Automatic generation of a view to a geographical database.

Mats Dunkars
SWECO Position
mats.dunkars@sweco.se

Abstract

This paper concerns object oriented modelling and automatic generalisation of geographic information. The focus however is not on traditional paper maps, but on screen maps that are automatically generated from a geographical database. Object oriented modelling is used to design screen maps that are equipped with methods that automatically extracts information from a geographical database, generalises the information and displays it on a screen.

Keywords

Object Oriented Modelling, The Unified Modelling Language, Digital Cartography, Automated Map Generalisation

Introduction

Object oriented modelling in geographical information is often used to design the database. This paper takes a different approach and uses object oriented modelling to design screen maps that are part of a user interface. The algorithms and parameters that determine how the generalisation shall be performed is stored within the object classes that belong to the screen map rather than being stored in the database. Most objects (roads, railways, houses) shall be generalised differently in different map contexts. If this knowledge is stored within the database this implies that we need to store a large set of different generalisation algorithms and parameters for the general object class in the database and also information about which algorithm and parameters to use for which map series. In this paper the database contains multi-purpose object classes and objects that are stored with as high accuracy as possible in a general database, but has no information on how to be displayed or generalised. This is stored in the object classes that belong to a certain user interface.

A model has been designed using the Unified Modelling Language (UML) which is described by Booch et al. (1999). This model consists of a set of typical object classes in a topographical map and has been partly implemented within the Geographic Information System LAMPS2.

Design criteria

According to Booch et al. (1999) a model is a simplification of reality to solve a certain problem. The problem addressed here is how to generate a user interface that retrieves information from one or several geographical databases and generalises the information before it is displayed in a map. A typical example is a municipal GIS which consists of several databases that are owned, maintained and updated continuously by different organisations. Several different users such as urban planners, the environmental agency, maintenance personnel of the different utility departments, the fire department etc combine information from different geographical databases and use their own symbolisation to highlight certain aspects of a problem. There is an obvious risk that this will lead to conflicts due to lack of generalisation.

The user interface that is considered here is typically rather simple and designed for a particular user group. It gives the user the ability to view a map and zoom in and out to get an overview or to see details. The information that is displayed changes with scale. The user can retrieve additional information about the objects that are shown in a view, such as the owner of a house. It is also possible to perform different analyses depending on the application. In a car navigation system for instance it is possible to find the best route and to compute distances between locations. The view to the database is static in a sense that the user can not add additional information to the view. The user can choose to see subsets of the information that is available to a view however, by marking information in a legend

General Design

The user-interface should have the ability to zoom in and out and to see information at various levels of resolution. In the model that is presented here this is handled by letting the user-interface consist of static maps defined at different scales. As the user zooms the appropriate map is shown. This is a simpler approach than the one presented by Petzold et al. (1999) that has automatic name placement that is continuously changing with scale. However, a map that is constantly changing even with small changes in resolution might be rather confusing to the map-reader.

A static map that is a part of a user-interface will be called a level for the remaining part of this paper and the issue is how the level is generated. A level has the following characteristics:

- A level is static and is defined to be viewed at a certain scale range.
- An object class is defined in one and only one level.
- The representation is defined for each object class, which means that an object class can only be represented in one way.

This implies that a category, such as building, will be represented by several object classes where each of these object classes is defined for a certain level.

When a new level is defined the cartographer creates a data model for the level. He defines the object classes that are members of the level, how they are represented in the level, the requirements for preservation of topological constraints between the different object classes etc. He also equips the object classes with methods that retrieve and analyse information from one or several databases, generalises the information and instantiates new objects. When all is defined, objects that are members of the level can be instantiated. As the new objects are created an analysis is performed in the new level to find a suitable location that does not cause conflicts with objects that already have been created.

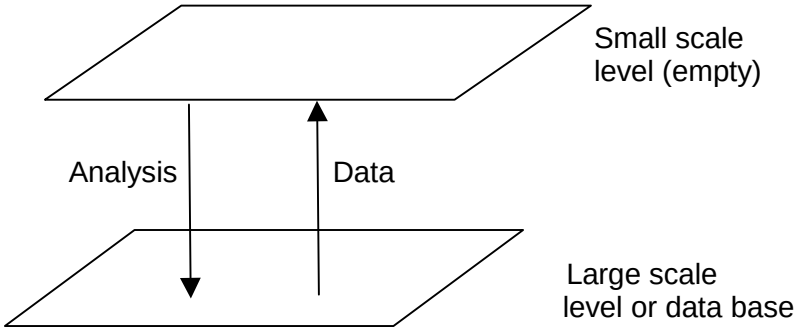

Figure 1 Generalisation

If the new objects create conflicts in the map that can not be solved it can be decided not to create the object or perhaps to merge the new object with an already existing object. What is done depends on the application and the object classes involved. The analysis is performed at two levels simultaneously: the data source and the level that is being created.

Based on these ideas a model has been created that contains a subset of categories from a topographical map. The categories have been chosen to get a wide representation of different types of geographical categories that interact in different ways.

