Active Object and Agent Based Approaches to Automated Generalisation

Kelvin Haire
Laser-Scan Limited, Science Park, Milton Road, Cambridge CB4 0FY
Tel +44 (0)1223 420414, Fax +44 (0)1223 420044
E-mail: kelvinh@lsl.co.uk

Abstract

The ability to derive mapping products at several scales from a single database is highly desired by mapping agencies around the world. Key to the success of a system demonstrating this ability is an effective way to generalise automatically the features of the map. The ideal system would, amongst other characteristics, be able to generate an entire map series with minimal user interaction, coping with a large variation in the land types and uses being generalised.

In this paper three approaches to map generalisation are described, each based upon the use of methods on object classes within a database. The three approaches, display methods, process methods and active agents, can be considered to be of increasing complexity and flexibility. It is recognised that a fully automated system for map generalisation could make use of all three.

The application of the approaches to actual map production and delivery systems is also described, with particular emphasis being given to the use of agents by Kort and Matrikelstyrelsen (Denmark) in their production of 1:50000 scale maps from 1:10000 scale source data.

Keywords

Automated Generalisation, Agents, Object-Oriented, Display methods, Process Methods

Introduction

The Object-oriented approach

A logical way to represent spatial data is through the use of software objects. In an object-oriented (O-O) environment, distinct features within a database are stored as separate 'objects'. Each object has its own data associated with it. Each object also has access to a set of functions known as 'methods', common to all features of the same type or 'class'.

The processing of an object containing database would tend to involve querying or setting the attributes stored on the objects, or sending messages to the objects to trigger appropriate methods. For example, a user of such a spatial database might wish to find the name of the owner of a particular building (querying an attribute), and find the shortest distance from that building to another point on the map (triggering a method).

A fully object-oriented system should have the following characteristics:

- The data are encapsulated within objects, together with methods which the object can trigger in response to a message from elsewhere.
- All objects belong to particular 'classes', with each object being a specific example of its abstracted class. All object of the same class have access to the same methods, and contain the same attributes, though with different values stored for those attributes.
- Classes can be formed into an inheritance hierarchy. One class can inherit the properties of one or more 'superclasses'. For example, houses are types of buildings. The house class could inherit from the

building class, meaning that houses would inherit certain characteristics common to all buildings. Both methods and attributes can be inherited in this way.

- References can be set between particular objects, giving one object direct knowledge of one or more other relevant objects.
- Polymorphism means that the same message can be interpreted differently by members of different classes. For example, a request for the size of a church and the size of a road would invoke different responses.

The Laser-Scan Gothic O-O database is one such system with the characteristics described above. All examples described in this paper are based on the Gothic family of products (Figure 1). More information on this family can be found in reference [Hardy 1999].

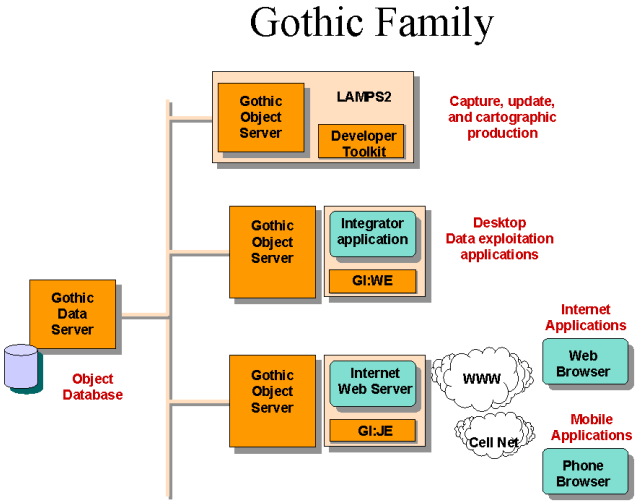

Figure 1: The Laser-Scan Gothic family of object-oriented applications.

Display methods

Of particular relevance to the tasks of map generalisation are the methods which exist on the individual objects within the database. These allow the objects to execute functions in response to messages sent to them by the system as a whole.

