CLARITY - A NEW ENVIRONMENT FOR GENERALISATION USING AGENTS, JAVA, XML AND TOPOLOGY

Paul Hardy, Melanie Hayles, & Patrick Revell,
Laser-Scan Ltd., Cambridge, UK
paul.hardy@laser-scan.com

ICA Generalisation Workshop, Paris, April 2003

KEYWORDS: Generalisation, Object, Agent, Topology, Java, XML, Clarity

ABSTRACT

Over the past few years, many national and regional mapping agencies have built a basic scale dataset covering their territory, often now referred to as 'framework' data. At the same time, there have been increasing requirements for rapid creation of multiple products at different scales and specifications, combined with great pressures to cut costs and reduce staffing. These conflicting requirements have given a real incentive to use automated generalisation. However, production-quality, intelligent tools for contextual generalisation have been scarce or unavailable.

The advantages of object data modelling, active object toolkits and software agents have been proven by research projects, and by successful deployments of prototypes in production flowlines at national mapping agencies. Work is now underway, guided by the MAGNET consortium to develop a new production-ready object generalisation environment (***Clarity***). This is developed from the AGENT prototype, but building on experience gained, and adopting standard technologies such as XML and Java.

This paper overviews the current and future developments and product strategy of a leading automated solution for cost-effective object and agent generalisation. It particularly describes the new architecture with its use of XML and Java, and the importance of topology.

1 INTRODUCTION

1.1 Requirement for Generalisation

Over the past few years, many national and regional mapping agencies have built a basic scale dataset covering their territory. The scale of such datasets varies from country to country. Britain is unusual in having a consistent dataset at scales of 1:1250 (urban) and 1:2500 (rural). Countries such as Denmark, Netherlands, France have 1:10,000 as their basic scale dataset. In other countries like Germany, the basic scale landscape model (Base DLM) is at about 1:10,000, but is produced separately by the regional mapping agencies.

This basic data was usually captured to automate the production of basic scale paper mapping. Other uses such as analysis in customers' GIS systems were secondary. It has emerged that such GIS applications require cleaner and more structured data than that for cartography, necessitating data re-engineering such as that done by the Ordnance Survey to produce OS MasterMap [Hardy 2001]. Using the basic scale data as a starting point for automated derivation of other scales of mapping has always been on the agenda for these agencies, but has been pushed back while the urgent task of getting complete basic scale coverage was completed, and an updating service established.

While this has been going on, there have been increasing requirements for rapid creation of multiple products at different scales and specifications, combined with great pressures on agencies to cut costs and reduce staffing. An example is KMS Denmark, which has had to produce up-to-date 1:50K mapping for its military, at the same time as shrinking the organisation. These conflicting requirements have given a real incentive to use automated generalisation.

1.2 The MAGNET project

Mapping Agencies Generalisation NETwork (MAGNET) is a project to consolidate and extend generalisation capabilities in line with the business requirements of its members. The MAGNET group comprises organisations (principally European national and regional mapping agencies) interested in ensuring suitable generalisation capabilities are produced. They do this by influencing and contributing to the development of the new generalisation product from Laser-Scan, called ***Clarity***.

2 ACTIVE OBJECTS AND AGENTS

2.1 Scope of Generalisation

The real advances in automated generalisation came with the realisation that good generalisation requires:

- Contextual analysis - you can't generalise one map feature at a time in isolation. You have to consider groups of objects as a whole.
- Adaptive processing - you can't apply a single algorithm to all features (even of a single class). You have to choose appropriate algorithms according to the circumstances of that feature.
- Backtracking - you can't get it right first time every time. You have to be prepared to assess whether an operation has made things better or not, and be prepared to undo it and try something else.

To implement these requirements necessitates a holistic rather than procedural generalisation environment [Hardy 2000]. Also, generalisation is not just applicable to visual cartography (paper maps, or web or mobile screen). Increasingly, mapping agencies are being driven to derive lower resolution GIS data products from their framework data. An example is in Germany, where the German Länder have captured a Base DLM in the ATKIS data model, but find that many GIS users find this too large data volume. Hence the need to derive a 1:50K landscape model by 'model generalisation', which needs a subset of the techniques needed for cartographic generalisation (no need for displacement or representation).

Beyond the visual map and the GIS dataset, generalisation will become increasingly important to the new generation of 'Location-Based Services' (LBS) [Hardy & Haire 2000]. These services, provided to the new media like mobile phones, PDAs and tablets, will need to extract and present only the information that is relevant at the time to a mobile customer.