The Model

The model consists of two levels: the green and the blue level. The green level is defined at the scale of 1:10 000 and is the base level which is initially filled with data from the National Land Survey of Sweden (NLS). The blue level is automatically created by retrieving and analysing data from the green level. Depending on the choice of symbolisation and generalisation parameters the blue level can be instantiated at scales in a range from 1:30 000 to 1:100 000. In general there are no requirements on the how the source data shall be structured. In this model however it is assumed that all linear objects form a

node-link structure, called a network and that the surface objects form a complete surface cover without any gaps.

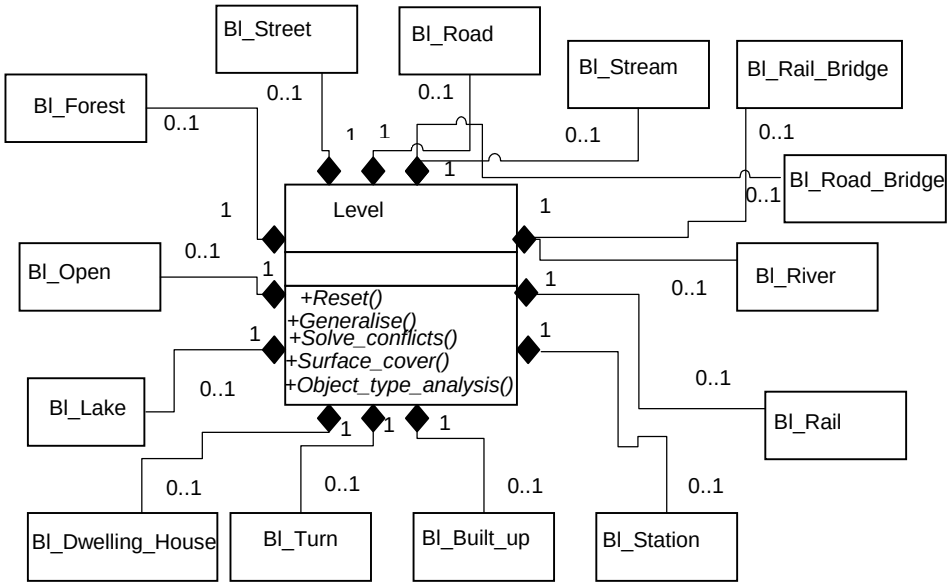

Figure 2 A UML-diagram showing the Bl object classes are members of the blue level.

Levels

Figure 2 shows the geographical object classes that are members of the blue level. In figure 5 it is possible to see how the Road object class is split into different road types. The connection between the object classes is an aggregation meaning that a level consists of geographical objects. The green level has a similar set of classes and contains the data while the blue level is empty at the start. A process is initiated in the blue level and data is extracted from the green level, analysed and inserted into the blue level. The process ends when all seed objects (described below) in the green level have been treated.

Inheritance

The model uses inheritance structures, which are shown in Figure 3. There are three different inheritance structures for objects represented as points, lines and areas. At the most general level is a virtual object class called bl_Object. In this class the three methods Select(), Create() and Simplify() are defined. These methods are later implemented further down in object class hierarchy. All linear object classes inherit from the bl_Network object class. There is an association within the class that describes how linear objects form node-link structures. Most of the attributes concerning symbolisation and constraints such as accuracy requirements or the minimum length an object should have to be displayed at this level are defined in the bl_Network object class. The values of the attributes are then given in the leaf object classes.

Object classes such as station and rail-bridge has associations with railroad. These associations illustrate topological relations between these object classes. A station or a rail-bridge has to be connected to a railroad.