Display methods control they way an object is rendered following a request for that object to be displayed. The same data can be made suitable for display at several different scales through the use of display methods. For example:

- The width of a road symbol can be calculated from the scale.
- Objects can be displayed or not depending on the scale and theme of the map.
- An object's outline can be replaced by a symbol at smaller scales.
- Each object can have several outlines stored which are relevant to different scale bands.
- The positioning of text labels can be altered to suit the scale being used, with fewer labels being used at smaller scales and "halos" being used to enhance readability.

Figure 2 shows an example of the use of a display method within "CamMap.com" (http://www.cammap.com), an interactive web map of the Cambridge area. In the figure, the same area has been displayed at two different scales. The insets show the way the representation of a college building has been altered in moving to the smaller scale, with an appropriately pre-generalised outline being displayed.

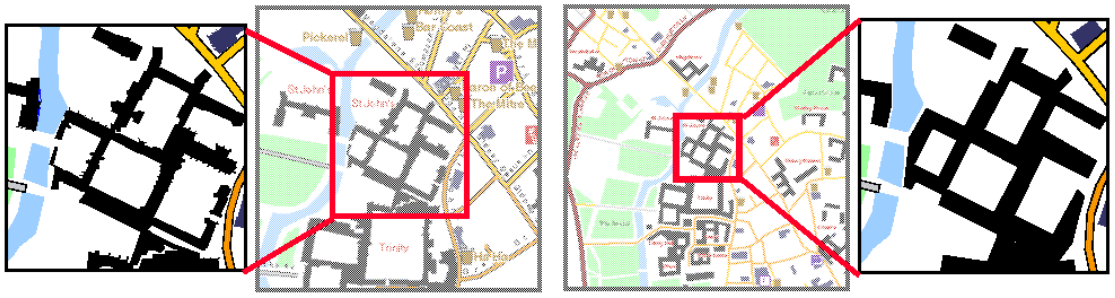

Figure 2: The use of display methods to represent the same feature at two different scales. At the smaller scale a pre-generalised alternative outline is used.

Process methods

Display methods allow the results of previously performed generalisation operations to be displayed, and allow some very basic generalisation operations to be performed. For more complicated generalisation operations, other methods associated with the objects can be triggered, with the results being stored in the database.

Many generalisation operations can be completed by "batch processing" of the objects within the database. Within Gothic, the mechanism used to perform this batch processing uses methods known as process methods. The use of process methods has two components:

1. The methods themselves, which exist on base classes and are inherited by the real-world classes which are to use them.
2. A process sequencer which sends a message to each object in turn and handles any return messages such as the notification of failure.

By placing the process methods onto base classes rather than the class being processed, the same process method can be applied to many different classes which may exist within a map. Figure 3 shows the result of a process method which aggregates area objects which are closer together than a user-defined distance. In the figure, the areas being aggregated represent the outlines of built-up areas, but the same operation could easily be applied to woodland areas for example.

