transformation hierarchy.

From the above text and from the syntax of the collision node (see Figure 5-4) it can be concluded that the only information going out if collision takes place is the collideTime. A certain time doesn't give information about which objects are involved in the collision. So if the collision node is used as starting point for the constraints checking, additional scripting is necessary within the application.

Furthermore the collision detection only gives information about the collisions between the avatar and the geometries of the objects in the 3D model, support for inter-object collision is not specified. Collision detection within VRML is therefore not a solution for the constraints checking in an application consisting of a DBMS and VRML.

5.1.4 Touch sensors in VRML

From [URL 1] the following information about the TouchSensor node is given.

A TouchSensor node tracks the location and state of the pointing device and detects when the user points at geometry contained by the TouchSensor node's parent group. A TouchSensor node can be enabled or disabled by sending it an *enabled* event with a value of TRUE or FALSE (see Figure 5-5). If the TouchSensor node is disabled, it does not track user input or send events.

```
TouchSensor {
  exposedField SFBool  enabled TRUE
  eventOut     SFVec3f hitNormal_changed
  eventOut     SFVec3f hitPoint_changed
  eventOut     SFVec2f hitTexCoord_changed
  eventOut     SFBool  isActive
  eventOut     SFBool  isOver
  eventOut     SFTime  touchTime
}
```

Figure 5-5: syntax of the TouchSensor node in VRML

The *isOver* eventOut reflects the state of the pointing device with regard to whether it is pointing towards the TouchSensor node's geometry or not. When the bearing of the pointing device intersects geometry belonging to a TouchSensor, an *isOver* TRUE event is generated. When the pointing device moves to a position where it no longer intersects the geometry, or some other geometry is obstructing the TouchSensor node's geometry, an *isOver* FALSE event is generated. These events are generated only when the pointing device has moved and changed 'over' state. Events are not generated if the geometry itself is animating and moving underneath the pointing device.

Each movement of the pointing device, while *isOver* is TRUE, generates *hitPoint_changed*, *hitNormal_changed* and *hitTexCoord_changed* events. *hitPoint_changed* events contain the 3D point on the surface of the underlying geometry, given in the TouchSensor node's coordinate system. *hitNormal_changed* events contain the surface normal vector at the *hitPoint*. *hitTexCoord_changed* events contain the texture coordinates of that surface at the *hitPoint*. The values of *hitTexCoord_changed* and *hitNormal_changed* events are computed as appropriate for the associated shape.

If *isOver* is TRUE, the user may activate the pointing device to cause the TouchSensor node to generate *isActive* events (e.g., by pressing the primary mouse button if a mouse is the pointing device). When the TouchSensor node generates an *isActive* TRUE event, it grabs all further motion events from the pointing device until it is released and generates an *isActive* FALSE event.

The eventOut field *touchTime* is generated when all three of the following conditions are true:

a. The pointing device was pointing towards the geometry when it was initially activated (*isActive* is TRUE).
b. The pointing device is currently pointing towards the geometry (*isOver* is TRUE).
c. The pointing device is deactivated (*isActive* FALSE event is also generated).

If objects in the VRML scene are implemented as children of the TouchSensor node, the movements and actions done with the pointing device (in most cases a mouse) are followed. When the bearing of the mouse touches an object an isOver TRUE event is generated and when the user holds a mouse button down while pointing to an object an isActive TRUE event is generated. When the user releases the mouse button the *touchTime* EventOut is generated with the current time.

Also the hitPoint_changed, hitNormal_changed and hitTexCoord_changed are generated where the hitPoint_changed is of interest here, because this is a 3D vector. This 3D point is located on the surface of the object and could be used as input coordinates for the constraint checking.

The necessary geometric information and information about which objects and surfaces are involved in the 'touch' cannot be substracted using the TouchSensor node. The geometric information is available in the appearance of the hitPoint_changed. However, the 3D vector lays arbitrarily on the object that is part of the TouchSensor. Also information about which objects and ground surfaces are involved isn't available with this approach.

In other words, the TouchTime generated by the TouchSensor can only serve as starting point for a script that should check for constraints, but the TouchSensor doesn't provide enough information for a proper constraint checking. This information can for example be gathered while running the script. Therefore the scripting possibilities are described next.

5.1.5 VRML Script and routing

Some nodes can produce output events and/or can receive input events. In Figure 5-4 and Figure 5-5 some EventIn and EventOuts belonging to the collision and touch sensor nodes can be found. The incoming events are messages sent by other nodes to change some state (field) within the receiving node and outgoing events are used to send messages (events) to destination nodes. In order to connect a node generating a message (event) with another node receiving a message you must use a 'ROUTE' statement. A node that produces events of a given type can be routed to a node that receives events of the same type (e.g. string, boolean, float, vector) with the following syntax [URL 3]:

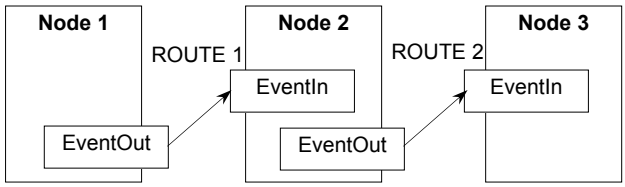

Figure 5-6: Route/event model in VRML.
Node 1 is for example a sensor and produces an eventOut which is routed to the EventIn of node 2, for example a script. Node 2 sends an EventOut to the EventIn of Node 3, that is for example an object.

```
ROUTE NodeName.eventOutName TO NodeName.eventInName
```

Figure 5-6 shows this route/event model in VRML. If the route/event model of Figure 5-6 would be the initial structure of the constraint checking after some actions concerning an object took place, node 1 can for example be a TouchSensor where the objects of the 3D model are all in the group of the TouchsSensor, node 2 can be some scripting and node 3 can be some feedback about the validity of the actions.

In VRML a Script node enables scripting. The syntax of the script node can be seen in Figure 5-7. The URL in this script node can be a reference to a JavaScript or the script can directly be implemented in the Script node. Within the JavaScript programming code can be generated to implement the constraint checking. With programming a lot is possible, but all code has to be created, tested and debugged by the designer (in GIS's or DBMSs some standard functions and operations exist that can be used, see section 5.2) Moreover, the constraints are not stored on a central place when implementing them somewhere in the programming code. Adding, removing or changing constraints can therefore be difficult. So in some cases it can be more efficient implementation the constraints in the database. This will be discussed in the next section.

```
Script {
  exposedField MFString url           []
  field        SFBool   directOutput  FALSE
  field        SFBool   mustEvaluate  FALSE
And any number of:
  eventIn      eventTypeName eventName
  field        fieldTypeName fieldName initialValue
  eventOut     eventTypeName eventName
}
```

Figure 5-7: syntax of script node

5.1.6 Conclusion of implementation possibilities in DVM

The possibilities of VRML for implementing constraints are discussed in this section. The collision node of VRML is not suitable for implementing constraints. This is mostly because only collisions between the avatar and other geometric objects can be detected with this standard node of VRML and inter-object collisions are not supported. The TouchSensor can serve as starting point for a script node. The scripting can be based on the programming language Java. With Java programming a lot is possible, but all code must be created, tested and debugged by the designer. Also central storage and a good accessibility of the constraints cannot be guaranteed when using script nodes. Therefore the possibilities to implement constraints in the DLM are discussed next.

5.2 Implementing constraints in DLM

Implementing constraints in an application means storing the constraints in a proper way and make sure that a good connection exists between the stored constraints and the rest of the application. Implementing constraints in the DLM means therefore storing constraints in the server side of an application (DBMS) and make a connection to the DVM. Besides storing constraints in the DBMS also communication and computation aspects are very important for a good implementation. A closer look to integrity constraints and standard operations and functions of existing DBMSs is desirable to find out if all types of constraints can be managed by these standard possibilities. If this is true, the amount of programming should be considerably less than implementing constraints by programming them in Java.

Many DBMSs exist, but within this research the geo component of the application is of interest. Therefore geo-DBMSs should be used. The benefits of geo-DBMSs are:
1. Geo-DBMSs can manage geometric data;
2. In ArcGIS 8.3 for example, there are many conversion tools;
3. The data can be used for other (geo-) analyses and applications;
4. Some geo-databases, like Oracle, are object oriented.

There are many different geo-DBMSs, among others IBM DB2, MySQL with MapInfo's SpatialWare, Informix, Ingres, PostgreSQL (with PostGIS) and Oracle. Each of these geo-DBMSs have standard functions and operations to adapt or query geometric data and these possibilities are more or less the same for all geo-DBMSs. Investigating only one of the geo-DBMSs is sufficient to get an overview of the standard functions and operations. MySQL and Oracle (Spatial) are both available at WUR.

MySQL is an open source DBMS. MySQL is multithreaded and runs on many platforms. On the client side, the primary API is written in C (for greater portability), and most application and scripting languages use this library under the hood. The MySQL client/server protocol is public. (Lentz, 2003)

Oracle Spatial is commercial DBMS that provides a SQL schema and functions that facilitate the storage, retrieval, update and query of collections of spatial features in an Oracle database. Oracle Spatial consists of the following components (Oracle, 2002c):
- A schema (MDSYS) that prescribes the storage, syntax, and semantics of supported geometric data types;
- A spatial indexing mechanism;
- A set of operators and functions for performing area-of-interest queries, spatial join queries, and other spatial analysis operations;
- Administrative utilities.

For this research Oracle (Spatial) is used (also for the implementation of some example constraints in SALIX-2), because:
- Oracle is a widely used commercial DBMS;
- experience exists with Oracle (at WUR and TUDelft);
- Oracle has good geo-functionality;
- Oracle has orderly tutorials.

Aspects of interest for implementing constraints in the DBMS are:
- communication between the DBMS and the rest of the application;
- communication within the DBMS;
- standard functions and operations (of Oracle);
- additional programming possibilities (PL/SQL, Java, SQL).

In the next subsection, first an introduction to the Structured Query Language (SQL) is made. This language allows querying, which can be seen as filter operations, and all (if not almost all) geo-DBMSs have the possibility of using SQL. This section is followed by a discussion in 5.2.2 about integrity constraints in DBMSs in general. Integrity constraints are available in almost all DBMSs. In section 5.2.3 Oracle Spatial and its functionality is looked at in more detail, followed by a short description of CDM RuleFrame in section 5.2.4 and ArcSDE in section 5.2.5.

5.2.1 Structured Query Language

Database query languages exist to request or get responses from databases. SQL is the best-known query language for relational databases and almost all geo-DBMSs support SQL. SQL is not only a query language but can also be used for storage, retrieval or updating of simple geospatial feature collections (having spatial and non-spatial attributes). The following text comes from (ESRI, 2002):

> SQL provides an interface to relational tables that allows you to select rows based on the values contained in the fields. An SQL statement can range from very simple to very complex, allowing you to compose virtually any type of query from basic column types.

The result of a query is a set of rows meeting the criteria established by the SQL statement. A list of SQL data manipulations (DML) and queries and SQL operators can be found in Appendix B. The SQL

SELECT command can be used to obtain information from (a combination of) tables in the database and is useful for this research to check whether the stored constraints are violated or not. The syntax of the SELECT command can be found in Table 5-1.

Table 5-1: The syntax of the SQL SELECT statement

Syntax select statement (from Haan, 1993)	Types of conditions for the where component (from [URL 4])
SELECT [DISTINCT] select_expr[,…] FROM table_expr[,…] [WHERE cond] [GROUP BY expr[,…] [HAVING cond]] [{UNION [ALL]|INTERSECT|MINUS} query] [ORDER BY {expr|pos} [ASC|DESC][,…]]	{ simple_comparison_condition | group_comparison_condition | membership_condition | range_condition | null_condition | exists_condition | like_condition | compound_condition }

Note that a select statement only concerns existing tables. This means that querying the data for the constraint checking can only be done after the table is adapted and filled with the new information. If the data does not satisfy the constraints a rollback has to take place to make the updating of the table undone (or just before the final commit).

The SELECT and FROM clause select statement are compulsory and specify where the data must come from. The WHERE, GROUP BY, query and ORDER BY clauses are optional. The WHERE component is used for restricting the selection and this restriction is realized by a **SQL condition**. A SQL condition is the optional logical condition that restricts the selected set of rows to those for which the condition is true. There are nine different types of conditions and they are listed in Table 5-1.

In this section the central question is whether the standard spatial GIS functions and operations are sufficient for implementing all different constraints or not. SQL is a way of interacting with databases and is actually part of a database, therefore a closer look to the possibilities of Oracle concerning SQL is necessary and this is done in section 5.2.3.

5.2.2 Integrity constraints in DBMSs

All DBMSs have the possibility to implement integrity constraints. A constraint in this context is more comprehensive than only the constraints for geo-VR, which are defined in chapter 3. Date and Darwen (1997) divide the integrity constraints of DBMSs into the following broad categories:

1. **domain constraints** – are associated with specific domain, and apply to every column (in every base table) that is defined on that domain.
2. **general constraints** (or assertions) – apply to arbitrary combinations of columns in arbitrary combinations of base tables.
3. **base table constraints** (including 'column constraints') – are associated with some specific base table 'column constraints'.

Domain constraints are constraints concerning (in this context) the value domain of the attributes. These constraints are not of interest for implementing the constraints that are defined for geo-VR applications.

General constraints have the following syntax: CREATE ASSERTION (assertion_name) CHECK (constraint_body). General constraitns are often referred to as assertions. An assertion is an expression, which never can be result in false, and the constraint_body in the asserton is a boolean SQL expression. An example assertion for SALIX-2 can be seen in Figure 5-8.

Base table constraints can refer not only to one base table, which could be concluded from the name, but can refer to many base table columns. There are three kinds of base table constraints:

1. candidate key definition – {PRIMARY KEY | UNIQUE} (column-commalist);
2. foreign key definition – FOREIGN KEY (column-commalist) references-definition;
3. check constraint definition – CHECK (conditional-expression).

```
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'))
)
```

Figure 5-8: Example of a general constraint (assertion).
The assertion represents the constraint: '**Bushes never lie inside water**'. Constraint_1 is the assertion_name and all text after CHECK is the constraint_body.

The general definition of integrity constraints for databases can be used to search for implementation possibilities of constraints for geo-VR applications in databases. The general constraints and base table constraints are able to define the same contents, so for the constraint implementation the general constraint can be used as well as the check base table constraint. In both integrity constraints an expression must be given that specifies the content of the constraint. To find out if the different types of constraints for geo-VR applications can be implemented using these expressions, a real database (Oracle) with its possibilities is looked at. This is done in the next subsection.

5.2.3 Integrity constraints in Oracle Spatial

In Oracle Spatial querying the data can be done using SQL. Furthermore Oracle Spatial provides some integrity constraints, they are (Oracle, 2002a):

- **not null** – a domain constraint that requires a column of a table contain no null values;
- **unique** – a base table constraint that requires that every value in a column is unique;
- **primary key** – a base table constraint, where the values in the group of one or more columns subject to this constraint constitute the unique identifier of the row;
- **foreign key** – a base table constraint, where the column or set of columns included in the definition of the referential integrity constraint reference a referenced key. The referenced key is the unique key or primary key of the same or different table that is referenced by a foreign key.
- **check** – a base table constraint requiring the specified condition to be true or unknown for every row of the table.

These are all base table or domain constraints and no general constraints appear. This is correct, because Oracle doesn't provide general constraints. The outcome of further investigation to other databases (Informix, Ingres, MySQL, PostgreSQL (PostGIS), IBM DB2) is that general constraints neither exist in these databases. This implies that general constraints only exist in theory! However, a general constraint can be reformulated in a base table constraint. So first an integrity rule can be formulated as a general constraint and than rewritten as a base table constraint. The advantage of general constraints is that the constraint doesn't have to refer to a table.

The check constraint of the base table constraints is the only constraint with the ability to formulate a specified condition. However, in Oracle this condition cannot contain any subqueries (same is true for other environments). This implies that the possibilities of the check constraint within Oracle are restricted to rather simple conditions.

For the more complicated integrity constraints, Oracle provides database triggers (and stored procedures) as integrity rules with a non-declarative approach. So Oracle provides not null, unique, primary key, foreign key and check constraints for 'simple' integrity rules and for complicated integrity constraints triggers and procedures can be used.

For geo-VR applications, mostly complicated constraints are defined, so it is worth looking at the possibilities for defining complicated constraints using database triggers. In Oracle a trigger definition consists of the components described in Table 5-2.

Within a database trigger the trigger event must be specified. In the trigger event the concerning table is specified and also the actions that make the trigger run. E.g. when the trigger event is of the type 'insert or update on <table>', then the trigger will be run when an insert or update on the table takes place.

The trigger time point specifies whether the trigger must be run before or after the trigger event. If the time point is 'before', the table is only mutated after the trigger is run and the mutation satisfies the

trigger body. If the time point is 'after', the table is first mutated and afterwards the trigger is run to check if the trigger body is satisfied. If not, an implicit rollback takes place to undo the mutation of the table.

The trigger type specifies if the trigger must be run for each row in the insert or update statement or not. If 'for each row' is not specified, the trigger runs only once for each statement.

When a for each row trigger is created a restriction can be specified in the form of a when condition. The trigger is only running for the rows in the statement that satisfy the when condition.

```
declare
        <constants>
        <variables>
        <cursor>
        <user defined exceptions>
begin
        <PL/SQL statements>
exception
        <exception handling>
end;
```

Figure 5-9: syntax of PL/SQL body in database trigger

The trigger body is a PL/SQL block where the actual constraint checking takes place. The structure of a PL/SQL block can be found in Figure 5-9. SQL and spatial operations and functions available within Oracle can be used inside the trigger body. The spatial operations and functions are listed in Appendix B.

Table 5-2: Components of the database triggers in Oracle

Component	syntax
trigger name	**create** [**or replace**] **trigger** < trigger name>
trigger time point	**before | after**
trigger event(s)	**insert or update** [**of** <column(s)] **or delete on** <table>
trigger type (optional)	**for each row**
trigger restriction (only for **for each row** triggesr)	**when** (condition)
trigger body	<PL/SQL block>

The possibilities for implementing the constraints that are formulated for this research using database triggers are listed below:

- The *spatial topology* constraints can be implemented with the SDO_RELATE operation, because this operation provides all topological relationships.
- The *spatial metric* constraints are divided in distance and directions. Distance constraints can be implemented with the SDO_WITHIN_DISTANCE operation. For direction constraints trigonometry operations (like sine, cosine and tangent) can be used in the PL/SQL block. These trigonometry operations have to be implemented in the PL/SQL block, so a little extra programming is necessary.
- The *temporal* constraints can be implemented using some timestamps and dates. Dates can be compared with each other in Oracle. This comparison of dates must be programmed in the PL/SQL block.
- The *quantity* constraints can be implemented with the count function of SQL (SELECT count(*) FROM <table_name>).
- The *thematic* constraints can be implemented with the standard SELECT-FROM-WHERE statement and specifying the right thematic attributes in the WHERE clause.

All different types of constraints for geo-VR applications can be implemented in Oracle using database triggers and for most types of constraints existing operations and functions can be used inside the trigger body. For implementing the directional and temporal constraints a little extra programming is necessary to check the constraints.

Besides using integrity constraints for simple constraints and database triggers for the more complex constraints in Oracle, also additional software can be used to implement constraints. Some examples are given in the next sections.

5.2.4 Implementing constraints using Oracle's CDM RuleFrame

Oracle provides a development tool called Custom Development Method (CDM) for generating code for the DBMS. The CDM RuleFrame is the business rules implementation framework of CDM. The components of the CDM RuleFrame can be seen in the Business Logic Layer of Figure 5-10.

When an insert statement is done, the Table API (TAPI) is called. This TAPI can handle the 'simple' integrity rules/constraints, like domain checks. However, for the more complex constraints, the TAPI calls the Custom API (CAPI).

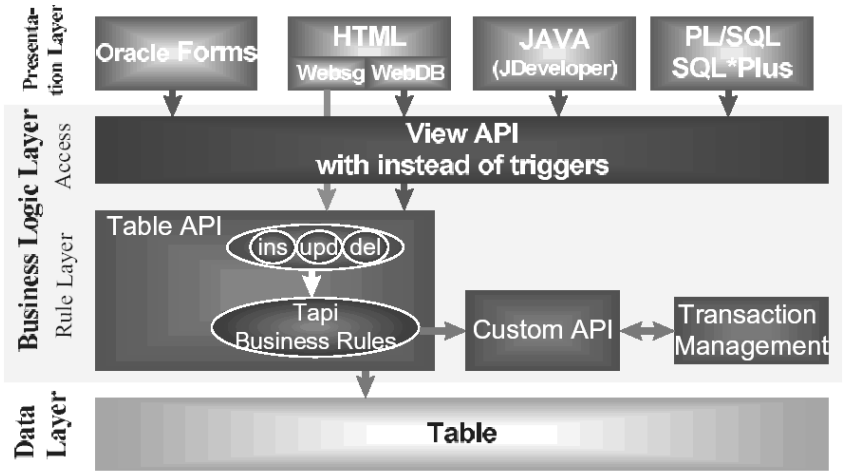

Figure 5-10: The structure of CDM RuleFrame.
The components of the CDM RuleFrame can be seen in the Rule Layer.
(from Muller, 2000)

The CAPI consists of one package per table for all custom rules. In other words, a container for all rules that Oracle Designer does not generate. All business rules (comparable to complex constraints) in the CAPI have a predefined structure. 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 (which is indicated in the CAPI) 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.