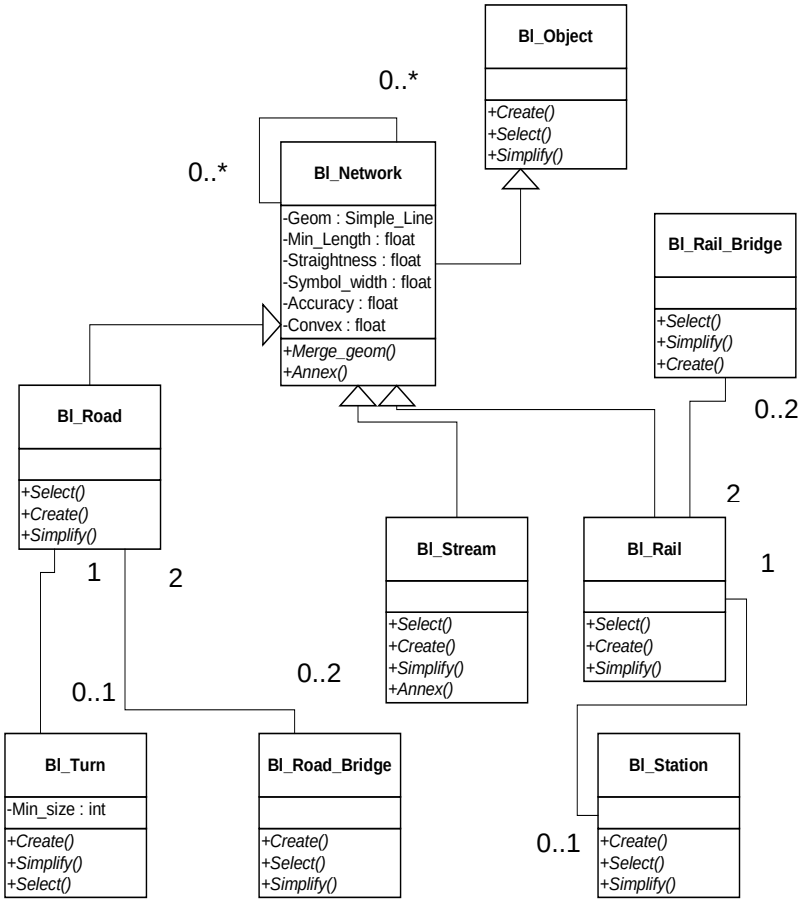

Figure 3 The inheritance structure of object classes in the blue level that are represented with linear geometry.

In the initial stages of this work an attempt was made to model the hydrographic network using multiple inheritance. The hydrographic network includes both surface objects such as lakes and rivers and linear objects such as streams. The idea was that the river object class to some extent has the characteristics of a lake and to some extent has the characteristics of a stream and ought to inherit functionality from both. As the functionality was thought out in detail it turned out that functions that the river object had a need to inherit where defined within the surface object class. The characteristics of the network object class, the river class needs are the requirements on how data should be structured. To make sure that crossings of linear features, e.g. roads, and rivers are maintained properly the river is divided into a new object at every crossing. Figure 4 shows how a river object is formed between two road-bridges.

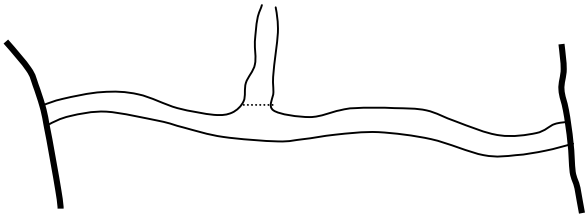

Figure 4 Example of a river object that ends where the road crosses the river. There is also a split into separate objects where the two river objects meet.

The part of the road that crosses the river, the bridge, has different requirements on how it shall be generalised compared other parts of the road. A road bridge should be located "on" the river and other parts of the road should be located "on" land. Therefore all road and railway bridges are treated in separate classes. Where a road crosses a stream there is no need to form a bridge for the purpose of generalisation. The node where the two objects cross is maintained by the network structure.

The different road types are treated as separate road classes since they have different symbolisation. As can be seen in figure 5 however they have the Simplify() method in common which is implemented in the general Bl_Road class. The only methods that are implemented in the leaf object classes are Create() which is the constructor of the class and Select() which determines if the information retrieved from the source database is suitable to instantiate a new object(e.g. is this cul-de-sac long enough).

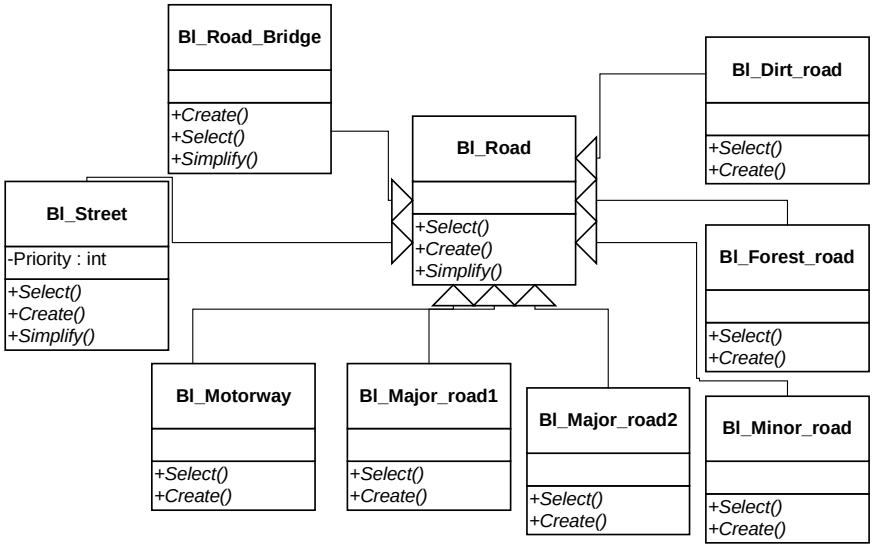

Figure 5 Different road types inherit from the general road object class.

Dependencies

There is a part of the model that defines which objects might be effected if an object is moved. This is illustrated using dependencies, which have a specific meaning in this model. The lake class for instance has dependencies with all other object classes. This means that if a lake object is moved, conflicts might occur with objects of any other object class. In Figure 6 we see a diagram that illustrates how the move of a house object effects objects in other classes. The interesting information is what object classes are left out. As can be seen the object classes bl_forest and bl_open are not considered to be effected when a house object is moved. This means that, if a house object is located "on" a forest object or "on" an open-area object is irrelevant in this level. This is a design choice for this particular level and in a different level it might be decided to let the house object class have dependencies to all other object classes.