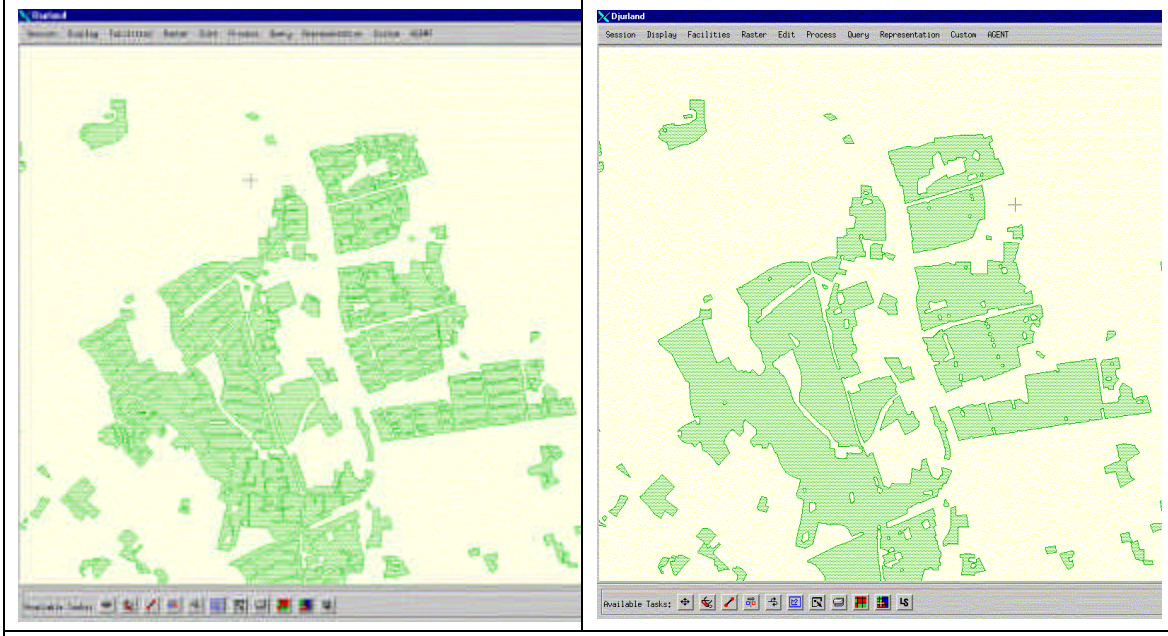

Figure 3: An example of a process method used to aggregate area objects closer together than a threshold distance. (aggregated on the right).

Process methods can be used in a number of ways during the complete map generalisation process. For example they can be used to:

- Remove unnecessary objects from the database (or label them as "deleted") at a particular scale based upon their size.
- Reclassify objects in preparation for a later processing step.
- Construct supporting data structures such topological faces, links and nodes in preparation for a later processing operation.
- Identify and label particular features within the data, such as motorway interchanges in a road network or the outlines of cities.
- Perform generalisation operations on an object by object basis.

Complex algorithms can be implemented as process methods, in effect giving the database itself advanced generalisation capabilities.

The LAMPS2 Generaliser

LAMPS2, Laser-Scan's main desktop application based upon Gothic, includes a generalisation module. This module contains a number of process methods specifically designed for tackling problems applicable to map generalisation, together with a user interface for setting the parameters to be used.

The Generaliser contains the following types of operations:

- **Clustering** to create an outline around groups of objects which are closer together than a specified distance.
- **Simplification** to reduce the amount of co-ordinate data used to represent an object.
- **Typification** for selectively removing objects from a group whilst maintaining the overall pattern present.
- **Aggregation** for combining objects which are within a specified tolerance of each other.
- **Collapsing** to simplify the representation of objects or groups of objects by altering the basic types. E.g. areas are collapsed to lines or points, parallel lines are collapsed to a single line.
- **Exaggeration** to enhance certain characteristics of an objects appearance.
- **Refinement** to improve the visual appearance of objects.
- **Displacement** to remove conflicts between closely neighbouring objects.

Figure 4 shows an example of the user interface used for setting the parameters for displacement generalisation. In the figure, the green contours of two road objects have been displaced away from their original (red) positions adjacent to a central road where the original separations are less than a specified distance.