The CAPI offers some standard services to support the enforcement of rules. These services are (Muller, 2000):

- *Get char/num/date value.* These statements return the value of any column of the CAPI's table, to use when another CAPI needs the value;
- *Exists Row.* This statement determines whether a row exists with the properties provided;
- *Display label.* This statement returns a meaningful identification of a row. For example, the display label of a row in the plantation layout table of SALIX-2c could be a concatenation of the kind and type of an object: 'tree, FraxExc'. This can for example be used in error messages;
- *Aggregate value.* This statement performs calculation services (max, min, count, avg, sum) for a set of rows satisfying given conditions.

Look for example to the constraint: 'the tree must be of type 'FraxExc' '. For this constraint the service get char value can be used. The code of the constraint becomes:
constraint_ok := plantation_layout_capi.get_char_value(treetype='FraxExc');
The syntax here is: constraint_name := table.get_char_value(condition). This code is shorter and easier than using a SQL select statement (with the select, from, where syntax). However, the CDM RuleFrame is not further discussed here because it's too complex for this research. More information can be found in (Muller, 2000) or (Boyd, 2000).

5.2.5 Implementation constraints using ArcSDE

Besides or actually on top of a DBMS also other software can be used for a smoothly constraint implementation, for example ArcSDE, part of the ArcGIS product. ArcSDE is an abstract layer on top of a DBMS where defining integrity rules is possible (see Figure 5-11). On [URL 8] the following information can be found:

> ArcSDE is a middleware that allows you to store and manage spatial data in your chosen relational DBMS (RDBMS). ArcSDE works with IBM DB2, Informix, Microsoft SQL Server and Oracle.

ArcSDE (also) works as an application server, delivering spatial data to many kinds of applications and serving spatial data across the Internet.

ArcSDE manages the integrity of the point, line, and polygon information added to the database and won't allow ill-formed feature geometry to be inserted (for example, polygon boundaries must be closed). In addition, you can use the ArcSDE gateway with ArcInfo and ArcEditor to implement **additional integrity constraints** on the data model that aren't practical to implement in the DBMS itself. For example, you can add connectivity rules for utility networks.

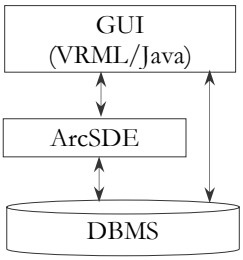

Figure 5-11: Possibilities for the implementation architecture of a geo-VR application.

The integrity constraints that can be implemented in ArcSDE are:
- attribute validation rules;
- network connectivity rules;
- relationship rules (based on topology).

The attribute validation rules check the validity of the attributes, e.g. 'polygons should not have a point with the same co-ordinates in the description' so the same point is not stored twice. These rules can also constrain the values allowed in any particular attribute for a table, feature class or subtype (ESRI, 2002b). This kind of integrity constraints is not of interest for this research.

Topology is the mechanism to find whether the network connectivity rules and the relationship rules are correct. The available topology rules in ArcSDE are listed in Appendix B. These rules can be implemented with a graphical user interface wizard and the user does not need SQL knowledge.

A network connectivity rule constrain the type of network features that may be connected to one another and the number of features of any particular type that can be connected to features of another type (ESRI, 2002b). If for example in SALIX-2 the roads and bridges were line features, a network connectivity rule could be that the roads should always be connected through bridges to reach the other side of water areas.

Relationship rules control which object subtypes from the origin class, can be related to which object subtypes in the destination class (ESRI, 2002b). An example of a relationship rule for SALIX-2 could be: a tree (point object) must be properly inside a grass area.

However, with the ability of defining topology rules, only a limited number of the spatial constraints can be implemented. The rest of the constraints (spatial metric, temporal, quantity and thematic constraints) cannot be implemented by defining topology rules between features or feature classes. This is not only a disadvantage for the ArcGIS software, but for all implementation approaches using only topology rules.

For the non-topological constraints the SQL possibilities within ArcSDE could help for the implementation. One can also use standard SQL in the ArcSDE software's API to perform attribute-only queries. So in addition to the topology rules, which are easy to implement in ArcSDE, SQL can be used for the thematic (attribute) queries. However, this can also be done in Oracle without ArcSDE so this software tool is not of further interest for this research.

5.2.6 Conclusion of implementation possibilities in DLM

The DLM provides enough possibilities to implement integrity constraints. However, the assertions (which are most suitable for the constraint implementation) are not available in existing DBMSs. DBMSs provide their own solutions for maintaining integrity.

Oracle Spatial provides domain and base table constraints for the simple constraints. For the more complicated constraints database triggers can be used in Oracle Spatial. Within the PL/SQL trigger body standard operations and functions, which are available in Oracle Spatial, can be used as well as SQL statements. Not all constraints for geo-VR applications can be implemented using the standard spatial functions and operations. For these constraints (temporal and directional) a little additional programming is necessary in the PL/SQL block of the triggers.

Other software packages exist to implement integrity constraints in a more user-friendly way, e.g. Oracle's CDM RuleFrame and topology rules in ArcSDE. In Oracle CDM RuleFrame the code for some constraints can be generated automatically (based on high level specification, like OCL) and in ArcSDE

topology rules can be defined with a graphical user interface wizard and programming is not necessary for these topology rules. This could be an advantage for implementing constraints in the DLM.

However, Oracle's CDM RuleFrame is beyond the scope of this research and ArcSDE is not sufficient enough for implementing all constraints, so programming is still necessary. Moreover, additional software packages are necessary for the geo-VR application. More complex system structures exist (and probably more expensive) and the question arises if this counterbalances the benefits the extra software provides. This question has to be answered for each geo-VR application separately, because the final purposes for each geo-VR application are different. With this also the discussion starts about using freeware or commercial software. This discussion continues in the next section.

5.3 Freeware or commercial software?

Before starting with the development (or modification) of an application, it should be clear who the end users of the application are and what the available budget for the application is. This information is necessary to decide what the final application structure should be. The question 'which software should be implemented on the client side and which software on the server side' is of importance. If an application is developed for interactive spatial planning for 'normal' citizens, the software and hardware requirements on the client side cannot be very high, because then hardly anyone can use the application. So in such cases one should rather choose for a simple client side, which can be constructed using freeware or possibly some additional standard hardware and software.

Looking for example to SALIX-2, the conclusion can be drawn, that only some plugins (freeware) are necessary on the client side and MSAcces is the used database, which is part of MS Office and a rather simple DBMS.

When implementing the constraints in the DLM of a geo-VR application, using Oracle Spatial and possibly ArcSDE on top of Oracle Spatial, the application will not be a simple application (anymore). Licenses for commercial software are expensive. However, using existing commercial software can have advantages in the field of the expectation of already implemented functions that are needed, the ability of getting more support and the reliability of existing software. Furthermore, the commercial software packages only have to be installed once on the server-side of the application and the client side of the application can still use freeware.

So for each application the structure, the budget and the final purposes must be determined. After that the software can be chosen that fits the conceptual model of the application. Geo-DBMSs are available as commercial geo-DBMSs, but also as free geo-DBMSs. The commercial geo-DBMSs are for example Oracle, IBM DB2, Informix, Ingres etc. and the free geo-DBMSs are MySQL and PostgreSQL/PostGIS. Within all these DBMSs the integrity constraints described in section 5.2.2 and database triggers (or similar solutions) are available. Which geo-DBMS eventually must be used for which application is, besides the budget and the final purposes, also dependent of the experience of the organisation developing the application.

5.4 Criteria for best implementation approach of constraints

In this chapter many implementation possibilities are described. There's not a best implementation approach for all applications. A list of criteria can help to decide which implementation approach is most desirable for an application. The most important criteria concern:

- end users (final purpose);
- budget;
- usage of the application;
- freeware / commercial software;
- expertise within organisation developing the application;
- interaction with constraints.

These criteria can be used as basis for a kind of decision table, as can be seen in Table 5-3. The combination of the decisions can give a good **indication** of the side where the constraints have to be implemented. A broad distinction can be made in:

1. implementing constraints in the DLM using freeware (e.g. MySQL);

2. implementing constraints in the DLM using a commercial software (e.g. existing DBMSs like Oracle or existing GIS software like ArcSDE);
3. implementing constraints in the DVM with a rather simple GUI and using freeware (e.g. VRML or Java);
4. implementing constraints in the DVM with a more complex GUI, based on commercial software (like the 3D Analyst extension of the ArcGIS package).

Each of these four possibilities is represented by a separate column in Table 5-3. The explanation of how this table works is done with an example.

Table 5-3: decision table for place of constraint implementation.
When a criterium does not fullfill the requirements of the application,

Four impl. approaches / Criteria	**DLM** Serverside implementation		**DVM** Client side implementation	
	1 freeware DBMS	**2** Commercial DBMS	**3** Simple GUI Freeware (e.g. VRML and Java)	**4** More complex GUI Com. software (e.g. 3D Analyst)
Budget	low	high	low	high
Experienced GIS users	yes or no	yes or no	no	yes
Wide usage of application	yes or no	yes or no	yes	no
Expertise of developers	If yes anywhere, use this expertise as much as possible			
Interaction with constraints	yes	yes	no	yes or no

The example concerns an application that has to be made:
- with low budget (columns 1 and 3);
- where the end users do not have experience with GIS (columns 1, 2 and 3);
- which should be widely used (columns 1, 2 and 3);
- by developers who have no experience with DBMSs, but with Java (column 3);
- where interaction with constraints is not necessary (columns 3 and 4).

After each application requirement, the column number is given which fulfils these requirements. For the described application, only column 3 fulfils all application requirements. This is an indication that the constraints can probably best be implemented in the DVM (or client side) of the application with freeware and a simple GUI, based on Java.

A comment can be made to the decision table. The table can only give an indication for the most suitable implementation approach, the other solutions are not excluded from being good implementation possibilities. Moreover, the applications of interest are geo-applications. It is desirable to use geo-DBMSs for the storage when working with geo-appliations. When using already a geo-DBMS for an application, the most suitable implementation approach for constraints is in almost all cases on the DLM side of the application. Expertise already exists and DBMSs offer existing operations and functions that can be of help for the constraints implementation.

5.5 Conclusion

Implementing constraints in an application can be done in the DVM and the DLM. VRML and Java can be used for the constraints implementation in the DVM. However, VRML offers not the right functionality for constraints. In Java a lot is possible with programming, but everything has to be designed, tested and debugged by hand.

In the DLM existing DBMSs offer standard functions and operations that can be of help for the constraint implementation. For geo-VR applications geo-DBMSs should be used for the storage of all data. Within Geo-DBMSs database integrity is available. However, these standard integrity possibilities are mostly not sufficient for the constraints for geo-VR applications. Other functionality, such as database triggers, also exist for the more complicated constraints.

Additional software, like CDM RuleFrame and ArcSDE, can be used for the implementation of constraints. The advantages are:
- orderly implementation possibilities for complex constraints with CDM RuleFrame
- easy implementation of topological constraints with ArcSDE

The disadvantages are:
- additional software is necessary, so the application becomes more complex
- with ArcSDE still programming is necessary for implementing all non-topological constraints.

For this research they are not used, because the advantages do not counterbalance the disadvantages and CDM RuleFrame is too complex for this research.

The **best** implementation approach for constraints in an application cannot be given, because each application has its onwn requirements. To get an indication of the most suitable implementation approach for an application, the requirements for each application have to be filled in in the decision table (Table 5-3). Also other solutions are possible, because they are not excluded from being good implementation approaches.

6. Conceptual model of SALIX-2c

SALIX-2 is used as example application to implement some constraints. For this constraint implementation, the most suitable approach is followed, which is found by filling in the decision table. This is described in section 6.1. Furthermore the conceptual model of SALIX-2 with constraints (SALIX-2c) is described in this chapter, preceded by an introduction of the Unified Modelling Language (UML) in section 6.2 and the Object Constraint Language (OCL) in section 6.3. UML can be used to model an application and OCL can be used to model the constraints. After these introductions the static system structures of SALIX-2 and SALIX-2c are given in 6.4. In section 6.5 the communication between the application and the user is discussed. This communication is part of the dynamic system structures, which can be found in 6.6. After designing the system structure of SALIX-2c is also the constraints must be designed, this is done in section 6.7.

6.1 Most suitable implementation approach

The most suitable implementation approach can be found by filling in the application requirements in the decision table (Table 6-1). The application requirements for SALIX-2 are:

- the budget is very low, but the DBMS Oracle is already available;
- the users are mostly students with some experience with GIS;
- the application is ment for wide usage and distribution over the internet;
- there is experience with Oracle and MySQL;
- interaction with the constraints is desirable.

Table 6-1: decision table with requirements for SALIX-2

Four impl. approaches / Criteria	**DLM** Serverside implementation		**DVM** Client side implementation	
	1 Freeware DBMS	**2** Commercial DBMS	**3** Simple GUI Freeware (e.g. VRML and Java)	**4** More complex GUI Com. software (e.g. 3D Analyst)
Budget	Low		Low	
Experienced GIS users	Yes	Yes		Yes
Wide usage of application	Yes	Yes	Yes	
Expertise of developers	Yes (MySQL)	Yes (Oracle)		Yes
Interaction with constraints	Yes	Yes		Yes

The low budget criterium is very hard. So it is necessary to use as much as possible existing expertise, already available (also commercial) software and the existing program code. Furthermore, the DVM implementation is less suitable than the DLM implementation (more crosses in the decision table).

So a DLM implementation with usage of existing expertise, software and program code is most suitable. Existing geo-DBMSs within Alterra and WUR are Oracle and MySQL and also expertise with both DBMSs is available. Oracle is used for this research (widely used DBMS and orderly tutorials). Although Oracle is an expensive commercial DBMS, this DBMS is already available and can act as a server. So the low budget criterium is not violated. The client side of the application will stay simple to fulfil the low budget and wide usage criterium.

6.2 Unified Modelling Language

UML is a standard used for object oriented system development. It offers some diagrams that together form the model of the system. The available diagrams within UML are (Warmer, 2001):

- use-case diagrams – shows how the system can be used by external entities such as human users;

- class diagram – the static structure of a software system as classes and their relations;
- object diagram – the static structure of the software system as objects and their relations;
- sequence diagram – the order in time of messages send and received in the system;
- collaboration diagram – shows how objects collaborate to reach a goal;
- state diagram – shows the possible states of the objects during it´s lifetime;
- activity diagrams – shows activities executed by the different parts of the system;
- component diagram – shows the system components and their relations;
- deployment diagram – shows the usage of the software components.

The possible diagrams can be used to conceptualise, analyse, design and implement the system. All objects, components and relations can be outlined. In this report the class diagrams are used to model the static system structures of SALIX-2 and SALIX-2c. The state diagrams are used to model the dynamic system structures.

UML class diagrams show the classes of the system, their inter-relationships, and the operations and attributes of the classes [URL 10]. A class is a collection of objects with similar properties. The class description gives the name of the class and the description of the properties and instances. These properties are divided into attributes and operations. The design of a class can be found in Figure 6-1.

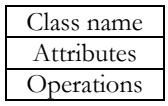

Figure 6-1: design of a class

Between classes some associations and relations (e.g. composition) can exist. Also some notes can be added to the diagram to give additional information. These notes can be related to a specific class.

State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system. [URL 9]

6.3 Object Constraints Language

OCL is a notational language for analysis and design of software systems and is part of UML. OCL enables one to describe expressions and constraints on object-oriented models and other object modeling artifacts. In this context an *expression* is an indication or specification of a value and a *constraint* is a restriction on one or more values of (part of) an object-oriented model or system [URL 5].

The following types of constraints exist in OCL [URL 5]:

- 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. (these constraints are used for SALIX-2c).
- A *precondition* to an operation is a restriction that must be true at the moment that the operation is going to be executed.
- A *postcondition* 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. An example of the attribute constraint 'a QueRob (one of the treetypes in SALIX-2) can never become older than 35 years' is:

context QueRob **inv**:
self.age <= 35

The context of the constraint is the QueRob, the constraint type is invariant and the constrained attribute is age. 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.

In OCL the standard operation types are integer, real, string and Boolean. These 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.

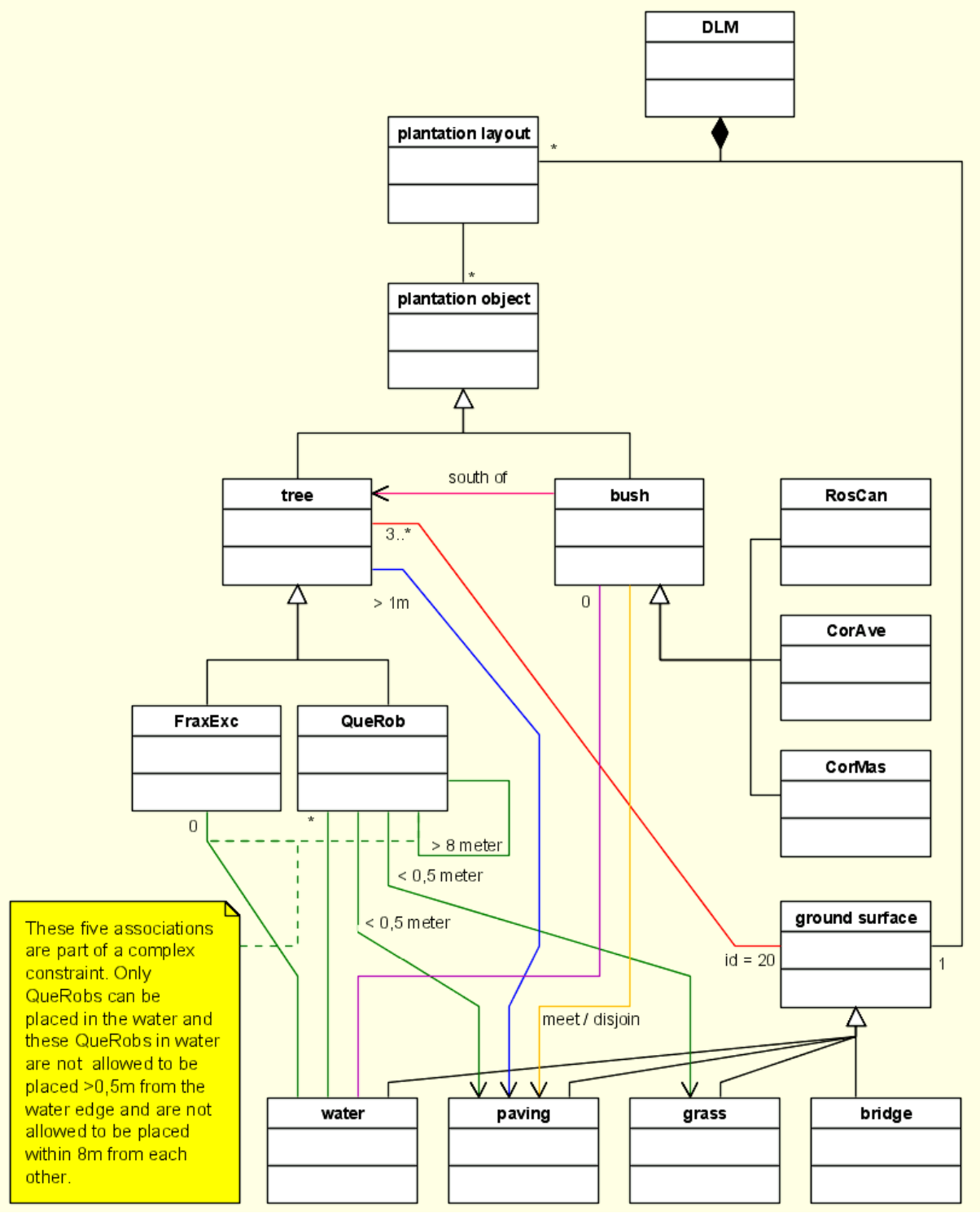

Figure 6-2: UML class diagram representing the objects in the DLM of SALIX-2c. The constraints are visualized as restricted associations between object classes.