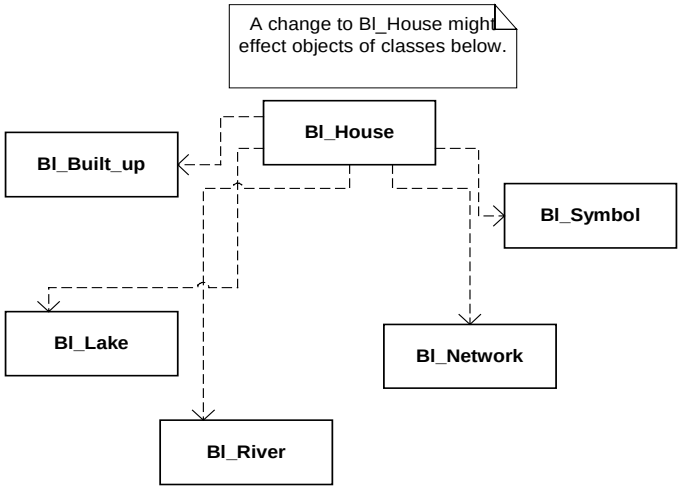

Figure 6 This figure illustrates which object classes that might be effected if a house object is moved. A house may not be located on water or on the wrong side of a road but it might be moved from a forest to an open area without severely deteriorating the quality of the map.

Similar relations are created for each object class. Even though the house and forest object classes do not have dependencies, they can effect each other indirectly through other object classes. A house object, for instance, can be moved in such a manner that it effects a lake object, which in turn, effects the forest object when it is moved. All object classes in the level have direct or indirect dependencies.

Links between levels

There are two types of links between object classes defined in different levels, an association and a dependency, which is illustrated in Figure 7. The association is formed for the object classes where the relation is intuitively simple to form or where the relation is needed when the data in the new level is created. The association between bl_lake and gr_lake is simple to form since it is considered to be obvious which objects in the two classes that refer to the same real world feature. The association between bl_river and gr_river is more vague and forms a many-to-many relation.

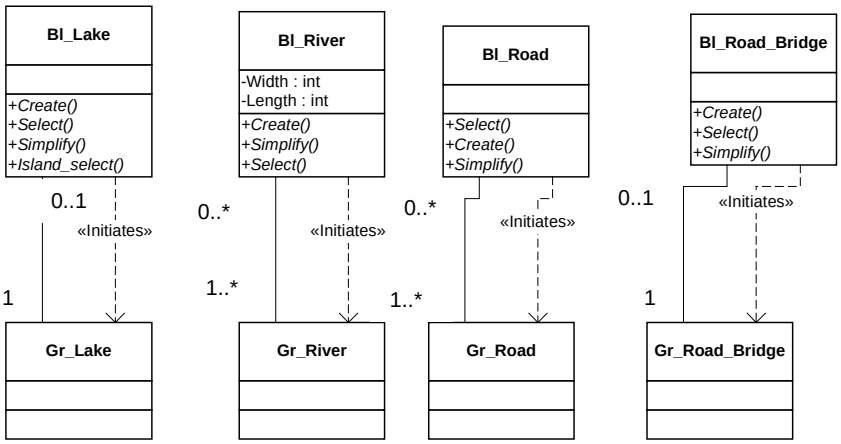

Figure 7 There are two kinds of links between object classes in different levels, an association and a dependency.

The other relation that is formed between object classes is a dependency called “initiate” that has a different meaning than the dependency in the former section that illustrates topological relations. During the creation process the object classes in the bl_level analyses the gr_level to find objects and groups of objects that can form objects in the bl_level. The first part of this analysis is to locate an object that can be a seed for the creation of objects in the bl_level. The lake objects of the gr_lake class are the seeds for creation of bl_lakes. The gr_houses and gr_built_up contains the seeds for the creation the built-up areas in the bl_level. This relation is illustrated with the “initiate” dependency.

Creation

The creation of new objects follows a similar pattern for all object classes. However, since different object classes have different characteristics and different relations to other object classes there are variations.

In general the creation of all objects in a class consists of the following steps.

1. The objects that can act as seeds to form objects in the bl level are selected and stored in an array. The seed relations are shown with the “initiate” dependency in Figure 7. The array is stepped through and the objects are analysed one at a time. Step 2-4 is done for an object at a time until the end of the array is reached.

2. First the select() method determines if this particular object is suitable to form an object in the bl_level. For lakes it is a simple decision if the lake is large enough. In the select method other objects in the vicinity of the current object can also be analysed. An island for instance is selected if it is larger than a certain size and does not have any objects on the island that shall be members of the bl_level. An example of such an object could be an important building. This is determined by calling the select methods of the objects on the island. The select function does not give the complete answer to the question if an object shall be displayed or not. At a later stage in the construction process an analysis is made how much space is available for this particular object in the bl level. If there is not sufficient space it might be that the object is not instantiated even though the select function has returned a true.