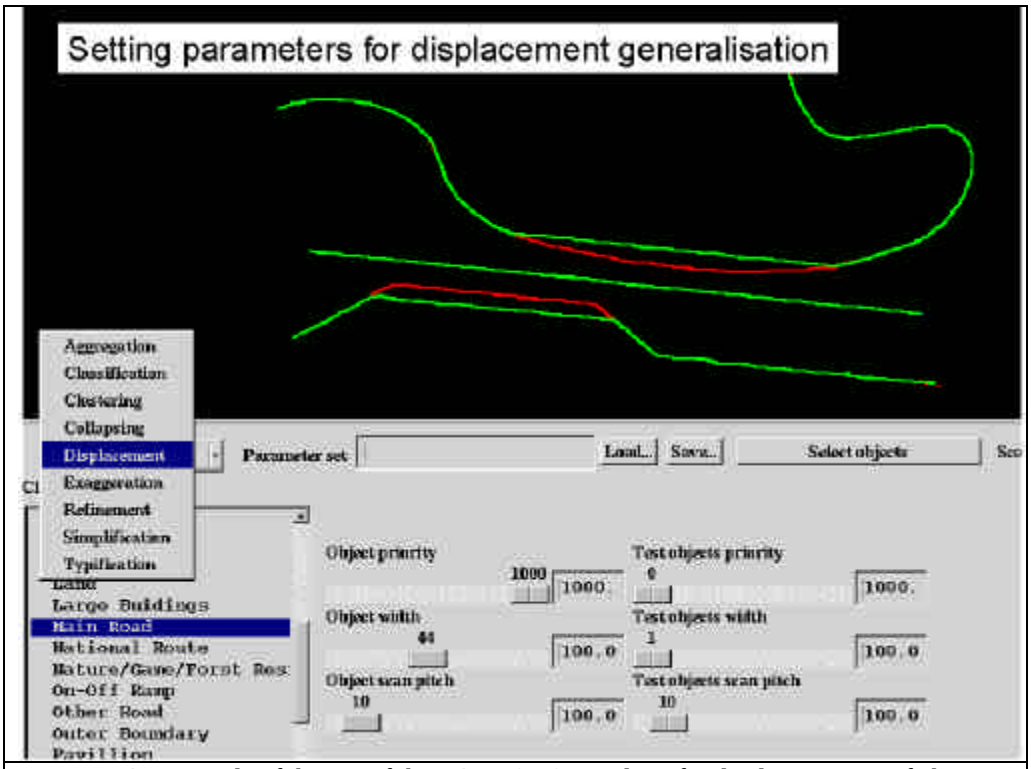

Figure 4: An example of the use of the LAMPS2 Generaliser for displacing parts of objects which are too close to one another.

AGENT based generalisation

Process methods offer a powerful way for algorithms to be run on objects within the database. Complex algorithms can be devised to overcome certain types of generalisation problem. There is a significant limitation with this approach which means that process methods, or any other type of batch processing will never be able to generalise an entire data set automatically. The main limitation is that a batch procedure will blindly apply the same set of algorithms to all similar features across the whole map. With innumerable variations in the data possible, it seems extremely unlikely that the same algorithms will work across the entire data set.

In many cases the generalisation operations performed on individual features will result in conflicts between features. For example, a building might be enlarged to make it legible at the final map scale, but might then overlap a road. These conflicts must be resolved in such a way that the aims of the map are maintained. In the example just given, either the building or the road could be displaced slightly to remove the conflict. If there are other objects nearby, or if the positional accuracy is of paramount importance then a different strategy must be employed for resolving the conflict. The possible conflicts which could be encountered are of course innumerable and each would require a slightly different treatment to resolve it.

If an automated generalisation system is to emulate a cartographer it needs to have a degree of intelligence to decide upon the most appropriate course of action to take with each object or group of objects being generalised, and to detect and overcome the new conflicts introduced as side-effects of other generalisation processes.

The Automated Generalisation New Technology or AGENT project was designed to develop new techniques for generalising features within a geographic data set which would address the problems described in the preceding paragraphs. This was a three year collaborative project sponsored by the European Commission (ESPRIT 24939) involving IGN France, University of Zurich, University of Edinburgh, INPG Grenoble and Laser-Scan Ltd.

The strategy

The AGENT system incorporates a strategy which aims to resolve conflicts not by describing in great detail how certain conflicts should be resolved, but by describing the desired final characteristics of the feature. For example, the system might incorporate the desired outcome that no building should be smaller than a certain size, that it should not be closer than a certain distance to another building and that it must not be moved by more than a certain distance from its original starting position. The outcome of any generalisation operation or set of operations is compared against this set of guidelines. This comparison is used to decide whether further operations are required, or whether the result should be discarded and a different operation applied. In this way, individual map features can be generalised in a way which is sensitive to their particular situation; with similar features potentially having quite different operations applied to them.