6.4 Static system structure of SALIX-2c

Besides formulating constraints using OCL, the constraints can also be visualized in a UML class diagram. This is done for SALIX-2c. A class diagram representing the static system structure of SALIX-2 is given in Appendix A. This class diagram is rewritten to a class diagram representing the objects of the application in the DLM. These object classes including the constraints can be seen in Figure 6-2. The constraints are visualized as restricted associations between object classes. Note that this is only the DLM part of SALIX-2c.

The plantation objects in Figure 6-2 are categorized in trees and bushes, the trees and bushes are in turn also categorized in sub-classes (FraxExc, QueRob, RosCan, CorAve, CorMas). In this way the relations between specific object classes can be visualized more clearly. The associations between object classes can have directions, visualized by arrows at the association ends, and can also represent specific relations, which are visualized by text in the middle of the association. With these notations a complex system structure with constraints can be visualized in a class diagram.

6.5 Communication between application and user

The constraints that are visualized in the UML class diagram in Figure 6-2 can be implemented in the application. If constraints are implemented, the user has to know which constraints exist. When the user edits the 3D model (for SALIX-2c this would be modifying the VRML plantation layout) the user also needs some feedback about the validity of the modifications he/she made. A picture is more than thousand words, so a visual feedback is desirable. However, the detail of the feedback information can be limited when using visual feedback. Furthermore, it can be more complicated to implement visual feedback (instead of textual feedback). Therefore a list of constraints (textual) should be presented when the application is started in combination with textual feedback when a constraint is violated.

So feedback to the user can be given on several moments:
- before any changes are made, for example show transparent red areas where plantation objects are not allowed to be placed (by adding an additional layer) or show a list of all implemented constraints;
- after changes are made, for example colour the plantation objects red or give textual error message;
- a combination of these two.

The multi-view approach described by E.Verbree et al. (1999, described in section 2.2) would be a very nice solution for displaying the constraints on a 2D map and simultaneously in the 3D model. If also additional textual feedback is given, the user knows exactly what can be done and what not. This solution is however too extensive for the example implementation of SALIX-2. For this implementation only textual feedback is given to the user in the textbox of the Java console. This is also implemented in the flowcharts of actions in the next section.

6.6 Dynamic system structure of SALIX-2c

The dynamic structures of SALIX-2 and SALIX-2c are discussed in this section. For each interaction in the application a state diagram can be made. The possible interactions are:
- starting the application;
- loading a plantation layout;
- making a new plantation layout;
- adding an object;
- deleting an object;
- dragging an object to another position;
- saving the plantation layout.

The state diagrams are given for both SALIX-2 as SALIX-2c. When the state diagrams of SALIX-2c are not different from the state diagrams of SALIX-2, only one state diagram is given.

The most remarkable changes in SALIX-2c are:

- MSAccess is replaced by the geo-DBMS Oracle Spatial;
- Database triggers are used to implement the constraints in the database;
- The ground surface is stored in Oracle Spatial as a set of geo-objects, but for the visualization still an image is used in the VRML scene;
- All data is stored as geodata in RD coordinates in Oracle Spatial. VRML becomes very slow (too slow) when working with the large RD coordinates, so within the VRML browser the application works with local VRML coordinates. A transformation must take place when loading the geodata from the DBMS and also when saving the plantation layouts in the DBMS.

6.6.1 Starting the application

The start of the application is the same for SALIX-2 as for SALIX-2c. The order of the actions that take place while the application is being opened, is described below and these actions can be seen in Figure 6-3.

1. SALIX-2/SALIX-2c has to be started with index.htm.
2. A VRML browser is started and loaded with the base scene (land, sky, ground surface) and the java panel with interaction possibilities appears (buttons, scroll down menus).
3. A connection is made between the java panel and the VRML browser.
4. A connection to the database is made to add the available treetypes and plantation layouts to the scroll down menus in the java panel.

The database in SALIX-2 is MSAccess and the database in SALIX-2c is Oracle Spatial.

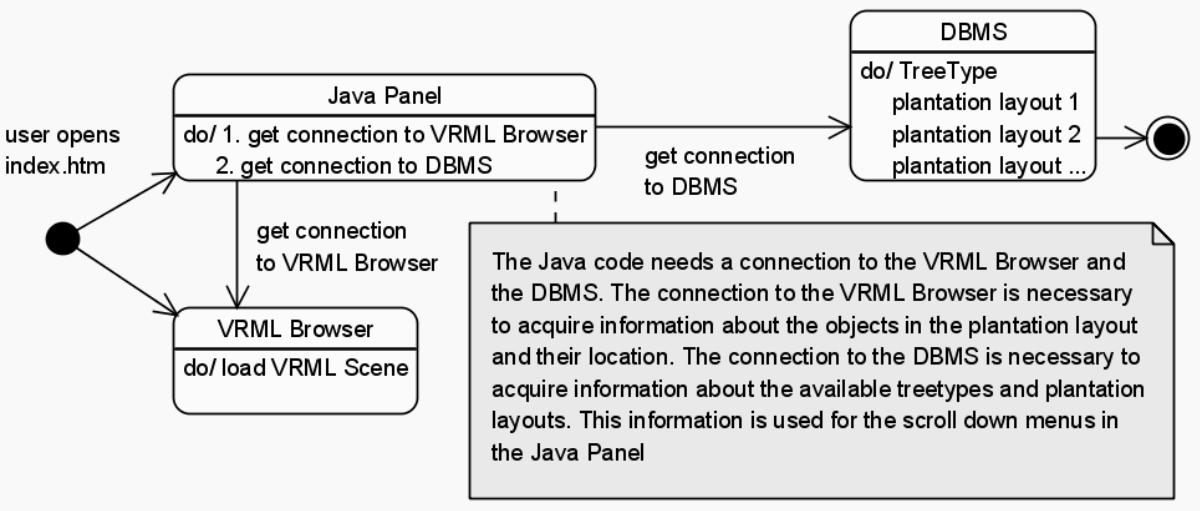

Figure 6-3: state diagram when starting SALIX-2 and SALIX-2c

Now SALIX-2/SALIX-2c is ready for use. A choice can be made to load an existing plantation layout or make a new one. After a plantation layout is loaded and the objects are displayed in the new VRML scene or after a plantation layout is created, interaction is possible in SALIX-2/SALIX-2c.

6.6.2 Making a new plantation layout

Making a new plantation layout in SALIX-2 and SALIX-2c is actually making a new table in the database with the specified name (CREATE TABLE 'table_name' (TreeID NUMBER PRIMARY KEY, TreeType STRING, TreePosX DOUBLE, TreePosY DOUBLE, TreePosZ DOUBLE, TreeAge DOUBLE, Solitair BIT). The accompanying state diagram can be seen in Figure 6-4.

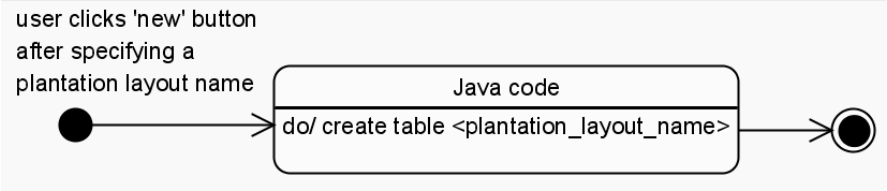

Figure 6-4: state diagram of creating a new plantation layout in SALIX-2

For SALIX-2c it is also necessary to assign the triggers to the correct plantation layout table. Furthermore the constraints for that plantation layout must be listed in the textbox of the Java console, so the user knows which constraints exist. The state diagram for SALIX-2c can be seen in Figure 6-5.

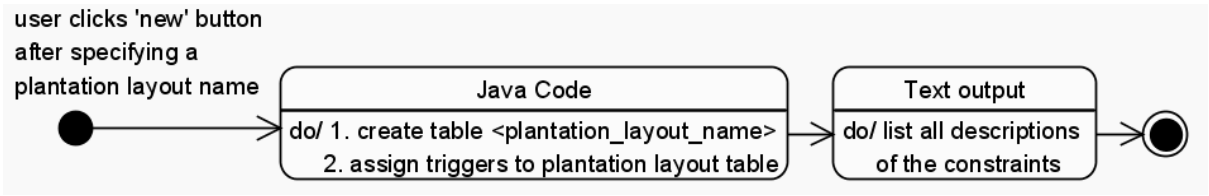

Figure 6-5: state diagram of actions when making a new plantation layout in SALIX-2c.

6.6.3 Loading a plantation layout

Loading a plantation layout in SALIX-2/SALIX-2c is actually getting the object attributes from a plantation layout table and storing these attributes in the vector DBObjects. This vector is used to store the object attributes (objectid, type, x, y, z, age, solitair) temporarely, so the frequency of connecting to the database is limited. For SALIX-2 the object attributes are acquired with the SELECT * FROM 'table_name' statement. The object attributes are used to import the objects in the VRML scene (create VRML from string). For SALIX-2 the 'load' operation is finished. The state diagram of these actions can be seen in Figure 6-6.

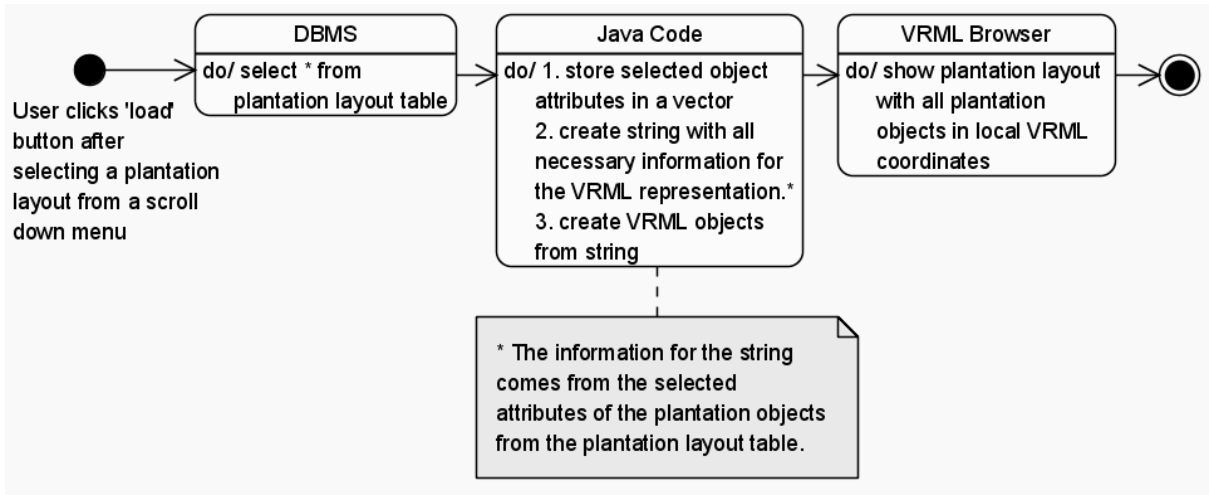

Figure 6-6: state diagram when loading a plantation layout in SALIX-2

For SALIX-2c the location of the plantation objects is stored as sdo_geometry in RD coordinates. These coordinates have to be selected separately (xrd, yrd and zrd), because it is necessary to transform the RD coordinates to local VRML coordinates. When all attributes are available, including the local VRML coordinates, a 'VRML from string' creation can be done and the objects can be imported in the VRML scene. Then the constraints for this plantation layout must be listed in the textbox of the Java panel to inform the user about the constraints. The flowcharts of all these actions can be seen in Figure 6-7.

When a plantation layout is visualized in VRML (after making a new one or loading an existing one) the user can add, delete or replace objects. In SALIX-2 these changes are temporarily stored to the vector with all object attributes (including the local VRML coordinates). Only when the plantation layout is saved this vector is used to insert all data in the database (with local VRML coordinates).

Maintaining this structure within SALIX–2c, the constraints are only checked when the plantation layout is saved, that is when the user clicks the 'save' button. It can be desirable to check every change separately, so the 'save' button must be clicked after each change. However, also constraints can exist that can only be true when more than one change is made (e.g. the grass surface with id 20 must have 3 trees placed on it). The user has to wait with clicking the 'save' button till all necessary plantaion objects are placed. Feedback is only given when the 'save' button is clicked. So the user has to know that it is desirable to click the 'save' button regularly to get feedback. This information can be given in the users

guide of the application. The state diagram when the plantation layout is changed (add, delete, replace objects) are the same for SALIX-2 and SALIX-2c and are shown in Figure 6-8 till Figure 6-12.

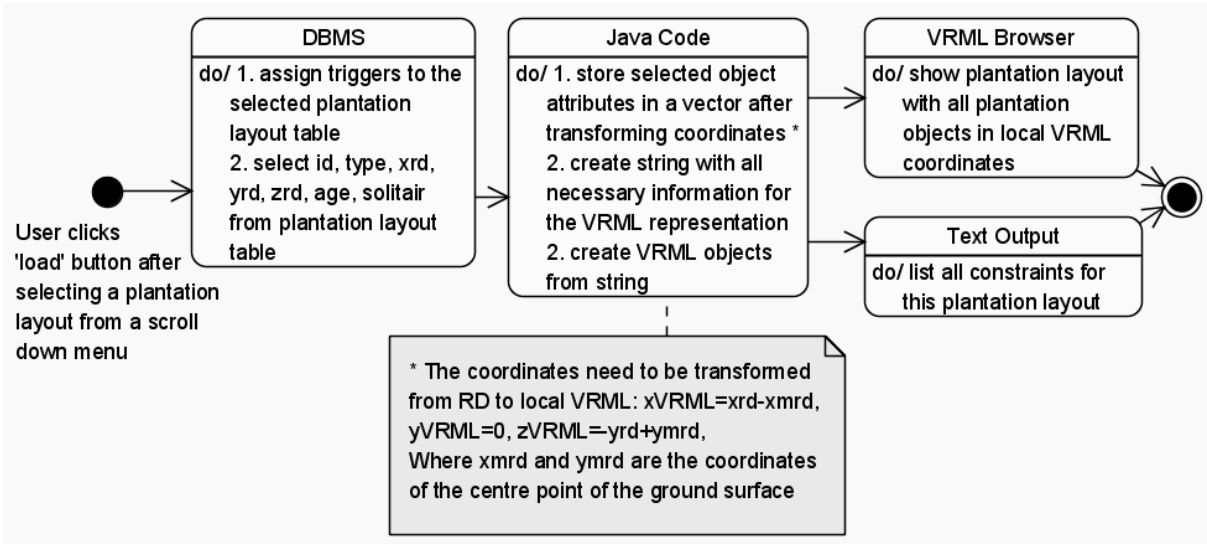

Figure 6-7: state diagram when loading a plantation layout in SALIX-2c

6.6.4 Adding a new object

One of the interaction possibilities is adding a new object to the VRML scene. The button 'create' has to be clicked after selecting the type and behaviour of the new object from the scroll down menus. For SALIX-2 and SALIX-2c the actions that follow are:

3. the Vrml.ToggleLocationSelection is set to true (in the program code). This activates the callback method and the VRML browser. The text on the 'create' button is changed in 'click place'.
4. When a location is clicked in the VRML Scene, the callback method is invoked. This method enables Java to catch information (location) from the VRML browser.
5. the object is added to the DBObjects vector. For this the selected type, behaviour and the clicked location in the VRML browser is necessary.
6. the action addNode is invoked to create the object in the VRML world. The browser creates a VRML from the string and adds the object to the scene.

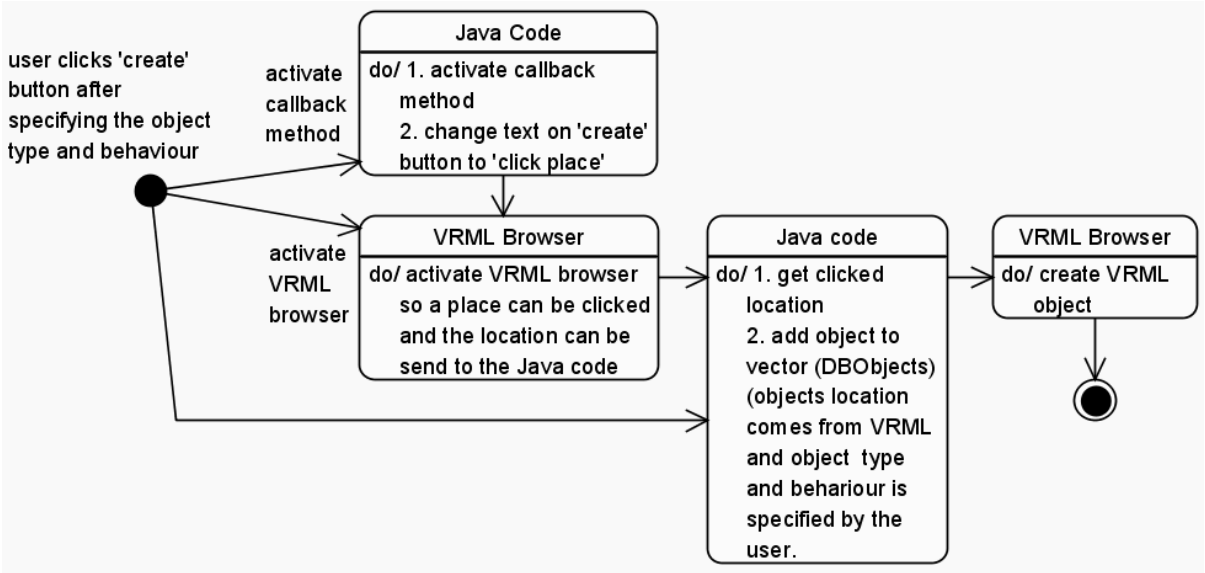

Figure 6-8: state diagram when making a new object in the plantation layout

6.6.5 Drag object to another position

Another interaction with the plantation plan is dragging an object to another position. Actions that take place in SALIX-2 and SALIX-2c are:

1. the object is dragged to another position, is visualized in the VRML Scene and an eventOutSFVec3f translation is generated
2. This translation is sent to the callback method of the Java code and the position in the VRML scene is acquired. This is necessary to save the new tree position in the DBObjects vector (only set new position (x,y,z) of the right object in the vector).
3. if the object is not the last moved object, the Architectural object where this tree is involved in, is removed and the tree is given the right colour. If the object is not the last moved object, all necessary actions are done.

During these actions, there is neither a connection to the database. The state diagram of these actions can be seen in Figure 6-9.

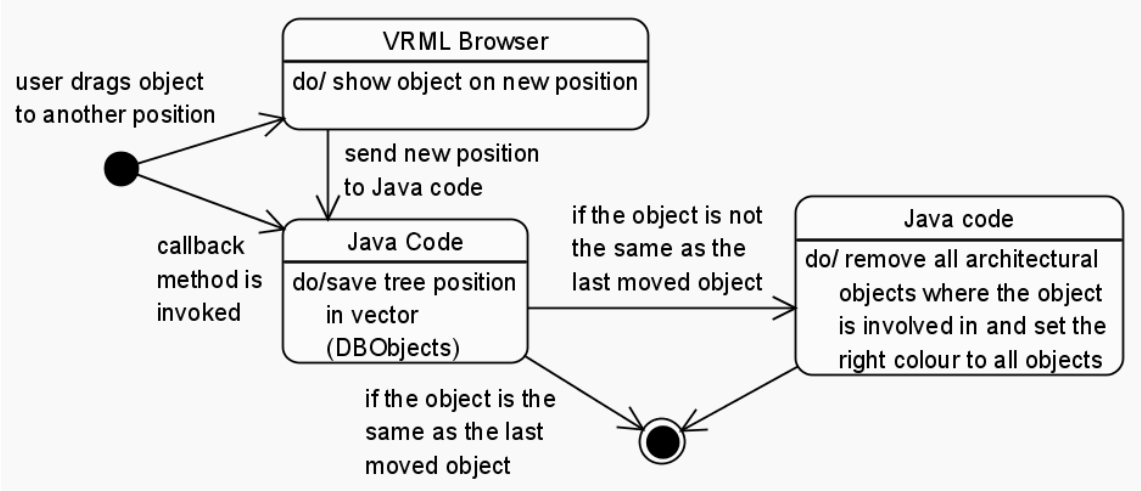

Figure 6-9: state diagram of actions when dragging an object in SALIX-2/SALIX-2c

6.6.6 Deleting an object

An object can be deleted from the plantation layout of SALIX-2 and SALIX-2c by clicking the button 'delete' and after that clicking an object in the VRML Scene. The object disappears from the VRML scene but not from the DBObjects vector. The object will only be set to invisible. This is done to overcome problems with the object id's of the objects. When all objects are still in the vector, a new object can get a new unique object id, directly following on the highest object id existing in the vector. In the VRML Scene all architectural objects are removed where the deleted object was involved in. Figure 6-10 shows the state diagram of actions when deleting an object from the plantation layout.