3. Based on the accuracy requirements of the object class an area is created around the object that shows where the new object may be located in the blue level. If there are other objects in the blue level within this area that the current object can be in conflict with the area is modified until no objects are located within it, using the merge_geom() method. Which object classes a particular class can be in conflict with is modelled as dependencies, which is described in the chapter about dependencies above. The result is an area where the new object may be located.

4. The simplify method computes the geometry of the new object using the geometry of the seed object and the area where the new object may be located.

Roads

The creation of roads is more complex then the general outline above since each road object only stretches between two nodes in the network structure. This means that a single road object might stretch between a crossing with a stream and a crossing with a railway. Harrie (1998) describes the analysis made at the National Land Survey of Sweden when determining if a dirt road is to be selected as a member of the green map (1:50 000). A simplified version of the selection process is to say that a dirt road that is a cul-de-sac is not created if it is shorter than 500 meters and does not lead to a lake or a house. This analysis has been implemented in the select function for dirt roads. A recursive method called Annex() (implemented in the Bl_network class) is used to collect the road objects that lead between two road crossings or form a cul-de-sac. This set of road objects is then analysed according to the above criteria.

If a new road object shall be created in the blue level, the extent of this road object has to be determined. Linear objects should have a node-link structure similar to the structure in the green level. However, we do not know if the other network objects, e.g. a stream, at the endpoints of the current object will be displayed in the blue level. A collection of objects in the green level that can form one object in the blue level has to be formed. Another recursive function calls the select functions of the network objects (e.g. a stream) that connect to the endpoints of the current object. If these objects shall not be displayed in the blue level the road object is added to the collection. When the function ends, we have a collection of road objects that shall form one road object in the blue level.

An area where the new object may be located is created and modified in a similar manner to that described above. When this is done it is an advantage to treat individual links in the network structure as objects. Figure 8 and 9 show how a stream crosses a road and then make a turn and moves along the road. In Figure 8 the individual road object stretches between two road crossings and thus runs over the crossing with the stream. In Figure 9 the road is split at the stream and the individual road object only leads between the crossing with the stream and the crossing with the road. As has been described above, the area where the new object may be located should not have any conflicting objects within it and is modified until all conflicting objects are outside the area. If an individual road object would stretch over crossings with other linear objects, e.g. a stream, we would have to accept the stream object within the area where the new object may be located. This is the case we have in Figure 8 where we have to accept the part of the stream that crosses the road within the area, while the part that runs next to the road should be outside the area. In Figure 9 all linear objects are treated in a node link structure and this problem does not occur. At the end nodes the buffer is modified so that the end node is located on the border of the buffer.

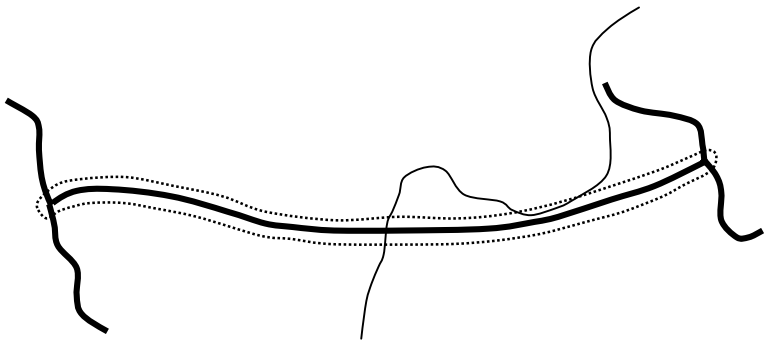

Figure 8, A stream crosses a road and then makes a turn and moves along the road. In this example the road object crosses the stream object. A buffer around the road shows where the new road object may be located. The stream is located within the buffer in two places, at the crossing with the stream and when the stream is close to the road.

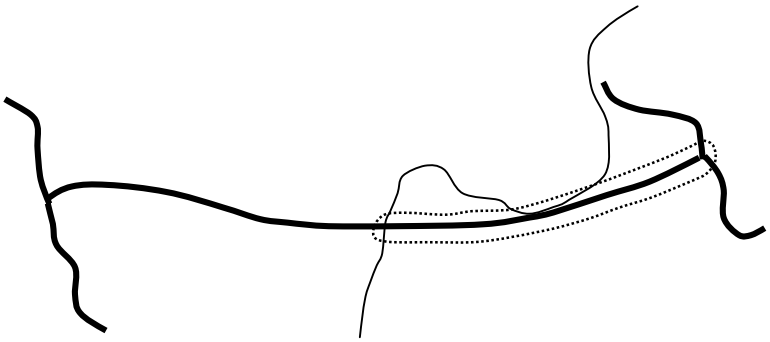

Figure 9, The same case as in Figure 8 but now the road object ends at the crossing with stream.

Finally a new road object is created in the blue level based on the area and the geometry of the road object in the green level.