This strategy has been implemented using a system of "Agents". An agent in this context is an active database object, capable of taking decisions and of communicating with other agents. In this sense, agents are an extension of the more familiar database objects, which have associated properties and methods. Any object within a map such as a building, a forest or a road has the potential to become an agent. It would then be able to generalise itself based on its current situation according to its particular set of guidelines using the strategy outlined in the previous paragraph.

Many generalisation problems can be solved by simply empowering the individual features as agents, allowing them to try to resolve their particular conflicts. However, this does not address the fact that a map contains information at many different scales, with many individual features being grouped together to form a larger compound feature. For example, a collection of buildings and roads can form a town; individual roads connect together to form a road network. Any generalisation process must consider these compound features which are in many respects more important than the individual features. For example, as the scale is decreased, built-up areas within towns could be represented by shaded polygons at intermediate scales, and aggregated into a single point at very small scales. Many minor roads would be removed from the road network, but it is important that no part of the road network is disconnected by the operation. Individual features which have become agents cannot cope with these "higher level" considerations. This naturally leads to the concept of a hierarchy of agents, with higher level agents controlling groups of sub-agents and hence compound features.

The terminology used to describe this hierarchy is as follows:

- **Micro-agents** control individual atomic features. They have no overview of how they fit into the overall map, but have guidelines for generalising themselves.
- **Meso-agents** control groups of lower level agents. They are able to consider the interrelations between the sub-agents they are managing. There can be several levels within the meso-agent hierarchy.

One example of an agent hierarchy can be found by considering a city. Micro-agents will control each building, road segment etc. Buildings will be collected into urban-blocks and industrial areas which will be controlled by meso-agents. The urban and industrial blocks will form districts which will be controlled by a meso-agent. Finally, the districts will be collected together to form a city which will have its own meso-agent. Within an urban block, the meso-agent would be able to delete, displace or aggregate buildings. The district meso-agent would be able to aggregate or remove individual urban blocks. The city meso-agent

would be able to aggregate or alter individual districts. These hierarchies can be extended as required by the particular map being used. The situation of a two-level hierarchy of meso- and micro-agents is illustrated in figure 5.

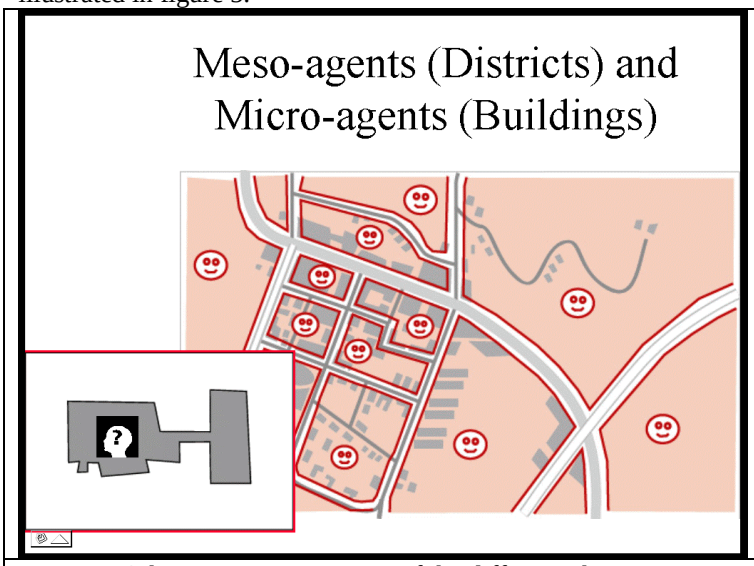

Figure 5: Schematic representation of the difference between meso- and micro-agents as applied to the generalisation of urban areas. Each district is controlled by a meso-agent. Within each district are several buildings which act as micro-agents.

Each agent within the system employs the same general strategy: generalisation operations are attempted on the entity under their control, with the outcome being compared with a set of guidelines and used to decide upon the next stage. The strategy is considered to be very versatile as it does not require the generalisation operations to be pre-defined for any type of feature.

System overview: Constraints, Measures, Plans and the Life-cycle

Apart from the agents themselves, the AGENT system has a number of essential components which allow it to function.