2.2 Features, Objects and Agents

Early attempts at automatic generalisation involved a GIS or digital mapping software application applying geometric algorithms, one at a time, to simplify, displace, exaggerate, aggregate, collapse or otherwise modify an individual map feature. The limitation of this approach is that the algorithms operate in isolation - a shortcoming that entailed heavy manual intervention by expensive cartographers to complete the process.

A major step forward was the introduction of object-oriented data models and associated object-oriented spatial toolkits. This enabled algorithms to operate in the context of the feature, so that it could use information about its neighbourhood to modify the effect of the algorithm.

The third generation of generalisation involves software agents [Lamy et al 1999]. In an agent-based system, the individual map features are activated as self-aware software entities, which strive to improve their 'happiness'. Instead of trying to specify which algorithms should be applied to which features; the user defines constraints and associated measures, and lets the agent decide for itself.

2.3 Agents and the AGENT project

The AGENT (Automated GEneralisation New Technology) research project [Ruas 2000] was initiated as an EC funded project (Esprit 24939) in the late 1990s. Laser-Scan joined the multi-national consortium as the software supplier. Other members included IGN (the French national mapping agency in Paris) in the lead role, and three universities: Zurich and Edinburgh (for their expertise in geography and cartography), and Grenoble (for its work in artificial intelligence). Three years of intensive research and development resulted in a prototype agent-based generalisation system, based on Laser-Scan's Gothic LAMPS2 object-oriented software.

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 that is sensitive to their particular situation, with similar features potentially having quite different operations applied to them.

The AGENT prototype was made available in the commercial LAMPS2 Generaliser product and has been extended for production use at several NMAs including KMS Denmark [Sheehan 2001] and IGN France.

2.4 Agent lifecycle

During the AGENT project and then subsequently during the deployment of the prototype at NMAs, it became clear that the agent lifecycle in the prototype had limitations. Consequently, as part of the current work that is creating ***Clarity***, the agent core is being redesigned and re-implemented using rigorous software engineering. The new lifecycle has the following properties:

1) it handles the life of an agent, from activation to passivity
2) it invokes constraints (possibly with associated measures), plans and algorithms as needed
3) it is implemented using object methods (behaviours)
4) it is highly customisable by the overriding of methods
5) it implements 'hill-climbing' algorithms to explore possible states from execution of plans, and choose the best
6) it can preserve history of state for subsequent analysis of how and why
7) it has breakpoints for debugging and investigation

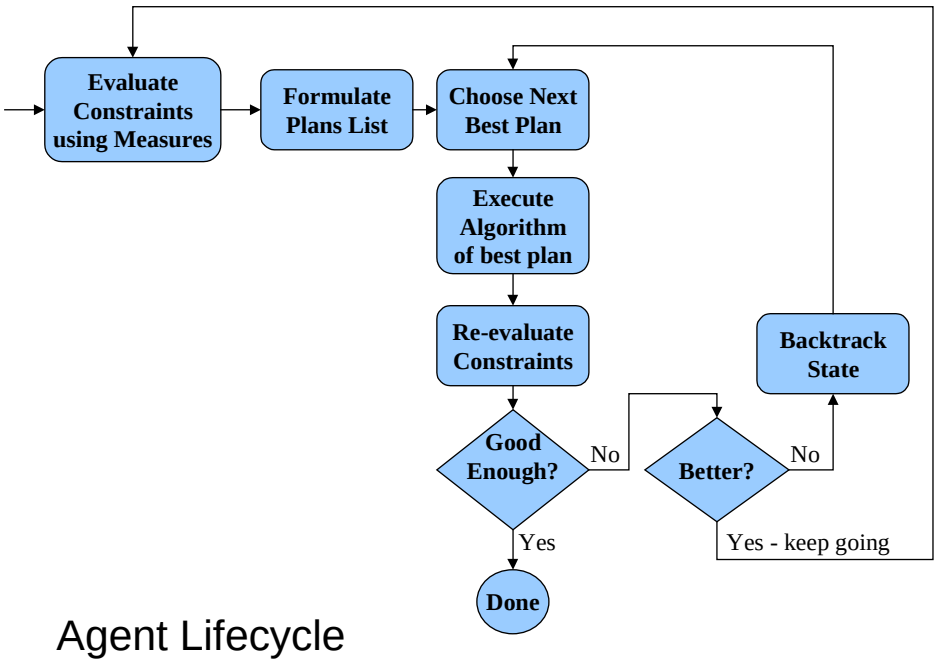


Fig. 1 - Simplified Agent Lifecycle

Note that in the Agent prototype, measures were always implemented as independent method behaviours. In Clarity, the same can be done, but is not compulsory. Instead, the equivalent of a measure can be calculated within the constraint, using functions from the underlying spatial toolkit as needed.

