Diploma Thesis

GOMS

A Geographical Object Management System

Raphael Huber, IIIC
raphael.huber@gmx.ch

August 3rd 2001

Institute for Information Systems
Swiss Federal Institute of Technology (ETHZ)

Diploma Professor:
Prof. Moira C. Norrie

Supervisors:
Beat Signer
Adrian Kobler

Abstract

Modelling Geographical Information Systems(GIS) using standard notations, such as ER or UML, forces the modelling engineer to build enities containing attributes holding information for geographical representation. Furthermore, there are no facilities to include frequently used spatial constraints into the model. This diploma thesis presents a framework to overcome this lack by specifying spatial abstract data types and geographical constraints. Part of the framework is also a generic viewer and editor application called *Geographical Object Desktop* which makes use of the spatial abstract data types to visualize any GIS built using *Geographical Object Management System* (GOMS). The *Geographical Object Desktop* provides an intuitive way of managing geographical data. The GOMS core is is structured in two layers: The middle layer is to decouple front-end applications from persistence systems which enables portability of former ones and OMS Java at the base is the current persistent Object Management System.

Contents

Chapter 1

Introduction

About one year ago, I joined a lecture about Geographical Information Systems (GIS). One of the readers was C. Parent presenting the model principles described in [1] and [2].

Later I was occupied in terms of a semester work to create a data model for public transportation networks. There were two main goals to reach:

1. It must serve as geographical data basis for the schematization algorithm. An example of such an schematized map is shown in figure 1.1. The algorithm to generate schematic maps is described in [7]

2. It should answer general queries about the transportation network, e.g. *What is the nearest station to the football stadium?*

During the semester work, whose results can be found in [8] and [9], I remembered the concepts presented in the lecture described above - while building abstract spatial types by myself on the model level. There were several geographical constraints which we could not model directly. I also implemented a viewer to visualize the transportation network's geographic reality. At this moment it came to my mind that it would be very nice to have a generic viewer visualizing types from any GIS model. Therefore, it is also important to have predefined geographical abstract data types as basic elements.

After all, I decided to develop such a framework including a generic viewer and editor application within the scope of this diploma thesis.

As persistence base, OMS Java seemed predestined because it is open, extensible and the scope of research in the *GLOBIS group* where I developped already my semester work.

To be compatible with OMS Java and to be platform independent I chose Java as implementation environment. But I was sceptical whether it would

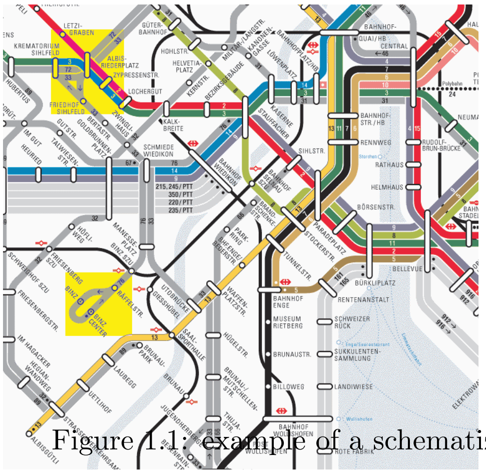


Figure 1.1: example of a schematized map

be possible to realize a fast graphical subsystem - this is one of the most important aspects for practical use from geographic engineer's point of view, if visualisation were slow, work becomes very resinous and intuitive insight to data is decreased.

The set of spatial abstract data types and the spatial constraints as well as the overall architecture of the *Geographical Object Management System* (GOMS) is described in Chapter 2. The Design of the middle layer for basic data structures and spatial indexing is the scope of Chapter 3. Chapter 4 provides design documentation for the viewer and editor application *Geographical Object Desktop*. Conclusions are presented in Chapter 5. Finally there is an appendix containing detailed class documentation to enable future extensions and adaptions.

Chapter 2

Basic Concepts

The basic concepts are based on the principles developed at *Laboratoire de Base de Données* at *EPFL* in [1] and [2]. There were two fundamental principles we took into account:

i) A slim and orthogonal set of basic abstract types for geographical[1] objects. The hierarchy within this set is shown in the figure 2.1. The idea is to define own *normal* concrete types inherited from a leaf or even *generic* concrete types inherited from a branch - this will be explained in section 2.1.

ii) Spatial constraints for associations enumerated in figure 2.2.

2.1 Hierarchy of abstract geographical types

We decided to support only *simple geo* types from figure 2.2 in order to keep the framework manageable. And beside this reason the complex types are more rare and can be built out of the simple types. The resulting hierarchy of GObject[2] types including the basic attributes and methods is shown in figure 2.3.

Examples of derived concrete geographic types are those of table 2.1. Where Source, Pond, River and Lake are derived from WaterBody as well as from their particular abstract type. Similar with Village which is a Town *and* a GPoint - analogously a City is a GArea *and* a Town.

An advantage of generic concrete types, such as WaterBody is to be able to define common attributes there. An example were *pollution* which is meaningful for all, Source, Pond, River and Lake thus it would be placed

[1] 'geographical' and 'spatial' can be considered as synonym in this report

[2] stands for 'geographical object' through all over the project we abbreviated geographical and geographically resp. by 'G' followed by the related substantiv or verb

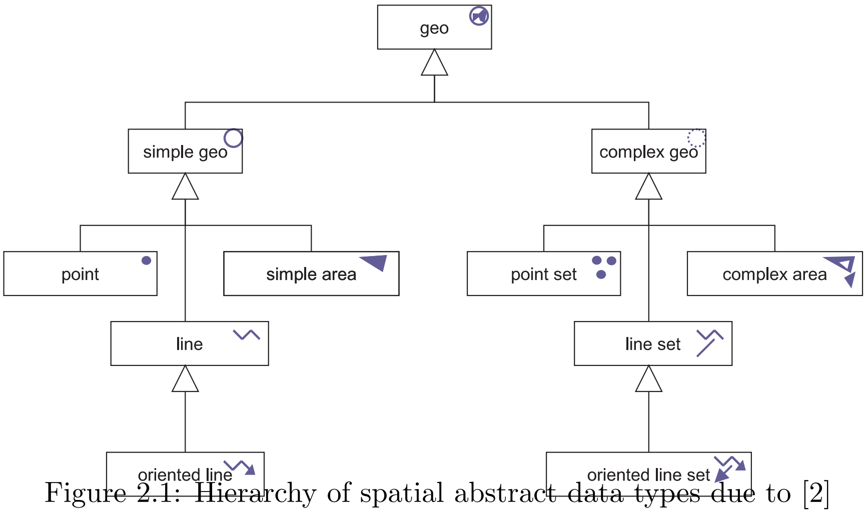


Figure 2.1: Hierarchy of spatial abstract data types due to [2]

spatial type	icon	definition
disjunction		the linked objects have spatially disjoint geometries
adjacency		geometry sharing without common interior
crossing		sharing of some part of the interior, such that the dimension of the part is strictly inferior to the higher dimension of the linked objects
overlapping		sharing of some part of the interior, such that the dimension of the shared part is equal to the dimension of the linked objects
inclusion		the whole interior of one object is part of the interior of the other object
equality		sharing of the whole interior and of the whole envelope (valid for spatial types of the same dimension)

Figure 2.2: Spatial constraints on associations due to [2]

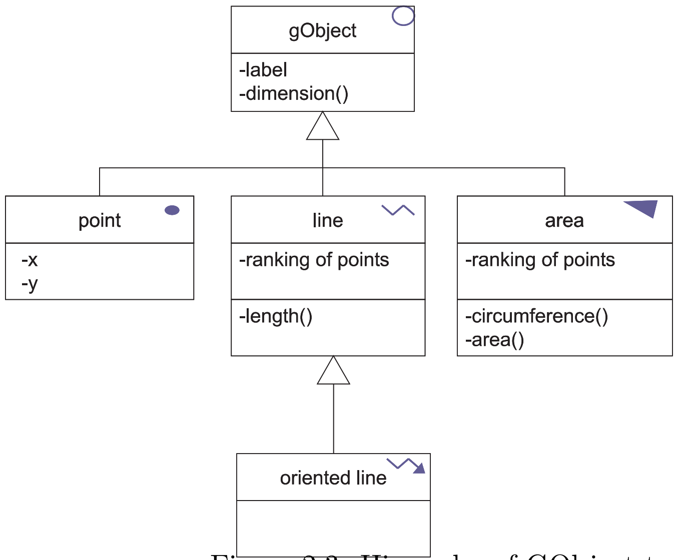


Figure 2.3: Hierarchy of GObject types

abstract type	derived concrete types
GObject	WaterBody, Town
GPoint	Source, Tree, Antenna, Pond, Village
GLine	Street, Wire
GArea	Country, Lake, Island, City
GOrientedLine	River, Pipeline

Table 2.1: examples of concrete geographical types

into the WaterBody type. Another advantage is the common treatment which could be applied to all subtypes of a generic type - specially declaring associations anchored at a collection of generic type. Figure 2.4 gives an example for the data model and DDL[3] for the WaterBodies example.

This concept of generic types needs multiple inheritance, which is not provided directly in Java nor in the current version of OMS Java - for that reason this is not part of the present implementation. But it is possible to partly imitate the generic types by defining *pseudo-generic collections* having GObject as their membertype. The members of such a collection can profit from common treatment but not from having same attributes declared just once. Figure 2.5 shows the WaterBodies example using pseudo-generic collections.

[3]Data Definition Language

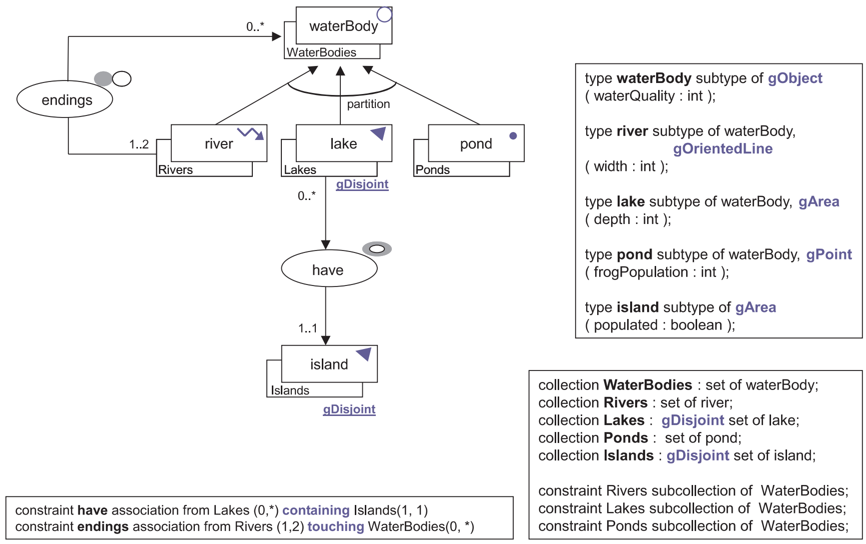


Figure 2.4: the WaterBodies example

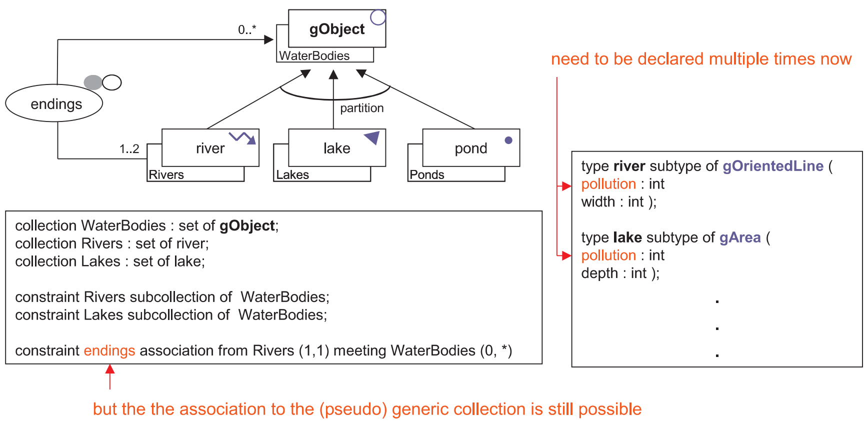


Figure 2.5: WaterBodies example using *pseudo-generic collections*

2.2 Geographical Constraints

Geographical constraints express spatial conditions of and between gObjects[4]. While modelling a database, the DB engineer often needs to specify such constraints to force consistency. For example, lakes need to be spatially disjoint or islands need to be within a lake.

Without the ability of specifying such geographical constraints, the engineer would have to implement algorithms by himself to check the geographical consistency. To use a GIS providing geographical constraints seems to be the preferable solution for two main reasons:

- From an abstract geometric perspective there are just a manageable amount of constraints. Thus it seems unneccessary and fault-prone to implement checking algorithms on a higher logical layer again and again.
- If constraints can be integrated to the data model directly, this improves the comprehensibility of the whole project. Imagine a visual data model including constraints compared to a list of specifications for the checking-algorithms.

Further, geographical constraints make it possible to chose from 3 different association managing alternatives:

1. Associations are generated automatically. Consider the following example: Lakes *have* Islands, where *have* is association with *inclusion* (see figure 2.2) constraint. A pair (lake l, island i) is inserted automatically into *have*, if i is geometrically included within l. This approach leads to redundancy - which is an advantage in terms of accelerated queries.
2. Associations are not stored at all. This is possible, because the geometry of the gObjects in the involved collections defines implicitely which pairs are members of the association. This alternative does not lead to redundant storing. This approach is only possible for associations which are declared to have the spatial constraint as sufficient condition for a pair to be contained. (In "Lakes *have* Islands" were *have* such an example but in "Countries *have capital* Towns" is *have capital* an example, where the *containing* constraint is not sufficient.)
3. The User specifies the contents of associations manually. In this case, The Constraint checker becomes the role of a verifyer. This solution

[4]types are indicated with a capital first letter whereas objects(instances) of a certain type are written by that typename with a small first letter

constraint	*object type*	*specification*
meanderShape	GLine/GArea:	The angles of every two neighboured edges must be either 90 or 270 degrees
rectShape	GLine/GArea:	Must have four edges, formed to a rectangle
straightLine	GLine	Must have exactly 2 vertices
cycleLine	GLine	1st and last vertex must be at same position
openLine	GLine	Must not intersect itself - touching is not allowed either

Table 2.2: object constraints

leads to redundancy as well - which is good for quality assurance in this case.

Additionally to the constraints on associations described conceptually at the beginning of this chapter, we introduce the object and layer constraints here, thus we distinguish between 3 types of geographical constraints now:

Object constraints specify restrictions on gObjects without involving any other. Those are defined on a certain layer[5], which tells latter to contain just gObjects fulfilling the particular constraint, i.e. if a gObject does not satisfy the constraint it is set as invalid. Table 2.2 shows an expandable list of object constraints.

Figure 2.6 shows examples of valid and invalid gObjects in context of the object constraints.

Object constraints could be combined by boolean operators, for example:

```
collection PathAroundAcres: (meanderShape ∩ cycleLine) set
                         of gLine;
```

This concept is not part of our implementation and would need a checking algorithm to verify such boolean expressions, for example it is prohibited to specify:

```
(straightLine ∪ cycleLine)
```

Figure 2.7 gives a basic idea of possible combinations and could serve as a base for such an algorithm.

[5] *Layer* denotes a collection with geographical membertype

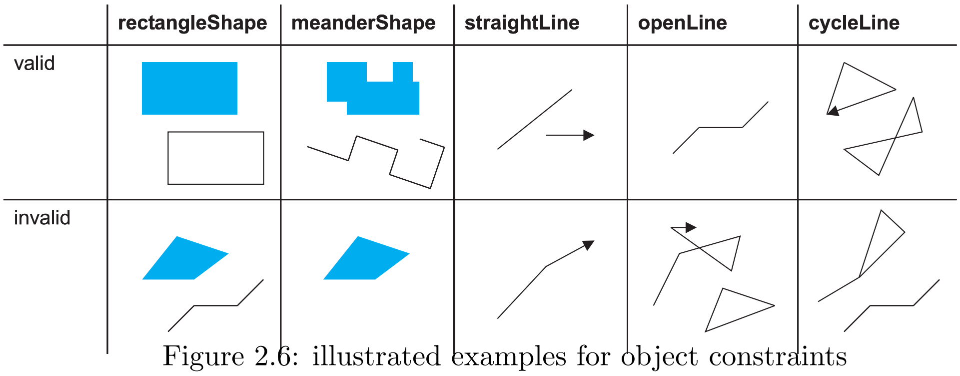


Figure 2.6: illustrated examples for object constraints

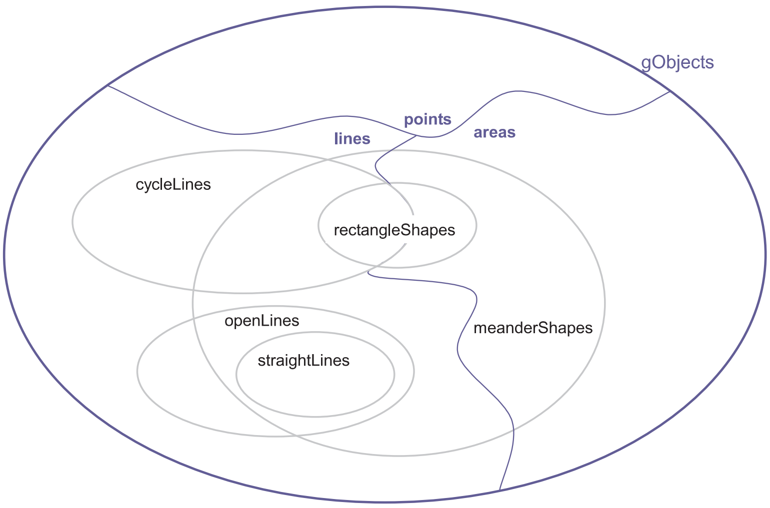


Figure 2.7: set diagram for object constraints

constraint	*specification*
disjoint	None of the gObjects must have any common point with any other gObject in this layer
touching	Each pair of gObjects within the layer must be disjoint or at most touching.

Table 2.3: layer constraints

Layer constraints are used to describe conditions between the various gObjects within a layer. Those restrictions must be valid for all possible pairs within the layer. If a pair failes, its two gObjects are invalidated. Table 2.3 gives a list of possible layer constraints.

Association constraints specify a spatial condition that need to be established for every pair of gObjects within a certain association.

These constraints are combined with the conventional specification of cardinalities, which have the be fulfilled as well. Thus looking at the WaterBodies example, there need to be for every Island i exactly 1 pair in the *have* association relating to a lake whose area contains the area of i completely. For a lake there can exist an arbitrary amount of associated islands.

Table 2.4 shows an expandable list of association constraints. Those are strongly related to the rows in figure 2.2. In that table the first four constraints are symetric the next two are directed and the last one is a special case of spatial aggregation. All the association constraints are to be understood in context of specified cardinalities, thus the *containing* constraint, for example, would be written in its full version as follows:

Any object from source layer must contain between *tmin* and *tmax* gObjects from target layer. And any object from target layer must be contained within between *smin* and *smax* gObjects from source layer.

Where *(smin, smax)* and *(tmin, tmax)* are the specified cardinalities for source and target layer.

In the following, we will describe the possible spatial relationships between a pair of gObjects. Note that some are redefined compared to figure 2.2. These redefinitions seem to be less pure than the original ones, but more intuitive and suitable for concrete geographical reality[6]. Figures 2.8, 2.9 and 2.10 illustrate the specifications below.

disjoint Every two gObjects are defined to be disjoint if they share not

[6]But also with these adapted specifications we are still sceptical - practical use of the framework must decide about further refinements, specially in *touching* and *crossing*

constraint	*specification*
disjoint	Objects must have disjoint geometries
touching	Objects must touch
overlapping	Objects must overlap
equal	Objects must be of equal geometry
crossing	Source object must cross target object
containing	Source object must contain target object
consisting of	Union of target objects must form source object

Table 2.4: association constraints

even one common point.

touching Two points are touching if they are equal. A point is touching a line or an area, if it is at a vertex position. A line is touching a VertexShape if both are disjoint with exception of one or both of line's endings which must coincide with a vertex of the vertexShape. Two areas are touching, if they are disjoint except sharing of edges.

overlapping Two gObjects are defined to be overlapping, if they are of the same dimension and share a common part which has the same dimension as themselves.

equal Equal are those two gObjects that have identical geometric representations.

crossing A source gObject *go* is crossing a target gObject *to*, if *go* is of linear type on one hand.
On the other hand:
if *to* is a point, *to* needs to be a vertex of *go*.
if *to* is a line, *go* and *to* need to have a point of intersection, which is none of the four endings.
if *to* is an area, there must be a common part of linear dimension.

containing One gObject is containing another one, if every point of the latter is part of the former.

consistingOf One gObject is consisting of others, if the union of the latter ones result in geometry of the former one.