- **Constraints** are the mechanism by which the generalisation process is guided. Each agent, micro- or meso-, has a set of constraints which represent the guidelines for generalisation. For example, a constraint on a building might say that it must only have square corners or that it must be larger than a certain size; a constraint on an urban area might say that the buildings it controls must not overlap. The agent can assess the degree to which each of its constraints are violated.
- **Measures** are the functions used by constraints to assess their degree of violation. A road agent might have a measure to find out its level of self intersection; a building agent might have a measure to calculate its size.
- **Plans** are the algorithms and their associated parameters proposed to overcome violated constraints. An agent with several constraints violated will have several different plans proposed. The order in which these plans are applied is dependent upon the degree of violation of the constraint proposing the plan, and the importance given to resolving each particular constraint.

Each agent has a special method called the **life-cycle** which allows it to intelligently process the information given to it by its constraints and to apply the plans proposed in an appropriate way. A simplified flow diagram for the life cycle is shown in figure 6. As the life cycle is followed, a “research tree” is built up. This is also shown in figure 6.

An agent begins its life-cycle by *characterising* its situation which involves each of the constraints assessing their degree of violation and the agent aggregating this information to give it an overall “happiness”. The initial “state” being characterised corresponds to the highest node in the research tree

These constraints *propose* sets of plans which are ranked by the agent according to the importance of the constraint proposing them, the degree of violation of that constraint and the weighting given to the plan by the constraint itself. The agent then *acts the best plan* from the list and reassesses its situation. This takes the system down to the next node in the research tree corresponding to a new state. If the plan enacted is considered to have improved the situation, the new state is kept and re-characterised, with the life cycle re-starting from the new state. If the new state is considered to be worse than the old state, the system is backtracked to the previous state (corresponds to climbing back up to the previous node in the research tree) and the next plan in the list is attempted. In the event that there are no more plans at a particular node in the tree, the system climbs back up to the previous node. The process continues until all plans at all states have been attempted. At the end of this process the best state seen throughout the research process is restored. It should be noted that if at any point the state attained is considered to be "perfect" then the life cycle is ended and the perfect state is kept without researching any further nodes of the tree.

This approach to generalisation means that the exact set of algorithms applied to similar features can be different, and well suited to the particular circumstances of each feature. It also means that the system will take care of any side effects of particular operations, provided there is a sufficiently broad set of algorithms available.

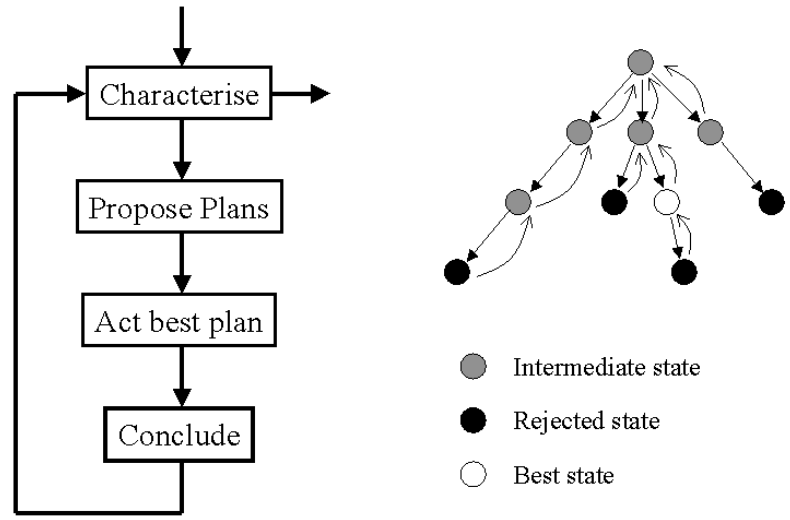

Figure 6: A simplified flow diagram of the agent life-cycle (left) together with a schematic representation of the research tree which might be constructed as an agent enacts its life cycle (right).

Figures 7 and 8 show examples of the algorithms which are available at the micro and meso levels respectively for the generalisation of an urban area.