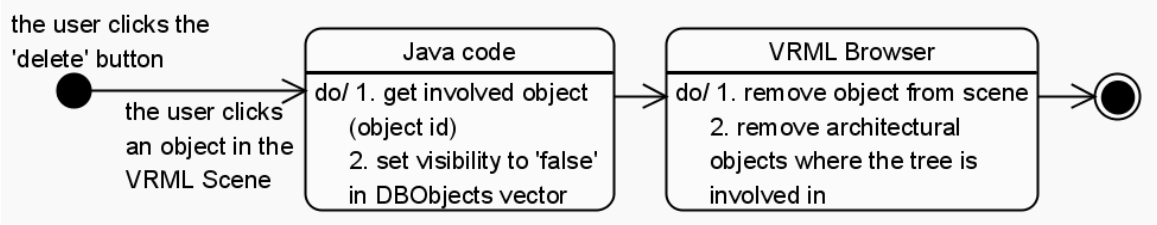

Figure 6-10: state diagram when deleting an object from the plantation layout in SALIX-2/SalIX-2c

6.6.7 Saving a plantation plan

The actions when saving the plantation layout in SALIX-2c is very different from the actions when saving in SALIX-2. When the user wants to save the plantation layout in SALIX-2, the 'save' button must be clicked. The actions that follow can be seen in Figure 6-11 and are:

1. First of all the selected plantation layout name is set as table name
2. the user is notified that some actions are taking place by the message: 'saving file…'
3. In the DBMS everything in the plantation layout table is deleted (delete * from <table_name>)

4. In the Java code the DBObjects vector is organised: all the objects that are set to invisible during interaction with the VRML scene, are now deleted from the DBObjects vector and the id's of the remaining objects are replaced by new (continuous) id's.
5. The new DBObjects vector is used to insert the necessary data in the plantation layout table in the DBMS.
6. The user is notified that the 'save' operation is finished by the message 'done'.

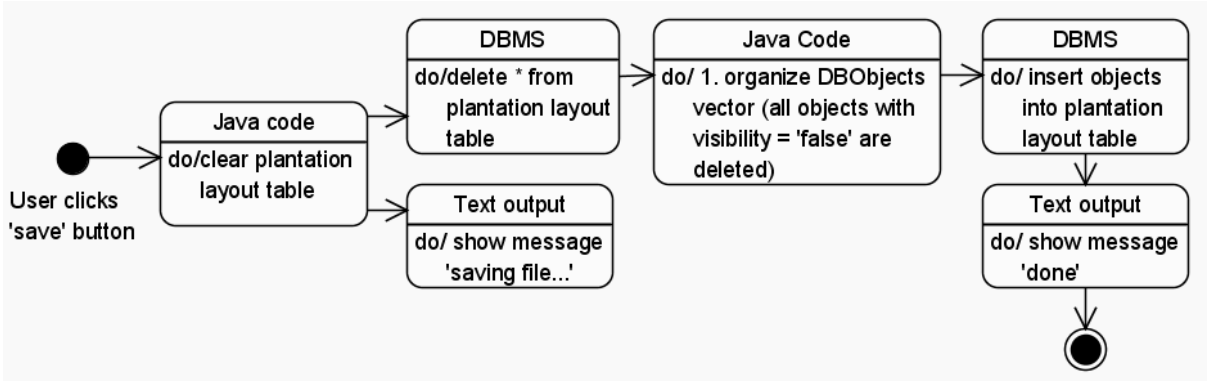

Figure 6-11: state diagram of actions when saving a plantation layout in the DMBS for SALIX-2

For SALIX-2c some additional actions are necessary, because this is the moment of the constraint checking. However, the first part is the same. The actions in SALIX-2c are:

1. the table_name is set to the selected plantation layout name
2. all objects are deleted from the plantation layout table in the DBMS
3. the message 'saving file...' appears in the textbox
4. the DBObjects vector is organized.

From here the actions are different for SALIX-2c:

5. The local VRML coordinates in the DBOBjects vector have to be transformed to RD coordinates. These RD coordinates are used to store the geometry of the objects (as sdo_geometry)
6. the objects are inserted in the plantation layout table. This is all done by separate insert statements. For each insert statement the triggers are checked. Note that constraints that can only be true when more than one change is made at the same time, cannot be implied in SALIX-2c.
7. when all triggers are satisfied for all insert statements, the message 'done' appears
8. when a trigger is not satisfied, that insert statement is made undone and the remaining objects are inserted. When in the end all insert statements are finished, error messages appear in the textbox output.

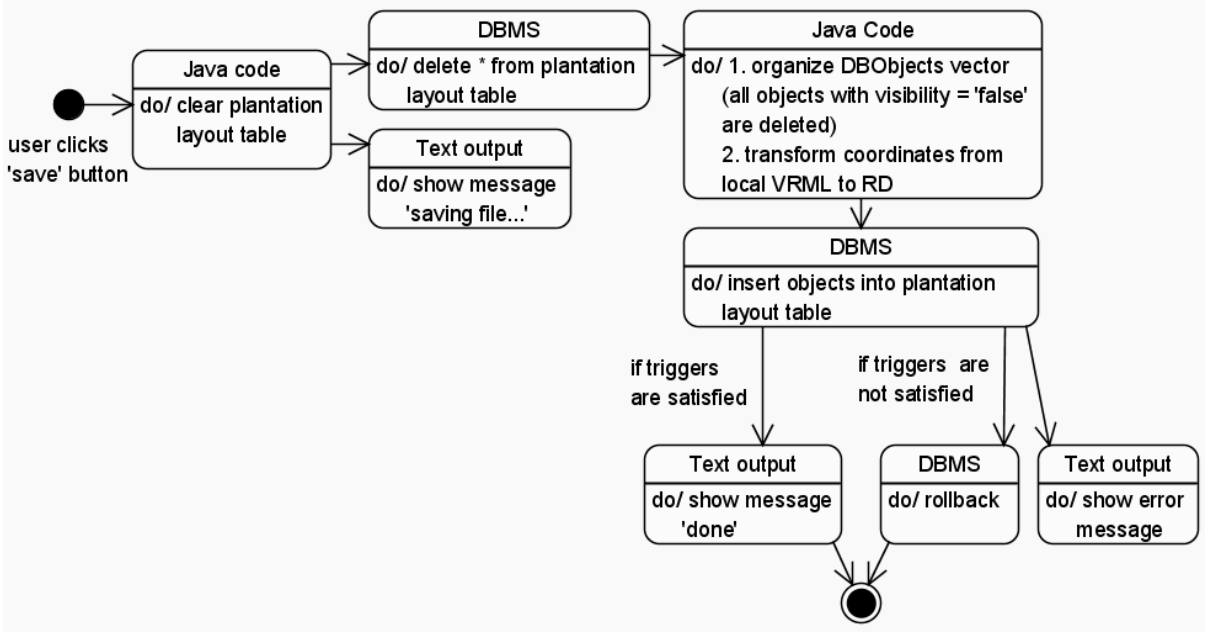

Figure 6-12: state diagram when saving a plantation layout in SALIX-2c

6.7 Conceptual Design of triggers to implement constraints

Besides designing the dynamic system structure for SALIX-2c, also the real constraint checking must be designed. The possibilities for the constraint implementation in the database are already described (section 5.2). In Oracle database triggers can be created for the implementation of the more complex constraints. The PL/SQL statement that forms the trigger body can contain spatial operations and geometry functions of Oracle, so the topological relationships can be tested, buffers can be made and distances can be computed. With these options and some if-then-else statements, the example constraints can be implemented in SALIX-2c. When stored procedures are used for the constraint checking, the same functionality (e.g. check direction of an object) can be re-used.

Before the triggers are created it is recommended to first formulate the constraints as general constraints (assertions). Although general constraints are not available in Oracle Spatial, the assertions have the advantage of only referring to just one database table. However, select statements can be used in the expressions of the assertion and select statements always concern database tables, so then one or more of the involved tables must be defined. After the constraints are formulated as general constraints, they can be rewritten as database triggers. Database triggers can be created (some (semi-) automatic with CDM RuleFrame) in Oracle Spatial.

6.7.1 The example constraints as assertions

In this subsection the example constraints for the implementation are formulated as assertions. The tables of SALIX-2 that are used in the assertions are:

- prcv_treesrd_point: a certain plantation layout of plantation objects;
- prcv_gvkrd_poly: the ground surface;
- involved_object: a temporary table filled with the attributes of the object that is inserted. (For the real implementation of database triggers in Oracle, this table is not needed anymore, because a package is used to store attributes of the involved object.)

For each example constraint a separate assertion is formulated.

1. 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'))
```

2. 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 (this location comes from the table involved_object) and can be used in the assertion.

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

3. 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'))
```

4. 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. A bush always has to meet or disjoint paved areas

The opposite of meet or disjoint is for point objects inside. This relation is used in the assertion below.

```
create assertion constrain_5 check
(not exists     (SELECT * FROM prcv_treesrd_point t, prcv_gvkrd_poly g
                WHERE t.treetype IN ('CorAve', 'CorMas', 'RosCan')
                AND g.descript IN ('paving', 'soft_paving')
                AND sdo_relate (t.geom., g.geom., 'mask=INSIDE,
                querytype=window')='TRUE'))
```

6. a) the distances between trees inside water is > 8 meter AND
b) the distance between the tree and the edge of the water always is < 0,5 meter AND
c) the only treetypes that are allowed in the water are QueRobs (trees of type Quercus)

This complex constraint concerns all objects in the water. For constraint 6 b) the geometry of water is necessary for checking whether the QueRobs are in the water. But the grass and paving geometries are also necessary to check whether the QueRob in water is not >0,5m from the water edge. This is necessary because a negative buffer of water areas is not possible, so all other areas except the water (and the bridges) are used for this constraint checking.

```
create assertion constraint_6 check
(not exists   (SELECT * FROM prcv_treesrd_point t, prcv_gvkrd_poly g, involved_object i
              WHERE t.treetype = 'QueRob'
              AND g.descript='water'
              AND i.treetype = 'QueRob'
              AND t.treeid != i.treeid
              AND sdo_relate (t.geom, g.geom, 'mask=INSIDE,
              querytype=window')='TRUE'
              AND sdo_relate (i.geom, g.geom, 'mask=INSIDE,
              querytype=window')='TRUE'
              AND sdo_within_distance (t.geom, i.geom, 'distance=8')='TRUE')
AND
not exists    (SELECT * FROM prcv_gvkrd_poly g, involved_object i
              WHERE i.treetype = 'QueRob'
              AND g.descript='water'
              AND sdo_relate (i.geom, g.geom, 'mask=INSIDE,
              querytype=window')='TRUE'
              AND
              exists   (SELECT * FROM prcv_gvkrd_poly g, involved_object i
                       WHERE sdo_relate (i.geom, sdo_geom.sdo_buffer(g.geom, 0.5, 0.005),
                       'mask=disjoint, querytype=window')='TRUE'
                       AND g.descript IN ('paving', 'soft_paving', 'grass')))
AND
not exists    (SELECT * FROM prcv_treesrd_point t, prcv_gvkrd_poly g
              WHERE t.treetype != 'QueRob'
              AND g.descript='water'
              AND sdo_relate(g.geom, t.geom, 'mask=CONTAINS,
              querytype=window')='TRUE'))
```

6.7.2 The example constraints as database triggers

The assertions listed in the previous subsection are all general constraints and can't be implemented in Oracle (or other DBMS) because the DBMSs do not support general constraints. The assertions have to be converted to PL/SQL code to use inside database triggers and this conversion must be done by hand (although CDM Ruleframe has automatic conversion possibilities for some assertions).

Within assertions it is not specified whether the check has to take place before any mutation on a table takes place or after a mutation took place. Database triggers on the other hand, can be before triggers or after triggers, which is described in section 5.2.3. Also a difference is made in for each row triggers and statement triggers.

Before the conversion takes place from an assertion to a database trigger, it must be clear whether the database trigger will become a before or after and a for each row or a statement trigger. If more than one trigger is created in an Oracle database, the order of execution of the triggers is (ORACLE, 2002a):

1. all before statement triggers;
2. all before row triggers;
3. all after row triggers;
4. all after statement triggers.

When there are two or more triggers of the same type the order of execution of these triggers is arbitrarily. For the example constraints it doesn't matter in which order they are checked, because they do not influence each other.

Besides the order of execution also other aspects influence the decision of making before or after for each row or statement triggers. Inside the assertions and triggers data about the plantation layout and the ground surface is necessary to check the constraints. This data is acquired by querying the plantation layout table and the ground surface table. The constraint checking must be done for the new plantation layout, so the triggers should be after triggers. Triggers are also created for one plantation layout table and (in Oracle) this table can only be queried inside the trigger body of after statement triggers. This implies that all triggers that query the plantation layout table have to be after statement triggers or the constraint check must be done inside stored procedures.

So first the table is updated and secondly the database triggers on the plantation layout (prcv_treesrd_point) table are run and all necessary data is acquired from the plantation layout table and the ground surface table. When in SALIX-2c the plantation layout is saved, the objects are inserted into the database table one by one. This implies that it doesn't matter if the after triggers are row triggers or statement triggers. When the triggers are violated, an implicit rollback takes place to undo the changes of the table.

In the above the assumptions is made to use one trigger per constraint. In Oracle it is possible to enable and disable database triggers, so when creating separate triggers for each constraint the interaction possibilities of the constraints by the user can be extended.

A description of the technical implementation of creating the database triggers and al lot of remarks and changes to this initial conceptual design of the database triggers is given in the next chapter.

7. Constraint implementation in SALIX-2c

After the conceptual design of the constraint implementation using database triggers, the implementation of this conceptual design must be done. Before the database triggers are implemented, some required modifications must be made, which are described in Appendix D. Section 7.1 describes the implementation of the conceptual model. When the database triggers are created in the database, the visual environment of SALIX-2c must also be changed slightly. A description of the implemented adaptations is given in section 7.2.

7.1 Constraint implementation in DLM

In the conceptual design for the constraint implementation the decision was made to use separate after statement triggers. Inside the trigger body the already modified plantation layout table is used to acquire information that is necessary for the computations and comparisons for the constraint checking. The plantation objects are inserted one by one in the plantation layout table and the id's of the involved objects must be known for the constraint checking. The new attribute values (including the id) of the object in the insert statement can only be invoked in a before each row trigger. This implies that a before each row trigger is required to get the id of the involved object beforehand, then the table is updated followed by the constraint checking in the after statement triggers.

The new object id and object type of the involved object are stored as variables in a package. These variables can be queried in all other triggers and also in stored procedures and functions. The stored procedures and functions in turn, can be invoked (more than once) inside e.g. trigger bodies. The stored procedures and functions can also be stored in packages. One package can be used to combine all procedures, functions and variables belonging to each other. The re-usage and the organized way of storing the functions, procedures and variables make structured programming of the database triggers possible.

Finally one before each row trigger is used to store the new object id and an after statement trigger is used to invoke the procedures and functions that do the actual constraint checking. For each constraint checking a separate stored procedure is used. The disadvantage of using only one trigger for the constraint implementation is that the possibility to enable or disable separate constraints (as database triggers) by the user disappears. This implies that the constraint interaction possibilities are rather limited. However, using only one trigger is less complex. So the eventual constraint checking is organized as follows:

- For each insert statement[1] of one object on the plantation layout table, a before each row trigger is invoked. This trigger first stores all new attribute values of the involved object as variables in a package, called pck_salix. These attributes are easily accessible and can be used in all procedures and functions.
- All procedures and functions are also stored in the package pck_salix, so they are stored in an organized way.
- all constraints that cannot have the involved object as starting point are checked in the after statement trigger (for this research this will be the quantity constraint that concerns a specific ground surface polygon).
- In the next part of the after statement trigger 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).

The codes of all triggers, packages, procedures and functions are described in Appendix D.

The eventual objects for the constraint checking are:

- Table prcv_treesrd_point with one plantation layout (the test plantation layout)

[1] delete and update are left out, because in SALIX-2/SALIX-2c first all objects are deleted, followed by insert statements

- Table prcv_gvkrd_poly with ground surface
- Table treetype with all possible object types in SALIX-2/SALIX-2c and their description
- Pck_salix is the package with definition of variables, procedures and functions
- Pck_body_salix is the package body with the source of the procedures and functions, used for the constraint checking.
- Brt_all_salix is a before row trigger that gives the predefined variables (id and type) the values of the new involved object.
- Ast_salix is an after statement trigger where all procedures and functions are invoked to check constraints.

After the trigger, functions, procedures and packages are created, some insert statements were done to test the database trigger. Below the results of two insert statements can be found. The textual feedback that is generated by the trigger and the elapsed time for the constraint checking are also given.

```
SQL> @ c6-insert_tree1_in_water.sql;
SQL> -- try to insert tree too close to paving.
SQL>
SQL> INSERT INTO prcv treesrd point
  2  (treeid, TreeType, TreeAge, Solitair, Kind,
  3  geom)
  4  VALUES
  5  (236, 'QueRob', 12, 1, 'tree',
  6  MDSYS.SDO GEOMETRY(2001, NULL, MDSYS.SDO POINT TYPE(105094.12, 482778.40, N
ULL), NULL, NULL));
4: there are enough trees (>= 3)
      on polygon 20
the involved_object is a tree
3: the tree is placed >1m from the paving
6a: the tree of type QueRob is placed far enough from other QueRobs inside the
water.
6b: the object is placed <0.5m from the wateredge

1 row created.

Elapsed: 00:00:02.05
SQL>
SQL> @ c6-insert tree2 in water.sql;
SQL> -- try to insert tree too far from water edge
SQL>
SQL> INSERT INTO prcv treesrd point
  2  (treeid, TreeType, TreeAge, Solitair, Kind,
  3  geom)
  4  VALUES
  5  (237, 'QueRob', 12, 1, 'tree',
  6  MDSYS.SDO_GEOMETRY(2001, NULL, MDSYS.SDO_POINT_TYPE(105097.02, 482777.40, N
ULL), NULL, NULL));
INSERT INTO prcv treesrd point
*
ERROR at line 1:
ORA-20061: 6a: The tree of type QueRob and with x= 105097.02 and y= 482777.4,
is placed too close to another QueRob inside water.
ORA-06512: at "ORAGIS02.PCK SALIX", line 358
ORA-06512: at "ORAGIS02.AST SALIX", line 84
ORA-04088: error during execution of trigger 'ORAGIS02.AST SALIX'


Elapsed: 00:00:02.02
```

In Figure 7-1 and Figure 7-2 the map representations are given of the locations of both trees. Figure 7-1 shows the location of the first tree (most right tree) that is inserted in the plantation layout. This tree is placed inside the water, but not too close to other trees in the water and within a distance of 0,5m of the edge of the water. Figure 7-2 shows the location of the first and the second tree that is inserted. This second tree is also placed in the water, and also within a distance of 0,5m from the water edge. However, the distance to the first inserted tree is less than 8 meters, so they are too close to each other. That's why the database trigger generates an error message followed by an implicit rollback.