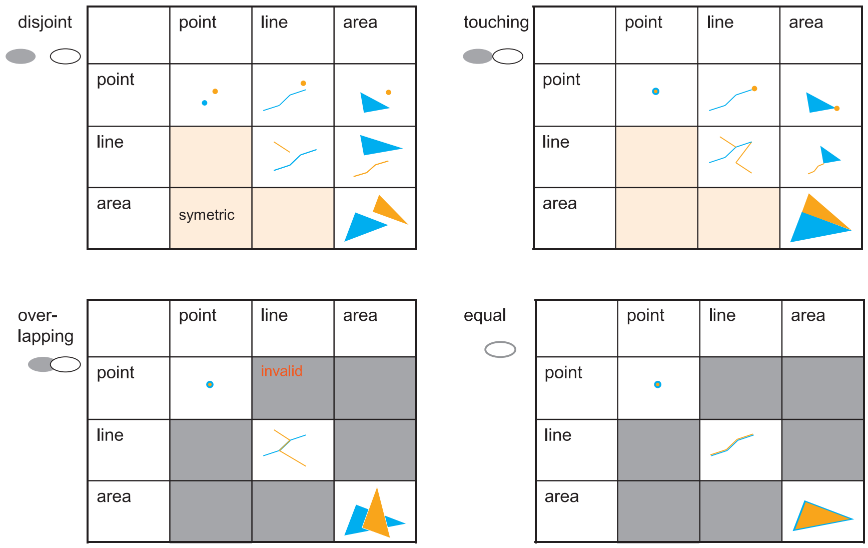


Figure 2.8: symetric spatial relationships

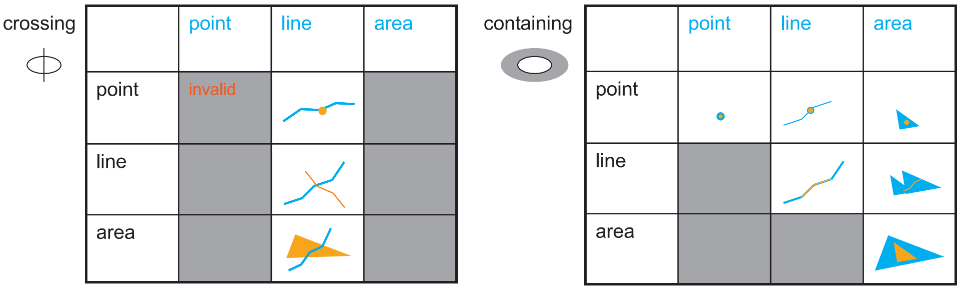


Figure 2.9: non-symetric spatial relationships

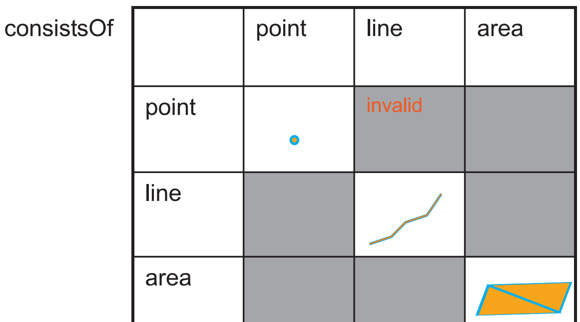


Figure 2.10: spatial aggregation

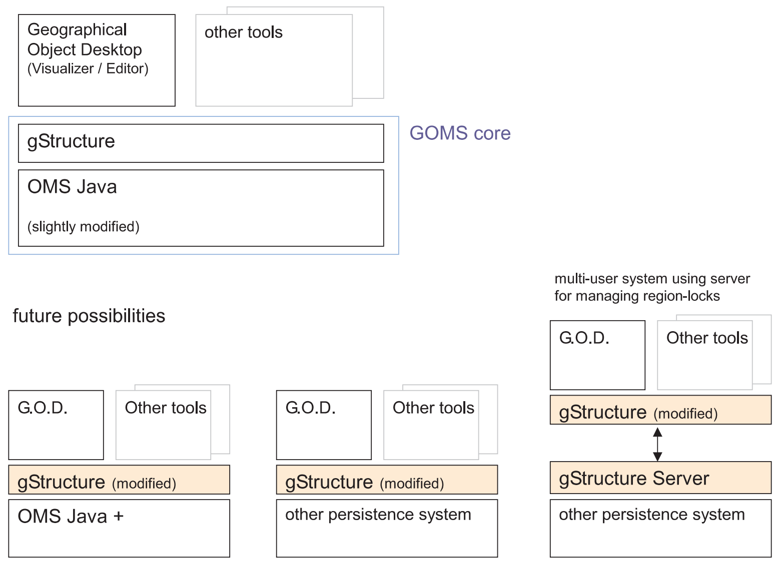


Figure 2.11: architecture of GOMS

2.3 The GOMS Architecture

GOMS consists of OMS Java ([3], [4] and [6]) and the overlaid layer for geographical structures *gStructure*. This design is intended to decouple client applications from persistent systems. So the existing client application GOD[7], for example, can be ported without changes using other architectures providing their adapted version of gStructure. Figure 2.11 illustrates these principles.

In the current implementation, the extensions in OMS Java are summarized to a package called *gisExtensions* and limited to DDL-Parser modifications supporting the specification of geographical constraints.

Other useful front-end applications are for example the following:

- A very slim web viewer without features to modify data
- A tool to manage import and export facilities

The design of the gStructure for OMS Java and the client application GOD are documented in the following two chapters.

[7]Geographical Object Desktop, pronounced "Geowdee", to avoid confusion.

Chapter 3

Geographical Structure Base

Due to figure 2.11 there is an abstraction layer called *gStructure* between persistence system and *Geographical Object Desktop* and other client applications. So those client applications could be used - without any changes - on other persistence systems if *gStructure* has been adapted for it. There are two more advantages of having the *gStructure* layer:

1. Encapsulation of index-revalidating after modifications on gObjects. All spatial indices are based on the principle of partitioning the space into cells. As a result queries which depend on a given location must not scan all the gObjects anymore, but only those cells which are at that location. But if any gObject is translated for example, it might fall into a different cell than before. Thus every spatial modification requires the index structure to be revalidated, which is not trivial and hidden from client applications through the gStructure.

2. For performance reasons there are a lot of low level representations and redundant attributes within gStructure which need not to be visible for a client application.

Figure 3.1 gives an overview of *gStructure* and its relations to the underlying and the client layer, whereas figure 3.2 shows an UML class diagram of it.

The centre of gStructure is the Model, which is the representation of the data model with its collections, associations and constraints. It is also the ressource of AbstractViews to get its data to visualize.

Another core class is GObject which represents the base of all gObjects.

GSet is an abstract class to contain a set of gObjects. Concrete subclasses typically use spatial index structures to manage those sets.

Class Div is an auxiliary class which contains static methods for general purposes.

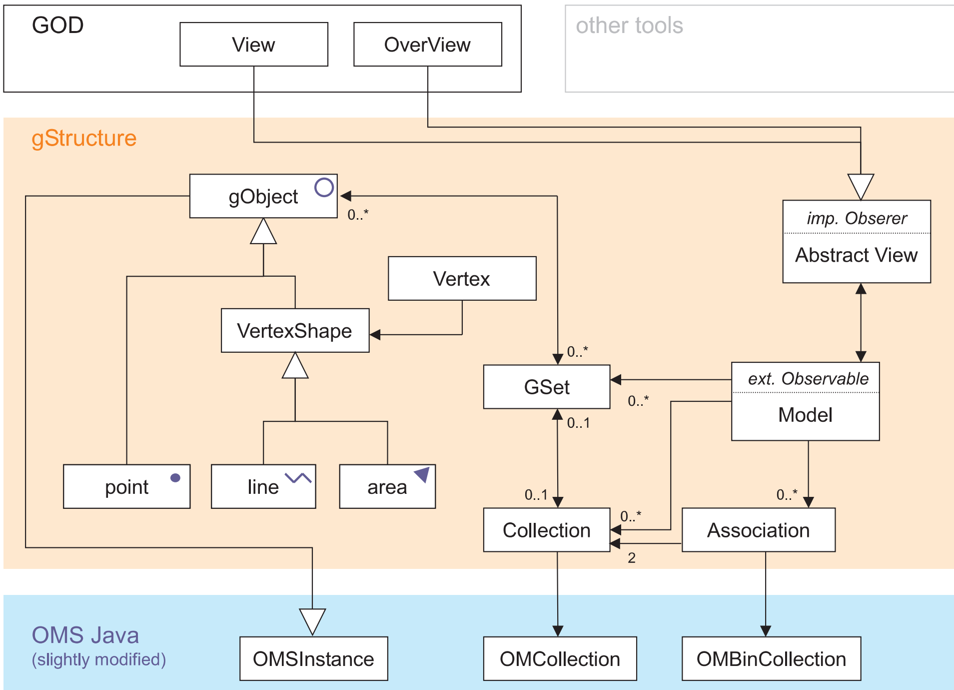


Figure 3.1: anatomy of the geographical structure base

The details concerning the classes in gStructure can be read in the following sections.

3.1 GObjects

GObject

In the present implementation of gStructure based on OMS Java, GObject is a subclass of OMSInstance. All coordinates are integer numbers. The Decisive points for those instead of floating point are the following:

- Exact and fast arithmetics
- Transparent to users and programmers: Users see the model coordinates in a high zoom level as squares and thus can easily recognize what is touching or intersecting etc. Programmers do not need to care about additional FP complexity.
- Same density overall the range of maps. FP numbers have higher granularity around 0.

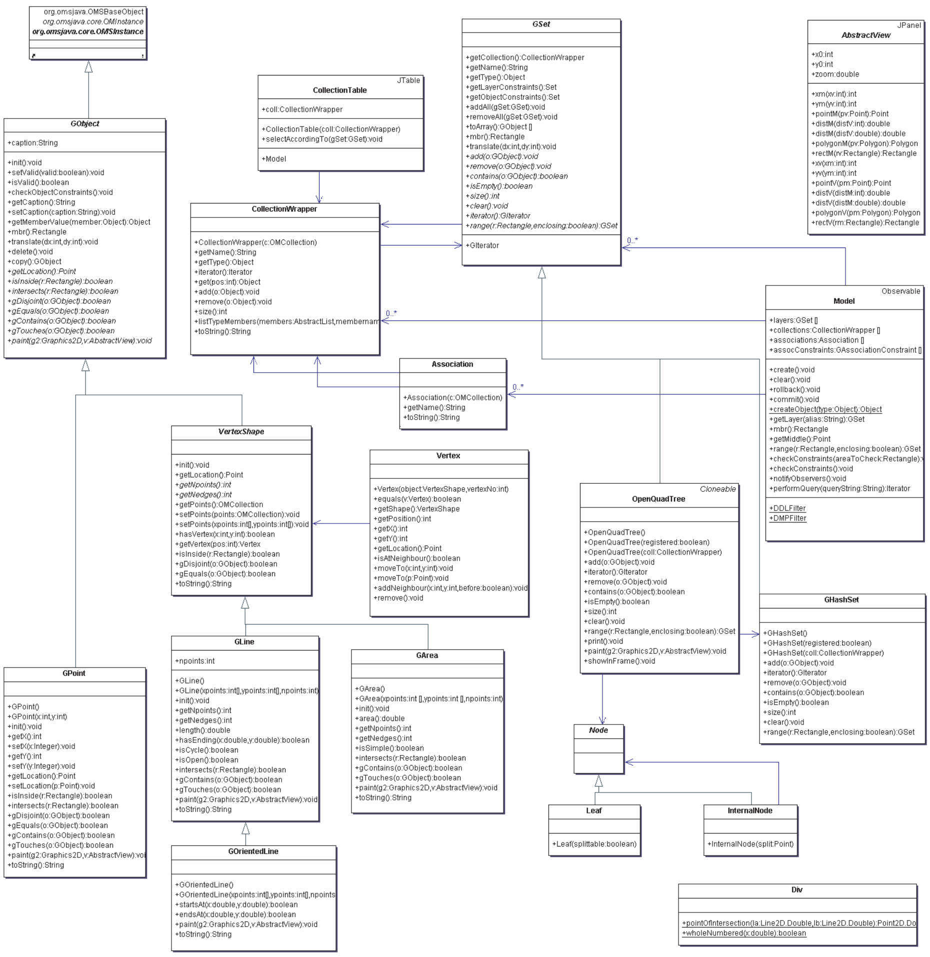


Figure 3.2: class diagram of gStructure

- Fractional numbers could be emulated in Dialogs by defining some digits as decimal places.

We chose *int* instead of *long* to be compatible with OMS Java and the classes from *java.awt* package.

The common functionallity of the subclasses is implemented in this abstract class, which contains the following fields and methods: There is a field `registeredSets` where all gSets are stored that contain this gObject and want to be informed of spatial modifications to revalidate themselves.

There are the final methods `translate(..)` and `copy()` which call translateConcrete(..) and `copyConcrete(..)` in the concrete subclass between execution of general code. For more details refer to the API Reference appendix

Other important methods to mention are `getMemberValue(..)` and setAttributeValue(..) which are the public interface towards client applications to access non-spatial data of the gObjects.

All the boolean methods of spatial conditions where constraints are based on, are abstactly declared in GObject, so it is guaranteed to constraint checkers that every gObject is able to know about its spatial relations to others. Those are:

- `boolean gDisjoint(GObject o)`
- `boolean gEquals(GObject o)`
- `boolean gContains(GObject o)`
- `boolean gTouches(GObject o)`
- `boolean gOverlaps(GObject o)`
- `boolean gCrosses(GObject o)`

There is a public `init()` method in GObject as well. This is called by framework to initialise transient fields of the gObject. The concrete subclasses can overwrite the `init()` method (but need to call the `super.init()` at the beginning) to initialise their own transient fields. There is a mechanism to prevent clients from unintended initialising of existing gObjects - which would violate consistency with the registered indices: Further calls of `init()` on a certain gObject are without effect.

There is an abstract paint operation which must be implemented within every concrete subclass. This paint method needs to know about graphics context to draw within as well as about offset and scale factor (which are attributes of AbstractView). Thus its signature is the following:

```
public abstract void paint(Graphics2D g2, AbstractView v);
```

VertexShape & Vertex

This is an abstract class to summarize common fields and behaviour of GLine and GArea. Moreover it enables common treatment of those under certain circumstances.

The most important fields are the two arrays xp and yp representing the coordinates of the vertices of the VertexShape. This low-level reprentation is though fast for computational geometry, but prone to inconsistency due to wrong handling. Thus these arrays are just package-visible.

As a public interface to modify these vertices there exist the Vertex class. Client layers of gStructure get a vertex from both GLine and GArea via

```
public Vertex getVertex(int pos)
```

where `pos` is the number in the desired vertex within the whole sequence. On the received Vertex object there are 3 operations available to modify the VertexShape in its form:

- `public void moveTo(int x, int y)`
- `public void addNeighbour(int x, int y, boolean before)`
- `public void remove()`

where `before` is an indicator whether the new vertex shall be inserted before or afterwards this vertex within the whole sequence.

These modifying operations involve a revalidating of spatial indexed containers - as descibed in the beginning of chapter 3 - which is performed automatically, hidden from the client layer.

There are some more methods in the Vertex class to get information about vertices. Refer to figure 3.2 or the API Reference appendix to get more details.

GPoint

GPoint is the concrete class from which any point-shaped objects will inherit from. Examples are sources, trees, antennas etc. - of course it depends on the circumstances whether an object might be modelled as point-shaped, linear or as area.

The operations available for GPoint are quite intuitivly comprehensable and will not be documented further here. For more information refer to figure 3.2 or the API Reference appendix.

GLine

GLine consists of an ordered set of vertices inherited form VertexShape, which result in a sequence of edges. Its public `length()` method thus just summarizes the length of all segments.

Most of GLine's functionallity is covered by VertexShape. But there are two methods which might not be underestimated: The implementations of `boolean gContains(GObject o)` and `boolean gTouches(GObject o)` - they are more complicated than one might guess, but documented in detail within code.

GOrientedLine is a subclass of GLine without any additional fields - the set of vertices was already ordered in GLine and gets the semantics of direction quite naturaly. GOrientedLine overwrites just the `paint` method as a hint for visualisation and adding two more methods for further thinkable constraints:

- `boolean startsAt(double x, double y)`
- `boolean endsAt(double x, double y)`

GArea

GArea is the base of concrete 2 dimensional entities. It is specified to be a simple polygon, which is defined as being free of self-intersections, if this condition is violated, it will be set to invalid. Checking the *simplicity* of an area is performed as an implicit constraint in GObject.checkObjectConstraints(). Similar to GLine, GArea consists of a sequence of points. The chosen design is the same as in java.awt.Polygon: A GAera with n vertices consists of n points. (In other systems it would be built out of n+1 points where $point_0$ and $point_n$ share the same position - **not in gStructure** This decision allows us to profit from all built-in methods of java.awt.Polygon. GArea is wrapping java.awt.Polygon - more precisely even *infiltrating* - because it is coupled not just by one reference to the polygon: While constructing, the Polygon is instantiated first and afterwards those coordinate arrays as well as the bounds attribute are referenced by the own attributes. This is illustrated in figure 3.3. This design can also be seen as a so-called *Proxy*, which is described in [18].

Doing so, we can use those attributes of the polygon directly without need of synchronizing own attributes with those of the polygon. One might ask why not just extending GArea from Polygon - the answer is that Java does not allow multiple inheritance and GArea must be a GObject, a VertexShape more precisely. So another might ask why not just declaring

- `abstract Rectangle getBounds()`

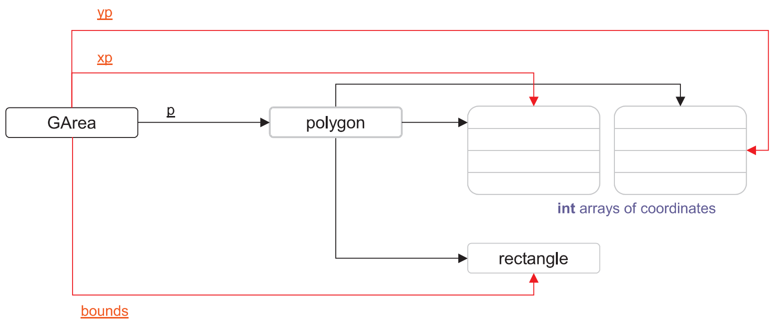


Figure 3.3: GArea infiltrates java.awt.Polygon

- `abstract int [] getXp()`
- `abstract int [] getXp()`

in GObject and VertexShape and then implement those accessor methods for GArea as follows:

```
Rectangle getBounds()  return p.bounds;
```

(analogous for the others). This would beware from complicated infiltrating mechanism. Well, the reason is again graphic performance on the one hand - accessing fields is faster than invoking methods. And on the other hand, using the attributes directly makes the code more readable and smaller in size.

Another thing to mention is the algortihm to calculate polygon area, which is not trivial. The principle of the algorithm is illustrated by an example in figure 3.4

3.2 Index structures - GSets

GSet is an abstract container for GObjects only - thus the signature of the basic methods defined on GSet are the following:

- `add(GObject)`
- `remove(GObject)`
- `boolean contains(GObject)`
- `GSet[] toArray()`

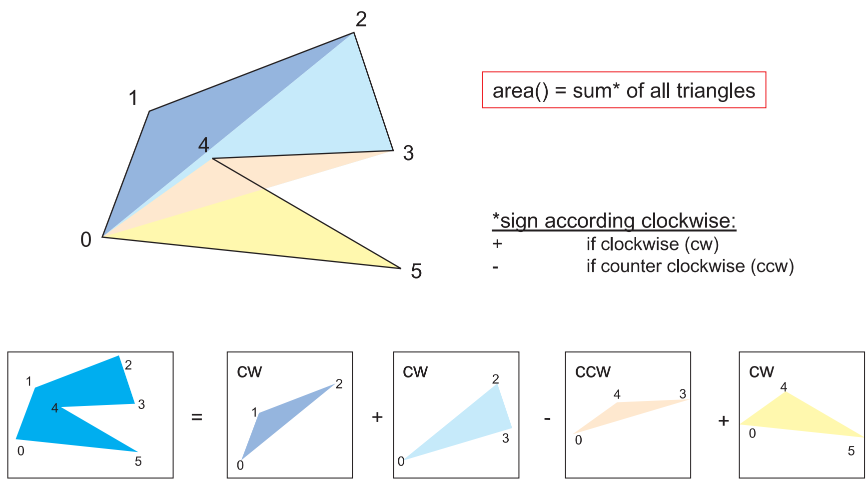


Figure 3.4: calculation of polygon area

- `GSet.GIterator iterator()`

where GSet.GIterator returns a GObject instead of Object in its `next()` method.

There is a lot of additional operations defined for GSet whose purpose can be understood intuitively and will not be documented further in this part. For more details see API Reference appendix.

GSet may - as its name indicates - contain every gObject just once without storing any sequential order on them. For managing multiple and ordered occurrences we use the Collection[1] class. These two - GSet and Collection - can be used in combination, where the task of GSet is the spatial indexing of the collection.

But gSets as well as collections may exist in single form too. For example, results of range queries are of GSet type and collections having non-geographical mebertype have of course no spatial indices.

There are at the moment two different implementations of GSet:

- OpenQuadTree - spatially indexed based on Quad-Tree principles
- GHashSet - non spatially indexed

We will not write here about GHashSet, because it is mainly a wrapper of java.util.HashSet. Refer to the API Reference appendix to get more detailed information. But it is worth to write more about OpenQuadTree.

[1] in current implementation with old name CollectionWrapper

OpenQuadTree uses a data-driven[2] indexing strategty, which divides the space recursively into four sub-partitions. Therefore there are classes InternalNode and Leaf, which are both extensions of abstract Node class. We called this class *Open*QuadTree, because the indexed area is not needed to be limited: border-partitions always represent a region of infinite area. This bewares from rebuilding the tree everytime gObjects are inserted, which are outside model's MBR [3]

InternalNode points to its four sub-partitions which are representeed by static type Node - dynamically they can be either InternalNode or Leaf, depending on whether this partition is devided again or not.