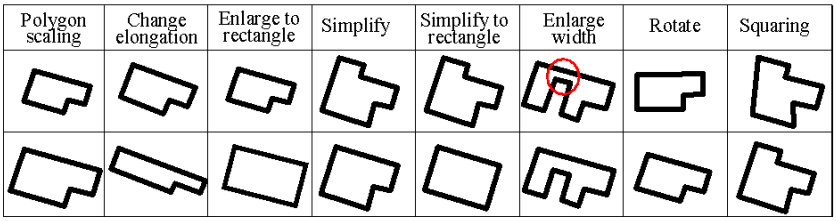

Figure 7:Examples of algorithms which can be applied to the building micro-agents.

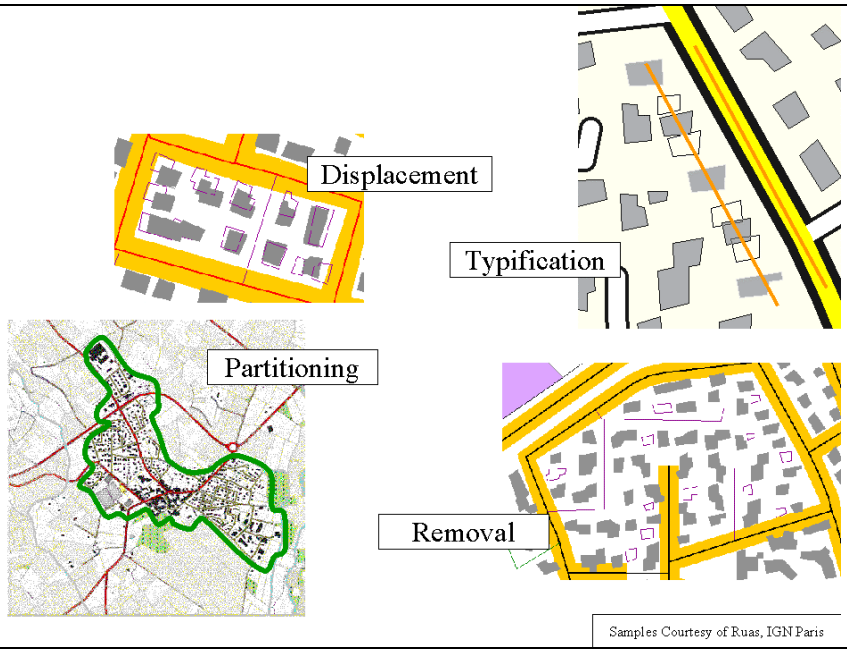

Figure 8: Examples of algorithms which can be applied to urban area meso-agents.

Building Generalisation for KMS

Laser-Scan has been working with Kort and Matrikelstyrelsen (KMS), the Danish national mapping agency, to produce topographic maps at a scale of 1:50000 from a source data set suitable for representation at a scale of 1:10000. The AGENT system is being used to automate all aspects of the building generalisation process.

Within the KMS data, the buildings which are to remain at the final scale can be classified in three ways:

- Buildings in urban-areas including buildings in industrial areas and some other areas classified as built-up.
- Farm buildings.
- Buildings outside urban areas which do not belong to farms.

All other buildings are removed using a process method.

Generalisation of Urban-Blocks

KMS have a number of guidelines covering the generalisation of buildings within urban blocks. These guidelines have been communicated to the AGENT system through the use of constraints on micro and meso-agents. Furthermore, a suitable meso-agent hierarchy has been constructed to process effectively the urban block.

One particular requirement is that distinctive visual patterns formed by groups of buildings should persist after the generalisation process. Without this requirement it is likely that these visual patterns would be destroyed as individual buildings are displaced, deleted, rotated and merged with other buildings to overcome other conflicts. To tackle this requirement, a high level meso-agent, assigned to the entire urban block identifies clusters or groups of buildings which have characteristic shapes. These could be made up of a small number of buildings which are particularly close to each other compared with other buildings, or groups of buildings which form geometric shapes such as lines or rings. Each of these groups has a lower level meso-agent assigned to it to ensure that the characteristic shape of the group are maintained. Figure 9 shows two examples of such groupings within ungeneralised urban blocks.