Built-up area

When built-up areas are created the seed can be either a built-up area or a dwelling house in the green level. The surroundings of the seed object are analysed. If there are any built-up areas or dwelling houses that are located close enough they are annexed. However, they may not be located on the other side of a river or a lake. This process continues recursively for the annexed objects until no more seed objects can be added. The select function determines if the objects cover an area that is large enough to be represented with a built-up area in the blue level. If so the new area is created using the simplify method. In this case there is no need to analyse if there are any conflicting objects since the only objects that has been created so far that can have conflicts with this object are other surface objects. This knowledge can be obtained from the model by looking at the dependencies between different object classes and the creation sequence. New surface objects that overlap already existing surface objects are modified by the simplify method until there is no overlap.

Buildings

There are three kinds of buildings in the green as well as in the blue level: Dwelling houses, churches, and other buildings. Churches and other houses are rather uncomplicated and follow the general pattern for creation of new objects described above. Dwelling houses are more complicated. The seed to create new dwelling houses in the blue level are dwelling houses in the green level. The select function determines if an individual dwelling house object is suitable to be created in the blue level. However, a group of dwelling house objects in the blue level represents both the individual dwelling house features and the pattern formed by a group of real world features. Thus we have to form a collection of dwelling houses that can be analysed to determine which dwelling house objects that shall represent the pattern of buildings. This collection is created, by analysing the surroundings of the seed dwelling house and adding neighbouring dwelling houses to the collection. To be added to the collection a dwelling house should not be located on the other side of a linear

object such as a road or a river. When the surroundings of the seed house has been analysed the process continues recursively with the dwelling houses that have been added to the collection until no more dwelling houses are added. When we have a group of dwelling houses, the area they can occupy in the blue level is determined, which is shown by the dotted line in Figure 10. If the accuracy requirements for dwelling houses are low, there is a risk that they can be located at the wrong side of a road. This can be avoided by using the links that are formed between linear objects in different levels. The roads in Figure 10, for instance, are connected with links between the levels. If, for instance, a road object is located west of the buildings we can use the links to determine which road object in the blue level that has been formed from this road and make sure that the buildings are located west of this road object. When we know the area within which the buildings should be located, the simplify method finds conflicts among the dwelling houses due to symbolisation and selects a subset of the dwelling houses to be represented in the map.

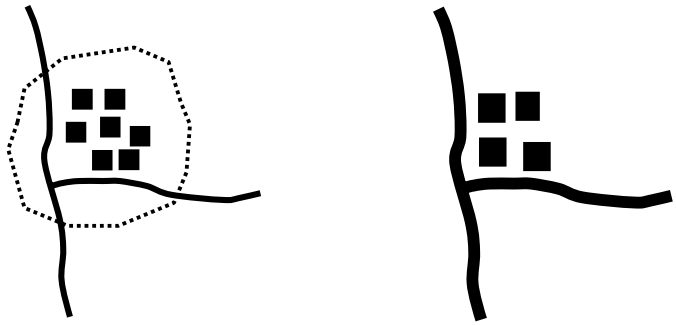

Figure 10 Left is group of buildings in the green level that form a group of buildings in the blue level to the right.

Implementation

The model described above has been implemented for three levels. The green level contains information from the GGD database of the NLS, which is defined at an approximate scale of 1:10 000. The data set covers an area of approximately 60 km^2. The blue level, which is initially empty, exists in two different versions defined at different scales. The first version has a resolution that corresponds to a scale of approximately 1:50 000 and the second one has a resolution that corresponds to an approximate scale of 1:100 000.

The two different versions of the blue level contain essentially the same functionality but the parameters of the object classes are tuned differently for the different scales.

The green level contains the object classes: Lake, river, roads divided into six separate classes, railroad, road-bridge, rail-bridge, stream, dwelling house,

buildings not used as dwelling houses, and church. The blue level contains the object classes: Lake, river, roads divided into 5-6 classes, railroad, road bridge, rail bridge, built-up area, dwelling house, buildings not used as dwelling house. The features are generated in essentially the manner that has been described above. A detailed description is given for roads, built-up area, dwelling houses and buildings not used for dwelling to highlight how the different parameters are selected.

Roads

Roads are treated as one object class that is divided into 5-6 different types using different attribute values. Each road type is given a different symbolisation. This does not comply with what has been said about how object classes previously. Each object class should only be symbolised in one way and therefore each road type should be treated as an object class. However, when roads and railways are created, the topological methods within LAMPS2 are used to find the objects that meet at the end nodes of the objects. Since the topological methods in LAMPS2 does not support inheritance this means that treating the road types as separate object classes is rather cumbersome. For simplicity I have chosen to treat them as one class.

The creation process for roads has been described above. However since the blue level now is becoming more populated we can see how conflicts occur with other road objects as well as with railroad objects. Figure 11 shows an example where the method works quite well. A road runs along a railway and then crosses the railway. There are no other objects in the area that can create conflicts. The road has been moved so that the symbols are not overlapping each other.