A Leaf contains those gObjects which are lying within or overlapping its region. If a specified amount is exceeded, it will be changed to an InternalNode and its region split into four new Leaves. The geometrical center of those four new leaves is set to the center of gravity of the gObjects contained. Refer to figure 3.5 for illustration of splitting priciple.

There is a special case in which splitting will not be performed: If the region represented by the leaf became so small that it is about average size of contained gObjects. Splitting under those circumstances would lead to infinit splitting, because all gObjects contained in the original leaf would be pushed to all children as well! Such Leaves are set to be unsplittable.

There are a lot of other indexing strategies which would be implemented in further GSet extensions. General information about spatial indexing data structures can be found in [10], [17] and [15]

3.3 Model & View

The model is the center of gStructure, since it contains the following 3 arrays as attributes:

- `public GSet [] layers;`
- `public Collection [] collections;`
- `public Association [] associations;`

where *layers* are those collections which have a geographical membertype and thus are referenced by their indices. The model structure is illustrated[4]

[2]Means: partitions are spread according to object density (vs. *space-driven*: where space is divided homogenously

[3](rectangular) minimum bounding region

[4]all over this report rounded rectangles are used to idicate instances whereas *normal* rectangles are used for classes. Furthermore rounded rectangles containing other rounded rectangles indicate referencing.

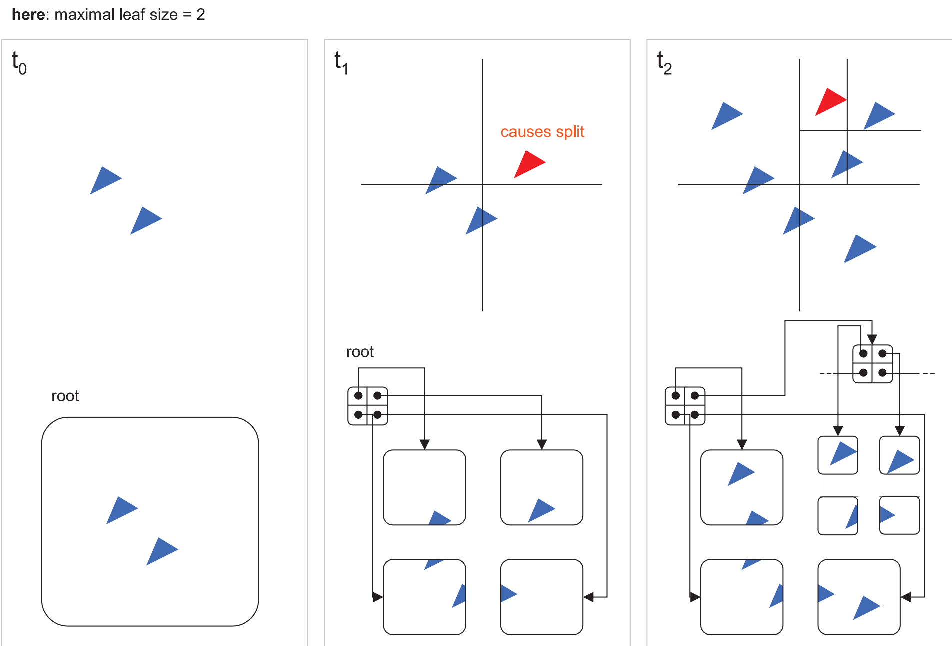


Figure 3.5: sketch of an OpenQuadTree

in figure 3.6 The basic operations to build and access the model strucuture can be found in the API Reference appendix

Collection and Association are wrapper types referencing OMCollection and OMBinCollection in the present implementation of gStructure. Wrapping is neccessary to provide an independent abstraction to client applications. Within Collection and Association there are the two fields `parents` and `covered`. Former is to establish the sub-/super-collection structure with the other collections and latter is to specify the collection to be fully covered by sub-collections. This means that all the contained objects need to exist in at least one sub-collection. If the collection is set to `covered`, no objects can be added directly to it - to ensure consistency. Thus, the objects must be added to a sub-collection from where they are inserted automatically also to its parents. The principle of indexing a collection by an overlaid gSet can be seen in figure 3.7 (in combination with figure 3.5 where OpenQuadTree class as an extension of GSet is illustrated)

The Model class provides the following 3 important geometric operations:

- `public Rectangle mbr()`
- `public Point getMiddle()`
- `public GSet range(Rectangle r, boolean enclosing)`

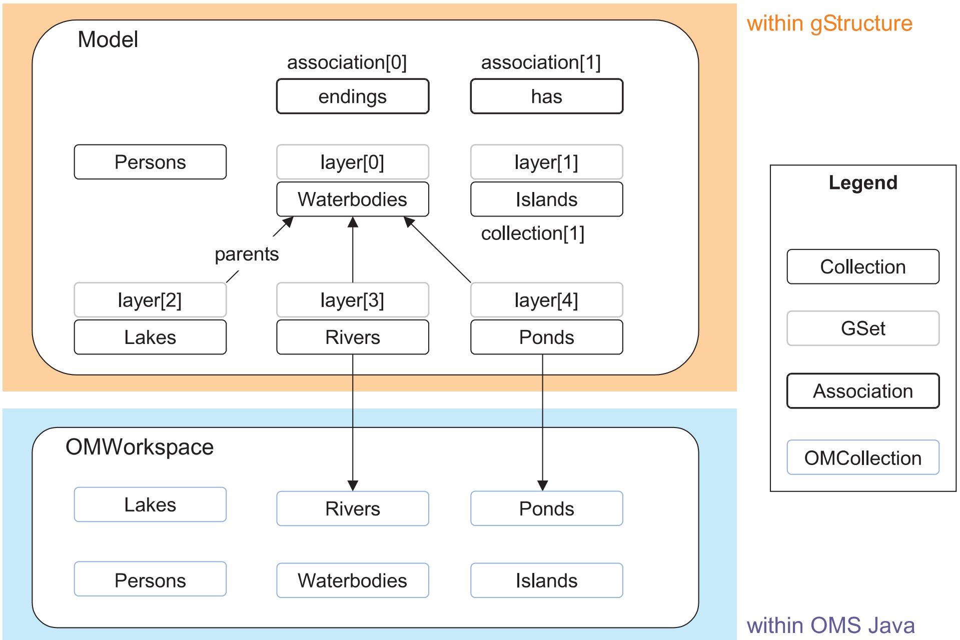


Figure 3.6: model structure for the WaterBodies example

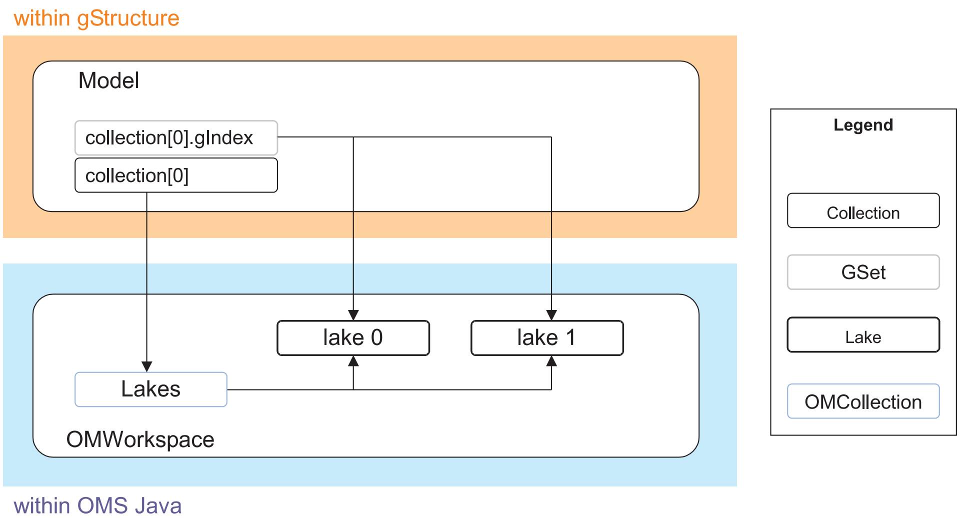


Figure 3.7: collection indexed by a gSet

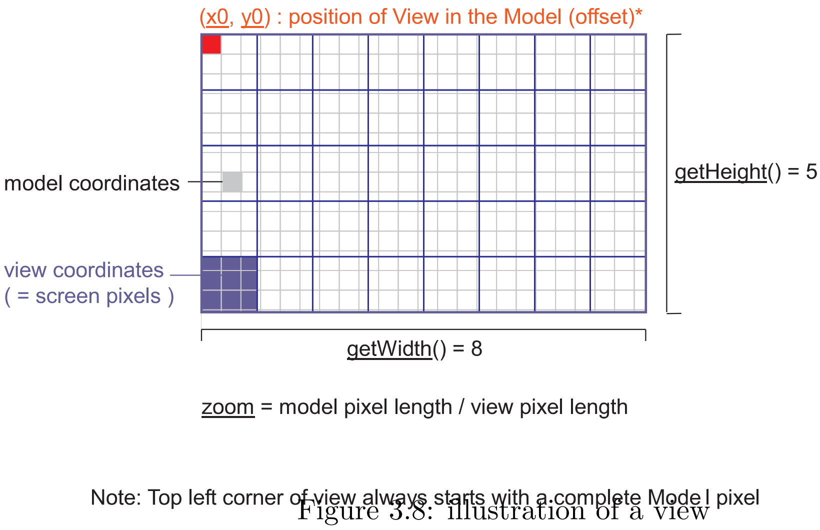


Figure 3.8: illustration of a view

where `mbr()` gives an MBR of the whole model content and `range(..)` returns the gObjects in the specified rectangle looking at all its layers.

The base of all views is class AbstractView. It provides fields[5] for offset and scale as shown in figure 3.8.

Additionally, there are methods for coordinate transformations, illustrated in figures 3.9 and 3.10. Coordinate transformations are used for example to determine object which was clicked (view to model) and to paint the object in a view (model to view).

The Principle for coordinate transformation is the following

view to model:	Find the model pixel (xm, ym) which contains the middle of a given view pixel (xv, yv)
model to view:	Vice versa

This leads to the following formula for coordinate transformation of a point:

$$xm(xv) = x0 + \left\lfloor \frac{xv+\frac{1}{2}}{zoom} \right\rfloor$$
$$ym(yv) = y0 + \left\lfloor \frac{yv+\frac{1}{2}}{zoom} \right\rfloor$$

[5]fields are indicated in figures by underlined text

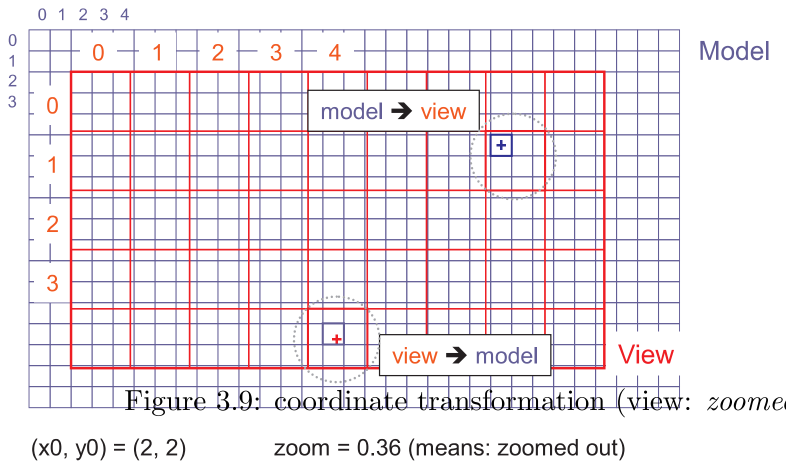


Figure 3.9: coordinate transformation (view: *zoomed out*)

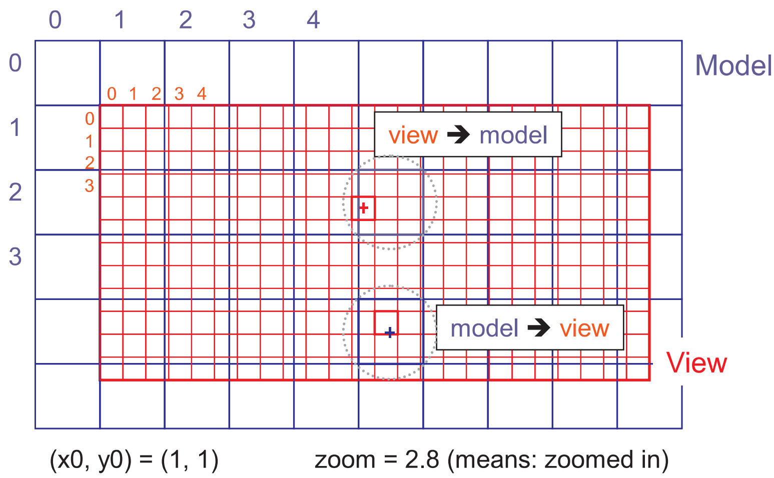


Figure 3.10: coordinate transformation (view: *zoomed in*)

$$xv(xm) = \left\lfloor (xm - x0 + \tfrac{1}{2}) * zoom \right\rfloor$$
$$yv(ym) = \left\lfloor (ym - y0 + \tfrac{1}{2}) * zoom \right\rfloor$$

In the AbstractView class there are methods for transforming rectangles and polygons as a whole, i.e. they just transform all the vertices.

No view does store any data itself, it just visualizes data from the model. More about this principle - called MVC - can be read in [18]. In the current implementation, the MVC is realized by the Observer Pattern - explained in [18] as well.

3.4 Constraints

All the classes representing constraints are separated to the sub-package *gStructure.constraints*. Figure 3.11 shows an overview of that package.

As described in chapter 2.2 there are 3 types of geographical constraints: object, layer and association constraints. Those are represented by basic abstract classes GObjectConstraint, GLayerConstraint and GAssociation-Constraint.

GObjectConstraint forces its subclasses to implement the following method:

```
public abstract boolean check(GObject o);
```

where concrete constraint classes, such as GCycleLineConstraint have code to check whether the given gObject satisfies the condition or not. All the object-constraints are checked by invoking the `checkObjectConstraints()` of a GObject instance - if any of the constraints failed, this gObject will be set to be invalid.

GLayerConstraint and GAssociationConstraint have similarly the

```
public abstract boolean checkConcrete(GObject source, GObject
                          target);
```

method which must be implemented by those concrete subclasses.

Within GLayerConstraint, there is the method check(Rectangle areaToCheck) to check a concrete layer constraint within a specified area. If invoked, all the gObjects o of that layer and within areaToCheck will be checked to satisfy the checkConcrete(o, candidate) condition - where candidate is every other gObject within areaToCheck. If a gObject failes, its valid attribute will be set to false.

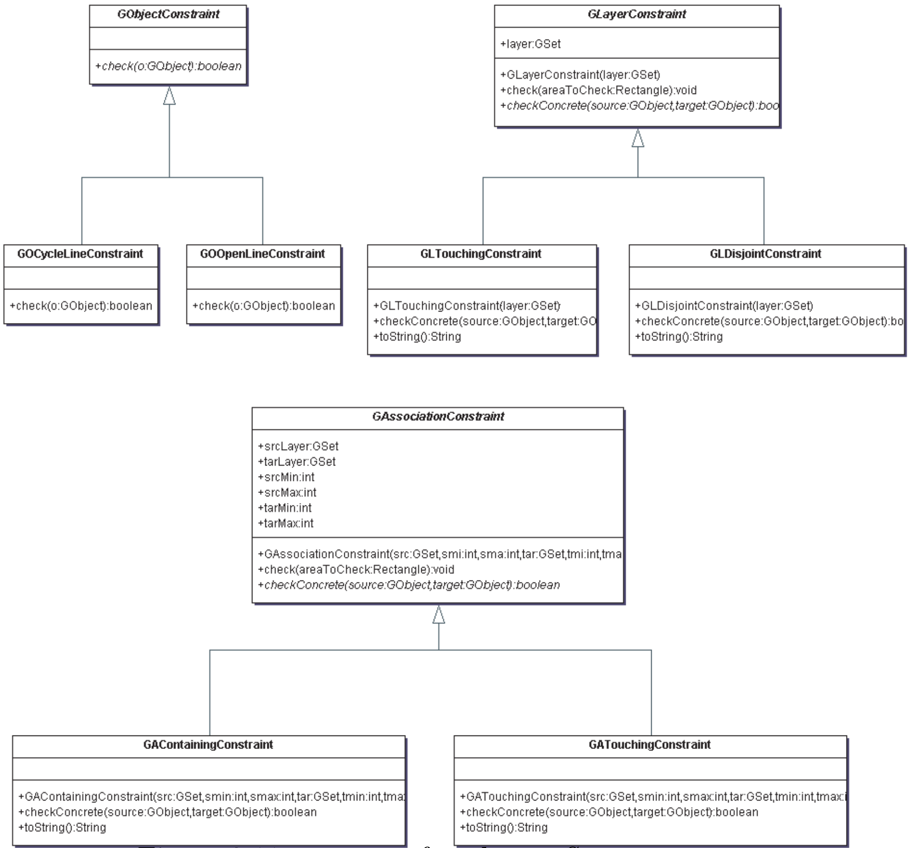


Figure 3.11: content of package *gStructure.constraint*

GAssociationConstraint as well has a method check(Rectangle areaToCheck) to check a concrete association constraint within a specified area. This check method is outlined as follows:

1. Check every gObject o from source layer to satisfy the checkConcrete(o, t) for tmin to tmax gObjects t from target-layer: If failed: o.setValid(false)

2. Check every gObject o from target layer to satisfy the checkConcrete(s, o) for smin to smax gObjects s from source-layer: If failed: o.setValid(false)

where source and target-layer as well as smin, smax, tmin, tmax are the attributes declared within GAssociationConstraint to specify the constraint. As an example, the *have* constraint from the WaterBodies example has Lakes as source layer, Islands as target layer and (1, 1), (0, *) as (smin, smax), (tmin, tmax).

Chapter 4

Geographical Object Desktop

The Geographical Object Desktop is a front-end application built on top of the *gStructure* layer. The main purposes of GOD are the following:

- visualizing and thus increase comprehensability of correlations in geographical data
- editing of both logical and spatial data, where spatial editing is performed by mouse clicking and dragging
- interface for queries, where spatial results are mapped to visual output

Figure 4.1 shows a screenshot of the *geographical Object Desktop*. And figure 4.2 shows the structure of the GUI components, paricularly the custom ones - comparison to figure 4.1 might increase the readability.

There are 3 tables involved each accessing its own TableModel. All those TableModels have not a direct graphical representation but are included to figure 4.2 to show the essence of the tables. All the TableModels are embedded as member classes within a related outer class. To give an understanding of TableModel-principle, we present the method signatures they implement in table 4.1.

All these methods are called by Java's table-renderer and -editor to plug in the custom functionality. All three custom TableModels are extensions of AbstractTableModel from the *javax.swing* library. More information about TableModels in general can be found in [14].

After this brief introduction to GUI structure and TableModels in particular, we provide an overview of the whole *god* package in UML notation in figure 4.3.