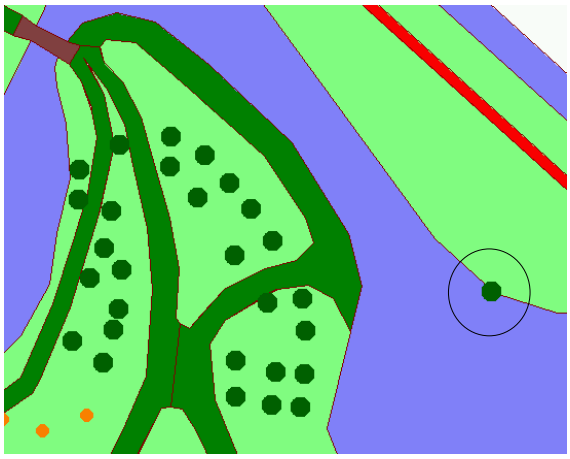

Figure 7-1: position of first tree (c6-insert_tree1_in_water.sql)
The first insert can be done, because the tree is a QueRob in the water, but within a distance of 0,5m from the water edge.
(This 2D image is created in GeoMedia Professional)

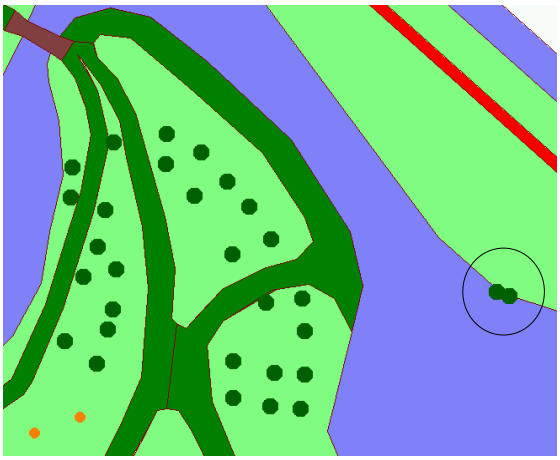

Figure 7-2: position of first and second tree (c6-insert_tree2_in_water.sql)
The second insert cannot be done, because the second tree (QueRob in water) lies within a distance of 8 meters from the first QueRob.
(This 2D image is created in GeoMedia Professional)

The question whether a before trigger is faster then an after trigger for the constraint checking arose, because when using before triggers the table is not modified and a rollback is not necessary when the triggers are not satisfied. To check if before triggers are indeed faster than after triggers, the example constraints are besides using only one before each row trigger, also implemented as a combination of one before each row trigger (for updating the temporary table) and an after statement trigger (for the constraint checking). For both solutions some insert statement were done (like the above described statements) and the computation time was registered, both for a valid insert statement as for an invalid insert statement. The computation time using only the before each row trigger just exceeede one second and the computation time for the after statement trigger just exceeded two seconds in a 'hot' situation. The before each row trigger was a little bit faster, but in this trigger, a separate table (involved_object) was used for the constraint checking, while for the after statement trigger the plantation layout table (prcv_treesrd_point) was used to acquire all necessary attributes of the involved object. So the choice can be made to use only a before each row trigger with an additional table, or an after statement trigger without using an additional table. The last choice is made here..

7.2 Post constraint implementation

After creating the database triggers and necessary package, procedures and functions in the database, also some other adaptation are necessary. The flowcharts of SALIX-2c have some adaptations in comparison with SALIX-2. For this research only the minimum required adaptations are made to test SALIX-2c on the constraint checking. These minimum adaptations are:

1. change the database connection;
2. the local VRML coordinates were previously stored as separate numbers stored in separate columns (TreePosX, TreePosY, TreePosZ), but in SALIX-2c these coordinates have to be derived from the geometry (sdo_geometry) stored in the plantation layout tables in the geo-DBMS;
3. When saving a plantation layout the local VRML coordinates (TreePosX, TreePosY, TreePosZ) have to be transformed to a point in RD coordinates before saving this geometry (sdo_geometry) in the plantation layout table;
4. the textual feedback generated by the database triggers and procedures must be send to the textbox in the Java panel, so it reaches the user.

Changes 2 and 3 are necessary to guarantee consistency; both the visual environment and the database use the same RD coordinates as basis for all actions and computations.

The mentioned minimum adaptations all concern the communication between the database and the rest of the application. The only java class that maintains the communication between the application and the database is the DBHandler class. The changes in the DBHandler class are described below in chronological order. The eventual DBHandler java file can be seen in Appendix F.

Change database connection in DBHandler
First of all the connection to the MSAccess database in the DBHanler class must be changed in a connection to the Oracle database. In Oracle 9*i* JDBC Developer's Guide and Reference information can be found about connecting to an Oracle database. The next text comes from this manual (Oracle, 2002b).

> Connecting to an Oracle database can on different ways. The Java Database Connectivity (JDBC) is a standard Java interface for connecting from Java to relational databases. Oracle has two JDBC drivers; the thin driver and the Oracle Call Interface (oci) driver.
>
> The JDBC thin driver is written entirely in Java. It does not require additional Oracle software on the client side. The Thin driver communicates with the server using TTC, a protocol developed by Oracle to access the Oracle RDBMS.
>
> The JDBC OCI driver is for use with client-server Java applications. This driver is quicker than the thin driver, but requires an Oracle client installation. Therefore it is Oracle platform-specific and not suitable for applets.

The Java code of SALIX-2/SALIX-2c consist of Java applets. Furthermore, SALIX-2c must be a 'portable' application, because it's going to be used widely. This implies that an oci connection with an Oralce client installation on the client side is not desirable. The JDBC connection is thus a thin connection.

Converting coordinates from stored RD to local VRML for visualization
The location of the plantation objects in the VRML scene must be derived from the geometry stored in the DBMS. This in comparison to the old situation where the x,y,z coordinates were stored as numbers in separate columns. The RD coordinates are stored in the DBMS, but they cannot be used in the VRML visualization, because these coordinates are too large. So the separate RD coordinates of the plantation objects must be selected from the database and these coordinates have to be transformed to local VRML coordinates. The transformation consists of a translation and a reflection. The reflection is necessary to go from the x,y plan in RD to the x,z plane in VRML, which are both the base planes for visualization. For the translation the coordinates of the centre point of the ground surface are used (so the plantation layout is concentrated around the y-ax of VRML). For the reflection xVRML is xRD (minus xmRD) and zVRML is the opposite of yRD (plus ymRD). The coordinate axis of both systems can be seen in Figure 7-3. The text belonging to this figure also illustrates the tranformation.

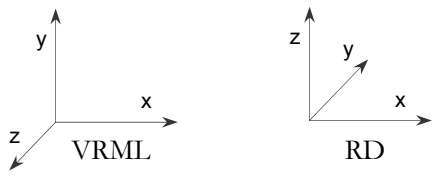

Figure 7-3: The VRML and RD coordinate systems.
Necessary transformations from RD to VRML in SALIX-2c:
zVRML = – (yRD – ymRD)
yVRML = zRD (yVRML is always 0 for SALIX)
xVRML = xRD – xmRD
where xmRD and ymRD are the coordinates of the centre point of the ground surface.

Also in the other way, when saving a plantation layout, the local VRML coordinates where the visual environment of the new SALIX application works with, have to be transformed to the RD coordinates and after that, the plantation layout table can be updated with the adapted or new plantation layout.

Retrieving error information for feedback
You can retrieve basic error information with the SQLException method getMessage(). For errors originating in the JDBC driver, this method returns the error message with no prefix. For errors originating in the RDBMS, it returns the error message prefixed with the corresponding ORA number. The database triggers generate error messages and they are part of the RDBMS. So the output is an error message prefixed with the ORA number. The following example prints output from a getMessage() call (ORACLE, 2002b).

```
catch(SQLException e)
{
System.out.println("exception: " + e.getMessage());
}
```

Other desirable changes to the system structure
When in SALIX-2c a plantation layout is saved by the user, first all objects are deleted from the plantation layout table and then all objects of the new plantation layout are inserted separately. The constraint checking is not very fast (2 seconds for each object), so for a plantation layout with many objects, the save plantation layout operation will become very slow. It would be more efficient only deleting those objects that are removed or replaced in the plantation layout followed by insert statements of the replaced and added objects in the plantation layout. Less objects have to be inserted and the number of constraint checking is reduced. However, SALIX-2 is only used to implement some example constraints and the intention was not to change this application rigorously. That is why this is not changed for SALIX-2c, but this should be improved in the future.

7.3 Conclusion

The technical implementation of the example constraints for SALIX-2 started with creating database triggers (one before each row and one after statement) in the Oracle database. The before each row trigger stores the new object id and object type of the involved object as variables in the package pck_salix. These variables in combination with the changed plantation layout table (here prcv_treesrd_point) and the ground surface table (here prcv_gvkrd_poly) are used to get the necessary information for all computations and comparisons for the constraint checking. These computations and comparisons are done by the stored procedures and functions that are created and are also stored in the package. The procedures and functions in the package are invoked from the after statement trigger. The code of the triggers and procedures can be found in Appendix E. The constraints are implemented in an orderly way using triggers, procedures and functions, but the user is not able to interact with the constraints somehow. Even enabling and disabling constraints is impossible because all constraints are implemented in one trigger.

After creating the database trigger in Oracle for the constraint checking, SALIX-2c had to be modified. All changes in the plantation layout have to be checked for constraints and feedback to the user about the validity of the changes must be given. For this constraint checking the database connection had to be changed to an Oracle DBMS connection. The geometry of the plantation objects in the plantation layout tables had to be used for both the constraint checking by the database trigger as well as for the visualization in the VRML scene. The error messages, generated by the database trigger and stored procedures, had to be send to the textbox of the Java panel to give feedback to the user. All these changes concern only the DBHandler class of the application and this class with its adaptations can be found in Appendix F.

8. Conclusions and recommendations

In the previous chapters the research to constraints in geo-VR applications is described. All findings and remarks of this research are discussed in this chapter. Conclusions and recommendations are therefore listed in sections 8.1 and 8.3.

8.1 Conclusions

The object of this research was to **define a way to specify and implement constraints in a geo-information model**. To reach this objective, some questions are answered. These questions including the answers are summarized below.

1. *Is there already been a research on this area?*

None of the investigated geo-VR applications had constraints to restrict the virtual world in a way that it looks more like the real world (with all it's natural rules). So a general definition of constraints (Molenaar, 1998) was converted to a definition of constraints for geo-VR applications. This definition is: a condition that must always be true for **objects** in a 3D model

2. *What are constrained objects and what types of constraints exist (e.g. geometrical, temporal, thematic, topological)?*

Objects can be described by their attributes, behaviour and **relations**. Constraints can be formulated in the description of an object class as a relation to another object class. The different types of constraints are:

- spatial topology;
- spatial metric;
- temporal;
- quantity;
- thematic.

When an object class has two or more restricted relations as a part of its description, a check is necessary if the constraints of all object classes do not conflict each other. This check is manually done in the conceptual phase. Besides constraints between object classes also constraints between object instances are possible.

3. *What is the current application structure of SALIX-2?*

SALIX-2 consist of an html page with on the left side the VRML world for visualization and on the right side a Java applet. The Java applet has a box for textual output and a Java console for more complex interaction possibilities with buttons and scroll down menus. All data is stored in a DBMS; for SALIX-2 MSAccess is used. The communication between VRML and Java is done with EAI and the communication with the DBMS is done in a separate class of the Java code (DBHandler).

4. *Which 2D constraints can be implemented in SALIX-2 as a good example?*

For the implementation a representative set of constraints should be implemented. A representative set of constraints can be obtained by defining an example for each type of constraint. However, the temporal example is of less interest for SALIX-2c, because there's not much change over time in SALIX-2c. So for the example implementation only spatial topological, spatial metric, quantity, thematic and complex constraints are used.

5. *What is the best way to implement these constraints (storage in database or VRML environment, accessibility of constraints, interactivity to change the constraints).*

Geo-VR applications consisting of VRML, Java and a DBMS have two implementation approaches:

1. the DVM (VMRL and Java), which can be compared to the client side of a client-server architecture
2. the DLM (DBMS), which can be compared to the serverside of a client-server architecture.

The question 'what is the best way to implement constraints' cannot be answered for all geo-VR applications, because each application is unique

Constraints can only be implemented in the Java code of the DVM. VRML doesn't offer the right possibilities for the constraint implementation and because Java is a programming language, a lot is possible. However, for the constraint checking everything has to be designed and programmed from scratch and the constraints are not stored on a central place.

All types of constraints can be implemented in the DBMS of the DLM. Literature about DBMSs all mention the possibilities of implementing integrity constraints, but these possibilities are also limited to rather simple constraints. General constraints (assertions) and base table constraints should be able to implement the more complex constraints. However, general constraints only exist in theory and no DBMS has implementation possibilities for general constraints and the base table constraints cannot contain subselects in the check. The only solution of implementing complex constraints in the DBMS is by triggers and procedures.

To decide which implementation approach (DVM/DLM) is most suitable for which application, a list of criteria has to be taken into account.

6. *How and when can the user of the application be informed about the constraints? (Show the user a list of defined constraints and a good and detailed feedback is necessary)*

The user can best be informed about the constraints when the application is started. A combination of visual and textual feedback is the best solution. Visual feedback is desirable, because a picture is more than thousands words and textual feedback is desirable because this can give more detailed information. Also when the user violates a constraint, a direct feedback in textual and visual form should be given.

8.2 Discussion

The objective of this research was to define a way to specify and implement constraints in a geo-information models. It appeared that two implementation approaches exist for geo-VR applications: the DVM and DLM. To decide which implementation approach is most suitable for which application, the next **decision criteria** should be taken into account:

- who are the end users? (how much computer experience do they have, what computers do they have)
- What is the final purpose of the application? (wide usage of application?)
- what is the available budget to create/adapt the application? (freeware/commercial software)
- what expertise exists already within the organisation developing the application?
- Is interactive constraint definition desirable?

These criteria can be placed in a decision table (Table 5-3) to find the most suitable approach. The outcome is not the only possible solution, so the developers can deviate from this outcome when some criteria have a higher weight then others.

When the most suitable implementation approach is found, the constraint can be implementation. First a conceptual model is necessary; the constraint and the way of implementing them are defined. Then this model can be formalized; the code for the constraints can be generated. When the code is generated, the constraints can be implemented in either the DLM or DVM (according to the most suitable implementation approach). Points of interest in this conceptualization – formalization – implementation process are the system structure, the constraint definition and the feedback to the user.

For SALIX-2c the most suitable implementation approach was the DLM implementation, using already available software (Oracle). The constraints were defined in the conceptual model. It is desirable to **change the constraint definitions** when working with the application, but in SALIX-2c the constraint definitions cannot be changed. Even enabling and disabling constraints separately is not possible for the example implementation, because the group of constraints is implemented as one trigger (a trigger can be enabled/disabled in Oracle) in combination with stored procedures and functions. This should be improved somehow in the future.

Furthermore the constraints are implemented by using triggers and procedures in Oracle. The code for the triggers/procedures is generated by hand. Of course **automatic generation of triggers/procedures** would be much more efficient. Maybe OCL or assertions can be used to define the constraints and then a tool can be used to generate triggers/procedures automatically from these defined constraints.

The last point of discussion concerns the **feedback to the user**. A combination of before, after, visual and textual feedback is the best, however not easy to implement. Red areas can for example be

used for the visual feedback beforehand, but the geometry of this area must actually be constructed in the DBMS and then a conversion to a VRML node with a certain appearance is necessary. The construction of a VRML node in Oracle is difficult so for the implementation in SALIX-2c only textual feedback is implemented. Constructing VRML nodes from geometry is probably possible in software with 3D extensions, but this is not investigated in this research.

8.3 Recommendations

This thesis to constraints in geo-VR applications is only the beginning of the search to solutions for constraints in geo-VR. A lot of further research can be done to find out more about this subject:

1. This research can be extended to geo-VR application structures different from the structure used for this research (VRML, Java and Oracle). Maybe geoVRML, gaming environments or existing geo software with 3D extensions have more suitable implementation possibilities for constraints in geo-VR applications.

2. Other geo-DBMSs can be investigated, for example MySQL. The question why general constraints are not implemented in DBMSs can be investigated. Oracle has for example an implementation tool (CDM) to implement constraints (which are called business ruels). Maybe also UML and OCL can be used to implement triggers/procedures. An investigation to the ability of using such tools for constraints implementation is desirable. These tools offer probably better opportunities for constraint implementation then only database triggers and procedures.

3. The types of constraints in this research concern binary relations. The list of different types of constraints should be extended with unary and set relations to get a complete overview.

4. Define a good check whether the defined 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. E.g. topology constraints can be implemented in the ESRI software, 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.

5. This visual feedback for VRML visualization has to be a VRML node (like red or green areas) and this VRML node should be derived from geometry in a DBMS. An investigation should take place if these nodes can be created inside the DBMS or with GIS software (like 3D Analyst of ArcGIS).

6. An investigation should take place if the nodes for the feedback (e.g. red/green areas) can be derived from the given set of constraints/assertions in case the user wants to insert objects. This is an extension of the previous recommendation.

7. An investigation should take place how a list of constraints automatically can be generated from the triggers in the DBMS to guarantee consistency in the application. Maybe some implementation tools (like CMD for Oracle), SQL assertions or OCL can be expanded to be of help for generating this list.

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

9. This research can be extended to 2½D or 3D. This extension concerns the objects of interest and the constraints. The objects of interst in SALIX-2c are limited to point objects (plantation objects) and polygon objects (ground surfaces). Functions and operations for 2½D and 3D geometry are necessary in DBMSs for implementing 2½D and 3D constraints concerning 2½D or 3D objects.

10. The last recommendation concerns the 'save' operation of SALIX-2/SALIX-2c. When saving a plantation layout in the DBMS, all objects are deleted from the plantation layout table. Then all objects are inserted. This can be done more efficient by only deleting those objects that are deleted from or moved to another position in the plantation layout. Then only the objects that are added or moved in the plantation layout have to be inserted.

Literature

URL's

[URL 1] http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/index.html
Title: VRML 97. ISO/IEC 14772
Subject: VRML Part 1 (ISO/IEC 14772-1) defines the base functionality and text encoding for VRML (including node references). VRML Part 2 (ISO/IEC 14772-2) defines the base functionality and all bindings for the VRML External Authoring Interface.

[URL 2] http://deslab.mit.edu/DesignLab/courses/13.016/visualization/second/
Title: Introduction to VRML
Subject: description of Concepts, Geometry, Positioning objects and Building a larger world

[URL 3] http://tecfa.unige.ch/guides/vrml/vrmlman (consulted on 16-10-03)
Title: VRML Primer and Tutorial
Subject: Introduction to VRML (including EAI)

[URL 4] http://www.cit.uws.edu.au/docs/oracle/sqlref/expressi.htm#1002893
Title: Oracle8*i* SQL Reference Release 8.1.5. Chapter 5: Expressions, Conditions, and Queries
Subject: description of expressions, conditions and queries possible in Oracle 8i.

[URL 5] http://www.klasse.nl/ocl/inde.html (consulted on 29-02-04)
Title: Welcome to the OCL Center
Subject: A short introduction to the OCL, including examples that show how and when OCL is useful

[URL 6] www.geovrml.org
Title: About GeoVRML
Subject: explanation and specifications concerning GeoVRML

[URL 7] http://3dgraphics.about.com/library/weekly/aa011001a.htm
Title: GeoVRML an Overview
Subject: short explanation of GeoVRML including description of nodes

[URL 8] http://ArcSDEOnline.esri.com
Title: ArcSDE devoloper help guide
Subject: All about ArcSDE Client API for C programmers and ArcSDE Client API for Java programmers.

[URL 9] http://unicoi.kennesaw.edu/~jbamford/csis4650/uml/UML_tutorial/state.htm
Title: State Diagrams
Subject: How to construct and use UML State Diagrams

[URL 10] http://www.agilemodeling.com/style/classDiagram.htm\
Title: UML Class Diagram Guidelines
Subject: guidelines about how to construct a UML class diagram

[URL 11] http://eureka.lucia.it/vrml/tutorial/eai/sgi/ExternalInterface.html#EventOutObserver
Title: Cosmo Player Developer Tools & Docs
Subject: Reference for the External Authoring Interface

[URL 12] www.k2vi.com
Title: Key to virtual insight – interactive virtual reality software platform
Subject: information about the K2Vi product

URL 13] http://www.tec.army.mil/TD/tvd/survey/Pavan.html
Title: Commercial terrain visualization software product information
Subject: information about the Pavan software

[URL 14] http://www.esri.com/software/arcgis/index.html
[Title: ESRI GIS and Mapping Software – ArcGIS
Subject: information about the ArcGIS package

URL 15] www.sense8.com
Title: Sense8
Subject: information about the application development tools WorldToolKit, World Up and World2World.

Books and articles

ArcNews Vol.24 No.2: ArcGIS 8.3 *Brings topology to the geodatabase*.

ArcNews Vol.24 No.4: *ArcGIS 8.3 focuses on topology and editing*

Baars, Marco. 2003. *A comparison between ESRI Geodatabase topology and Laserscan Radius Topology*. Delft, 2003.

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*. Lecture notes in computer science, Vol. 367, pp. 457-472.

ESRI, 2002. *Understanding ArcSDE* (pdf file from www.esri.com)

ESRI, 2002b. W*orking with the geomdatabase: powerful multiuser editing and sophisticated data integrity*. An ESRI white paper, February 2002.

Haan, Lex de. 1993. *Leerboek Oracle-SQL*. Academic Service, Schoonhoven. ISBN 90-6233-939-5.

Hartman, Jed and Josie Wernecke. 1996. *The VRML 2.0 Handbook: building moving worlds on the web*. Silicon Graphics, Inc.

Haung B., Jiang B. and Hui L. 2001. *An integration of GIS, virtual reality and the Internet for visualization, analysis and exploration of spatial data*. International Journal of Geographical Information Science, 2001, Vol. 15, No. 5, pp. 439-456.