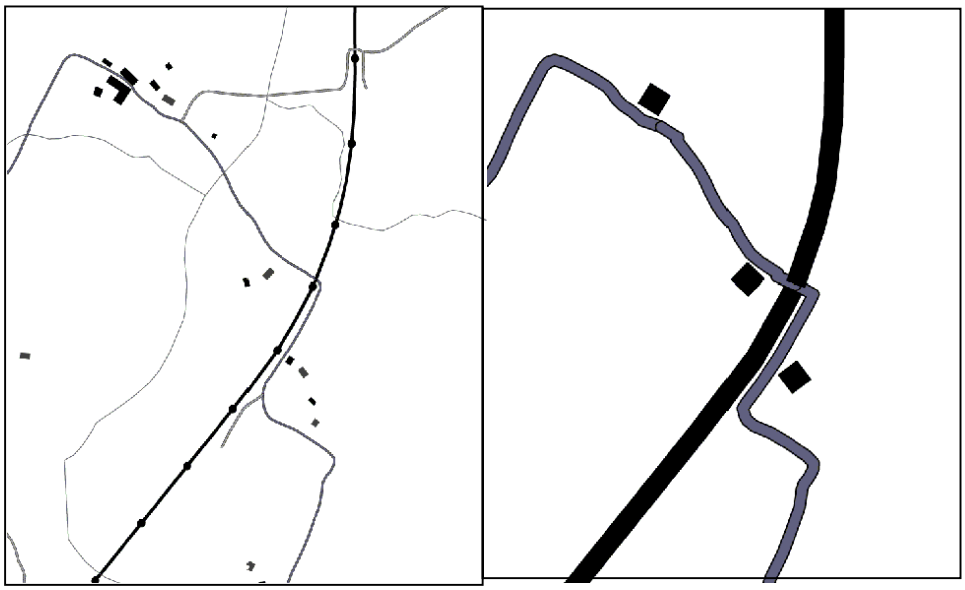

Figure 11 The top map shows the green level and the map below shows the small scale version of the blue level. The road that runs along the railway has

been moved so that the symbols are not overlapping. The node where the road crosses the railway has been maintained properly. The roads that are not shown in the blue level are dirt tracks which are not shown at all at this scale.

Figure 12 shows a similar case where a road has been moved to not have conflicts with another road. Here we can also see an example where the simplify method is not sophisticated enough. The symbol size of the first object infringes on the area where the new object may be located to such an extent that the second point of the geometry is displaced irregularly.

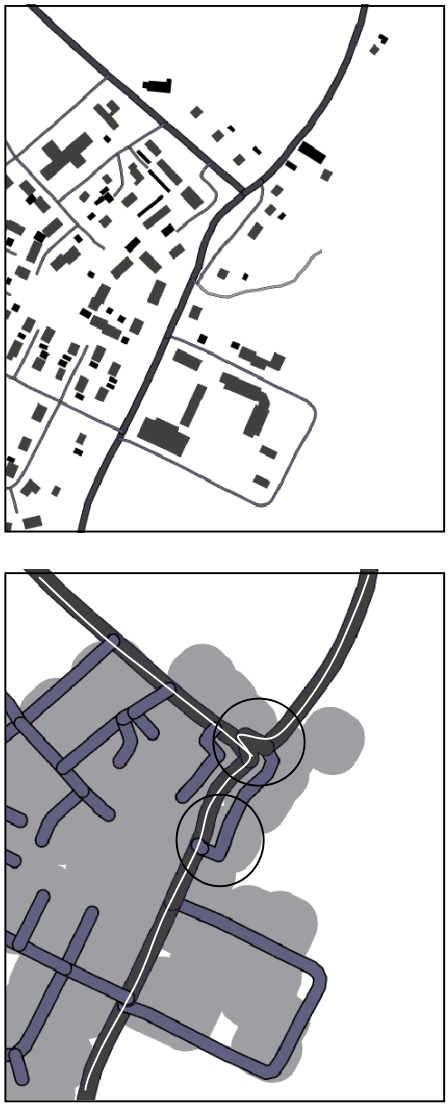

Figure 12 The top map shows three green level and the map below shows the small scale version of the blue level. The lower circle illustrates how the road has been moved to avoid conflicting symbols. The upper circle illustrates a how the second point of a road link has been moved irregularly due to a conflict with a symbol of the neighbouring road.

Built up area

In the test area the green level does not contain any built-up area objects and the large scale version of the blue level is designed in such a manner that no built-up areas are created. In the small scale version of the blue level however the building symbols has to have such a size that they can not represent densely populated areas. A built up area has to be created instead. Since there are no built up areas in the test data set the method is rather simple. Dwelling house objects are annexed recursively in the manner that is described above. The parameters for the annexation has been set so that the size of a dwelling house has to be larger than 90m^2 to be selected for the blue level. The distance between two dwelling houses to be annexed has to be shorter than 40m. When the annexation is finished we have to have a group of at least 20 dwelling houses to form a built-up area. The built-up area is formed by first creating a built-up area around each dwelling house using a buffer method. When buffers has been created for all selected dwelling houses, all buffers that are overlapping are merged into one built up area and the small buffer objects can be deleted. It would also be possible to delete the "islands" within the built up areas but this has not been implemented. Figure 12 shows how the built-up areas are shown in the small scale version of the blue level. Of course this is far from the quality of a topographic map.