3 THE PRODUCTION FLOWLINE ENVIRONMENT

3.1 Interoperability and Spatial Warehouses

A global transition is under way in that location data (including mapping) is becoming centralised, and being handled by mainstream information technology. Increasingly, spatial data is being held in large databases (typically Oracle 9i), which act as a central warehouse to the enterprise. However, deriving multiple products at different scales and specifications by generalisation has intensive requirements for spatial access to data. These requirements put great demands on the pure relational model, which although good for generic queries and for short transactions, is not optimised for the local lookups and reactive consequences needed for good generalisation.

Therefore the preferred flowline approach is to extract from the warehouse the subset of data needed for a particular product, and load it directly into an object-oriented subsystem which is optimised for localised access and rapid modification. At the end of the generalisation processing, the results are transferred back to the warehouse.

3.2 Object Data Models and Data Re-Engineering

One point that is not always realised, is that to do good generalisation requires clean data in a good data model, with the following properties:

1) Features with different characteristics must be in separate feature classes. It's no use just having a class 'roads', if you want motorways to behave very differently to urban streets during generalisation.
2) Feature coding must be consistent. If a line between a road and a house is sometimes in class 'fence or wall', sometimes in class 'building', and sometimes in class 'road edge', then good generalisation will be difficult.
3) Data must be geometrically clean. Generalisation is hard enough, without having to guess whether two lines should join or not, or where an area with two seed points should really be split.
4) Data must be continuous. Trying to generalise data a sheet or tile at a time and then join it afterwards introduces immense problems.
5) Structure must be explicit stored, for fast traversal. Generalisation requires repeated analysis of feature relationships, particularly neighbourhood relationships. Such analysis can be done using repeated spatial index searches followed by repeated geometric analysis, but the time taken is likely to be unacceptable. Much better is to build explicit structure before generalisation.

The above properties are needed for generalisation, but not necessarily all needed for long-term storage, so it is usual to precede the generalisation processing stage with a 'data re-engineering' stage to get the data cleaned, structured, and into a suitable object data model [Hardy 2001].

3.3 Generalisation flowlines

The functional requirements for a generalisation flowline depends on several factors including, the scale of the target data, the classification of the data, and structure of data, e.g. whether it is polygonised or not.

Each flowline has to be built up independently using appropriate generalisation algorithms and processes. ***Clarity*** is designed so that algorithms and processes along with their associated parameters, developed as part of one flowline, can be easily incorporated into other flowlines. This is achieved largely through the use of XML.

3.4 Object processing

In object-oriented generalisation, the map features themselves are objects on which generalisation methods and properties have been defined. This means that the knowledge is embedded in the data model and not within the application. The application merely provides a framework for invoking and sequencing the generalisation processes [Ormsby and Mackaness 1999].

The Gothic toolkit includes powerful and versatile mechanisms for object processing. This is being extended for ***Clarity*** through the use of Java and XML-based tools to simplify the definition of such processes.

In Figure 2, the user defines the aims of the generalisation task, using Java forms to set up a Map Specification, plus parameters for processes, and process sequences. He or she then instigates the process sequence. This can then invoke process methods to call algorithms directly on objects, or it can activate agents via the scheduler, which then evaluate constraints (using measures) in order to plan which algorithms to call.