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

Lammeren, R. van, V. Clerc (SERC), H. Kramer. 2003. *SALIX-2. Simulatie Agenten voor Landschapsarchitectonisch Design in virtual reality (x)*. Wageningen, Alterra, Research Instituut voor de Groene Ruimte. Alterra-rappport 715, ISSN 1566-7197.

Lentz, Arjen. 2003. *MySQL® RoadMap, What we have now & where we are heading*. MySQL AB.

Lovett, Andrew. et al. 2002. *Visualizing sustainable agricultural landscapes*. Chapter 9 (p.102-130) of *Virtual reality in geography,* P. Fisher and D. Unwin. Taylor & Francis, London. ISBN 0-7484-0905-X.

Maren, Gert van. 2003. *H11: Key to virtual insight: a 3D GIS an Virtual reality system*. In: Planning support systems in practice, by Geertman and Stillwell. ISBN 3540437193.

Molenaar, Martien. 1998. *An introduction to the theory of spatial object modelling for GIS*. Taylor & Francis, London. ISBN 0-7484-0774-X.

Morelli, Ralph. 2000. *Java, Java, Java, Object-Oriented Problem Solving*. Prentice-Hall, Inc. New Jersy. ISBN 0-13-011332-8.

Muller, Sandra. 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)

Oosterom, P. van, W. Vertegaal, M. van Hekken, T. Vijlbrief. 1994. *Integrated 3D modelling within a GIS*. Presented at the International GIS workshop AGDM'94 (Advanced Geographic Data Modelling), Delft, The Netherlands, 12-14 September 1994, pages 80-95.

ORACLE. March 2002a. *Oracle 9i Database concepts*, release 2 (9.2), Part No. A96524-01. (Chapter 17 – Triggers, Chapter 21 – Data Integrity).

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

ORACLE. March 2002c. Oracle Spatial - User's Guide and Reference, release 9.2, Part No. A96630-01.

Papadias D., Karacapilidis N. and Arkoumanis D. 1999. *Processing fuzzy spatial queries: a configuration similarity approach*. In International Journal of geographical information science. Vol. 13, No. 2, 93-118.

Schans, René van der. 1997. *A quest for optimal expression of objects and actions in GIS*. In Proceedings Best-GIS Workshop "GIS-Interfaces for environmental Control", Utrecht, 19-20 February 1997. (p. 11-20)

Verbree E., Maren G. van, Germs R., Jansen F. and Kraak M-J. 1999. *Interaction in virtual world views – linking 3D GIS with VR*. In International Journal of geographical information science, 1999, Vol. 13, No. 4, 385-396.

Vries, M.E. de and J.E. Stoter. 2003. *Accessing a 3D geo-DBMS using Web technology*. ISPRS Joint Workshop on "Spatial, Temporal and Multi-Dimensional Data Modelling and Analysis", Qu\351bec, Canada, October 2003.

Wachowicz, M., Vullings, L.A.E., Broek, M. van den, Ligtenberg, A. 2002. *Games for interactive spatial planning: SPLASH a prototype strategy game about water management.* Wageningen, Alterra, Research Instituut voor de Groene Ruimte. Alterra-rapport 667.

Warner, Jos en Anneke Kleppe. 2001. *Praktisch UML 2de editie*. Addison Wesley. ISBN 90-430-0494-4.

Worboys, Micheal F. 1998. *GIS: A computing perspective*. Taylor&Francis, London. ISBN 0-7484-0064-6.

Xiang L. and Hui L. 2002. *Participatory comprehensive plan based on virtual geographical environment*. Asian Conference on Remote Sensing (session GIS, GPS and data integration).

Zlatanova S. 2000. *3D GIS for urban development*. Thesis ICGV, GrazUT, Austria and ITC, The Netherlands. ISBN 90-6164-178-0.

Abbreviations

CDM	Custom Development Method
CGI	Centre for geo-information
DBMS	Database Management System
DLM	Digital Landscape Model
DVM	Digital Visualization Model
GUI	graphical user interface
JDBC	Java DataBase Connection
OCL	Object Constraint Language
OCI	Oracle Connection Interface
PL	Procedural Language
RDBMS	Relational Database Management System
SALIX	Simulation Agents for Landscape architectural design In virtual reality (x)
SQL	Structured Query Language
UML	Unified Modelling Language
VGE	Virtual Green Environment
VRML	Virtual Reality Modelling Language
WGDM model	Model of real world (W), graphical representation (G), digital model (D), ant the mental model (M)
WUR	Wageningen University and Research centre

List of figures

List of tables

Appendix A. Static structure SALIX-2

SALIX-2 will serve as case study for the implementation of some constraints. Therefore it is necessary to know what the application structure of SALIX-2 looks like, to decide which constraints can be implemented and how they can be implemented.

As can be seen in figure A-1 SALIX-2 consists of a VRML representation of the world on the left and a Java applet on the right side of the window. When SALIX-2 is started (index.htm) the VRML world is created including the frontpanel (in the upper-left corner of the scene), but without plantation objects. On the right the java applet is created including a textbox for output messages and an interaction panel including several buttons, which enable interaction with the VRML world and the database. The external authoring interface (EAI) realises the link between Java and the VRML model.

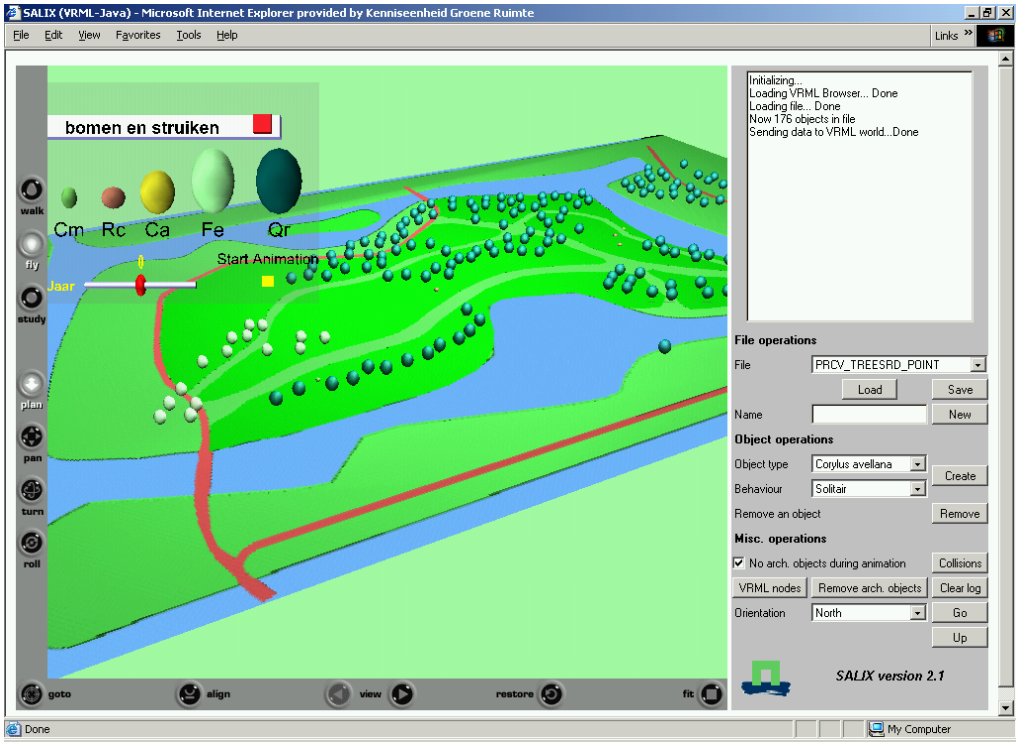

Figure A - 1: the appearance of SALIX-2

After SALIX-2 is started, first a plantation layout has to be loaded from the database to import objects to the VRML scene or a plantation layout has to be created and then objects can be added to the VRML scene. All plantation layouts are stored as separate tables in the database, as well as a table containing descriptions of the different types of plantation objects. To get a good overview of all the objects in the SALIX-2 application and the static system structure, a Unified Modelling Language (UML) class diagram is made. This class diagram can be seen in figure A-2.

The class diagram of SALIX-2 is a static system structure. When the plantation layout is loaded, the plantation objects are added to the VRML scene. After they are added, interaction is possible in two ways: interaction in the VRML scene and interaction through buttons on the Java panel. All possible interactions in the VRML scene are:

- enlarge/reduce the size of the frontpanel;
- drag the frontpanel to another position;
- change the age of the plantation objects in the scene;
- start an animation (shows the growth of the plantation objects from the age of -37 to 37).

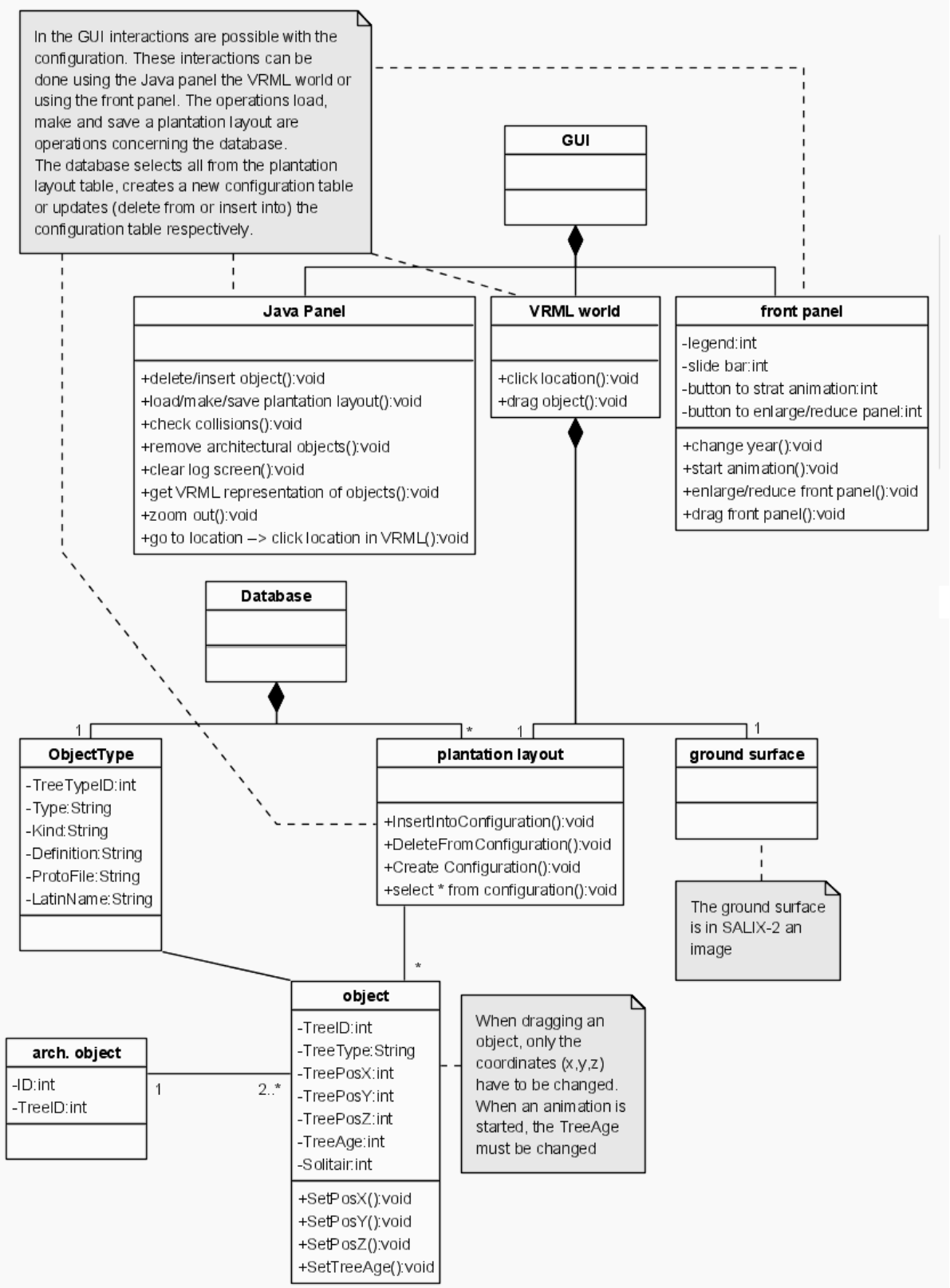

Figure A - 2: class diagram of the static system structure of SALIX-2.
The two basic components of the system are the database and the GUI, consisting of a VRML world and a Java panel. All plantation layouts are stored as separate tables and each object in the plantation layout is a row of this table. Two or more objects can also form an architectural object.
The VRML world consists of the ground image and some objects belonging to a plantation layout or an empty plantation layout. In SALIX-2c, the ground surface image will stay the same, but the image is also digitized and stored with all necessary data as a separate table in the database.

All possible interactions through buttons on the java panel are:
- Loading and saving files and making new ones;
- Creating and removing objects;
- Checking for collisions and architectural objects;
- Get the VRML representation of the objects in the scene;
- Go to a location you click in the VRML scene;
- And zoom out (go up above the location you were).
- Remove the architectural objects (so replacement of individual objects, that were a part of an architectural object, is possible);

Architectural objects are objects consisting of two or more plantation objects. Plantation objects (like trees and bushes) can be placed very close to each other in the VRML scene, so close that they all together form a larger object, a so-called architectural object. An architectural object is only constructed when the original plantation objects have a minimum (specified) overlap. The shape of architectural objects can vary. In (Lammeren et al., 2003) a summary of the different shapes of architectural objects can be found:
- *Base plane:* a solid element, which raises the base. This plane is mostly used to divide an environment into sections;
- *Overhead plane:* a solid element, which constructs a roof. This plane is being used to create some privacy or 'windows' in the environment;
- *Vertical plane:* a solid element, which constructs a wall. This plane is being used to fulfil a spatial separation.

```
<html>
<head>
   <title>SALIX (VRML-Java)</title>
</head>
<body>
    <embed src="vrml/rdbos.wrl" border="0" WIDTH="720" HEIGHT="650">
    <applet archive="salix2.jar" code="java.OutputApplet.class"
            mayscript WIDTH="260" HEIGHT="650">
    </applet>
</body>
</html>
```

Figure A - 3: The html code of the index.htm file, which is the starting file of the SALIX-2 application

The index.htm file is the start of the application and the code of this file can be seen in figure 3.2. The structure of SALIX-2 doesn't become clear from this code. For that, a closer look to the application code is necessary. The VRML file rdbos.wrl has various links to other VRML files and the java class OutputApplet has also many links to other java classes. A complete list of all VRML files and java classes is shown in Table A - 1.

Table A - 1: all VRML files and Java classes of the SALIX-2 application

All **VRML files** in SALIX-2 are: (with *.wrl extension)		All **java classes** in SALIX-2 are: (with *.java extension)	
David	Slider_p	ArchitecturalObject	SalixObject
Halo	corave3_p	Behaviour	SalixObjectType
HudPanel_p	cormas4_p	DBHandler	VRMLHandler
Land	fraxexc7_p	Orientation	VRMLTypes
Rdbos	querob3_p	OutputApplet	
sky	roscan3_p	SalixException	

Besides these files of VRML and Java code, also a database exists. This database, with the name 'myTrees', contains the following tables:
- Tree – a plantation layout;
- TreeOrg – a plantation layout;
- TreeRotateAge – a plantation layout;
- TreeSmall – a plantation layout;
- TreeType –the descriptions of the five plantation objects.

Each plantation plan table contains the rows TreeID, TreeType, TreePosX, TreePosY, TreePosZ, TreeAge and Solitair. The table TreeType contains the rows TreeTypeID, Type, Kind, Definition, ProtoFile, LatinName.

Still nothing is said about the links between the files and the database and the order of execution of the actions in SALIX-2. This will be described in the following sections.

Appendix B. Querying databases

Many different queries exist to get information from a database. The Structured Query Language (SQL) is the most common used query language. Databases have their own query possibilities, mostly including SQL possibilities. An overview of the diversity of querying databases is given in this appendix.

B.1 SQL

SQL data manipulations (DML) and queries (from Haan, 1993)

```
COMMIT
DELETE FROM table_name [WHERE cond]
INSERT INTO table_name [(column,…)]
    {VALUES(value,…)|query}
ROLLBACK [TO SAVEPOINT sp_name]
SAVEPOINT sp_name
SELECT [DISTINCT] select_expr[,…]
    FROM table_expr[,…]
    [WHERE cond]
    [GROUP BY expr[,…] [HAVING cond]]
    [{UNIOIN[ALL]|INTERSECDT|MINUS} query]
    [ORDER BY {expr|pos} [ASC|DESC] [,…]
TRUNCATE table_name
UPDATE table_name [alias]
    SET {col=expr[,…]|(col[,…])=(subquery)}
    [WHERE cond]
```

SQL operators (from Haan, 1993)

equation operators:
=
!= ^= <>
> >=
< <=
IN
NOT IN
ANY
ALL
BETWEEN x AND y
EXISTS
LIKE
[ESCAPE 'X']
IS NULL

Mathematical operators
+ - (positive or negative expression)
* /
+ - (plus or minus)

alphanumeric operators
||

Logical operators:
NOT
AND
OR

Set operators:
UNION [ALL]
INTERSECT
MINUS

Other operators:
(+)
*
DISTINCT

B.2 MSAccess

Types of queries you can create in MSAccess

Action queries:
- delete
- Update
- Append
- Make-table

SQL queries:
- Union
- Pass-through
- Data-definition
- Subquery

Select
Parameter
Crosstab

B.3 Oracle Spatial 9*i*:

Geometry functions in Oracle Spatial 9*i*

Geometry functions can be used for:
- determining relationships between geometries
- finding information about single geometries
- changing geometries
- combining geometries.

These functions all take into account two dimensions of source data. If the output value of these functions is a geometry, the resulting geometry will have the same dimensionality as the input geometry, but only the first two dimensions will accurately reflect the result of the operation.

The geometry functions are not discussed in more detail, because the constraints for geo-VR environments concern mostly (if not always) one geometry comparing to all other geometries. The geometry functions mentioned in table A.2 all concern relations between two specified geometries.

Table B - 1: geometry functions available in Oracle

Geometry function in Oracle	Short description of geometry function
SDO_GEOM.RELATE	Determines how two objects interact.
SDO_GEOM.SDO_ARC_DENSIFY	Changes each circular arc into an approximation consisting of straight lines, and each circle into a polygon consisting of a series of straight lines that approximate the circle.
SDO_GEOM.SDO_AREA	Computes the area of a two-dimensional polygon
SDO_GEOM.SDO_BUFFER	Generates a buffer polygon around a geometry
SDO_GEOM.SDO_CENTROID	Returns the centroid of a polygon
SDO_GEOM.SDO_CONVEXHULL	Returns a polygon-type object that represents the convex hull of a geometry object
SDO_GEOM.SDO_DIFFERENCE	Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects
SDO_GEOM.SDO_DISTANCE	Computes the distance between two geometry objects
SDO_GEOM.SDO_INTERSECTION	Returns a geometry object that is the topological intersection (AND operation) of two geometry objects
SDO_GEOM.SDO_LENGTH	Computes the length or perimeter of a geometry
SDO_GEOM.SDO_MAX_MBR_ ORDINATE	Returns the maximum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object
SDO_GEOM.SDO_MBR	Returns the minimum bounding rectangle of a geometry
SDO_GEOM.SDO_MIN_MBR_ ORDINATE	Returns the minimum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object
SDO_GEOM.SDO_POINTONSURFACE	Returns a point that is guaranteed to be on the surface of a polygon
SDO_GEOM.SDO_UNION	Returns a geometry object that is the topological union (OR operation) of two geometry objects
SDO_GEOM.SDO_XOR	Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects
SDO_GEOM.VALIDATE_GEOMETRY	Determines if a geometry is valid
SDO_GEOM.VALIDATE_LAYER	Determines if all the geometries stored in a column are valid
SDO_GEOM.WITHIN_DISTANCE	Determines if two geometries are within a specified distance from one another

Spatial operators for querying in Oracle Spatial
The available spatial operators in Oracle are listed in
Table B - 2 and their syntax is given below.

SDO_FILTER (geometry1, geometry2, params);
With the 'params' the querytype (window or join) and the indextables (for the geometries) can be specified.
SDO_NN(geometry1, geometry2, param[,number]);
The 'number' specifies the same number used in the call to SDO_NN_DISTANCE (when included)

SDO_NN_DISTANCE(number)

SDO_RELATE(geometry1, geometry2, params);

With the 'params' the mask and querytype must be specified and the index tables can be specified. The mask specifies the topological relation of interest. Valid values are one or more of the following in the 9-intersection pattern: TOUCH, OVERLAPBYDISJOINT, OVERLAPBYINTERSECT, EQUAL, INSIDE, COVEREDBY, CONTAINS, COVERS, ANYINTERACT, ON.