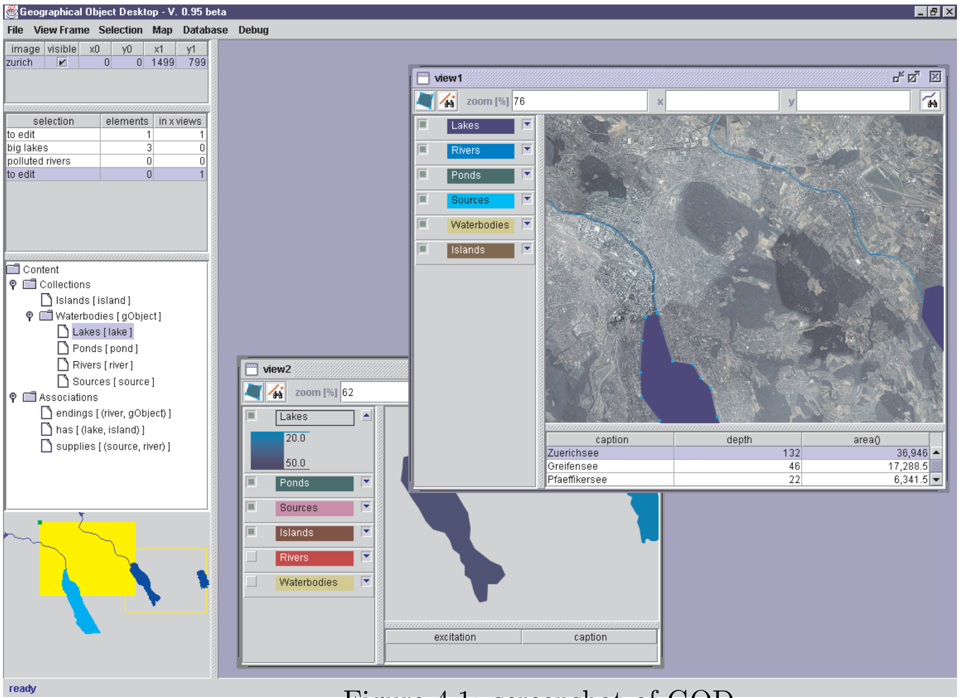


Figure 4.1: screenshot of GOD

returns	*method name*
`int`	`getColumnCount()`
`int`	`getRowCount()`
`String`	`getColumnName(int col)`
`Object`	`getValueAt(int row, int col)`
`Class`	`getColumnClass(int col)`
`boolean`	`isCellEditable(int row, int col)`
`void`	`setValueAt(Object value, int row, int col)`

Table 4.1: method signatures in custom TableModel classes

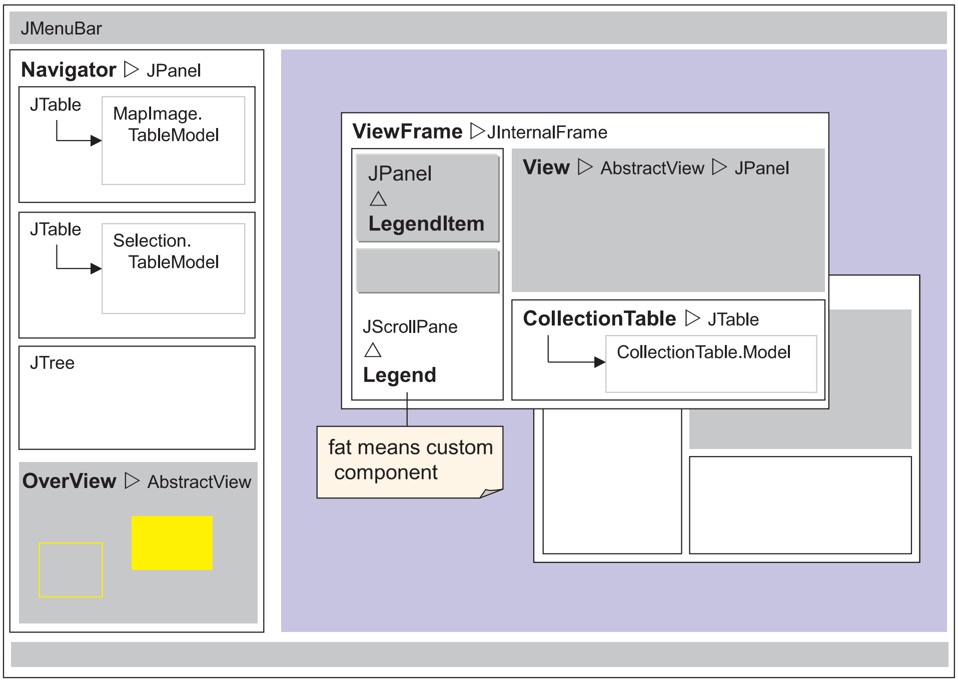


Figure 4.2: GUI structure of GOD

GOD class is the center of the application. It is actually an extension of JApplet for being able to be displayed within browsers as well[1] In God there are the important fields listed in 4.2 which indicate its central managing role:

type	*fieldname*
`JMenuBar`	`menuBar`
`JDesktopPane`	`desktop`
`Navigator`	`navigator`
`Clipboard`	`clipboard`
`Model`	`model`
`ArrayList`	`viewFrames`
`ArrayList`	`selections`
`ArrayList`	`mapImages`

Table 4.2: important fields within the God class

[1]For the current implementation, a policy file is neccessary to allow access to local files, refer to [11] for use of *policytool*

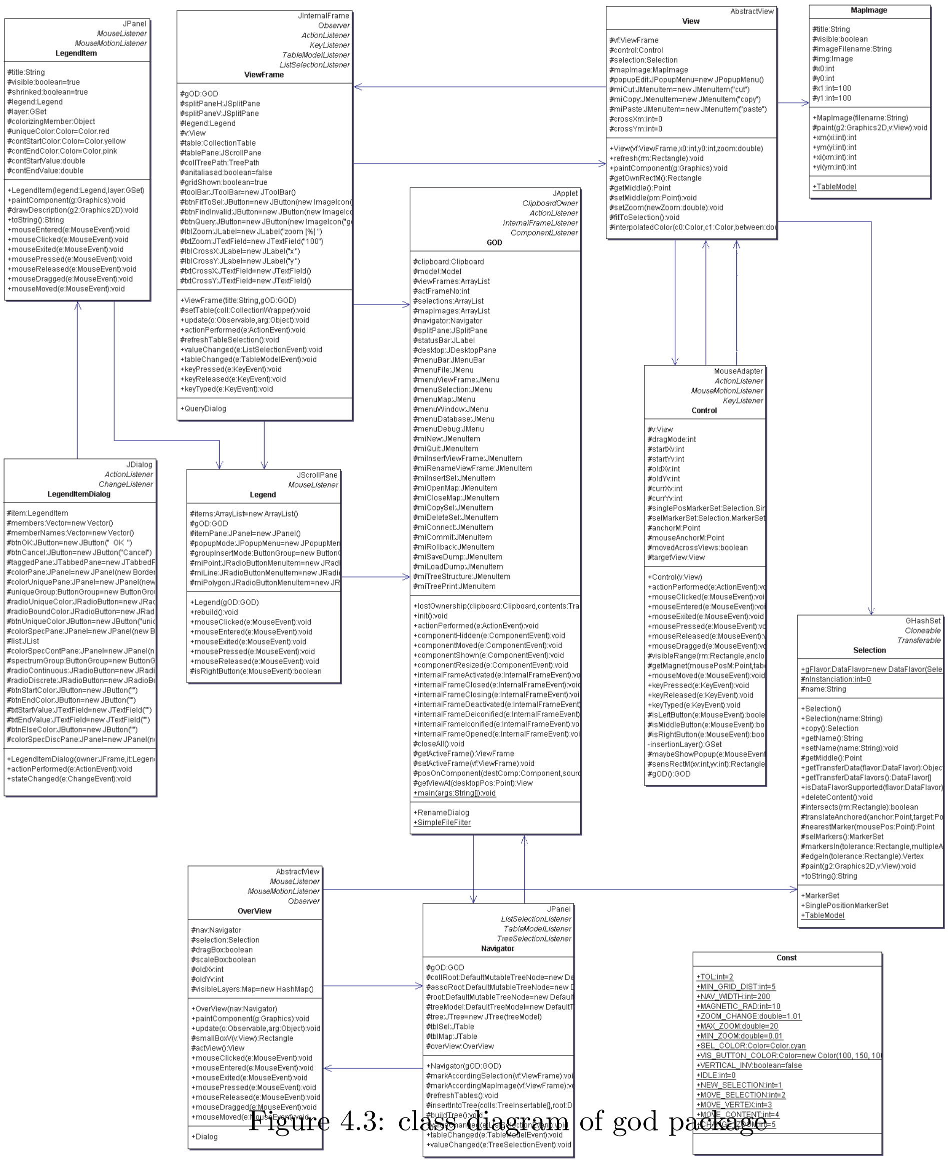


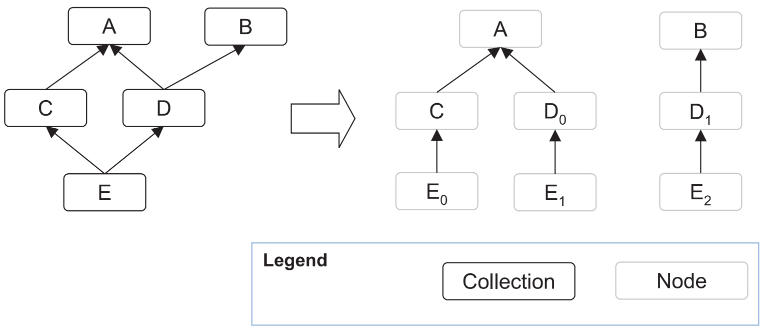
Figure 4.3: class diagram of god package

ViewFrame class represents a container of a view, legend and a collectionTable which can be seen in figure 4.2. It is an extension of JInternalFrame and thus able to be pushed to `desktop` declared in God. All the window operations, such as maximizing, minimizing, translating etc. are provided to JInternalFrame and thus did not need further code - except the listening to and handling of *InternalFrameEvents* to revalidate navigator's tables and overview in case of resizing, closing or activating a viewFrame.

ViewFrame is the anchor for the model to inform about updates. class ViewFrame implements the *Observer* interface and thus is able to be registered within an *Observable* class, which is inherited by Model. Every viewFrame registers itself in the model during execution of constructor.

There is a member class QueryDialog within ViewFrame which is just a simple dialog to get the `queryString` entered by the user. The contents of `queryString` is forwarded to model's `performQuery(..)` method which returns an iterator containing resulting objects.

All the other classes are documented in detail in the following sections.



algorithm – basic idea:

1. push all collections to *uninserted* list
2. loop over *uninserted* until it is empty, do for c_i:

   If all parents of collection c_i are inserted {
   insert nodes $n(c_i)$ as children of all nodes from parents*
   }
   else { continue; }

*auxiliary temporarily datastructure neccessary:
Map [inserted collection c_x ➔ set of all nodes $n(c_x)$ representing c_x]

Figure 4.4: construction of jTree

4.1 Navigator

The navigator is the container of the two tables for the existing selections and mapImages as well as of the jTree which shows the collections and associations in the open model. It furthermore contains an overView, which shows the whole Minimung Bounding Region (MBR) of the model in a very zoomed out manner.

To build up the jTree there exists a method `buildTree()` within which `insertIntoTree(..)` is called twice, once for the collections and once for the associations within the model. Because every collection as well as every association might have multiple parents, it is not trivial to map these structurs into a tree. The chosen solution inserts nodes which have multiple parents as children to all the nodes representing parents. Thus every collection and association might have multiple representations within the jTree. Figure 4.4 illustrates the basic idea of the algorithm.

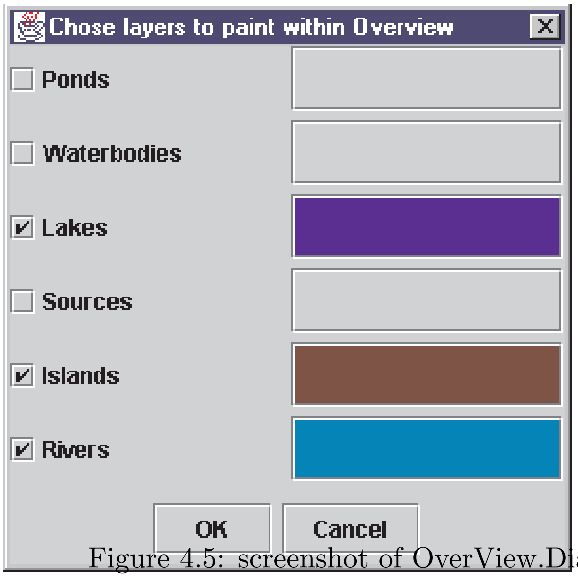


Figure 4.5: screenshot of OverView.Dialog

4.2 Overview

The Overview class is an extended JPanel which overwrites the paintComponent(..) method to draw the contents of chosen layers within the whole model area. Also the positions of all views are painted there.

The `visibleLayers` field holds references to the displayed layers. This design was chosen because it is a time-expensive operation to draw whole contents of layers - specially if they contain huge amount of gObjects. So the decision is up to the user, which layers give a good overview but in the same time do not contain too many gObjects (for performance reason).

This user interaction is provided by the member class *Dialog* which is allowed to modfify Overview's `visibleLayers` field. The OverView.Dialog does layout itself automatically due to available layers within the model. Figure 4.5 shows the Dialog.

4.3 View

The view class is inherited from gStructure.AbstractView where offset and scale related to model are defined. Coordinate system and relation to model's coordinate system is already described in the section *Model & View*.

View is connected via viewFrame.god.model which ensures the central managed model(s)[2] are used. A method - as in Control - `model()` could simplify code (encapsulting the reference-chain).

Further important references - taken into fields directly - are those of table 4.3.

type	*fieldname*
`Control`	`control`
`Selection`	`selection`
`MapImage`	`mapImage`

Table 4.3: important fields within view class

Where `control` is the handler of user interactions, described below, `selection` lists the gObjects to be highlighted and `mapImage` points to any bitmap which shall be painted as a background, e.g. a satellite image visualizing landscape's topography.

Most of View class' code is within its `paintComponent(..)` method to redraw the view anytime a window was modified or user interactions occur. The principle of the painting procedure is illustrated in figure 4.6

[2] In the current implementation there can only be one model opended at the same time.

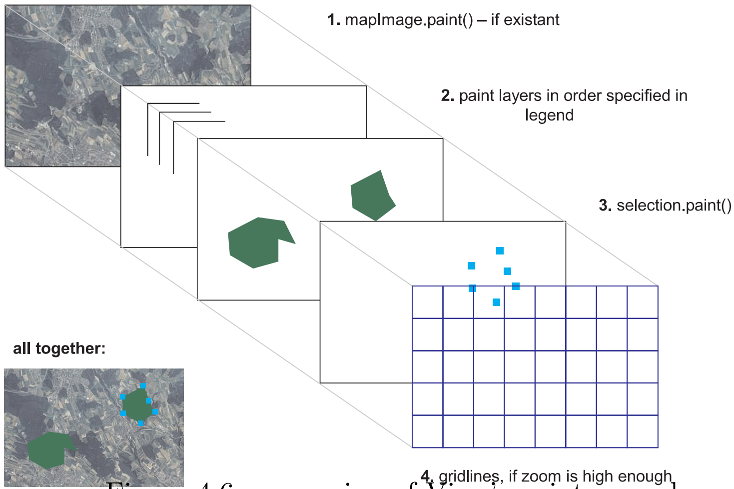


Figure 4.6: succession of View's paint procedure

4.4 Selection

The Selection class is basically a GSet, contained gObjects are highlighted in those viewers, which refer the selection. The fundamental question was whether the selection should belong to the model and thus being the same within all layers or whether it shall be part of every single view. We chose another more flexible solution where the user is free to decide how many selections exist and within which views those are visualized. This is achieved by encapsulating a selection to this separate class and having a `selection` attribute within the View class. Refer to figure 4.7 to see these connections illustrated.

There are, for example, the following situations in which the user might prefer different selection among different views: While comparing different groups of gObjects in terms of a certain (colorized) attributes, or while editing[3] at different map locations at the "same" time, i.e. simultanously

Sometimes users want to have the same gObject-groups visualized by different attributes at the same time which is an example situation where it is neccessary to have the same selection in multiple views.

Worth mentioning are the following methods (the others can be found in the API Reference appendix):

1. Vertex edgeIn(Rectangle tolerance)
2. void paint(Graphics2D g2, View v)

[3]editing always is performed on selected gObjects

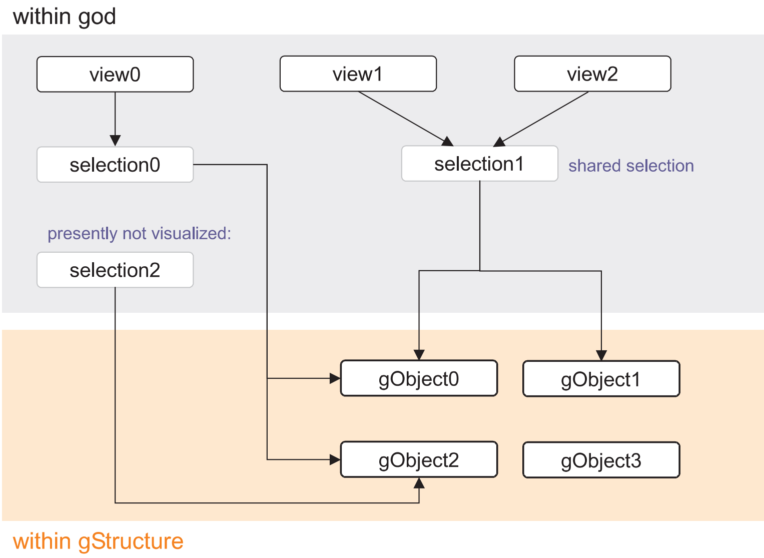


Figure 4.7: structure of views and their selections

where the first is to ask whether there is an edge of a vertexShape intersecting the specified (mouse) tolerance region. This is especially asked from a control which decides to insert a new vertex if the mouse was dragged on an edge. The second is due to figure 4.6 to draw all the selection markers within the view. All the other methods are described within the API Reference appendix.

Member classes

Besides TableModel, Selection has two more member classes: MarkerSet and the extension SinglePositionMarkerSet. A Marker has the meaning of a dot displayed at every gPoint and vertex the selection contains. Thus MarkerSet is a helper class to summarize such markers. A markerSet is needed as argument in Control's `getMagnet(..)` method to specify which markers are not possible as magnet. While translating gObjects, all the markers of selected gObjects are given to `getMagnet(..)` because those gObjects should latch at every other marker but not with itself. Method `selMarkers()` therefore returns a markerSet containing *all* markers of the selection. For a better understanding, see figure 4.8, particularly look at the markers denoted by *possible magnets.*. - all the other ones are those contained in markerSet.

While dragging just vertices - a single vertex or vertices of multiple selected gObjects at the same position - just the markers representing those vertices are given to `getMagnet(..)` as not valid. In this case a singlePositionMarkerSet is used, because while adding markers to it, they are checked to share exactly the same position as those which are contained already. This is important. Otherwise vertices which lie within tolerance region but do not share the same position exactly would dragged together - this would confuse the user. Method `markersIn(..)` is used to get such a singlePositionMarkerSet of the selection. For a better understanding, see figure 4.9, regard the markers denoted by *possible magnets..* - all the other ones are contained in singlePositionMarkerSet.

Furthermore there is a method `moveTo(Point p)` implemeted in SinglePositionMarkerSet to move all the dragged vertices using one line of code. Also a a method `release()` can be found, which is invoked to remove all the vertices which came to the position of one of its neighbours and the dragMarkerSet's storage is flushed there too.

4.5 Control

Control has all the public handler methods called by Java's event-manager after user-interactions. One of those is

```
public void actionPerformed(ActionEvent e)
```

where the events of the popup menu - to cut, copy and paste - are handled. Therefore the `clipboard` field within God class is used. Latter is of a library-type *Clipboard* where instances of the *Transferable*-implementing Selection class can be pushed into and taken from. Refer to [11] to read more about this paradigm.

Within `public void mouseClicked(MouseEvent e)` the view's selection is modified in case of a single click. A new default gObject - of the type from toppest visible layer within legend - is inserted in case of a double-click.

The drag engine

The following 4 methods form together the engine to handle the various drag actions:

1. `public void mouseMoved(MouseEvent e)`
2. `public void mousePressed(MouseEvent e)`
3. `public void mouseDragged(MouseEvent e)`
4. `public void mouseReleased(MouseEvent e)`

Furthermore, there are several attributes serving as variables to the *drag engine* which are listed in table 4.4

Within (1) the mouse cursor is set to different images depending whether it is *on vertex*, *on selection* or *else*. This indicates to the user, what kind of a drag is going to carry out.

Method (2) determines the dragMode and sets initial dragVariables. Table 4.5 shows the initial actions to be taken depending on mouse condition. Notice the toppest possible row is executed and all lower rows are ommitted.

The heart of *drag engine* is embedded in (3). According to `dragMode` the related drag actions are performed:

dragging variable field	*purpose*
`int dragMode`	the mode of *drag engine*
`int startXv, startYv`	position at beginning of drag action
`int oldXv, oldYv`	” at time of previous `mouseDragged` execution
`int currXv, currYv`	” at time of present `mouseDragged` execution
`singlePosMarkerSet`	markers involved in vertex moving
`selMarkerSet`	all markers of the selection
`Point anchorM`	model position of marker nearest to mouse
`Point mouseAnchorM`	model pos. of mouse
`boolean movedAcrossViews`	was the selection dragged to another view?
`targetView`	and which view was the target?

Table 4.4: variables for *drag engine*

mouse condition	*action*
left button	
on selection marker	set *move-vertex* mode
on area’s edge ∪ ALT + on line’s edge	insert vertex v; add v to empty singlePosMarkerSet; set *move-vertex* mode
on selected element	init anchors; set selMarkerSet; set *move-selection* mode
on unselected but visible element(s)	select them; refresh because of old selection; init anchors; set *move-selection* mode
else	set *new-selection* mode
middle button	init anchors; set *move-content* mode
right button	set *change-zoom* mode

Table 4.5: initialising `dragMode`

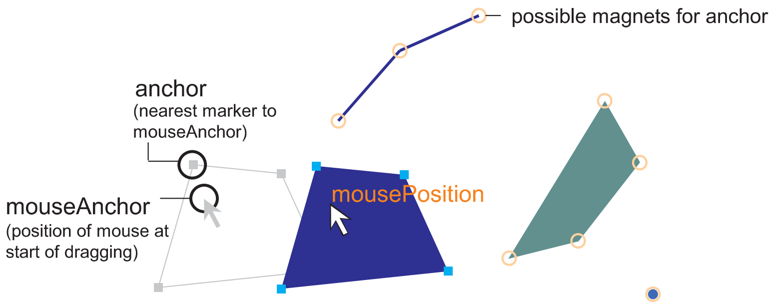
mode is move-selection :
actions due to figure 4.8;
repaint only neccessary area;

mode is move-content :
view.x0 += mouseAnchorM.x - mousePosM.x;
view.y0 += mouseAnchorM.y - mousePosM.y;
v.repaint();

mode is change-zoom :
store view-middle;
$zoom = zoom * const^{currYv-oldYv}$;
restore view-middle;
repaint();

mode is move-vertex :
actions due to figure 4.9;
repaint only neccessary area;

Finally, (4) selects elements in case of *new-selection* mode. It checks for vertices being moved to its neighbours and are thus neccessary to be removed in case of *move-vertex* mode. If mode was *move-selection* or *move-vertex*, the geographical constraints need to be checked. In any case, mode is set to *idle*, and the model is told to notify all views about updates and the navigator is told to refresh the selection table.