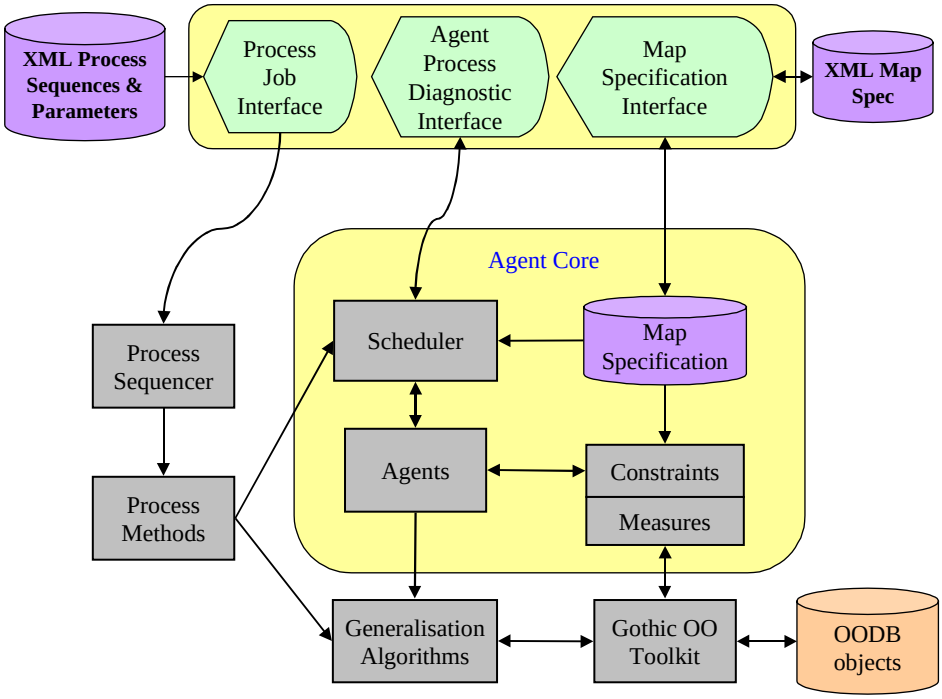


Fig. 2 - Flows of command and data during object processing

3.5 Client-Server architecture

Clarity is designed using a client-server architecture (see Figure 3), using interoperability standard interfaces (Java JNI) between the client and server parts.

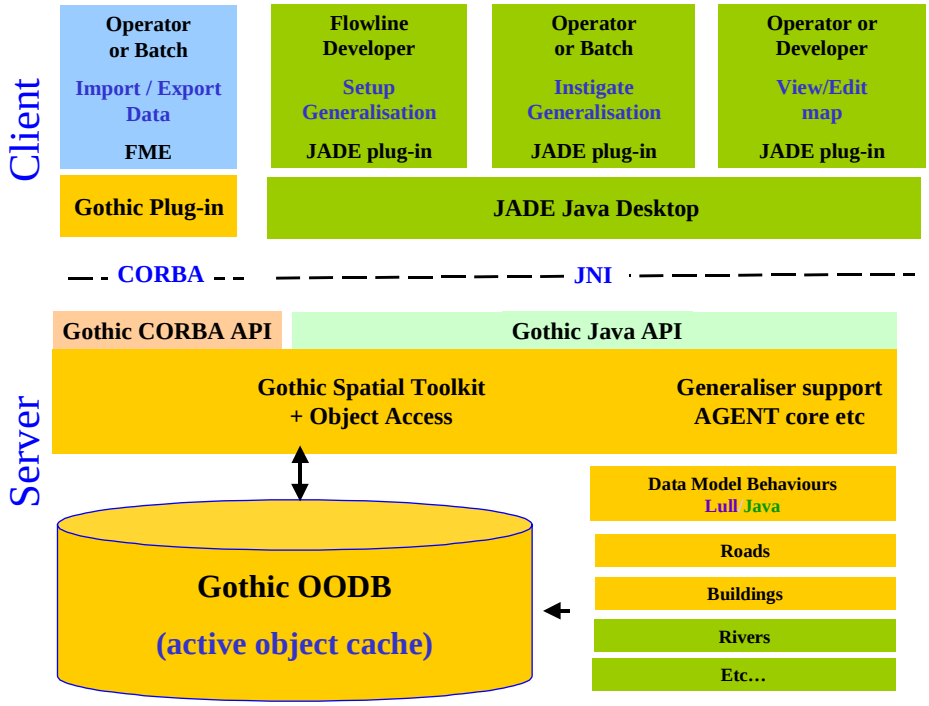


*Fig. 3 - **Clarity** Components*

The interactive clients all make use of the Gothic JADE Java desktop application, and are described further in section 5.3 onwards.

FME (Feature Manipulation Engine) is used to transfer data from external sources, notably from Oracle directly into the Gothic object database, which is being used as a persistent object cache. FME is similarly used after generalisation to put the derived data back into the warehouse.

3.6 Incremental Generalisation

Whenever the base dataset is updated, there is the possibility that the derived generalised dataset is now out of date. Note that this is not necessarily the case, as the changes at detailed scale may be irrelevant at reduced scale. However once significant change has accumulated, update information needs to be passed on to the derived generalised dataset.

While it would be possible to re-generalise the complete dataset, this is not a desirable solution for a number of reasons:

- It is quite time consuming.
- Manual corrections would have to be re-done, and this could lead to different results, as well as additional expense.

This situation can be avoided by using special incremental generalisation processes. The following illustration (Figure 4) shows the data flows in such an incremental generalisation system, that currently being developed by Laser-Scan for a consortium of the German Länder to generalise ATKIS Base DLM data to a 1:50K landscape model.

The incremental approach is made possible by the way that parentage information is determined at the time of the original generalisation, and stored on the derived features. This means that when changes happen to the base data, the effects of the change on the derived data can be quickly determined, and only the affected objects re-generalised.