The meso-agents managing the groups of buildings are responsible for ensuring that the buildings within them are generalised and that the characteristic shape is maintained. The meso-agent controlling the entire urban block is responsible for handling conflicts between the building groups and for controlling any other buildings not in a group (effectively in a group of one building). Where there is insufficient room to accommodate all of the groups within the urban-block, the top level meso-agent can request that the groups make them selves smaller. They can do this by deleting buildings within the group, by pushing members of the group closer together or by aggregating buildings and eroding the resultant combined building.
The set of generalisation requirements relevant to urban blocks and their AGENT based solutions are described in table 1.

Figure 9: Urban areas in which some visual structures have been formed by groups of buildings. (Courtesy KMS Denmark).

Generalisation requirement	**Solution**
Building smaller than a given size should be deleted.	Individual buildings are controlled by micro-agents. The micro-agents have constraints and measures associated with them which detect their sizes and change their symbols accordingly. Constraints which guide the squareness, granularity and concavity of the buildings take effect if the building is to be simplified.
Buildings within specified size bands should be scaled up or represented using pre-defined rectangular outlines.	
Buildings larger than a given size should be simplified	
Buildings should not overlap each other.	Meso-agents controlling groups of buildings detect overlaps between buildings and displace, rotate, delete or merge the buildings as appropriate.
Buildings should not overlap other objects such as roads.	Meso-agents controlling groups of buildings detect overlaps between buildings and other objects and displace, rotate or delete the buildings as appropriate.
Visibly distinctive shapes formed by ungeneralised buildings should remain after generalisation.	Two levels of meso-agents exist within the urban blocks. One controls the groups of buildings, ensuring that any displacement operations conserve the shape of the group. The other controls the entire urban block, co-ordinating the interactions between the groups of buildings and any other buildings not belonging to a group.

Table 1

Generalisation of Farms

Farms can be considered to be significant groups of buildings outside urban areas. Since farms constitute landmarks within a landscape, their buildings tend to persist in the generalised data where equivalently sized buildings in build-up areas would not. Farm generalisation tends to be accompanied by large increases in size. One important requirement is that the overall shape created by the farm buildings is maintained. This is achieved in a similar way to the generalisation of groups within the urban block – any displacements of buildings necessary to overcome conflicts must satisfy the constraint on the farm meso-agent that their relative positions are maintained. Where external objects such as roads or other farms cause limitations of space, the buildings within the farm are aggregated, enabling the buildings to take up less space while remaining legible.

Figure 10 shows an example of a groups of farm buildings which have been generalised. Note that the farm buildings have been enlarged and simplified and that conflicts between the farm buildings and the nearby road have been removed. The approximate relative positions of the buildings have remained unchanged however.

Figure 10: Generalisation of a group of farm buildings. The individual buildings have been simplified, enlarged and displaced, but the approximate relative positions have remained unchanged. (Courtesy KMS Denmark).

Conclusion

Generalisation can be achieved at several levels through the use of object-oriented database technology.

- Active representation enables simple generalisation operations to be performed on a database as it is being interrogated, extending the scope for using that particular database.
- Process methods enable algorithms to be applied to all objects of a similar type within a database. Such algorithms are suitable for well defined generalisation tasks such as aggregation of large areas, or pre-processing such as the identification of certain features.
- Giving the objects within the database intelligence by turning them into agents enables a wider ranges of generalisation problems to be tackled. In particular, the use of agents offers a strategy to the problems that similar features may require different treatments and that one operation may have side effects which cannot necessarily be predicted in advance.

Hierarchies of agents have been shown to be suitable for generalising complex features such as farms and urban blocks. These are currently being developed as part of a commercial map production environment.

References

Hardy P. G., 1999, "Active Object Techniques for Production of Multiple Map and Geodata Products from a Spatial Database", ICA/ACI Conference Proceedings, August 1999, Ottawa, Canada, or online at http://www.Laser-Scan.com/papers/ica99gwg/pghica99gwg.htm