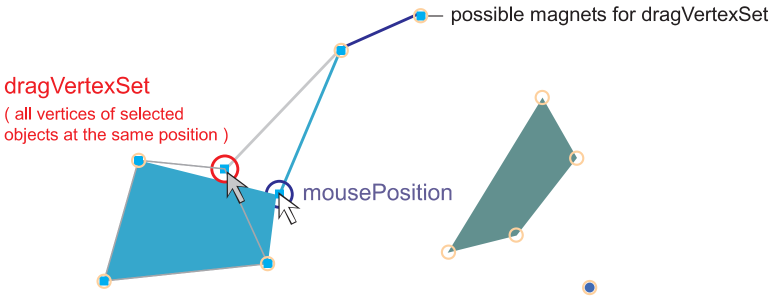


With every mouse-drag-event:

1. Translation* by (mousePosition – mouseAnchor)
2. If anchor is in sphere of a magnet m: {
   translation by (m – anchor)
   }

*means: translating all **selected objects** as well as **anchor** and **mouseAnchor**
(by difference of vectors (v1 – v0))

Figure 4.8: algortihm for magnetically moving objects



With every mouse-drag-event:

```
If mousePosition is in sphere of a magnet m {
        move dragVertexSet to m
}
else {
        move dragVertexSet to mousePosition
}
```

Figure 4.9: algortihm for magnetically moving vertices

4.6 Legend

The Legend class itself is just a slightly extended JScrollPane containing objects of the more code-intensive class LegendItem.

While constructing a legend, legendItems - one per layer - are inserted. They get a generated default color. There is furthermore a method `rebuild()` which must be invoked after having changed the order of the contained legenItems. The order of those - stored in the ArrayList attribute `items` - specifies the order of painting the corresponding layers, see figure 4.6.

LegendItem class contains in the present implementation among other the important fields listed in table 4.6

type	*fieldname*	*description*
`Legend`	`legend`	container
`GSet`	`layer`	layer, represented by the item
`boolean`	`visible`	shall the `layer`'s gObjects be painted or not?
`boolean`	`shrinked`	mode of graphical representation of the item
`Color`	`uniqueColor`	Color, which *all* the gObjects in the `layer` shall be painted with - `null`, if colorizing shall be performed according to a member
`Object`	`colorizingMember`	if `uniqueColor` is null, this field specifies the member which is used for colorizing the `layer`'s gObjects.

Table 4.6: important fields within LegendItem class

The `visible`, `uniqueColor` and `colorizingMember` fields are regarded while processing the paint procedure of the view. In combination with colorizingMember there are some more attributes declared in the LegendItem class to specify start- and end-Color and the according values. [4]

The method `mouseDragged(..)` is used to handle user's dragging interactions to change the order of the items within the legend. Whereas the `mouseClicked(..)` method handles toggling of the visibility and the shrinking button (see figure 4.1) and as well to bring the so-called LegendItemDialog to the screen in case of a double-click.

[4] see also section future work to read about mapping member values to graphical characteristics

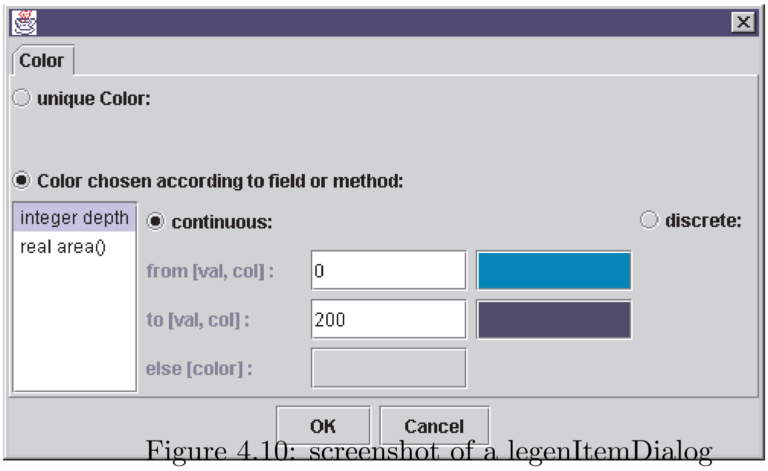


Figure 4.10: screenshot of a legenItemDialog

LegendItemDialog (figure 4.10) choses graphical representations of gObjects contained in `layer`. Its outermost component is a JTaggedPane, open for mapping values to other graphical characteristics than color, such as label and size of stroke. Thus, the `colorPane` to chose unique color or color according to member values, is put into the taggedPane as one among other possible panes. There is an `item` attribute to have access to the fields `uniqueColor`, `colorizingMember` etc. of the LegendItem, which are set after the user clicked the OK button.

type	*fieldname*	*description*
`String`	`title`	Name
`boolean`	`visible`	Should it be painted?
`String`	`imageFilename`	File in which the image has its persistent ressource
`transient Image`	`img`	Ressource of MapImage during runtime
`int`	`x0, y0`	Position of upper left corner in model
`int`	`x1, y1`	Position of lower right corner in model

Table 4.7: fields within MapImage class

4.7 Map images

Map images are bitmaps which can be used as background within views to increase orientation and visualizing quality. MapImage can be understood best by referring to table 4.7 and to figure 4.11.

Similar to the AbstractView class, there are operations available within MapImage to transform points from Model to MapImage coordinate system and vice versa. The following formula are used for this:

$$x_m(xi) = x0 + \left\lfloor \frac{xi+\frac{1}{2}}{w/(x1-x0+1)} \right\rfloor$$
$$y_m(yv) = y0 + \left\lfloor \frac{yi+\frac{1}{2}}{h/(y1-y0+1)} \right\rfloor$$

$$x_i(xm) = \left\lfloor (xm - x0 + \tfrac{1}{2}) * (w/(x1 - x0 + 1)) \right\rfloor$$
$$y_i(ym) = \left\lfloor (ym - y0 + \tfrac{1}{2}) * (h/(y1 - y0 + 1)) \right\rfloor$$

where w is the width and h the height of the mapImage.

Within its `paint(..)` method, a mapImage needs to transform the corners of the view to image coordinates to know which part of the image it shall paint:

$$xi0 = \min(w, \max(0, xi(vx0)))$$
$$yi0 = \min(h, \max(0, yi(vy0)))$$

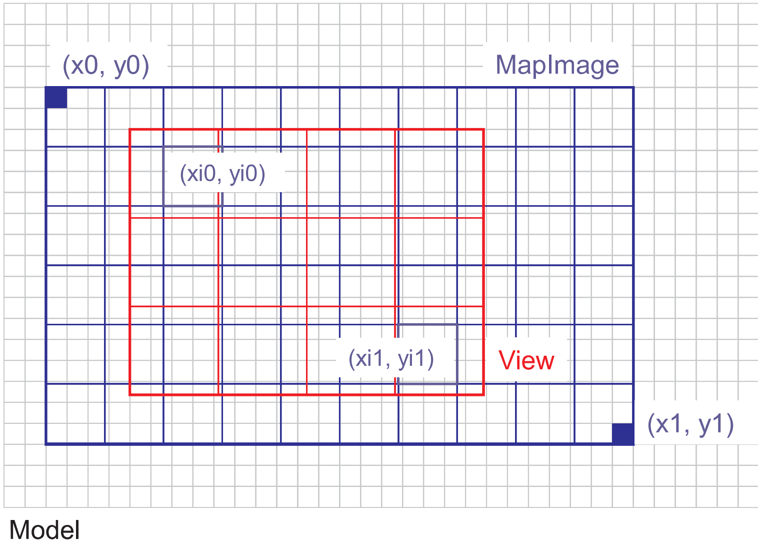


Figure 4.11: relations between MapImage, Model and View

$$xi1 = \min(w, \max(0, xi(vx1)))$$
$$yi1 = \min(h, \max(0, yi(vy1)))$$

where (vx0, vy0) and (vx1, vy1) are the antecedently calculated coordinates of the view in model-coordinates. The min(w,..) and max(0,..) functions are neccessary to ensure no painting occurs beyond the image's margin.

The special case - view is just overlapping the mapImage - needs some additional code to calculate the view coordinates[5] where the image's part is to be displayed. This is documented within the code.

[5]normally just (0, 0) and (w_{view}, h_{view})

Chapter 5

Conclusions

One main goal of this diploma thesis was to extend the persistent Object Management System (OMS Java) with basic geographical types for pointshaped, linear and areal entities. Furthermore it is possible to specify geographical constraints on them such as being geographically disjoint or containing.

Another goal was to implement a visual application to view and edit data stored in geographical information systems modelled in GOMS using Java environment. We were surprised about how fast graphic operations can be carried out although the application is evaluated by Java's virtual machine. The visual application covers the following functionality:

- View of data in a 2 dimensional area like on a map - with different levels of zoom
- Visualisation of user interests such as hiding certain collections of objects or assigning color according to value of a certain attribute
- Editing of data in an intuitive way, e.g. insertion of a new object shall be possible by clicking on the location it should be placed and changements of its form or location by dragging with the mouse.
- Interface to enter textual queries and mapping of spatial results to selection, highlighted in view.
- Integration of bitmap images, such as satellite pictures, as background to increase visualization of topographics or corellation to other data represented within the bitmap

Figure 4.1 shows a screenshot of this application, called *Geographical Object Desktop*.

So far, the current implementation is a single user system. We thought about directly realizing the architecture providing multi-user facility as shown in figure 2.11, but to design and implement such a server turned out to be too much work for diploma thesis.

5.1 Future Work

As mentioned, this system is a prototype. Thus, there is still a lot to implement before being useful for practical work. At least the following points need to be covered:

- Integration of *undo* / *redo* operations.
- Rebalancing operation on existing OpenQuadTree. At the moment, elements are expected to be inserted randomly (in terms of location) - and if this condition is not established, the tree might degenerate. Also if they are taken out of the index, latter should be rebalanced from time to time.
- Classes Collection[1] and specially Association are incomplete for practical use.
- Implementation of association insertion strategies described in the section *Geographical constraints* as well as a user interface to insert pairs of objects manually.
- In class AbstractView there exist a static `VERTICAL_INV` field, denoting that model coordinates should grow from buttom to top (usual on maps), which is the opposite to screen coordinate system. This field is not yet taken into account for coordinate transformation methods.
- Saving of environment (layers and colors in Overview, selections, legends etc.)
- Review of *protected* modifiers, we set them to protected by default. So existing modifiers other the *protected* are probably justified.
- Exception Handling
- Testing phase with concrete tasks to check stability of the system and to make perhaps refinements in the geometric specification of spatial constraints.
- Implementation of missing geometric operations, especially *gOverlaps* and *gCrosses* as well as of missing constraints.

[1]existing in the current implementation under its old name: *CollectionWrapper*

- Map Labelling

And it would improve GOMS to include the following features:

- Printing facility of view's and collectionTable's contents
- Optimization of geometric algorithms using *sweep line* principle.
- LegendItem.Dialog could have further panes to specify values of multiple members to be mapped to different graphical characteristics, such as label, size and color of stroke, icons to be used as patterns and points, bar plots and histogram integrated within/beside gObjects.

5.2 Acknowledgements

I thank Prof. Moira C. Norrie and Beat Signer for giving the opportunity to do a diploma project on a very interesting, they gave me *no* restrictions in whatever I was interested to investigate, and spent much time for coaching me. Special thank goes to Adrian Kobler who voluntarily entered the project as a cooperator - he made all neccessary modifications to OMS Java's DDL-Parser packaged to *gisExtensions* - and found time for answering questions and reviewing this report as well. I further acknowledge C. Parent, S. Spaccapietra, E.Zimányi, P. Donini, C. Plazanet and C. Vangenot from EPFL who inspired me by their papers [1] and [2]. Last but not least I thank R. Lamprecht who had the idea of the *smooth dragging zoom* user interface, to read more about his work, see [20].

Appendix A

Package gStructure

A.1 gStructure.AbstractView

java.lang.Object

  java.awt.Component

    java.awt.Container

      javax.swing.JComponent

        javax.swing.JPanel

---

public abstract *AbstractView* extends JPanel

Base class for model visualisation. Is declared abstract to avoid instantiating but actually no methods are abstract

Field Summary

Type	*Description*
protected static boolean	**VERTICAL_INV** : to indicate model coordinates should grow from buttom to top, which is the opposite to screen coordinate system
public int	**x0** : Position in model (offset)
public int	**y0** : Position in model (offset)
public double	**zoom** : proportion of view pixels per model pixels - thus 1 means zoomed in

Constructor Summary

Description
AbstractView() +

Method Summary

Returns	*Description*
public final double	**distM(double distV)** view to model transformation for stretch in view-system *Parameters:* **distV :** distance in view-system *Returns:* distance in model-system
public final double	**distM(int distV)** view to model transformation for stretch in view-system *Parameters:* **distV :** distance in view-system *Returns:* distance in model-system
public final double	**distV(double distM)** model to view transformation for stretch in model-system *Parameters:* **distM :** distance in model-system *Returns:* distance in view-system
public final double	**distV(int distM)** model to view transformation for stretch in model-system *Parameters:* **distM :** distance in model-system *Returns:* distance in view-system
public final Point	**pointM(Point pv)** view to model transformation for point in view-coordinates *Parameters:* **pv :** point in view-coordinates *Returns:* point in model-coordinates
public final Point	**pointV(Point pm)** model to view transformation for point in model-coordinates *Parameters:* **pm :** point in model-coordinates *Returns:* point in view-coordinates
public final Polygon	**polygonM(Polygon pv)** view to model transformation for polygon in view-coordinates. *Parameters:* **pv :** polygon in view-coordinates *Returns:* polygon in model-coordinates
public final Polygon	**polygonV(Polygon pm)** model to view transformation for polygon in model-coordinates. *Parameters:* **pm :** polygon in model-coordinates *Returns:* polygon in view-coordinates

public final Rectangle	**rectM(Rectangle rv)** view to model transformation for rectangle in view-coordinates. *Parameters:* **rv :** rectangle in view-coordinates *Returns:* rectangle in model-coordinates
public final Rectangle	**rectV(Rectangle rm)** model to view transformation for rectangle in model-coordinates. *Parameters:* **rm :** rectangle in model-coordinates *Returns:* rectangle in view-coordinates
public final int	**xm(int xv)** view to model transformation for x-view-coordinate *Parameters:* **xv :** x-view-coordinate *Returns:* x-model-coordinate
public final int	**xv(int xm)** model to view transformation for x-model-coordinate *Returns:* x-view-coordinate
public final int	**ym(int yv)** view to model transformation for y-view-coordinate *Parameters:* **yv :** y-view-coordinate *Returns:* y-model-coordinate
public final int	**yv(int ym)** model to view transformation for y-model-coordinate *Returns:* y-view-coordinate

A.2 gStructure.Association

java.lang.Object
gStructure.TreeInsertable

public *Association* extends TreeInsertable

Field Summary

Type	*Description*
OMCollection	**c**
CollectionWrapper	**sColl**
CollectionWrapper	**tColl**

Constructor Summary

Description
Association(OMCollection c) + Does nothing but assigning the c attribute

Method Summary

Returns	*Description*
public String	**getName()** *Returns:* name as it is specified in data model
public String	**toString()** To produce nice string representation of collection for tree nodes

A.3 gStructure.CollectionTable

java.lang.Object
 java.awt.Component
  java.awt.Container
   javax.swing.JComponent
    javax.swing.JTable

---

public *CollectionTable* extends JTable

Table to show objects with its members in a certain CollectionWrapper. CollectionTable. Model is basis for data extraction and inserting.

See Also:

- CollectionWrapper

Field Summary

Type	*Description*
public CollectionWrapper	**coll** : Collection to tabulate
protected ArrayList	**typeMemberNames** : Names of typeMembers
protected ArrayList	**typeMembers** : Attributes and methods (without parameters) of collections membertype

Constructor Summary

Description
CollectionTable(CollectionWrapper coll) + Constructs a new CollectionTable.

Method Summary

Returns	*Description*
public void	**selectAccordingTo(GSet gSet)** If coll has geographical membertype, all objects contained in gSet will be selected in this table *Parameters:* **gSet :** a set of gObjects

A.4 gStructure.CollectionTable.Model

java.lang.Object
  javax.swing.table.AbstractTableModel

---

public *CollectionTable.Model* extends AbstractTableModel

Base for CollectionTable. Provides methods to get and set values of table cells as well as title for headings and number of rows and columns

Constructor Summary

Description
CollectionTable.Model(CollectionTable coll) +

Method Summary

Returns	*Description*
public Class	**getColumnClass(int col)** Cell-Renderer and -Editor need to know type of values for certain column.
public int	**getColumnCount()** *Returns:* number of columns
public String	**getColumnName(int col)** *Parameters:* **col :** number of column *Returns:* title of column
public int	**getRowCount()** *Returns:* number of rows
public Object	**getValueAt(int row, int col)** Extracts the value for a certain cell by accessing the col-th member in row-th object of collection *Returns:* value object - is Integer, etc. for values of primitive type
public boolean	**isCellEditable(int row, int col)** *Returns:* true if member in col is an attribute, false for method
public void	**setValueAt(Object value, int row, int col)** Fills the value-object to the representing member (col) in the according object (row) in the collection.

A.5 gStructure.CollectionWrapper

java.lang.Object
gStructure.TreeInsertable

public *CollectionWrapper* extends TreeInsertable

Field Summary

Type	*Description*
OMCollection	**c** : Actual container which is wrapped
GSet	**gIndex** : If used in spatial context, this is the accelerating index.

Constructor Summary

Description
CollectionWrapper(OMCollection c) +   Does nothing but assigning the c attribute

Method Summary

Returns	*Description*
public void	**add(Object o)** Insert o into this collection.
public Object	**get(int pos)** *Parameters:* **pos :** position in collection of desired object *Returns:* the desired object
public String	**getName()** *Returns:* name as it is specified in data model
public Object	**getType()** *Returns:* type which all contained objects must be of
public Iterator	**iterator()** *Returns:* Iterator to walk over all contained objects

public void	**listTypeMembers(AbstractList members, AbstractList membernames, boolean includeTypename, boolean numeric, boolean bool, boolean string, boolean objectValue)** Fills members and their names into the argument lists. *Parameters:* **members :** prepared list from caller to receive the members **membernames :** prepared list from caller to receive the member's names as String objects (index coresponds to index in members) **includeTypename :** in case of true, there will be a prefix ('int ' etc.) before membername **numeric :** include numeric members (int, float..) **bool :** **string :** include string members **objectValue :**
public void	**remove(Object o)** Remove o from this collection and from gIndex - if esistant.
void	**setGIndex(GSet gIndex)**
public int	**size()** The number of elements within this collection
public String	**toString()** To produce nice string representation of collection for tree nodes

A.6 gStructure.Div

java.lang.Object

public *Div* extends Object

Constructor Summary

Description
Div() +

Method Summary

Returns	*Description*
public static-Point2D.Double	**pointOfIntersection(Double la, Double lb)** Calculates the point where two Line segments intersect. *Returns:* null, if they do not intersect or they overlap. the point of intersection otherwise.
public static boolean	**wholeNumbered(double x)** *Returns:* true, if x is of the form '+++.0'

A.7 gStructure.GArea

java.lang.Object
  org.omsjava.OMSBaseObject
    org.omsjava.core.OMSInstance
      gStructure.GObject
        gStructure.VertexShape

---

public *GArea* extends VertexShape

Field Summary

Type	*Description*
protected static int	**DEFAULT_N**
protected static int[]	**DEFAULT_X** : Default values for geometry of new gAreas
protected static int[]	**DEFAULT_Y**
protected Polygon	**p** : wrapped java.awt.Polygon to profit from built-in algorithms

Constructor Summary

Description
GArea() +   Constructs a new gArea with default geometry Note: init() is called here
GArea(int[] xpoints, int[] ypoints, int npoints) +   Constructs a new gArea with given geometry Note: init() is called here

Method Summary

Returns	*Description*
public double	**area()** To calculate the area of this GArea *Returns:* the area of the wrapped polygon
public boolean	**gContains(GObject o)** Tests if this GArea contains another GObject o (refer to report for specification of contains) *Parameters:* **o :** another GObject to test containing it *Returns:* true if this contains o, false otherwise
public int	**getNedges()** *Returns:* number of edges (same as nof vertices in case of gArea)
public int	**getNpoints()** *Returns:* number of vertices

public boolean	**gTouches(GObject o)** Tests if this GArea touches another GObject o (refer to report for specification of touches) *Parameters:* **o :** another GObject to test touching constraint with return true if they touch, false otherwise
public void	**init()** To initialise transient attributes as well as init of shared references between this GArea and the wrapped java.awt.Polygon
public boolean	**intersects(Rectangle r)** Tests if this GArea intersects a given Rectangle *Parameters:* **r :** Rectangle to check if this GArea intersects with *Returns:* true if this GArea intersects r, false otherwise
public boolean	**isSimple()** *Returns:* true, if gArea is free of selfintersections (touching is not allowed either) - false otherwise.
public void	**paint(Graphics2D g2, AbstractView v)** Paints this GArea on any viewing frame that extends AbstractView. *Parameters:* **g2 :** the Graphics context given from System to AbstractView.paintComponent(Graphics g) **v :** the AbstractView to paint within
void	**setNpoints(int npoints)** To set the number of vertices. *Parameters:* **npoints :** number of vertices this GArea shall have
public String	**toString()** Human readable text for this GArea
void	**translateConcrete(int dx, int dy)** Will be invoked from within translate(dx, dy), which performs general part for index consistency. *Parameters:* **dx :** translation in x direction **dy :** translation in y direction
protected int	**triangleArea2(int ax, int ay, int bx, int by, int cx, int cy)** Calculates the signed area of the triangle (multiplied by 2, because of nice integer arithmetics) This method is used to calculate polygon area. *Returns:* twice the area, positive if CCW from a to b to c

A.8 gStructure.GHashSet

java.lang.Object
    gStructure.GSet

---

public *GHashSet* extends GSet

Field Summary

Type	*Description*
private HashSet	**h** : wrapped HashSet as container for the inserted gObjects

Constructor Summary

Description
GHashSet() + Default constructor, instantiates a new GHashSet without registering and with no collection either
GHashSet(boolean registered) + Constructs a registered GHashSet, but without being gIndex of a collection
GHashSet(CollectionWrapper coll) + Constructs a GHashSet, which is gIndex of a collection, and is registered.