SDO_WITHIN_DISTANCE(geomertry1, aGeom, params);

Geometry1 specifies a geometry column in a table and aGeom specifies the object to be checked for distance against the geometry objects in geomery1. The params specify the distance, index table for geometry1, querytype (window or join) and unit of measurement.

Table B - 2: spatial operation available in Oracle

Spatial operation in Oracle	Short description of spatial operation
SDO_FILTER	Specifies which geometries may interact with a given geometry
SDO_NN	Determines the nearest neighbor geometries to a geometry
SDO_NN_DISTANCE	Returns the distance of an object returned by the SDO_NN operator
SDO_RELATE	Determines whether or not two geometries interact in a specified way
SDO_WITHIN_DISTANCE	Determines if two geometries are within a specified distance from one another

B.4 ArcGIS 8.3

The topology rules in ArcGIS 8.3 are listed in Table B - 3. (from Baars, 2003). The A, P and L in this table stand for Area, Point and Line respectively.

Table B - 3: Topology rules in ArcGIS 8.3

Area rules	
Must not overlap	A
Contains points	A:P
Must be covered by feature class of	A:A
Must not overlap with	A:A
Must not have gaps	A
Boundary must be covered by	A:L
Must be covered by	A:A
Must cover each other	A:A
Area boundary must be covered by boundary of	A:A

Line rules	
Must not have dangles	L
Must not overlap	L
Must not intersect	L
Must not intersect or touch interior	L
Must not overlap with	L:L
Endpoint must be covered by	L:P
Must not have pseudo-nodes	L
Must not self-overlap	L
Must not self-intersect	L
Must be single part	L
Must be covered by feature class of	L:L
Must be covered by boundary of	L:A

Point rules	
Must be properly inside polygons	P:A
Must by covered by boundary of	P:A
Must be covered by endpoint of	P:L
Point must be covered by Line	P:L

Appendix C. Agents

Intelligent agent technology could be used to communicate between Java and the DBMS. Agents can be seen as management or communication tools between the DBMS and the GUI of the application. In the current SALIX-2 application a Java applet takes care of this communication between DBMS and the GUI.

Current agent-based design methodologies extend the object-oriented design approach to intelligent agents operating in (distributed) environments. An agent is akin to a class of objects along with appropriate mechanisms for exhibiting intelligent behaviour. Thus, objects can be used to implement agents. From a practical point of view, such agent-based approaches for system development will enable effective management and revision control during software evolution. By virtue of the design techniques, such software systems will be easy to modify and maintain. (from Ramaswamy and Yan)

An agent has a design goal, behaviour, a state and a process. The abbreviation SALIX stands for 'Simulation Agents for Landscape architectonic design in Virtual Reality (x) and is based on Simulation Agents. This gives the impression that SALIX-2 is based on agent technology; this is only true for a small part of the application. In Salix-2 it is possible to form architectonic objects out of plantation objects that are planted close to each other (see Lammeren et al. for more details). For the transformation of plantation objects into architectonic objects the concept of agent technology was used.

A theoretical implementation of an agent could look as follows: the agents sensors automatically detect changes made in the VRML scene and the agent communicates these changes to the DBMS. In the DBMS a check whether the changes are allowed or not can take place. Another part of the agent detects that some information is send by the DBMS and sends this to the VRML for feedback and further action.

This is however pure theoretical. The implementation of agents isn't as simple as it looks like by the above illustration. It is too complicated for this research and therefore not further discussed here.

Appendix D. Required modifications of application before constraint implementation

Before database triggers can be created, first all tables from the MSAccess database had to be converted to Oracle Spatial tables. The plantation layouts were converted to Oracle Spatial tables (among others the prcv_treesrd_point table).

Ground surface image to geodataset conversion
The ground surface was digitised in ArcView. The created shapefile was converted and stored in Oracle Spatial to create a geodataset of the ground surface (prcv_gvkrd_poly table in Oracle Spatial). This conversion was done by FME universal translator, but before this conversion the user_sdo_geom._metadata table was filled with the boundaries of the dataset that has to be created.

During this digitisation process in ArcView to create a shapefile, it appeared that SALIX-2 didn't give a good representation of the real world. The ground surface in SALIX-2 should represent a piece of the Floriade terrain in the Netherlands. However, the ground surface image in SALIX-2 was a rotated image of the accompanying terrain (the picture was upside down). So the ground surface and all coordinates of the objects in plantation layouts of SALIX-2 first needed a transformation to give a correct representation of the real world. After this transformation the representation was still in local VRML coordinates, but it was a good representation of the real world. Only a translation was necessary to convert them to RD coordinates.

Create spatial indexes
For the constraint checking the sdo_relate statement is used very often. The sdo_relate statement requires spatial indexes on the used geometries. These spatial indexes should be the same for both geometries. So the same spatial indexes should be created on the prcv_treesrd_point table and the prcv_gvkrd_poly table (these tables are used in SALIX-2c).

Before creating spatial indexes, metadata must be specified. The tolerance specified in the user_sdo_geom._metadata is used by the index. The sql script to create metadata is:

```
-- first metadata must be created for all tables
-- involved in SALIX-2c.

DELETE FROM user_sdo_geom_metadata
   WHERE table_name = 'PRCV_TREESRD_POINT'
   AND column_name = 'GEOM';

INSERT INTO user_sdo_geom_metadata
   VALUES(
   'prcv_treesrd_point',
   'geom',
   mdsys.sdo_dim_array(
      mdsys.sdo_dim_element('x', 104500, 106000, 0.002),
      mdsys.sdo_dim_element('y', 482000, 484000, 0.002)),
      NULL
   );
commit;

---------
DELETE FROM user_sdo_geom_metadata
   WHERE table_name = 'PRCV_GVKRD_POLY'
   AND column_name = 'GEOM';

INSERT INTO user_sdo_geom_metadata
   VALUES(
   'prcv_gvkrd_poly',
   'geom',
   mdsys.sdo_dim_array(
      mdsys.sdo_dim_element('x', 104500, 106000, 0.002),
      mdsys.sdo_dim_element('y', 482000, 484000, 0.002)),
      NULL
   );
commit;
```

After specifying the metadata, the spatial indexes can be created. First the existing indexes must be droped. The new spatial index is an r-tree index, which is specified by 'indextype is mdsys.spatial_index'. The r-tree index is the default in Oracle (from version 8.1.7 and higher). The sql code to drop and create spatial indexes is:

```
----------
-- now spatial indexes can be made
----------

DROP INDEX i_treesrd_point FORCE;
DROP INDEX PRCV_TREESRD_P_SG7 FORCE;

CREATE INDEX i_treesrd_point ON PRCV_TREESRD_POINT(GEOM)
INDEXTYPE IS mdsys.spatial_index
PARAMETERS ('sdo_fanout=32 sdo_indx_dims=2');

----------

DROP INDEX i_gvkrd_poly FORCE;
DROP INDEX PRCV_GVKRD_POL_SG4 FORCE;

CREATE INDEX i_gvkrd_poly ON PRCV_GVKRD_POLY(GEOM)
INDEXTYPE IS mdsys.spatial_index
PARAMETERS ('sdo_fanout=32 sdo_indx_dims=2');

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

Appendix E. Trigger codes

Package

```
CREATE OR REPLACE PACKAGE pck_salix
IS
   -- the treeid and the treetype of the involved object in the insert
   -- statement is saved here. These attributes can be used in the triggers,
   -- procedures and functions. For the constraint checking in SALIX-2c
   -- they are used to determine if the object is a tree or a bush and
   -- only the corresponding constraints are checked

   treeid_io NUMBER;
   treetype_io varchar2(15);

   -- procedure to check constraint 1: a bush can never be placed inside water
   procedure pr_topology_c1;

   -- function to create the restricted area
   -- that is necessary for checking constraint 2
   function fu_restricted_area
      (angle2 number DEFAULT 5.68,
       angle3 number DEFAULT 0.60,
       distance number DEFAULT 20,
       geometry_io mdsys.sdo_geometry
      )
      RETURN MDSYS.SDO_GEOMETRY;

   -- procedure to check constraint 2: a bush may not be placed south of a tree
   -- south of can be changed by changing the angles
   -- and a maximum distance is added to reduce the
   -- search area
   PROCEDURE pr_direction_c2
      (first_angle number,
       second_angle number,
       distance number,
       geometry_io mdsys.sdo_geometry);

   -- procedure to check constraint 3:
   -- trees always have to be positioned > 1 meter from paving
   procedure pr_metric_c3;

   -- procedure to check constraint 4:
   -- There must always be at least # trees on surface #.
   -- In this case the default minimum number of trees is 3
   -- and the default surface id = 20 (this is a grass polygon)
   procedure pr_quantity_c4
      (surface_id integer DEFAULT 20,
       min_nr_of_trees integer DEFAULT 3);

   -- procedure to check constraint 5:
   -- A bush always has to meet or disjoint paved areas
   procedure pr_thematic_c5;

   -- procedures to check constraint 6. This is a complex constraint
   -- consisting of three simple constraints.
   -- procedure pr_complex_c6a checks constraint 6a:
   -- the distance between trees inside water is always > 8 meter
   procedure pr_complex_c6a;

   -- procedure pr_complex_c6b checks constraint 6b:
   -- The distance between the tree and the edge of the water
   -- always has to be < 0,5 m
   procedure pr_complex_c6b;

   -- procedure pr_complex_c6c checks constraint 6c:
   -- the species of the trees inside water must be Quercus
   procedure pr_complex_c6c;

END pck_salix;
/
```

package body

```
CREATE OR REPLACE PACKAGE BODY pck_salix AS

-- content:
--    procedure pr_topology_c1
--    function fu_restricted_area
--    PROCEDURE pr_direction_c2
--    procedure pr_metric_c3
--    procedure pr_quantity_c4
--    procedure pr_thematic_c5
--    procedure pr_complex_c6a
--    procedure pr_complex_c6b
--    procedure pr_complex_c6c

------
-- procedure pr_topology_c1
------
-- constraint 1: a bush can never be placed inside water
-- the existing table 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.

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_c1;


-------
-- function fu_restricted_area
-------
-- This function creates a geometry with the shape of
-- a piece of cake (the restricted area
-- for the direction constraint).
-- point 1 is the basis for this geometry.
-- This is the location of the bush that is to
-- be inserted or updated in the table.
-- Two angles must be given to calculate the
-- coordinates of points 2 and 3.
-- Also a middle point for the arc must be given
-- this middle point is calculated with an angle which
-- is the average of angle2 and angle3.
--        m
--        _._
--     /     \
--  2 .       . 3
--     \     /
--      \   /
--       \ /
--        .
--        1
--
```

```
FUNCTION fu_restricted_area(
   -- angle2 is the angle of the line from point 1 to point 2
   angle2 number DEFAULT 5.68,       -- when no angle is specified, 2pi-0.6 rad is used.
   -- angle3 is the angle of the line from point 1 to point 3
   angle3 number DEFAULT 0.60, -- when no angle is specified, 0.6 rad is used.
   -- distance is the length of the lines 1-2 and 2-3 (and also 1-m)
   distance number DEFAULT 20, -- when no distance is specified, 20 meter is used.
   -- the geometry of the involved object, which is the reference point for the restricted
areas
   geometry_io mdsys.sdo_geometry)
   RETURN MDSYS.SDO_GEOMETRY
IS

   dx2 number(10,2); -- difference in x coordinates between first and second point
   dy2 number(10,2); -- difference in y coordinates between first and second point
   dx3 number(10,2); -- difference in x coordinates between first and third point
   dy3 number(10,2); -- difference in y coordinates between first and third point
   dxm number(10,2); -- difference in x coord between point 1 and middle point of arc
   dym number(10,2); -- difference in x coord between point 1 and middle point of arc

   anglem number(10,2); -- angle to point m

   x1 number(10,2); -- x coordinate of first point
   y1 number(10,2); -- y coordinate of first point
   x2 number(10,2);
   y2 number(10,2);
   x3 number(10,2);
   y3 number(10,2);
   xm number(10,2);
   ym number(10,2);

BEGIN

   IF angle2>angle3 THEN -- 2pi must be added to angle3
      anglem := angle2 + (((angle3+6.283)-angle2)/2);
   ELSE -- angle2<angle3
      anglem := angle2 + ((angle3-angle2)/2);
   END IF;

   x1 := geometry_io.sdo_point.x;
   y1 := geometry_io.sdo_point.y;
   DBMS_OUTPUT.PUT_LINE('the coordinates of the involved object
   are: x='||to_char(x1)||', y='||to_char(y1));
   -- select i.geom.sdo_point.x, i.geom.sdo_point.y into x1, y1
   -- from involved_object i;

   dx2 := (sin(angle2))*distance;
   dy2 := (cos(angle2))*distance;
   dxm := (sin(anglem))*distance;
   dym := (cos(anglem))*distance;
   dx3 := (sin(angle3))*distance;
   dy3 := (cos(angle3))*distance;

   x2 := x1+dx2;
   y2 := y1+dy2;
   x3 := x1+dx3;
   y3 := y1+dy3;
   xm := x1+dxm;
   ym := y1+dym;

   RETURN MDSYS.SDO_GEOMETRY
   (  2003, NULL, NULL,
      MDSYS.SDO_ELEM_INFO_ARRAY(1,1005,2, 1,2,1, 5,2,2), -- compound polygon
      MDSYS.SDO_ORDINATE_ARRAY(x2,y2, x1,y1, x3,y3, xm,ym, x2,y2)
   );
END fu_restricted_area;


-------
-- PROCEDURE pr_direction_c2
-------
-- constraint 2: a bush may not be placed south of a tree
-- (or another direction, when specified otherwise in the
-- area_of_interest).

PROCEDURE pr_direction_c2
(first_angle number,
 second_angle number,
```

```
 distance number,
 geometry_io mdsys.sdo_geometry
)

IS
   i integer;
   bush_on_wrong_side_of_tree EXCEPTION;

BEGIN
   SELECT COUNT(t.treeid) INTO i FROM prcv_treesrd_point t
   WHERE t.treetype IN ('FraxExc', 'QueRob')
   AND SDO_RELATE(t.geom, fu_restricted_area(first_angle, second_angle, distance,
geometry_io),
   'mask=ANYINTERACT, querytype=window')='TRUE';

   IF i=0 THEN
      raise bush_on_wrong_side_of_tree;
   ELSE
      DBMS_OUTPUT.PUT_LINE
      ('2: the bush is placed in the right direction of '||i||' tree(s).');
   END IF;

EXCEPTION
   WHEN bush_on_wrong_side_of_tree THEN
      raise_application_error
      (-20002,'2: the bush (x='||to_char(geometry_io.sdo_point.x)||',
y='||to_char(geometry_io.sdo_point.y)||')
      is not placed in the right direction of any tree.');

END pr_direction_c2;


------
-- procedure pr_metric_c3
------
-- constraint 3: trees always have to be positioned > 1 meter from paving
-- this is metric constraint and should be done with a buffer and overlay operation
-- or with distance measurement with all paving polygons.
-- creating a buffer and then overlay works probably faster.

PROCEDURE pr_metric_c3

IS
   j integer;
   xrd_io number; -- := pck_salix.xrd_io;
   yrd_io number; -- := pck_salix.yrd_io;
   tree_too_close_to_paving EXCEPTION;

BEGIN
   SELECT i.geom.sdo_point.x, i.geom.sdo_point.y INTO xrd_io, yrd_io
   FROM prcv_treesrd_point i
   WHERE i.treeid = pck_salix.treeid_io
   group by i.geom.sdo_point.x, i.geom.sdo_point.y;

   SELECT count(distinct (g.id)) INTO j
   FROM prcv_gvkrd_poly g, prcv_treesrd_point i
   WHERE g.descript IN ('paving', 'soft_paving')
   AND i.treeid = pck_salix.treeid_io
   AND SDO_WITHIN_DISTANCE (g.geom, i.geom, 'distance = 1') = 'TRUE';

   IF j>0 THEN
      raise tree_too_close_to_paving;
   ELSE DBMS_OUTPUT.PUT_LINE('3: the tree is placed >1m from the paving');
   END IF;

EXCEPTION
   WHEN tree_too_close_to_paving THEN
      raise_application_error(-20003,
      '3: The tree (x='||to_char(xrd_io)||', y='||to_char(yrd_io)||') is placed inside or
within
      a distance of 1 meter from '||to_char(j)||' paving or soft paving surface(s).
      A tree must always be placed >1m from paving or soft paving.');
--   WHEN no_data_found then null;
END pr_metric_c3;
```

```
-------
-- procedure pr_quantity_c4
------
-- constraint 4: There must always be at least 10 trees in a piece of grass
-- the id of the involved grass polygon is 20 and the minimum number of trees is
-- changed in 3.
-- this must be an after statement trigger when inserting, updating or deleting.
-- the exceptionhandling must not give a rollback, so only a message with the right
-- information must be shown.

PROCEDURE pr_quantity_c4
   (surface_id integer DEFAULT 20,
   min_nr_of_trees integer DEFAULT 3)

IS
   i integer; -- integer to store the number of trees
   number_of_trees_too_low EXCEPTION;

BEGIN
   -- count the number of trees on the surface polygon with id=surface_id
   SELECT count(t.treeid) INTO i
   FROM prcv_treesrd_point t, prcv_gvkrd_poly g
   WHERE t.TreeType IN ('FraxExc', 'QueRob')
   AND g.id=surface_id
   AND SDO_RELATE(t.geom, g.geom, 'mask = ANYINTERACT, querytype = window') = 'TRUE';

   IF i < min_nr_of_trees THEN
      raise number_of_trees_too_low;
   ELSE DBMS_OUTPUT.PUT_LINE('4: there are enough trees (>= '||to_char(min_nr_of_trees)||')
      on polygon '||to_char(surface_id));
   END IF;

EXCEPTION
   WHEN number_of_trees_too_low THEN -- no application error can be raised, because than a
rollback takes place
      DBMS_OUTPUT.PUT_LINE('4: there are only '||to_char(i)||' trees placed on the grass
polygon with id '
      ||to_char(surface_id)||'. The minimum number of trees on this piece of grass must be
'||to_char(min_nr_of_trees));
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('4:there is no data found about the number of trees inside
      the grass polygon with id '||to_char(surface_id));
END pr_quantity_c4;

--------
-- procedure pr_thematic_c5
--------
-- Constraint 5: A bush always has to meet or disjoint paved areas.
-- Paved areas are ground surfaces with description 'paving' or 'soft_paving'
-- and the opposites of the topological relations meet and disjoint must be used
-- to search for bushes that do satisfy this opposite relationship. If they
-- exist, an application error must occur.
--
-- The opposites of meet and disjoint is contains for point objects.
-- The bushes are points and the paved areas are polygons,
-- so only the relation contains is of interest here!