Buildings not used for dwelling

This object class will be called house-other. If a house-other in the green level is suitable to form a house-other in the blue level depends on two requirements. The house-other has to be larger than a certain area, 600m^2 in the small scale version of the blue level and 200 m^2 in the large scale version. Furthermore the house other should not be located within a built up area in the blue level. If this is fulfilled an area where the object may be located in the blue level is computed using the buffer method. The area is then modified so that all objects that can have conflicts with the house-other are located outside the buffer. The geometry of the house-other in the green level is copied and the method tries to find a position for the geometry that is completely within the allowed area. However since a building might be surrounded by roads this could be impossible. The method tries to move the geometry in eight different directions (North, Northeast, West etc.) and tests if the geometry is inside the area. If none of the alternatives is accepted the geometry is created at its original position anyhow. The actual geometry of the house-other is not modified. Figure 13 shows an example of house-other objects. Some house-others has been moved while other stays put since they have no conflicts.

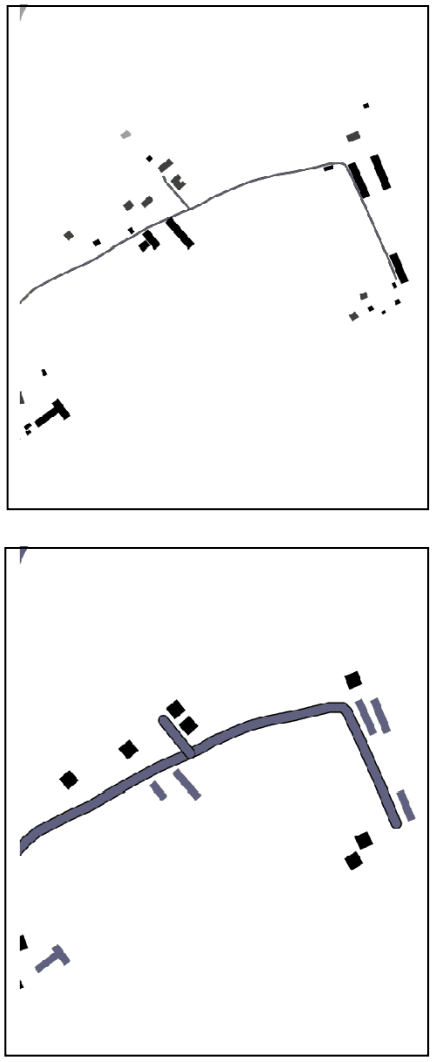

Figure 13 The top map shows the green level and the map below shows the large scale version of the blue level. The house-other objects that are close to the road have been moved to avoid conflicts.

Dwelling-houses

The creation of dwelling houses complies with what has been described in above. When the small scale version of the blue level is created the dwelling houses in the green level that have been used as seeds to create built-up areas are now marked and can not be used to create dwelling houses.

When we reach the simplification method we have a group of dwelling houses from the green level that shall be used to form a group of dwelling houses in the blue level and an area where the dwelling houses may be located. First we move the dwelling house in a similar manner as has been described for house-other above to try to get it within the accepted area. Then we check if there is a conflict with any dwelling house in the blue level that has already been created. If not, the dwelling house in the blue level is created.

Conclusion

The development of the model has to a large extent followed the steps suggested by Rumbaugh et al (1991). First a general sketch of the model is created which is gradually filled with more and more details. So far the model is not detailed enough to create maps with a reasonable quality. The simplify method which creates the geometry of a new object, for instance, simplifies the geometry of the seed object by picking out every second coordinate. The method also checks that every node in the new geometry is located within the area where it may be located and, if necessary, moves nodes to be within this area. Future work on this model is to make the model more detailed and use more appropriate generalisation algorithms.

The analysis of the green level often relies on recursive methods. The reason for this is that the green level should not contain any knowledge about how the blue level is generated. This requirement is necessary in an application where the blue level retrieves data from a database that belongs to a different organisation. We can not expect this organisation to modify their database to suit our application. Furthermore this organisation is most likely supplying data to several different applications and we can not burden the database with all kinds of application specific information. This implies that the constructor of an object class receives a seed object and can analyse its surroundings to find patterns of objects that are suitable to form new objects in the blue level. Since the surroundings of the seed object is unfamiliar recursive methods are very suitable for this analysis.

The main advantage with this approach to modelling seems to be the ability to structure the vast amounts of requirements and knowledge on how the generalisation shall be performed into manageable pieces using the tools of the UML.

References

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide.* Reading, Massachusetts: Addison-Wesley Longman.

Harrie, L. (1998) *Generalisation Methods for Propagating Updates between Cartographic Data Sets*. Licentiate Thesis. Lund institute of technology, Lund.

Petzold, I., Plümer, L., and Heber, M. (1999) Label placement for dynamically generated screen maps. *Proceedings of the 19th International Cartographic Conference,* Ottawa, Session 25 B.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorenson W. (1991) *Object oriented modelling and design* London: Prentice Hall.