Method Summary

Returns	*Description*
public void	**add(GObject o)** Insert o, update bounds and if registered, register this GHashSet in o
public void	**clear()** Removes all, if registered this GHashSet will be unregistered in all gObjects that were contained.
public boolean	**contains(GObject o)** *Returns:* true, if o within this GHashSet - false otherwise
public boolean	**isEmpty()** *Returns:* true, if no gObject within this GHashSet - false otherwise
public GSet.GIterator	**iterator()** To walk over all contained gObjects.
public GSet	**range(Rectangle r, boolean enclosing)** Range Query: returns all the objects contained in r. *Parameters:* **r :** range, where objects shall be returned. **enclosing :** if true, only gObjects count, that are fully within r, otherwise also those which just overlap into r count.
public void	**remove(GObject o)** Remove o, update bounds, if registered, unregister this GHashSet in o
public int	**size()** *Returns:* amount of gObjects contained within this GHashSet

A.9 gStructure.GLine

java.lang.Object
  org.omsjava.OMSBaseObject
    org.omsjava.core.OMSInstance
      gStructure.GObject
        gStructure.VertexShape

---

public *GLine* extends VertexShape

Field Summary

Type	*Description*
protected static int	**DEFAULT_N**
protected static int[]	**DEFAULT_X** : Default values for geometry of new gAreas
protected static int[]	**DEFAULT_Y**
public int	**npoints** : The total number of vertices.

Constructor Summary

Description
GLine() +   Constructs a GLine with default geometry.
GLine(int[] xpoints, int[] ypoints, int npoints) +   Constructs a GLine with given geometry.

Method Summary

Returns	*Description*
public boolean	**gContains(GObject o)** Tests if this GLine contains another GObject o (refer to report for specification of contains) *Parameters:* **o :** another GObject to test containing it *Returns:* true if this contains o, false otherwise
public int	**getNedges()** *Returns:* number of edges
public int	**getNpoints()** *Returns:* number of vertices

public boolean	**gTouches(GObject o)** Tests if this GObject touches another GObject o (refer to report for specification of touches) *Parameters:* **o :** another GObject to test touching constraint with return true if they touch, false otherwise
public boolean	**hasEnding(double x, double y)** *Returns:* true, if this GLine has first or last vertex at (x, y)
public void	**init()** To initialise transient attributes
public boolean	**intersects(Rectangle r)** *Returns:* true, if any point of this GLine lies within Rectangle r. (or on an edge / corner of r) - false otherwise.
public boolean	**isCycle()** *Returns:* true, if this GLine has first and last vertex at same position
public boolean	**isOpen()** *Returns:* true, if this GLine is free of self-intersections
public double	**length()** *Returns:* length of this GLine - the sum of the segments lengths
public void	**paint(Graphics2D g2, AbstractView v)** Paints this GLine on any viewing frame that extends AbstractView. *Parameters:* **g2 :** the Graphics context given from System to AbstractView.paintComponent(Graphics g) **v :** the AbstractView to paint within
void	**setNpoints(int npoints)** To set the number of vertices. *Parameters:* **npoints :** number of vertices this GArea shall have
public String	**toString()** Human readable text for this GLine
void	**translateConcrete(int dx, int dy)** Will be invoked from within translate(dx, dy), which performs general part for index consistency. *Parameters:* **dx :** translation in x direction **dy :** translation in y direction

A.10 gStructure.GObject

java.lang.Object
  org.omsjava.OMSBaseObject
    org.omsjava.core.OMSInstance

---

public abstract *GObject* extends OMSInstance

Field Summary

Type	*Description*
protected transient-Rectangle	**bounds** : Redundant minimum boundary region, for performance reason
public String	**caption** : For map labbeling
protected transient-HashSet	**registeredSets** : Set of registered containers (GSet), need to be informed of changes to validate their indexstructure (can be dependet from object location and shape)
protected transient-boolean	**valid** : Redundant, for performance reason - indicates whether constraints are satisfied for this GObject

Constructor Summary

Description
GObject() +   Default constructor, does nothing but overwrite public modifier by package access.

Method Summary

Returns	*Description*
final void	**addInSets(GSet[] sets)** Needs to be invoked after operations that performed spatial changes on the object - in combination with removeFromAllRegisteredSets before.
public void	**checkObjectConstraints()** Invoked (usually after object modification) to check whether it satisfies object- and collection-constraints.
public final GObject	**copy()** Performs copy-operation including some neccessary initialising for registered layers and wrapped objects. *Returns:* a GObject of the same dynamic type like the original

abstract void	**copyConcrete(GObject copy)** Needs to be overwritten due to specific operations for the concrete GObjectSubclass. *Parameters:* **copy :** by copy() newly generated copy of this GObject
public void	**delete()** Removes this GObject from all registered containers as well as from wrapped collections
public abstract boolean	**gContains(GObject o)** Tests if this GObject contains another GObject o (refer to report for specification of contains) *Parameters:* **o :** another GObject to test containing it *Returns:* true if this contains o, false otherwise
public abstract boolean	**gDisjoint(GObject o)** Tests if this GObject is geographically disjoint to another GObject o (refer to report for specification of disjoint) *Parameters:* **o :** another GObject to test geographical disjoint with *Returns:* true if this GObject has no common points with o, false otherwise
public abstract boolean	**gEquals(GObject o)** Tests if this GObject is geographically equal to another GObject o *Parameters:* **o :** another GObject to test geo-equality *Returns:* true if geographically equal to o, false otherwise
public final String	**getCaption()** getters / setters for coresponding attributes
public abstract Point	**getLocation()** To get an anchor position for this GObject *Returns:* A Point representing the anchor position of this GObject
public Object	**getMemberValue(Object member)** To gets the value of a member (attribute or method) of a concrete subclass type, which this must be instance of. *Parameters:* **member :** Object representing attribute or method
final GSet[]	**getRegisteredSetsArray()** Returns all the registered sets as an array, invoked by removeFromAllRegisteredSets and returned as backup array *Returns:* an array containing all the GSet that are registered
public abstract boolean	**gTouches(GObject o)** Tests if this GObject touches another GObject o (refer to report for specification of touches) *Parameters:* **o :** another GObject to test touching constraint with return true if they touch, false otherwise

public void	**init()** Every GObjectSubclass has its init() method, because init of transient fields cannot be placed into constructors.
public abstract boolean	**intersects(Rectangle r)** Tests if this GObject intersects a given Rectangle *Parameters:* **r :** Rectangle to check if this GObject intersects with *Returns:* true if this GObject intersects r, false otherwise
public abstract boolean	**isInside(Rectangle r)** Tests if this GObject lies within a given Rectangle *Parameters:* **r :** Rectangle to check if this GObject lies within *Returns:* true if this GObject lies within r, false otherwise
public boolean	**isValid()** To ask if this GObject established the constraints *Returns:* true if valid, false otherwise
public final Rectangle	**mbr()** Returns the minimum bounding region for this object. *Returns:* a reference to the original bounds attribute of this GObject
public abstract void	**paint(Graphics2D g2, AbstractView v)** Paints this GObject on any viewing frame that extends AbstractView. *Parameters:* **g2 :** the Graphics context given from System to AbstractView.paintComponent(Graphics g) **v :** the AbstractView to paint within
final void	**registerSet(GSet newSet)** Registers a container (GSet) which contains this GObject note: should only be called from within GSet.add(GObject), to not invalidate consistency of the bidirectional reference between [GSet, GObject]! *Parameters:* **newSet :** the GSet that contains this GObject

final GSet[]	**removeFromAllRegisteredSets()** Needs to be invoked before operations that perform spatial changes on the object - in combination with addInSets(returned backup array) afterwards. *Returns:* a backup array containing all the GSet that were registered before
public final void	**setCaption(String caption)**
public void	**setValid(boolean valid)** Is invoked by constraint checking algorithms *Parameters:* **valid :** true if this GObject establishes the constraints false otherwise
public final void	**translate(int dx, int dy)** There will "translateConcrete(dx, dy) after "bak = removeFromAllRegisteredSets()" and "addInSets(bak)" finally.
abstract void	**translateConcrete(int dx, int dy)** Needs to be overwritten due to specific operations for the concrete GObjectSubclass.
final void	**unregisterSet(GSet oldSet)** Unregisters a container (GSet) which does not contain this GObject (anymore) note: should only be called from within GSet.remove(GObject), to not invalidate consistency of the bidirectional reference between [GSet, GObject]! *Parameters:* **oldSet :** the GSet to unregister

A.11 gStructure.GOrientedLine

java.lang.Object
    org.omsjava.OMSBaseObject
        org.omsjava.core.OMSInstance
            gStructure.GObject
                gStructure.VertexShape
                    gStructure.GLine

---

public *GOrientedLine* extends GLine

Constructor Summary

Description
GOrientedLine() + Constructs a GLine with default geometry.
GOrientedLine(int[] xpoints, int[] ypoints, int npoints) + Constructs a GOrientedLine with given geometry.

Method Summary

Returns	*Description*
public boolean	**endsAt(double x, double y)** *Returns:* true, if this GLine has last vertex at (x, y)
public void	**paint(Graphics2D g2, AbstractView v)** Performs painting of GLine followed by drawing an arrow dot at end of the GOrientedLine
public boolean	**startsAt(double x, double y)** *Returns:* true, if this GLine has first vertex at (x, y)
public String	**toString()** Human readable text for this GOrientedLine

A.12 gStructure.GPoint

java.lang.Object
  org.omsjava.OMSBaseObject
    org.omsjava.core.OMSInstance
      gStructure.GObject

---

public *GPoint* extends GObject

Field Summary

Type	*Description*
private int	**x** : Coordinates of this GPoint
private int	**y** : Coordinates of this GPoint

Constructor Summary

Description
GPoint() + Constructs a new GPoint at (0, 0).
GPoint(int x, int y) + Constructs a new GPoint at (x, y).

Method Summary

Returns	*Description*
void	**copyConcrete(GObject copy)** Concrete code for copy operation invoked on GPoint. *Parameters:* **copy :** by copy() newly generated copy of this GObject
public boolean	**gContains(GObject o)** Tests if this GPoint contains another GObject o (refer to report for specification of contains) *Parameters:* **o :** another GObject to test containing it *Returns:* true if this contains o, false otherwise

public boolean	**gDisjoint(GObject o)** Tests if this GPoint is geographically disjoint to another GObject o (refer to report for specification of disjoint) *Parameters:* **o :** another GObject to test geographical disjoint with *Returns:* true if this GObject has no common points with o, false otherwise
public boolean	**gEquals(GObject o)** Tests if this GPoint is geographically equal to another GObject o *Parameters:* **o :** another GObject to test geo-equality *Returns:* true if geographically equal to o, false otherwise
public final Point	**getLocation()** *Returns:* position (x, y) of this GPoint as a java.awt.point object
public final int	**getX()** *Returns:* x coordinate
public final int	**getY()** *Returns:* x coordinate
public boolean	**gTouches(GObject o)** Tests if this GPoint touches another GObject o (refer to report for specification of touches) *Parameters:* **o :** another GObject to test touching constraint with return true if they touch, false otherwise
public void	**init()** To initialise transient attributes
public boolean	**intersects(Rectangle r)** *Returns:* true, if isInside(r) - false otherwise.
public boolean	**isInside(Rectangle r)** *Returns:* true, if extension of r contains this GPoint. Extension of r is defined as follows: same position as r, but rE.width = r.width + 1 and analogous for height. Reason: lower and right edge of rectangle shall be treated the same way as upper and left edge. See report for detailed information
public void	**paint(Graphics2D g2, AbstractView v)** Paints this GPoint on any viewing frame that extends AbstractView. *Parameters:* **g2 :** the Graphics context given from System to AbstractView.paintComponent(Graphics g) **v :** the AbstractView to paint within

public final void	**setLocation(Point p)** To set the position from outside of package, index consistency is bewared and bounds updated automatically.
public final void	**setX(Integer x)** To set the x coordinate from outside of package, index consistency is bewared and bounds updated automatically.
public final void	**setY(Integer y)** To set the y coordinate from outside of package, index consistency is bewared and bounds updated automatically.
public String	**toString()** Human readable text for this GArea
void	**translateConcrete(int dx, int dy)** Will be invoked from within translate(dx, dy), which performs general part for index consistency.

A.13 gStructure.GSet

java.lang.Object

---

public abstract *GSet* extends Object

Field Summary

Type	*Description*
protected Rectangle	**bounds** : Redundant minimum boundary region, for performance reason.
protected CollectionWrapper	**collection** : Collection which is indexed by this GSet, might be null
protected HashSet	**layerConstraints** : Set of all specified geographical layer constraints
protected HashSet	**objectConstraints** : Set of all specified geographical object constraints
protected boolean	**registered** : To indicate whether contained objects register this GSet as a container.

Constructor Summary

Description
GSet() +

Method Summary

Returns	*Description*
public abstract void	**add(GObject o)** To insert a gObject into this GSet.
public void	**addAll(GSet gSet)** To insert the content of another GSet at once.
public abstract void	**clear()** To remove all gObjects from this GSet.
public abstract boolean	**contains(GObject o)** *Returns:* true, if o is in this GSet, false otherwise.

public CollectionWrapper	**getCollection()** To get the collection this GSet is index of *Returns:* Indexed Collection - if existing. null otherwise
public Set	**getLayerConstraints()** To get the layer constraints from outside of package *Returns:* a clone, to be save against client corruption
public String	**getName()** If this GSet is just index of a wrapped collection, the name can be extracted here.
public Set	**getObjectConstraints()** To get the object constraints from outside of package *Returns:* a clone, to be save against client corruption
public Object	**getType()** To get the type of the contained GObjects *Returns:* Membertype of indexed Collection - if existing. null otherwise
public abstract boolean	**isEmpty()** *Returns:* true, if no gObjects contained at all.
public abstract-GSet.GIterator	**iterator()** To walk over all contained gObjects.
public Rectangle	**mbr()** *Returns:* a Rectangle that is the minimum bounding region of of the content
public abstract GSet	**range(Rectangle r, boolean enclosing)** Range Query: returns all the objects contained in r. *Parameters:* **r :** range, where objects shall be returned. **enclosing :** if true, only gObjects count, that are fully within r, otherwise also those which just overlap into r count.
public abstract void	**remove(GObject o)** To remove a gObject into this GSet.
public void	**removeAll(GSet gSet)** Removes all the gObjects from gSet, which are contained in this GSet
public abstract int	**size()** *Returns:* number of gObjects contained in this GSet
public GObject[]	**toArray()** Provides an array representation of this GSet. *Returns:* new GObject[size()] array, containing all gObjects in unspecified order
public void	**translate(int dx, int dy)** Translate all contained objects. *Parameters:* **dx :** translation in x direction **dy :** translation in y direction
protected void	**updateBounds()** To revalidate the bounds attribute.

A.14 gStructure.GSet.GIterator

java.lang.Object

---

public *GSet.GIterator* extends Object

Wrapper of java.util.Iterator to decorate it with inherent cast to GObject - which is the only allowed type in GSet.

Field Summary

Type	*Description*
private Iterator	**i**

Constructor Summary

Description
GSet.GIterator(GSet this\$0, Iterator i) + Can't be invoked from outside of GSet.

Method Summary

Returns	*Description*
public boolean	**hasNext()** Returns true if the iteration has more elements.
public GObject	**next()** Returns the next element in the iteration.

A.15 gStructure.InternalNode

java.lang.Object
gStructure.Node

---

final *InternalNode* extends Node

Field Summary

Type	*Description*
protected Node	**ll** : References to the four subtrees, u:upper l:lower / l:left r:right
protected Node	**lr** : References to the four subtrees, u:upper l:lower / l:left r:right
protected Point	**split** : Coordinates of the point, where the four subtrees coincide
protected Node	**ul** : References to the four subtrees, u:upper l:lower / l:left r:right
protected Node	**ur** : References to the four subtrees, u:upper l:lower / l:left r:right

Constructor Summary

Description
InternalNode(Point split) + To create a new InternalNode with subtrees meeting at given location

Method Summary

Returns	*Description*
protected Node	**add(GObject o)** Inserts a gObject to the node's subtrees
protected boolean	**contains(GObject o)** *Returns:* True, if the node's subtrees contains the gObject
protected void	**paint(Graphics2D g2, AbstractView v)** For debugging purpose, prints all the gObjects contained in subtrees at a certain view
protected void	**print(String levelTabs)** For debugging purpose, prints the subtrees at a certain tabulator level (according to depth of upper part of tree
protected void	**rangeFill(Rectangle r, boolean enclosing, GSet range)** Fills all gObjects from node's subtrees which are inside r to given range.
protected void	**remove(GObject o)** Removes a gObject from the node's subtrees

A.16 gStructure.Leaf

java.lang.Object
  gStructure.Node

---

final *Leaf* extends Node

Field Summary

Type	*Description*
protected HashSet	**content** : Anchor to contents mangager
protected static double	**CRITICAL_PART** : crititcal part of contents in a new child leaf just after having split if exceeded, this child will be set to unsplittable and thus can not produce any children itself.
protected static int	**MAX_SIZE** : Maximum gObjects that can be contained within leaves.
protected boolean	**splittable** : To specify whether this leaf could split if MAX_SIZE is exceeded

Constructor Summary

Description
Leaf(boolean splittable) + Instantiation of a new leaf

Method Summary

Returns	*Description*
protected Node	**add(GObject o)** Inserts a gObject to this leaf.
protected boolean	**contains(GObject o)** *Returns:* True, if this leaf contains the gObject
protected void	**paint(Graphics2D g2, AbstractView v)** For debugging purpose, prints all the gObjects contained in this leaf at a certain view
protected void	**print(String levelTabs)** For debugging purpose, prints the leaf contents at a certain tabulator level (according to depth of upper part of tree
protected void	**rangeFill(Rectangle r, boolean enclosing, GSet range)** Fills all gObjects from leaf which are inside r to given range.
protected void	**remove(GObject o)** Removes a gObject from this leaf

A.17 gStructure.Model

java.lang.Object
  java.util.Observable

---

public *Model* extends Observable

Field Summary

Type	*Description*
public GAssociationConstraint[]	**assocConstraints** : All the geographical association constraints contained in the data model Note: other geographical constraints are attached to layers
public Association[]	**associations** : All the associations contained in the data model
public CollectionWrapper[]	**collections** : All the collections contained in the data model
public GSet[]	**layers** : All collections in the model with geographic type
protected String	**schemaName** : Name of the data model schema within workspace
protected OMSWorkspace	**workspace** : Anchor to persistent OMS Java

Constructor Summary

Description
Model() +

Method Summary

Returns	*Description*
public void	**checkConstraints()** Performs checking of all gObjects to check fullfilling of layer- and association constraints.
public void	**checkConstraints(Rectangle areaToCheck)** Performs checking of gObjects in areaToCheck to check fullfilling of layer- and association constraints.
public void	**clear()** To flush all references
public void	**commit()** Performs storing to persistence system if all gObjects are valid (from dmp-file chosen in file selector)

public void	**create()** To initialise the model
public static Object	**createObject(Object type)** To generate a new object
protected CollectionWrapper	**getCollection(String alias)** To found a collection in the collections array by given name
public GSet	**getLayer(String alias)** To found a layer in the layers array by given name
public Point	**getMiddle()** *Returns:* The middle of the minimum bounding region mbr()
protected TreeInsertable	**getTreeInsertable(String alias)** To found a collection or association by given name in their arrays
protected void	**init()** Init of data model structure due to persistence system: Generation of arrays layers, collections and associations
protected void	**initConstraints()** Get the specified geographical constraints from workspace
public Rectangle	**mbr()**
public void	**notifyObservers()** Does inform the observers (views) about changes in the model.
public Iterator	**performQuery(String queryString)** Asks the workspace for results of given queryString.
public GSet	**range(Rectangle r, boolean enclosing)** Returns the objects contained in r, looking at all layers
public void	**rollback()** Performs reloading from persistence system (from dmp-file chosen in file selector)

A.18 gStructure.Node

java.lang.Object

---

abstract *Node* extends Object

Constructor Summary

Description
Node() +

Method Summary

Returns	*Description*
protected abstract Node	**add(GObject o)** Inserts a gObject to the node's subtree
protected abstract boolean	**contains(GObject o)** *Returns:* True, if the node's subtree contains the gObject
protected abstract void	**paint(Graphics2D g2, AbstractView v)** For debugging purpose, prints all the gObjects contained in subtree at a certain view
protected abstract void	**print(String levelTabs)** For debugging purpose, prints the subtree at a certain tabulator level (according to depth of upper part of tree
protected abstract void	**rangeFill(Rectangle r, boolean enclosed, GSet range)** Fills all gObjects from node's subtree which are inside r to given range. *Parameters:* **enclosed :** true, if objects must be inside r completely or just overlapping if false
protected abstract void	**remove(GObject o)** Removes a gObject from the node's subtree

A.19 gStructure.OpenQuadTree

java.lang.Object
gStructure.GSet

---

public *OpenQuadTree* extends GSet
implements Cloneable

Field Summary

Type	*Description*
private GHashSet	**iteratorCache** : Redundant linear structure with references of all contained gObjects.
private Node	**root** : The entrance of the datastructure
private int	**size** : Redundant field for performance reason

Constructor Summary

Description
OpenQuadTree() +   Default constructor, instantiates a new OpenQuadTree without registering and without being gIndex of a collection
OpenQuadTree(boolean registered) +   Constructs a registered OpenQuadTree, but without being gIndex of a collection
OpenQuadTree(CollectionWrapper coll) +   Constructs a OpenQuadTree, which is gIndex of a collection, and registered as well.