PROCEDURE pr_thematic_c5

IS
   j integer; -- integer to store the nr of paved areas
   xrd_io number; -- := pck_salix.xrd_io;
   yrd_io number; -- := pck_salix.yrd_io;
   bush_inside_paved_areas EXCEPTION;

BEGIN
   SELECT i.geom.sdo_point.x, i.geom.sdo_point.y INTO xrd_io, yrd_io
   FROM prcv_treesrd_point i
   WHERE i.treeid = pck_salix.treeid_io
   group by i.geom.sdo_point.x, i.geom.sdo_point.y;

   SELECT count(g.id) INTO j
   FROM prcv_gvkrd_poly g, prcv_treesrd_point t
   WHERE g.descript IN ('paving', 'soft_paving')
   AND t.treeid = pck_salix.treeid_io
   AND SDO_RELATE(g.geom, t.geom, 'mask= CONTAINS, querytype=window') = 'TRUE';

   IF j>0 THEN
```

```
      raise bush_inside_paved_areas;
   ELSE DBMS_OUTPUT.PUT_LINE('5: the bush is not placed inside the paving');
   END IF;

EXCEPTION
   WHEN bush_inside_paved_areas THEN
      raise_application_error (-20005,
      '5: a paving or soft_paving surface overlaps with the bush (x='||to_char(xrd_io)||',
y='||to_char(yrd_io)||').
      A bush must always meet or disjoint paved areas, so place the bush on another
location.');
END pr_thematic_c5;

---------------
-- constraint 6 is a complex constraint, consisting of a combination of
-- three simple constraints
---------------

-------
-- procedure pr_complex_c6a
-------
-- constraint 6 (a)
-- the distance between trees inside water is always > 8 meter

PROCEDURE pr_complex_c6a

IS
   k integer;
   xrd_io number; -- := pck_salix.xrd_io;
   yrd_io number; -- := pck_salix.yrd_io;
   trees_too_close EXCEPTION;

BEGIN
   SELECT i.geom.sdo_point.x, i.geom.sdo_point.y INTO xrd_io, yrd_io
   FROM prcv_treesrd_point i
   WHERE i.treeid = pck_salix.treeid_io
   group by i.geom.sdo_point.x, i.geom.sdo_point.y;

   SELECT count(t.treeid) INTO k
   FROM prcv_treesrd_point t, prcv_gvkrd_poly g, prcv_treesrd_point o
   where t.treetype = 'QueRob'
   AND g.descript='water'
   AND t.treeid != pck_salix.treeid_io
   AND o.treeid = pck_salix.treeid_io
   AND SDO_RELATE (t.geom, g.geom, 'mask=anyinteract, querytype=window')='TRUE'
   AND SDO_WITHIN_DISTANCE(t.geom, o.geom, 'distance=8')='TRUE';

   IF k>0 THEN
      Raise trees_too_close;
   ELSE DBMS_OUTPUT.PUT_LINE
      ('6a: the tree of type QueRob is placed far enough from other QueRobs inside the
water.');
   END IF;

EXCEPTION
   WHEN trees_too_close THEN
      raise_application_error(-20061, '6a: The tree of type QueRob and with x=
'||to_char(xrd_io)||' and y= '||to_char(yrd_io)||',
      is placed too close to another QueRob inside water.');
END pr_complex_c6a;

-------
-- pr_complex_c6b
-------
-- constraint 6 (b)
-- The distance between the tree and the edge of the water always has to be < 0,5 m.

PROCEDURE pr_complex_c6b

IS
   l integer;
   n integer;
   xrd_io number; -- := pck_salix.xrd_io;
   yrd_io number; -- := pck_salix.yrd_io;
   tree_too_far_from_edge EXCEPTION;

BEGIN
   SELECT i.geom.sdo_point.x, i.geom.sdo_point.y INTO xrd_io, yrd_io
```

```
   FROM prcv_treesrd_point i
   WHERE i.treeid = pck_salix.treeid_io
   group by i.geom.sdo_point.x, i.geom.sdo_point.y;

   SELECT count (g.id) INTO n
   FROM prcv_gvkrd_poly g, prcv_treesrd_point t
   WHERE t.treeid = pck_salix.treeid_io
   AND g.descript IN ('paving', 'soft_paving', 'grass')
   AND SDO_WITHIN_DISTANCE (g.geom, t.geom, 'distance=0.5')='TRUE';

       -- other solution could be:
       --
       -- SELECT count(g.id) INTO n
       -- FROM prcv_gvkrd_poly g, prcv_treesrd_point t
       -- WHERE g.descript IN ('paving', 'soft_paving', 'grass')
       -- AND t.treeid=pck_salix.treeid_io
       -- AND sdo_relate(g.geom, sdo_geom.sdo_buffer(t.geom, 0.5, 0.005), 'mask=anyinteract,
          querytype=window')='TRUE';

   IF n=0 THEN        -- there are only water surfaces or bridges within
              -- a distance of 0.5m from the object,
              -- so raise application error
      RAISE tree_too_far_from_edge;
   ELSE DBMS_OUTPUT.PUT_LINE('6b: the object is placed <0.5m from the wateredge');
   END IF;

EXCEPTION
   WHEN tree_too_far_from_edge THEN
      raise_application_error(-20062, '6b: there are only water surfaces
      within a distance of 0.5m from the object (x='||to_char(xrd_io)||',
y='||to_char(yrd_io)||'),
      so the object lies too far from the edge of the water. The distance between trees inside
water
      and the edge of the water must be < 0.5 m.');

END pr_complex_c6b;

------
-- procedure pr_complex_c6c
------
-- constraint 6 (c) the species of the trees inside water must be Quercus.
-- select all trees and bushes that are not of the type Quercus
-- and are inside water. If they exist, an application error must occur.

PROCEDURE pr_complex_c6c

IS
   description varchar2(10);
   n integer;
   xrd_io number; -- := pck_salix.xrd_io;
   yrd_io number; -- := pck_salix.yrd_io;
   object_not_Quercus EXCEPTION;

BEGIN
   -- if treetype != 'QueRob'THEN
   SELECT g.descript, g.id, t.geom.sdo_point.x, t.geom.sdo_point.y INTO description, n,
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=CONTAINS, querytype=window')='TRUE'
   group by g.descript, g.id,  t.geom.sdo_point.x, t.geom.sdo_point.y;
   IF description = 'water' THEN
      raise object_not_Quercus;
   ELSE DBMS_OUTPUT.PUT_LINE ('6c: There are no other objects than QueRobs placed inside
water');
   END IF;

EXCEPTION
   WHEN object_not_Quercus THEN
      raise_application_error(-20063, '6c: The object is not a QueRob
      and is placed (x='||to_char(xrd_io)||', y='||to_char(yrd_io)||') inside the
      water surface with id '||to_char(n)||'. Only trees of type QueRob can be placed inside
water.');

END pr_complex_c6c;

END pck_salix;
/
```

Before row trigger

```
CREATE OR REPLACE TRIGGER brt_all_salix
BEFORE INSERT ON prcv_treesrd_point
FOR EACH ROW

BEGIN
   pck_salix.treeid_io := :new.treeid;
   pck_salix.treetype_io := :new.treetype;
END;
/
```

After statement trigger

```
CREATE OR REPLACE TRIGGER ast_salix
AFTER INSERT ON prcv_treesrd_point

DECLARE
   geom_io MDSYS.SDO_GEOMETRY;
   description varchar2(10);

BEGIN
   -- check constraint 4 about number of trees
   -- inside a certain surface polygon.
   -- The syntax is:
   -- pr_quantity_c4
   -- (surface_id integer DEFAULT 20,
   -- min_nr_of_trees integer DEFAULT 3)

   pck_salix.pr_quantity_c4(20,3);

   -- now check whether the involved object is a tree
   -- or a bush and run all constraints that concern
   -- the new object

   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;

      -- Constraint 2 conscerns a certain direction between
      -- objects and a maximum distance. The direction
      -- range can be specified by 2 angles and the distance
      -- is the maximum distance allowed (actually this
      -- is a complex constraint)
      --
      -- The syntax of this procedure is:
      -- pr_direction_c2
      -- (first_angle number DEFAULT 5.68,
      -- second_angle number DEFAULT 0.6,
      -- distance number DEFAULT 20,
      -- geometry_io mdsys.sdo_geometry)
      --
      -- the geometry of the involved object must first
      -- be selected from the prcv_treesrd_point table.
      -- The geometry cannot be saved as a package variable,
      -- because a spatial indexes are necessary on all geometries
      -- used in the sdo_relate operation.

      select i.geom INTO geom_io
      from prcv_treesrd_point i
      where i.treeid = pck_salix.treeid_io;
      pck_salix.pr_direction_c2(5.41, 0.87, 30, geom_io);

      pck_salix.pr_thematic_c5;

      -- constraint 6 is a complex constraint. The third part
      -- checks if there are no other objects than QueRob
      -- in the water. When the object is a bush, this
      -- bush is not allowed to be placed inside water.
      -- This is checked with the next procedure:
      pck_salix.pr_complex_c6c;
```

```
   ELSIF pck_salix.treetype_io IN ('FraxExc', 'QueRob') THEN
      -- the object is a tree, so all constraints concerning
      -- trees must be run.
      DBMS_OUTPUT.PUT_LINE('the involved_object is a tree');

      pck_salix.pr_metric_c3;

      -- if the object is a FraxExc, a check must take place
      -- if this object is not placed inside water
      IF pck_salix.treetype_io = 'FraxExc' THEN
         pck_salix.pr_complex_c6c;
      END IF;

      -- for all QueRobs inside water, the procedures
      -- containing the complex constraints 6a and 6b,
      -- must be run.

      IF pck_salix.treetype_io = 'QueRob' THEN
         SELECT g.descript INTO description
         FROM prcv_gvkrd_poly g, prcv_treesrd_point i
         WHERE i.treeid = pck_salix.treeid_io
         AND SDO_RELATE(g.geom, i.geom,
         'mask=anyinteract, querytype=window')='TRUE';

         IF description = 'water' THEN
            pck_salix.pr_complex_c6a;
            pck_salix.pr_complex_c6b;
         END IF;

      END IF;

   ELSE raise_application_error(-20099,'the object type is not supported by SALIX-2.');

   END IF;

END;
/
```

Appendix F. DBHandler class

The light gray marked areas are the changes in the new DBHandler class. The adaptations concern:
- the connection to Oracle instead of the first used MSAccess database;
- the derivation of local VRML coordinates from the geometry stored in the DBMS instead of using numbers stored in separate columns for each coordinate;
- the derivation of RD coordinates from the local VRML coordinates (RD coordinates are used to store the geometry in the DBMS;
- the feedback to the user.

```
package java;

import java.sql.*;
import java.util.Vector;
import java.lang.*;

/**
 * This class provides the interface to the database containing information
 * on the objects to be visualized.
 * @author Viktor Clerc (SERC)
 * @version $Id$
 */

public class DBHandler {

    static final String USER_TABLE_PREFIX = "USER_";
    static final String DEFAULT_DATABASE  = "@gis";
    static final String DEFAULT_TABLE     = "PRCV TREESRD POINT";
    static final String CONN_JDBC_ODBC    = "jdbc:oracle:oci:";

    private String database;
    private String connType;
    private String table;

    private Connection c;

    /**
     * Create a default database handler
     */
    public DBHandler() {
        this(DEFAULT_DATABASE, CONN_JDBC_ODBC);
    }

    /**
     * Create a database handler with the given parameters
     * @param database The database that should be connected to
     * @param connType The type of connection to be made to the database
     */
    private DBHandler(String database, String connType) {
        this.database = database;
        this.connType = connType;
        table = DEFAULT_TABLE;
    }

    /**
     * Start the database
     * @throws SalixException if no connection could be made
     */
    public void start() throws SalixException {
        loadDBDriver();
        c = getConnection();
    }

    /**
     * Stop the database
     * @throws SalixException if the connection could not be closed
     */
    public void stop() throws SalixException {
        closeConnection(c);
        c = null;
    }
```

```
    /**
     * Load the database driver
     * @throws SalixException if the database driver class could not be
     * found
     */
    private void loadDBDriver() throws SalixException {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");

         /* for the 'old' salix2:
            Class.forName("com.ms.jdbc.odbc.JdbcOdbcDriver").newInstance();*/
        }
        catch(Exception ex) {
            try {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
            }
            catch(Exception e) {
                System.err.println(e.getMessage());
                throw new SalixException("loadDBDriver(): Can't find database driver
                classes:\n" + ex.getMessage() + " and " + e.getMessage());
            }
        }
    }

    /**
     * Get all configurations present in the database
     * @return Vector a vector containing all configurations
     * @throws SalixException if the configurations could not be read
     */

    public Vector getConfigurations() throws SalixException {
        Vector result = new Vector();
        ResultSet rs = null;
        if(c == null)
            c = getConnection();
        try {
            DatabaseMetaData dmd = c.getMetaData();
            rs = dmd.getTables(null, null, "%", null);

            while(rs.next()) {
                String tableName = rs.getString(3);
                String databaseType = rs.getString(4);
                if(databaseType.equals("TABLE")
                        && (tableName.startsWith(USER_TABLE_PREFIX))) {
                    result.addElement(tableName.substring(
                                    new String(USER_TABLE_PREFIX).length()));
                }
                //DEBUG testing
                if(tableName.equals(DEFAULT_TABLE)) {
                    result.addElement(tableName);
                }
            }
            if(result.isEmpty()) {
                System.err.println("No matching tables found");
                return null;
            }
       }
       catch(Exception e) {
            System.err.println(e.getMessage());
            throw new SalixException("getDatabase(): Could not read configurations from
            database");
       }
       finally {
            //Close any open connections
            try { rs.close(); }
            catch(SQLException e) {}
       }
       return result;
    }

    /**
     * Get all types present in the database
     * @return Vector a vector containing all types
     * @throws SalixException if the types could not be read
     */

    public Vector getTypes() throws SalixException {
```

```
        Vector result = new Vector();
        ResultSet rs = null;
               String query = "SELECT * FROM TreeType";
        Statement stmt = null;

        if(c == null)
                c = getConnection();
        try {
            stmt = c.createStatement();
            rs   = stmt.executeQuery(query);

            while(rs.next()) {
                result.addElement(new SalixObjectType(rs.getInt("TreeTypeID"),
                                                      rs.getString("Type"),
                                                      rs.getString("Kind"),
                                                      rs.getString("Definition"),
                                                      rs.getString("ProtoFile"),
                                                      rs.getString("LatinName")
                                                      )
                                  );
            }
        }
        catch(Exception e) {
            throw new SalixException("DBHandler.getTypes(): Could not read object types from
            database");
        }
        finally {
            try { rs.close(); }
            catch(SQLException e) {}
        }
        return result;
    }

    /**
     * Load the objects from the active configuration
     * @param types A list of all known types
     * @return A vector containing the objects
     * @throws SalixException if the objects could not be read
     */

    public Vector loadObjectsFromConfiguration(Vector types) throws
    SalixException {
        Statement stmt = null;
        String query = "SELECT t.TreeID, t.TreeType, t.geom.sdo point.x x,
                        t.geom.sdo point.y y, t.TreeAge, t.Solitair FROM " +
                        configurationName() + " t ORDER BY t.TreeID";
        ResultSet rs = null;

        Vector nodes = new Vector();

        if(c == null)
            c = getConnection();

        try {
            stmt = c.createStatement();
            rs   = stmt.executeQuery(query);

            while (rs.next()) {
                SalixObjectType objectType = null;
                String type = rs.getString("TreeType");
                if(types != null) {
                    //We have a Vector with types, so find the
                    //type of the current object
                    for(int i = 0; i < types.size(); i++) {
                        objectType = (SalixObjectType) types.elementAt(i);
                        if(objectType.getShortName().equals(type)) break;
                    }
                }
               float xrd = rs.getFloat("x");
               float yrd = rs.getFloat("y");
               float xmrd = 105058.32;
               float ymrd = 482698.40;
               float TreePosX = xrd - xmrd;
               int TreePosY = 0;
               float TreePosZ = ymrd - yrd;
               nodes.addElement( new SalixObject(rs.getInt("TreeID"),
                                                 objectType,
                                                 TreePosX,
```

```
                                        TreePosY,
                                        TreePosZ,
                                        rs.getInt("TreeAge"),
                                        false,
                                        rs.getBoolean("Solitair")
                                        )
                                );
            }
        }
        catch(SQLException e) {
            System.err.println(e.getMessage());
            throw new SalixException("loadObjectsFromConfiguration(): Could not load the
            database");
        }
        finally {
            try { rs.close(); }
            catch(SQLException e) {}
        }
        return nodes;
    }

    /**
     * Clears the current configuration
     * @throws SalixException if no connection could be made
     */
    public void clearConfiguration() throws SalixException {
        System.out.println("clear configuration");
        Statement stmt = null;
        String query = "DELETE * FROM " + configurationName() ;

        if(c == null)
            c = getConnection();
        try {
            stmt = c.createStatement();
            stmt.execute(query);
            stmt.close();
        }
        catch(SQLException e) {
            System.err.println(e.getMessage());
        }
    }

    /**
     * Insert a collection into the active configuration
     * @param v The vector to be added
     */

    public void insertInConfiguration(Vector v) throws SalixException {
        Statement stmt = null;
        String query = null;

        if(c == null)
            c = getConnection();
        try {
            for(int i = 0; i < v.size(); i++) {

                SalixObject tree  = (SalixObject) v.elementAt(i);
                int     id        =               tree.getID();
                float   TreePosX  =               tree.getPosX();
                float   TreePosY  =               tree.getPosY();
                float   TreePosZ  =               tree.getPosZ();
                float   xmrd      =               105058.32;
                float   ymrd      =               482698.40;
                float   xrd       =               TreePosX + xmrd;
                float   yrd       =               ymrd - TreePosZ;
                int     age       =               tree.getAge();
                String  type      =               tree.getType().getShortName();
                boolean solitair  =               tree.isSolitair();

                query = "INSERT INTO "+ configurationName() +" (treeid, TreeType, TreeAge,
                         Solitair, geom) VALUES (" + id + ", '" + type + "', " + age + ", " +
                         solitair + ", MDSYS.SDO GEOMETRY(2001, NULL, MDSYS.SDO POINT TYPE(" +
                         xrd + ", " + yrd + ", NULL), NULL, NULL))";

                stmt = c.createStatement();
                stmt.execute(query);
                stmt.close();
            }
```

```
        }catch(SQLException e) {
            System.out.println(e.getMessage());
            throw new SalixException ("DBHandler.insertInDatabase(): Could not insert");
        }
    }

    /**
     * Create a new configuration
     * @param name The name of the new configuration
     * @throws SalixException if the configuration could not be created or
     * if there already exists a configuration with the given name
     */

    public void newConfiguration(String name) throws SalixException {
        System.out.println("making new configuration");
        Statement stmt = null;
        String query = "CREATE TABLE " + USER TABLE PREFIX + name +
                       " (TreeID NUMBER PRIMARY KEY, TreeType STRING, " +
                       "TreeAge NUMBER, Solitair BIT, geom MDSYS.SDO_GEOMETRY)";

        if(c == null)
            c = getConnection();
        try {
            stmt = c.createStatement();
            stmt.execute(query);
            stmt.execute("commit");
        }
        catch(SQLException e) {
            System.err.println(e.getMessage());
            if(e.getMessage().indexOf("already exists") != -1)
                throw new SalixException("newConfiguration(): Database " + name +
                                         " already exists");

            if(e.getMessage().indexOf("tax error") != -1)
                throw new SalixException("newConfiguration(): Syntax error " +
                        "in query or configuration name");
        }
    }

    /**
     * Execute raw SQL on the database
     * @param query The SQL query to be executed
     */

    public void executeSQL(String query) {
        if(query.startsWith("SELECT")) {
            executeQuery(query);
            return;
        }
        System.err.println(query);
        Statement stmt = null;
        if(c == null) {

            try{ c = getConnection();}
            catch(Exception e){}
        }

        try {
            stmt = c.createStatement();
            stmt.executeUpdate(query);
        }
        catch(SQLException e) {
            System.err.println("Failed: " + e.getMessage());
        }
    }

    /**
     * Execute an SQL select query
     * @param query The SQL query to be executed
     */

    public void executeQuery(String query) {
        System.err.println(query);
        ResultSet rs = null;
        Statement stmt = null;
        if(c == null) {

            try{ c = getConnection();}
```

```
            catch(Exception e){}
        }
        try {
            stmt = c.createStatement();
            rs = stmt.executeQuery(query);
        }
        catch(SQLException e) {
            System.err.println("Failed: " + e.getMessage());
        }
        try {
            while(rs.next()) {
                double tableName = rs.getDouble(1);
                System.err.println("NUMBER " + tableName);
            }
        }
        catch(SQLException e) {
            System.err.println("Failed: " + e.getMessage());
        }
    }

    /**
     * Set an active configuration
     * @param configuration The name of the configuration to be set active
     */
    public void setConfiguration(String configuration) {
        this.table = configuration;
    }

    /**
     * Get a connection to the database
     * @return The connection
     * @throws SalixException if no connection could be made
     */
    private synchronized Connection getConnection() throws SalixException {
        Connection c = null;
        try {
            c = DriverManager.getConnection(connType+database, "oragis02", "dbms02");
        }
        catch(SQLException e) {
            System.err.println(e.getMessage());
            throw new SalixException("DBHandler.getConnection(): Could not open connection to
            database");
        }
        return c;
    }

    /**
     * Close a connection to the database
     * @param c The connection to be closed
     * @throws SalixException if the connection could not be closed
     */
    private synchronized void closeConnection(Connection c) throws SalixException {
        try {
            c.close();
        }
        catch(NullPointerException e) {
            System.err.println(e.getMessage());
            throw new SalixException("DBHandler.closeConnection(): Connection timed out,
            please reload Salix");
        }
        catch(SQLException e) {
            System.err.println(e.getMessage());
            throw new SalixException("DBHandler.closeConnection(): Could not close connection
            to database");
        }
    }

    /**
     * Get the name of the active configuration
     * @return The name
     */
    private String configurationName() {
        if(table.equals(DEFAULT_TABLE)) return table;
        return USER_TABLE_PREFIX + table;
    }
}
```