Method Summary

Returns	*Description*
public void	**add(GObject o)** Insert o, update bounds and if registered, register this OpenQuadTree in o
public void	**clear()** Removes all, if registered this OpenQuadTree will be unregistered in all gObjects that were contained.
public boolean	**contains(GObject o)** *Returns:* true, if o within this OpenQuadTree - false otherwise
private void	**init(CollectionWrapper coll, boolean registered)** Helper method for all the constructors

public boolean	**isEmpty()** *Returns:* true, if no gObject within this OpenQuadTree false otherwise
public GSet.GIterator	**iterator()** To walk over all contained gObjects.
public void	**paint(Graphics2D g2, AbstractView v)** Paint the tree structure grid to a view - For Debugging
public void	**print()** Prints the tree data in tabulator-structured form - For Debugging
public GSet	**range(Rectangle r, boolean enclosing)** Range Query: returns all the objects contained in r. *Parameters:* **r :** range, where objects shall be returned. **enclosing :** if true, only gObjects count, that are fully within r, otherwise also those which just overlap into r count. *Returns:* present implementation returns GHashSet
public void	**remove(GObject o)** Remove o, update bounds, if registered, unregister this OpenQuadTree in o
public void	**showInFrame()** Shows a frame, where the tree is painted within - For Debugging
public int	**size()** *Returns:* amount of gObjects contained within this OpenQuadTree

A.20 gStructure.TreeInsertable

java.lang.Object

---

public *TreeInsertable* extends Object

Field Summary

Type	*Description*
boolean	**covered** : To specify this to be fully covered by sub-container.
public Set	**parents** : To establish the sub/super structure with the other treeInsertables in the Model

Constructor Summary

Description
TreeInsertable() +

Method Summary

Returns	*Description*
public void	**addSuperColl(TreeInsertable c)** To specify parents of this container.
public boolean	**isCovered()** To ask this GSet of being fully covered by sub-container. *Returns:* true if covered, false otherwise
public void	**setCovered(boolean covered)** *Parameters:* **covered :** true to specify being covered, false otherwise

A.21 gStructure.Vertex

java.lang.Object

---

public *Vertex* extends Object

Field Summary

Type	*Description*
private VertexShape	**object** : The VertexShape, which this Verex is vertex of
private int	**pos** : The index in the ranking of vertices in the referenced VertexShape

Constructor Summary

Description
Vertex(VertexShape object, int vertexNo) +   Constructs a new vertex-object with given VertexShape and no.

Method Summary

Returns	*Description*
public void	**addNeighbour(int x, int y, boolean before)** Inserts a new Vertex next to this Vertex at the specified location. *Parameters:* **x :** the x coordinate of the new Vertex **y :** the y coordinate of the new Vertex **before :** true if new Vertex should be inserted before this one in the VertexShape's list of points, false if it should be inserted afterwards
public boolean	**equals(Vertex v)** Tests if this vertex is equal to another one (not just reference test!) *Parameters:* **v :** The other Verex to test for equality *Returns:* true if this Vertex has the same object and vertex no. as v, false otherwise

public Point	**getLocation()** Returns coordinates of this Vertex *Returns:* point representing coordinates of this Vertex
public int	**getPosition()** *Returns:* the position in its VertexShape this Vertex belongs to
public VertexShape	**getShape()** *Returns:* the VertexShape this Vertex belongs to
public int	**getX()** Returns x coordinate of this Vertex *Returns:* x coordinate
public int	**getY()** Returns y coordinate of this Vertex *Returns:* y coordinate
public boolean	**isAtNeighbour()** Checks if this Vertex has the same coordinates as one of its neighbours. *Returns:* true if this Vertex has the same coordinates than one of its two neighbours, false otherwise
public void	**moveTo(int x, int y)** Moves this Vertex to the specified location. *Parameters:* **x :** the new x coordinate **y :** the new y coordinate
public void	**moveTo(Point p)** Moves this Vertex to the specified location. *Parameters:* **p :** point specifiying the new coordinates
public void	**remove()** Removes this Vertex from the referenced VertexShape's points.

A.22 gStructure.VertexShape

java.lang.Object
  org.omsjava.OMSBaseObject
    org.omsjava.core.OMSInstance
      gStructure.GObject

---

public abstract *VertexShape* extends GObject

Field Summary

Type	*Description*
int[]	**xp** : The arrays of coordinates.
int[]	**yp** : The arrays of coordinates.

Constructor Summary

Description
VertexShape() +

Method Summary

Returns	*Description*
void	**copyConcrete(GObject copy)** Concrete code for copy operation invoked on VertexShape. *Parameters:* **copy :** by copy() newly generated copy of this GObject
public boolean	**gDisjoint(GObject o)** Tests if this GObject is geographically disjoint to another GObject o (refer to report for specification of disjoint) *Parameters:* **o :** another GObject to test geographical disjoint with *Returns:* true if this GObject has no common points with o, false otherwise
public boolean	**gEquals(GObject o)** Tests if this GObject is geographically equal to another GObject o *Parameters:* **o :** another GObject to test geo-equality *Returns:* true if geographically equal to o, false otherwise

public final Point	**getLocation()** Neccessary for operations that need anchors on objects
public abstract int	**getNedges()** *Returns:* number of edges
public abstract int	**getNpoints()** *Returns:* number of vertices
public final OMCollection	**getPoints()** Interface to OMS specific code.
public Vertex	**getVertex(int pos)** *Returns:* a new Vertex object, which is the only public interface to perform modifications on VertexShapes.
public boolean	**hasVertex(int x, int y)** *Returns:* true, if this VertexShape has a vertex at (x, y), false otherwise
public void	**init()** Pseudo constructor, in paricular for copy() and deserialising
public boolean	**isInside(Rectangle r)** *Returns:* true, if this VertexShape is fully within r (Touching does not violate 'being inside')
boolean	**isSelfIntersecting()** *Returns:* true, if any non-neighboured edges intersect (touching means intersecting as well here!)
abstract void	**setNpoints(int npoints)** To specify how many entries in the coordinates arrays are vertices
public final void	**setPoints(int[] xpoints, int[] ypoints)** To assign coordinate arrays attribute a new reference.
public final void	**setPoints(OMCollection points)** Interface to OMS specific code.
public String	**toString()** Returns (abstract) VertexShape part of a human readable textual representation for object of a concrete subclass
void	**updateBounds()** To revalidate redundant bounds attribute after modifications

Appendix B

Package gStructure.constraint

B.1 ..constraint.GAContainingConstraint

java.lang.Object
  gStructure.constraint.GAssociationConstraint

---

public *GAContainingConstraint* extends GAssociationConstraint

Constructor Summary

Description
GAContainingConstraint(GSet src, int smin, int smax, GSet tar, int tmin, int tmax) +

Method Summary

Returns	*Description*
public boolean	**checkConcrete(GObject source, GObject target)** *Returns:* True, if source contains the target gObject
public String	**toString()**

B.2 ..constraint.GAssociationConstraint

java.lang.Object

---

public abstract *GAssociationConstraint* extends Object

Field Summary

Type	*Description*
public final GSet	**srcLayer**
public final int	**srcMax**
public final int	**srcMin** : Refer to report for explanation of cardinalities
public final GSet	**tarLayer**
public final int	**tarMax**
public final int	**tarMin**

Constructor Summary

Description
GAssociationConstraint(GSet src, int smi, int sma, GSet tar, int tmi, int tma) +

Method Summary

Returns	*Description*
public void	**check(Rectangle areaToCheck)** Checks all the gObjects within areaToCheck to be valid in terms of the concrete associationConstraint, those that do not will be set to invalid.
public abstract boolean	**checkConcrete(GObject source, GObject target)** *Returns:* True, if source and target gObjects satisfy the concrete association-constraint condition

B.3 ..constraint.GATouchingConstraint

java.lang.Object
gStructure.constraint.GAssociationConstraint

public *GATouchingConstraint* extends GAssociationConstraint

Constructor Summary

Description
GATouchingConstraint(GSet src, int smin, int smax, GSet tar, int tmin, int tmax) +

Method Summary

Returns	*Description*
public boolean	**checkConcrete(GObject source, GObject target)** *Returns:* True, if source and target gObject are touching
public String	**toString()**

B.4 ..constraint.GLayerConstraint

java.lang.Object

---

public abstract *GLayerConstraint* extends Object

Field Summary

Type	*Description*
public final GSet	**layer** : The layer whose objects must fullfill the concrete layer-constraint

Constructor Summary

Description
GLayerConstraint(GSet layer) +

Method Summary

Returns	*Description*
public void	**check(Rectangle areaToCheck)** Checks all the gObjects within areaToCheck to be valid in terms of the concrete layer-constraint, those that do not will be set to invalid.
public abstract boolean	**checkConcrete(GObject source, GObject target)** *Returns:* True, if source and target gObjects satisfy the concrete layer-constraint condition

B.5 ..constraint.GLDisjointConstraint

java.lang.Object
gStructure.constraint.GLayerConstraint

---

public *GLDisjointConstraint* extends GLayerConstraint

Constructor Summary

Description
GLDisjointConstraint(GSet layer) +

Method Summary

Returns	*Description*
public boolean	**checkConcrete(GObject source, GObject target)** *Returns:* True, if source and target gObjects are geographically disjoint
public String	**toString()**

B.6 ..constraint.GLTouchingConstraint

java.lang.Object
gStructure.constraint.GLayerConstraint

---

public *GLTouchingConstraint* extends GLayerConstraint

Constructor Summary

Description
GLTouchingConstraint(GSet layer) +

Method Summary

Returns	*Description*
public boolean	**checkConcrete(GObject source, GObject target)** *Returns:* True, if source and target gObjects are geographically disjoint or touching
public String	**toString()**

B.7 ..constraint.GObjectConstraint

java.lang.Object

public abstract *GObjectConstraint* extends Object

Constructor Summary

Description
GObjectConstraint() +

Method Summary

Returns	*Description*
public abstract boolean	**check(GObject o)** *Returns:* True, if the gObject o does satisfy the concrete object-constraint

B.8 ..constraint.GOCycleLineConstraint

java.lang.Object
gStructure.constraint.GObjectConstraint

public *GOCycleLineConstraint* extends GObjectConstraint

Constructor Summary

Description
GOCycleLineConstraint() +

Method Summary

Returns	*Description*
public boolean	**check(GObject o)** *Returns:* True, if the gObject o is a line and a cycle

B.9 ..constraint.GOOpenLineConstraint

java.lang.Object
gStructure.constraint.GObjectConstraint

public *GOOpenLineConstraint* extends GObjectConstraint

Constructor Summary

Description
GOOpenLineConstraint() +

Method Summary

Returns	*Description*
public boolean	**check(GObject o)** *Returns:* True, if the gObject is a line and does not have self-intersections

Appendix C

Package god

C.1 god.Const

java.lang.Object

---

public *Const* extends Object

A container for global constants

Field Summary

Type	*Description*
public static int	**CHANGE_ZOOM**
public static int	**IDLE** : the different states for dragmode in view's control
public static int	**MAGNETIC_RAD** : radius of magnetic influence of other vertices [pixel]:
public static double	**MAX_ZOOM**
public static int	**MIN_GRID_DIST** : minimum distance between gridlines [pixel]:
public static double	**MIN_ZOOM**
public static int	**MOVE_CONTENT**
public static int	**MOVE_SELECTION**
public static int	**MOVE_VERTEX**

public static int	**NAV_WIDTH** : width of navigator pane [pixel]:
public static int	**NEW_SELECTION**
public static Color	**SEL_COLOR** : size for selection (how large is highlighting) :
public static int	**TOL** : tolerance for selecting elements with mouse [pixel]:
public static boolean	**VERTICAL_INV** : vertical inversion because of screen coordinate system (y starting at top):
public static Color	**VIS_BUTTON_COLOR** : color of visibilty LED in LegendItem
public static double	**ZOOM_CHANGE** : standard zoom in: zoom = zoom * the following const:

Constructor Summary

Description
Const() +

C.2 god.Control

java.lang.Object
  java.awt.event.MouseAdapter

---

public *Control* extends MouseAdapter
implements ActionListener,MouseMotionListener,KeyListener

This is the handler class for a view. All fields and local variables ending by *M* or *V* are for emphasising where they are related to: Model or View

Field Summary

Type	*Description*
protected Point	**anchorM** : Model position of marker nearest to mouse
protected int	**currXv** : position at time of current 'mouseDragged' execution
protected int	**currYv** : position at time of current 'mouseDragged' execution
protected int	**dragMode** : the mode of drag engine, see report for description of the drag engine
protected Point	**mouseAnchorM** : Model position of mouse
protected boolean	**movedAcrossViews** : Was the selection dragged to another view?
protected int	**oldXv** : position at time of previous 'mouseDragged' execution
protected int	**oldYv** : position at time of previous 'mouseDragged' execution
protected Selection.MarkerSet	**selMarkerSet** : All markers of the selection
protected Selection.SinglePositionMarkerSet	**singlePosMarkerSet** : Markers involved in vertex moving
protected int	**startXv** : position at beginning of drag action
protected int	**startYv** : position at beginning of drag action
protected View	**targetView** : If movedAcrossViews, which view was the target?
protected View	**v** : view which has this control to handle events

Constructor Summary

Description
Control(View v) +   Construction of a control for a given view

Method Summary

Returns	*Description*
public void	**actionPerformed(ActionEvent e)** Invoked when an action occurs, to handle popup menu (cut, copy, paste)
protected Point	**getMagnet(Point mousePosM, MarkerSet tabou)** Searches for a marker in specified Const.MAGNETIC_RAD(ius) from mousePos that is a valid magnet.
protected GOD	**gOD()** To avoid lot of link-chains (v.vf.gOD) in rest of class' code
private GSet	**insertionLayer()** *Returns:* The layer which is visible and highest in the legend order
protected boolean	**isLeftButton(MouseEvent e)**
protected boolean	**isMiddleButton(MouseEvent e)**
protected boolean	**isRightButton(MouseEvent e)**
public void	**keyPressed(KeyEvent e)** Invoked when a key has been pressed.
public void	**keyReleased(KeyEvent e)** Invoked when a key has been released.
public void	**keyTyped(KeyEvent e)** Invoked when a key has been typed, to handle short-cuts, at the moment: '+' and '-' to zoom
protected void	**maybeShowPopup(MouseEvent e)** Does show the popup menu on screen, if e is from right button
public void	**mouseClicked(MouseEvent e)** Invoked when the mouse has been clicked on the view.
public void	**mouseDragged(MouseEvent e)** Invoked when a mouse button is pressed on view and then dragged.
public void	**mouseEntered(MouseEvent e)** Invoked when the mouse enters the view.
public void	**mouseExited(MouseEvent e)** Invoked when the mouse exits the view.
public void	**mouseMoved(MouseEvent e)** Invoked when the mouse button has been moved on a component (with no buttons no down).

public void	**mousePressed(MouseEvent e)** Invoked when a mouse button has been pressed on view.
public void	**mouseReleased(MouseEvent e)** Invoked when a mouse button has been released on view.
protected Rectangle	**sensRectM(int xv, int yv)** calc sensitive area according to Const.TOL (mouse tolerance)
protected GSet	**visibleRange(Rectangle rm, boolean enclosing)** Returns gSet of all gObjects contained in rm and in a visible layer. *Parameters:* **rm :** **enclosing :** True, if gObjects must be inside completely, false if overlapping is allowed as well

C.3 god.GOD

java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Panel
java.applet.Applet
javax.swing.JApplet

---

public *GOD* extends JApplet
implements ClipboardOwner,ActionListener,InternalFrameListener,ComponentListener

Main class, startable as applet or application

Field Summary

Type	*Description*
protected int	**actFrameNo** : The currently activated viewFrame's number in viewFrames
protected Clipboard	**clipboard** : Container of cutten or copied selection
protected JDesktopPane	**desktop**
protected ArrayList	**mapImages** : All open bitmap backgrounds
protected JMenuBar	**menuBar**
protected JMenu	**menuDatabase**
protected JMenu	**menuDebug**
protected JMenu	**menuFile**
protected JMenu	**menuMap**
protected JMenu	**menuSelection**
protected JMenu	**menuViewFrame**
protected JMenu	**menuWindow**
protected JMenuItem	**miCloseMap**
protected JMenuItem	**miCommit**
protected JMenuItem	**miConnect**
protected JMenuItem	**miCopySel**
protected JMenuItem	**miDeleteSel**
protected JMenuItem	**miInsertSel**
protected JMenuItem	**miInsertViewFrame**
protected JMenuItem	**miLoadDump**
protected JMenuItem	**miNew**
protected JMenuItem	**miOpenMap**
protected JMenuItem	**miQuit**
protected JMenuItem	**miRenameViewFrame**
protected JMenuItem	**miRollback**

protected JMenuItem	**miSaveDump**
protected JMenuItem	**miTreePrint**
protected JMenuItem	**miTreeStructure**
protected Model	**model** : The center of all datastructure
protected Navigator	**navigator** : Custum swing component, containing tables, the jTree and the overView
protected ArrayList	**selections** : All open selections
protected JSplitPane	**splitPane** : standard swing components
protected JLabel	**statusBar**
protected ArrayList	**viewFrames** : all open viewFrames

Constructor Summary

Description
GOD() +

Method Summary

Returns	*Description*
public void	**actionPerformed(ActionEvent e)** Called to handle menu items
protected void	**closeAll()** To flush all viewFrames, mapImages, selections and the model
public void	**componentHidden(ComponentEvent e)** Invoked when the component has been made invisible.
public void	**componentMoved(ComponentEvent e)** Invoked when the component's position changes.
public void	**componentResized(ComponentEvent e)** Invoked when the component's size changes.
public void	**componentShown(ComponentEvent e)** Invoked when the component has been made visible.
protected ViewFrame	**getActiveFrame()** *Returns:* the viewFrame associated with actFrameNo
protected View	**getViewAt(Point desktopPos)** *Returns:* the view, which is located at certain desktop-relative coordinate. Specially to move objects across views
public void	**init()** init of components, layouting and loading default files, defined in Const

public void	**internalFrameActivated(InternalFrameEvent e)** Invoked when an internal frame is activated.
public void	**internalFrameClosed(InternalFrameEvent e)** Invoked when an internal frame has been closed.
public void	**internalFrameClosing(InternalFrameEvent e)** Invoked when an internal frame is in the process of being closed.
public void	**internalFrameDeactivated(InternalFrameEvent e)** Invoked when an internal frame is de-activated.
public void	**internalFrameDeiconified(InternalFrameEvent e)** Invoked when an internal frame is de-iconified.
public void	**internalFrameIconified(InternalFrameEvent e)** Invoked when an internal frame is iconified.
public void	**internalFrameOpened(InternalFrameEvent e)** Invoked when a internal frame has been opened.
public void	**lostOwnership(Clipboard clipboard, Transferable contents)** Notifies this object that it is no longer the owner of the contents of the clipboard.
public static void	**main(String[] args)** Although it is an applet it has this main method where a local frame is allocated and the applet trucked inside, so GomsView can be runned as an application as well
protected Point	**posOnComponent(Component destComp, Component sourceComp, Point sourceLoc)** For inter-component coordinate transformation.
protected void	**setActiveFrame(ViewFrame vf)** sets the a viewFrame to the active one, actassociated with actFrameNo

C.4 god.GOD.RenameDialog

java.lang.Object
 java.awt.Component
  java.awt.Container
   java.awt.Window
    java.awt.Dialog
     javax.swing.JDialog

---

public *GOD.RenameDialog* extends JDialog
implements ActionListener

Simple dialog to rename the activated viewFrame

Field Summary

Type	*Description*
protected JButton	**btnCancel**
protected JButton	**btnOK**
protected StringBuffer	**newName** : reference to object of client class, where it wants the result must be StringBuffer, because String is immutable!
protected JTextField	**txtNewName**

Constructor Summary

Description
GOD.RenameDialog(GOD god, StringBuffer newName) +

Method Summary

Returns	*Description*
public void	**actionPerformed(ActionEvent e)** Sets the newName to text, written by the user

C.5 god.GOD.SimpleFileFilter

java.lang.Object
  javax.swing.filechooser.FileFilter

---

public static *GOD.SimpleFileFilter* extends FileFilter

Field Summary

Type	*Description*
String	**description**
String	**extension**

Constructor Summary

Description
GOD.SimpleFileFilter(String extension, String description) +

Method Summary

Returns	*Description*
public boolean	**accept(File f)**
public String	**getDescription()**

C.6 god.Legend

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JScrollPane

---

public *Legend* extends JScrollPane
implements MouseListener

Field Summary

Type	*Description*
protected GOD	**gOD** : Anchor to central application
protected ButtonGroup	**groupInsertMode**
protected JPanel	**itemPane** : Standard swing components
protected ArrayList	**items** : The contained legndItems in specified order, which determines succession of painting the associated layers
protected JRadioButton-MenuItem	**miLine**
protected JRadioButton-MenuItem	**miPoint**
protected JRadioButton-MenuItem	**miPolygon**
protected JPopupMenu	**popupMode**

Constructor Summary

Description
Legend(GOD gOD) +   To instantiate a new Legend.

Method Summary

Returns	*Description*
protected boolean	**isRightButton(MouseEvent e)**
public void	**mouseClicked(MouseEvent e)** Invoked when the mouse has been clicked on a component.
public void	**mouseEntered(MouseEvent e)** Invoked when the mouse enters a component.Empty
public void	**mouseExited(MouseEvent e)** Invoked when the mouse exits a component.
public void	**mousePressed(MouseEvent e)** Invoked when a mouse button has been pressed on a component.
public void	**mouseReleased(MouseEvent e)** Invoked when a mouse button has been released on a component.
public void	**rebuild()** For revalidating Legend if order of contained legendItems has changed

C.7 god.LegendItem

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JPanel

---

public *LegendItem* extends JPanel
implements MouseListener,MouseMotionListener

Field Summary

Type	*Description*
protected Object	**colorizingMember** : Attributes to specify the color of the gObjects contained in the associated layer
protected Color	**contEndColor**
protected double	**contEndValue**
protected Color	**contStartColor**
protected double	**contStartValue**
protected GSet	**layer** : The associated layer
protected Legend	**legend** : Back-link to container
protected boolean	**shrinked** : A legendItem has two screen representations a short and an extended one to display additional information
protected String	**title** : The name written on it
protected Color	**uniqueColor**
protected boolean	**visible** : Shall the associated layer be drawn

Constructor Summary

Description
LegendItem(Legend legend, GSet layer) + To construct and init a new legendItem

Method Summary

Returns	*Description*
protected void	**drawDescription(Graphics2D g2)** To draw part of expanded representation, called from within paintComponent(..)
public void	**mouseClicked(MouseEvent e)** To handle buttons within legendItem and to bring a LegendItemDialog to screen

public void	**mouseDragged(MouseEvent e)** Invoked when a mouse button is pressed on a legendItem and then dragged.
public void	**mouseEntered(MouseEvent e)** Invoked when the mouse enters a component.
public void	**mouseExited(MouseEvent e)** Invoked when the mouse exits a legendItem.
public void	**mouseMoved(MouseEvent e)** Invoked when the mouse button has been moved on a legendItem (with no buttons no down).
public void	**mousePressed(MouseEvent e)** Invoked when a mouse button has been pressed on legendItem.
public void	**mouseReleased(MouseEvent e)** Invoked when a mouse button has been pressed on legendItem.
public void	**paintComponent(Graphics g)** Called from Java painting system to draw its contents
public String	**toString()** Human readable text of this LegendItem, good for debugging

C.8 god.LegendItemDialog

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog

---

public *LegendItemDialog* extends JDialog
implements ActionListener,ChangeListener

Field Summary

Type	*Description*
protected JButton	**btnCancel**
protected JButton	**btnElseColor**
protected JButton	**btnEndColor**
protected JButton	**btnOK**
protected JButton	**btnStartColor**
protected JButton	**btnUniqueColor**
protected JPanel	**colorPane**
protected JPanel	**colorSpecContPane**
protected JPanel	**colorSpecDiscPane**
protected JPanel	**colorSpecPane**
protected JPanel	**colorUniquePane**
protected LegendItem	**item** : Anchor to know, where to change attributes
protected JList	**list**
protected Vector	**memberNames**
protected Vector	**members** : All the attributes and methods with no parameters from type of associated layer
protected JRadioButton	**radioBoundColor**
protected JRadioButton	**radioContinuous**
protected JRadioButton	**radioDiscrete**
protected JRadioButton	**radioUniqueColor**
protected ButtonGroup	**spectrumGroup**
protected JTabbedPane	**taggedPane**
protected JTextField	**txtEndValue**
protected JTextField	**txtStartValue**
protected ButtonGroup	**uniqueGroup**

Constructor Summary

Description
LegendItemDialog(JFrame owner, LegendItem it) + Constructs a new Dialog to specify appearance of gObjects in the item's layer

Method Summary

Returns	*Description*
public void	**actionPerformed(ActionEvent e)** To bring user's specifications to the legendItems fields
public void	**stateChanged(ChangeEvent e)** To update which is visible: btnUniqueColor or colorSpecPane - depending on radio buttons

C.9 god.MapImage

java.lang.Object

---

public *MapImage* extends Object

Field Summary

Type	*Description*
protected String	**imageFilename** : File, where the image has its persistent ressource
protected transient Image	**img** : Ressource of MapImage during runtime
protected String	**title** : How it is called
protected boolean	**visible** : Should it be painted
protected int	**x0** : position of upper left corner in model
protected int	**x1** : position of lower right corner in model
protected int	**y0**
protected int	**y1**

Constructor Summary

Description
MapImage(String filename) + Constructs a new MapImage and initialises the transient img from file

Method Summary

Returns	*Description*
protected void	**paint(Graphics2D g2, View v)** To draw the mapImage on a given view, position and scale depends on relations to model specified in fields x0, y0, x1, y1
public final int	**xi(int xm)** Model to image transformation
public final int	**xm(int xi)** Image to model transformation
public final int	**yi(int ym)** Model to image transformation
public final int	**ym(int yi)** Image to model transformation

C.10 god.MapImage.TableModel

java.lang.Object
  javax.swing.table.AbstractTableModel

---

public static *MapImage.TableModel* extends AbstractTableModel

Field Summary

Type	*Description*
protected ArrayList	**mapImages**

Constructor Summary

Description
MapImage.TableModel(ArrayList mapImages) +   The ressource of table displaying opened mapImages.

Method Summary

Returns	*Description*
public Class	**getColumnClass(int col)**
public int	**getColumnCount()**
public String	**getColumnName(int col)**
public int	**getRowCount()**
public Object	**getValueAt(int row, int col)**
public boolean	**isCellEditable(int row, int col)**
public void	**setValueAt(Object value, int row, int col)**

C.11 god.Navigator

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JPanel

---

public *Navigator* extends JPanel
implements ListSelectionListener,TableModelListener,TreeSelectionListener

Field Summary

Type	*Description*
protected DefaultMutable-TreeNode	**assoRoot** : root node for all associations
protected DefaultMutable-TreeNode	**collRoot** : root node for all collections
protected GOD	**gOD** : Anchor to central application
protected OverView	**overView** : Custom component painting certain layers all over the model area
protected DefaultMutable-TreeNode	**root** : root node containing collRoot and assoRoot
protected JTable	**tblMap** : To display a list of all open mapImages
protected JTable	**tblSel** : To display a list of all open selections
protected JTree	**tree** : To visualise the collections and associations structure
protected Default-TreeModel	**treeModel** : Ressource of jTree's data

Constructor Summary

Description
Navigator(GOD gOD) +

Method Summary

Returns	*Description*
protected void	**buildTree()** Does invoke insertIntoTree(..) twice, once for collections and once for associations in the model
protected void	**insertIntoTree(TreeInsertable[] colls, Default-MutableTreeNode root)** Algorithm to constuct the jTree from data model's collection or association structure.
protected void	**markAccordingMapImage(ViewFrame vf)** To mark the given viewFrame's mapImage as row within tblMap

protected void	**markAccordingSelection(ViewFrame vf)** To mark the given viewFrame's selection as row within tblSel
protected void	**refreshTables()** Revalidates and repaints the contained tables
public void	**tableChanged(TableModelEvent e)** Called, when data in table is changed, to apply changes
public void	**valueChanged(ListSelectionEvent e)** Called, when selection-row or mapImage-row in tables is changed, to apply changes
public void	**valueChanged(TreeSelectionEvent e)** Called, when selected node in jTree is changed, to apply changes

C.12 god.OverView

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JPanel
          gStructure.AbstractView

---

public *OverView* extends AbstractView
implements MouseListener,MouseMotionListener,Observer

Field Summary

Type	*Description*
protected boolean	**dragBox**
protected Navigator	**nav**
protected int	**oldXv**
protected int	**oldYv**
protected boolean	**scaleBox**
protected Selection	**selection**
protected Map	**visibleLayers**

Constructor Summary

Description
OverView(Navigator nav) +

Method Summary

Returns	*Description*
protected View	**actView()**
public void	**mouseClicked(MouseEvent e)**
public void	**mouseDragged(MouseEvent e)**
public void	**mouseEntered(MouseEvent e)** Invoked when the mouse enters overView.

public void	**mouseExited(MouseEvent e)** Invoked when the mouse exits overView.
public void	**mouseMoved(MouseEvent e)** Invoked when the mouse button has been moved on overView (with no buttons no down).
public void	**mousePressed(MouseEvent e)** Invoked when a mouse button has been pressed on overView.
public void	**mouseReleased(MouseEvent e)** Invoked when a mouse button has been pressed on overView.
public void	**paintComponent(Graphics g)**
protected Rectangle	**smallBoxV(View v)** Calculates the box-representation (yellow rect) of a View
public void	**update(Observable o, Object arg)** Is called by model.notifyObservers, when changes occured.

C.13 god.OverView.Dialog

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog

---

public *OverView.Dialog* extends JDialog
implements ActionListener

Field Summary

Type	*Description*
protected JButton	**btnCancel**
protected JButton[]	**btnColors**
protected JButton	**btnOK**
protected JCheckBox[]	**cboxLayers**

Constructor Summary

Description
OverView.Dialog(OverView this\$0) +   Self-generating Dialog to chose displayed layers and their colors

Method Summary

Returns	*Description*
public void	**actionPerformed(ActionEvent e)** Changes the visibleLayers field

C.14 god.Selection

java.lang.Object
gStructure.GSet
gStructure.GHashSet

---

public *Selection* extends GHashSet
implements Cloneable,Transferable

Field Summary

Type	*Description*
public static final-DataFlavor	**gFlavor**
protected String	**name** : Displayed in table
protected static int	**nInstanciation** : To be able to give a unique title

Constructor Summary

Description
Selection() + Invokes Selection(name) with default generated name
Selection(String name) + Constructs a Selection that will be registered in the contained objects.

Method Summary

Returns	*Description*
public Selection	**copy()** *Returns:* A copy - referncing the same gObjects
public void	**deleteContent()** Invokes o.delete() for all contained gObjects o
protected Vertex	**edgeIn(Rectangle tolerance)** *Returns:* A vertex representing start of a edge within tolerance. Or null
protected Point	**getMiddle()** The center of mbr()
public String	**getName()**
public Object	**getTransferData(DataFlavor flavor)** Returns an object which represents the data to be transferred.
public DataFlavor[]	**getTransferDataFlavors()** Returns an array of DataFlavor objects indicating the flavors the data can be provided in.

protected boolean	**intersects(Rectangle rm)** Does the given Rectangle overlap any gObjects contained in the selection?
public boolean	**isDataFlavorSupported(DataFlavor flavor)** Returns whether or not the specified data flavor is supported for this object.
protected Selection.SinglePositionMarkerSet	**markersIn(Rectangle tolerance, boolean multipleAllowed)** *Returns:* A helper container where all points and vertices (must be at exactly the same position) of contained gObjects within tolerance are pushed in
protected Point	**nearestMarker(Point mousePos)** *Returns:* the point or vertex that is nearest to given position
protected void	**paint(Graphics2D g2, View v)** To draw all the markers highlighted by dots in given view
protected Selection.MarkerSet	**selMarkers()** *Returns:* A helper container where all points and vertices of all contained gObjects are pushed in
public void	**setName(String name)**
public String	**toString()** Human readable text for this Selection, good for debugging
protected void	**translateAnchored(Point anchor, Point target)** To translate the selection by vector _v (_target - _anchor)

C.15 god.Selection.MarkerSet

java.lang.Object

public *Selection.MarkerSet* extends Object

see report for detailed description of MarkerSet

Field Summary

Type	*Description*
protected Set	**gPointSet**
protected Set	**vertexSet**

Constructor Summary

Description
Selection.MarkerSet(Selection this\$0) +

Method Summary

Returns	*Description*
protected void	**addAll()**
protected boolean	**contains(GPoint p)**
protected boolean	**contains(Vertex v)**

C.16 god.Selection.SinglePositionMarkerSet

java.lang.Object
  god.Selection.MarkerSet

---

public *Selection.SinglePositionMarkerSet* extends Selection.MarkerSet

see report for detailed description of SinglePositionMarkerSet

Field Summary

Type	*Description*
protected Point	**location**

Constructor Summary

Description
Selection.SinglePositionMarkerSet(Selection this\$0) +

Method Summary

Returns	*Description*
protected void	**addAll()** not allowed in DragMarkerSet, so it is overwritten with empty statement
protected void	**addRequest(GPoint p)** A Gpoint can not be the 1st inserted point of the DragMarkerSet, and even if there are already other markers contained, the gPoint needs to be at the same postion than those
protected void	**addRequest(Vertex v)**
protected boolean	**isEmpty()**
protected void	**moveTo(Point p)**
protected void	**release()**

C.17 god.Selection.TableModel

java.lang.Object

javax.swing.table.AbstractTableModel

---

public static *Selection.TableModel* extends AbstractTableModel

The ressource of table displaying opened selections. Methods are called by Java's cell-renderer nad -editor to extract and modify data associated with the cells

Field Summary

Type	*Description*
protected ArrayList	**selections**
protected ArrayList	**viewFrames**

Constructor Summary

Description
Selection.TableModel(ArrayList viewFrames, ArrayList selections) +

Method Summary

Returns	*Description*
public Class	**getColumnClass(int col)**
public int	**getColumnCount()**
public String	**getColumnName(int col)**
public int	**getRowCount()**
public Object	**getValueAt(int row, int col)**
public boolean	**isCellEditable(int row, int col)**
public void	**setValueAt(Object value, int row, int col)**

C.18 god.View

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JPanel
          gStructure.AbstractView

public *View* extends AbstractView

Field Summary

Type	*Description*
protected Control	**control** : Its event-handler
protected int	**crossXm** : Coordinates of crossmarker, displayed, if selection is empty
protected int	**crossYm**
protected MapImage	**mapImage** : Background bitmap, may be null
protected JMenuItem	**miCopy**
protected JMenuItem	**miCut**
protected JMenuItem	**miPaste**
protected JPopupMenu	**popupEdit** : Standard swing components
protected Selection	**selection** : The set of selected elements within this View
protected ViewFrame	**vf** : The container of it

Constructor Summary

Description
View(ViewFrame vf, int x0, int y0, double zoom) +

Method Summary

Returns	*Description*
protected void	**fitToSelection()** Moves and scales the view such that it exactly contains all the selected gObjects
protected Point	**getMiddle()** *Returns:* The middle of the view (in model coordinates)
protected Rectangle	**getOwnRectM()** *Returns:* The represented rectangle in model of this View

protected Color	**interpolatedColor(Color c0, Color c1, double between)** Helper method to calculate the middle of c0 and c1 *Parameters:* **between :** must be in the range [0, 1]!
public void	**paintComponent(Graphics g)** Central method of View, does paint mapImage, visible layers in legend-specified order, selection and finally gridlines, if zoom is high enough
public void	**refresh(Rectangle rm)** Does repaint just a certain area in model
protected void	**setMiddle(Point pm)** To move the view to a certain model position
protected void	**setZoom(double newZoom)** Does set the scale factor by bewaring middle of the view at the same model position as before

C.19 god.ViewFrame

java.lang.Object
java.awt.Component
java.awt.Container
javax.swing.JComponent
javax.swing.JInternalFrame

---

public *ViewFrame* extends JInternalFrame
implements Observer,ActionListener,KeyListener,TableModelListener,ListSelectionListener

Field Summary

Type	*Description*
protected boolean	**anitaliased**
protected JButton	**btnFindInvalid**
protected JButton	**btnFitToSel**
protected JButton	**btnQuery**
protected TreePath	**collTreePath** : To store, which is the node in navigator's jTree chosen for that viewFrame
protected GOD	**gOD** : Anchor to central application
protected boolean	**gridShown**
protected JLabel	**lblCrossX**
protected JLabel	**lblCrossY**
protected JLabel	**lblZoom**
protected Legend	**legend** : The associated legend
protected JSplitPane	**splitPaneH** : Standard swing components
protected JSplitPane	**splitPaneV**
protected CollectionTable	**table** : The table to show gObjects member values
protected JScrollPane	**tablePane**
protected JToolBar	**toolBar**
protected JTextField	**txtCrossX**
protected JTextField	**txtCrossY**
protected JTextField	**txtZoom**
protected View	**v** : The displayed view

Constructor Summary

Description
ViewFrame(String title, GOD gOD) +

Method Summary

Returns	*Description*
public void	**actionPerformed(ActionEvent e)** Invoked when an action occurs.
public void	**keyPressed(KeyEvent e)** Invoked when a key has been pressed.
public void	**keyReleased(KeyEvent e)** Invoked when a key has been released.
public void	**keyTyped(KeyEvent e)** Invoked when a key has been typed.
protected void	**refreshTableSelection()** Called, to update table selection for being consistent to view's selection.
protected void	**setTable(CollectionWrapper coll)** To change the table to the one of another collection
public void	**tableChanged(TableModelEvent e)** called, when data in table is changed
public void	**update(Observable o, Object arg)** Called by model's notifyObservers to tell the viewFrame to repaint because of updates in data
public void	**valueChanged(ListSelectionEvent e)** called, when selection in table is changed

C.20 god.ViewFrame.QueryDialog

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog

---

public *ViewFrame.QueryDialog* extends JDialog
implements ActionListener

Field Summary

Type	*Description*
protected JButton	**btnCancel**
protected JButton	**btnOK**
public String	**queryString** : resulting String to invoke query-machine with
public String	**resSelName** : resulting name of Selection which will contain query results
protected JTextArea	**txtQuery**
protected JTextField	**txtSelection**

Constructor Summary

Description
ViewFrame.QueryDialog( ) + this constructor just initialises layout

Method Summary

Returns	*Description*
public void	**actionPerformed(ActionEvent e)** to handle ok and cancel button in case of ok button resSel is instantiated and queryString set according to input of user, in case of cancel the dialog will just be disposed and nothing further happens

Bibliography

[1] C. Parent, S. Spaccapietra and E.Zimányi *Spatio-Temporal Conceptual Models: Data Structures + Space + Time.* http://lbd.epfl.ch/e/publications/

[2] C. Parent, S. Spaccapietra, E.Zimányi, P. Donini, C. Plazanet and C. Vangenot *Modelling Spatial Data in the MADS Conceptual Model.* In Proceedings of the International Symposium on Spatial Data Handling, 1998. http://lbd.epfl.ch/e/publications/

[3] M. C. Norrie. *Distinguishing Typing and Classification in Object Data Models.* http://www.globis.ethz.ch (Publications)

[4] A. Kobler and M. C. Norrie. *OMS Java: An Open, Extensible Architecture for Advanced Application Systems such as GIS.* http://www.globis.ethz.ch (Publications)

[5] A. Wuergler and M. C. Norrie. *OMS Pro Introductory Tutorial.* http://www.globis.ethz.ch (Publications)

[6] A. Kobler, B. Signer and M. C. Norrie. *OMS Java Object-Oriented Framework and Data Management System Manual.* http://www.globis.ethz.ch (Publications)

[7] S. Avelar. *Generating Topologically Correct Schematic Maps.* Technical report 336, Department of Computer Science, ETH Zurich, October 2000.

[8] S. Avelar, R. Huber *Modelling a Public Transport Network for Generation of Schematic Maps and Location Queries.* In Proceedings of 20th International Cartographic Conference, Peking, August 6-10, 2001.

[9] R. Huber A Slim and Open Data Model for Public Transport Network using OMS raphael.huber@gmx.ch

[10] M. v. Kreveld, J. Nievergelt, T. Roos and P. Widmayer. *Algorithmic foundations of GIS.* Springer 1997.

[11] D. Flanagan *Java in a Nutshell - 3rd Edition.* O'Reilly 1999.

[12] G. Krüger *Goto Java 2.* Addison-Wesley 2000.

[13] M. Campione, K. Walrath The Java Tutorial, 3rd Edition Addison-Wesley 2001.

[14] M. Campione, K. Walrath The JFC Swing Tutorial Addison-Wesley 1999.

[15] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf *Computational Geometry - 2nd Edition.* Springer 2000.

[16] K. Schutte *An edge labeling approach to concave polygon clipping.* ACM Transactions on Graphics 1995.

[17] J. Nievergelt and K. H. Hinrichs *Algorithms & Data Structures.* Prentice Hall 1993.

[18] E. Gamma, R. Helm, R. Johnson and J. Vlissides *Design Patterns.* Addison-Wesley 1995.

[19] T. Oetiker, H. Partl, I. Hyna and E. Schlegl *The Not So Short Introduction to LATEX 2.* ftp://ftp.dante.de/tex-archive/info/lshort

[20] R. Lamprecht *Interactive Learning Components for the Study of Finite Automata.* http://www.tedu.ethz.ch/ifa