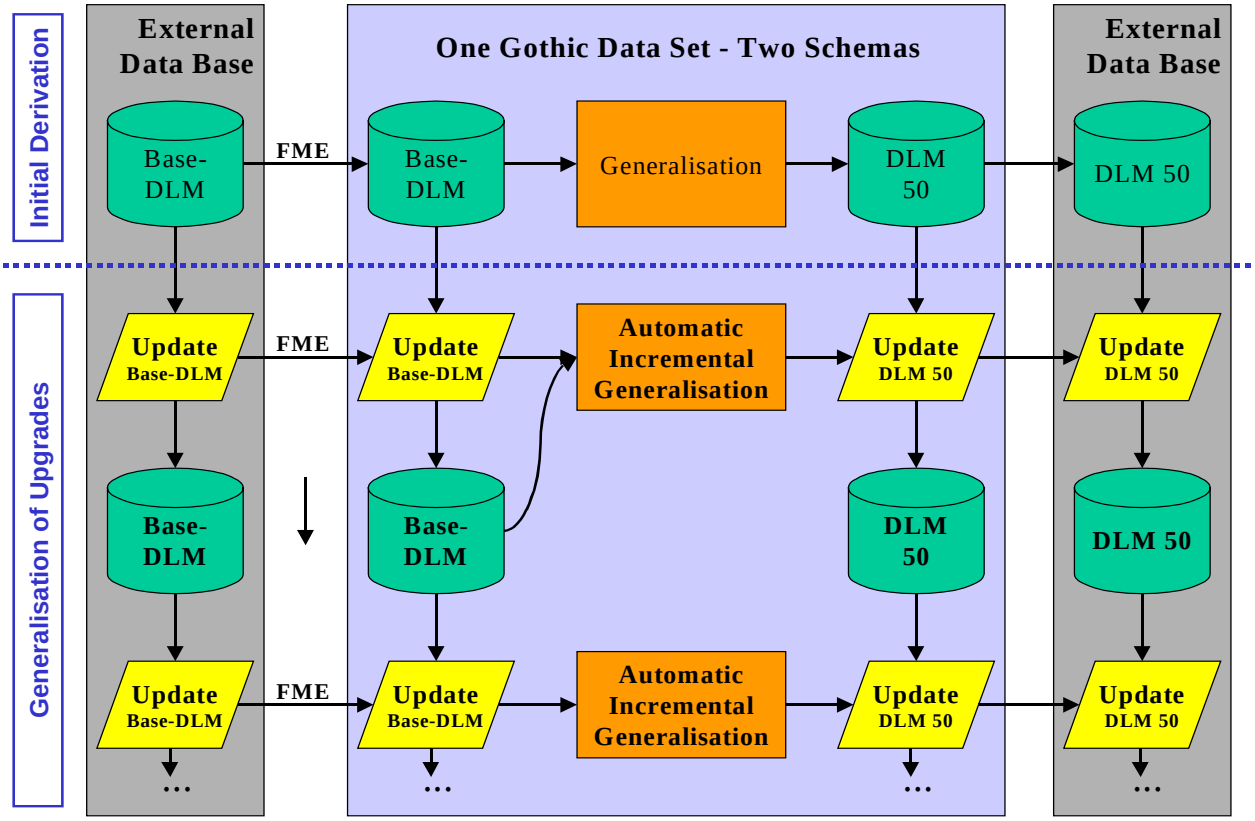


Fig. 4 - Incremental Generalisation

4 TOPOLOGY

Topology is the mathematical concept of spatial structure, sometimes defined as "Characteristics of geometry that do not change when the coordinate space is deformed". It expresses explicit geometric relationships, such as "connects to, touches, adjacent to, or within". A topological model consists of topological primitives (nodes, links, and faces) with references defining spatial relationships between them. References also specify how real-world features are constructed from these primitives. Laser-Scan's Gothic, which underpins ***Clarity***, has much better topology support than other GIS systems, as it implements a stored topology model covering a substantial subset of the ISO 19107 Spatial Schema standard, and includes support for dynamic update of the topological model in response to change to real-world features.

Topological relationships are present in most GIS data, but are often implicit. For good generalisation these need to be made explicit:

- Shared edges between land polygons
- Junctions between streets in the road network
- Colinearity of administrative boundaries with roads and streams
- Adjacency of buildings to roads

A good study of the use and management of topology is "the Balance between Geometry and Topology" [van Oosterom et al 2002]. An example of Dutch data (courtesy Topografische Dienst) is displayed below (Fig 5), showing the shared edges between polygons and the node points where edges meet.

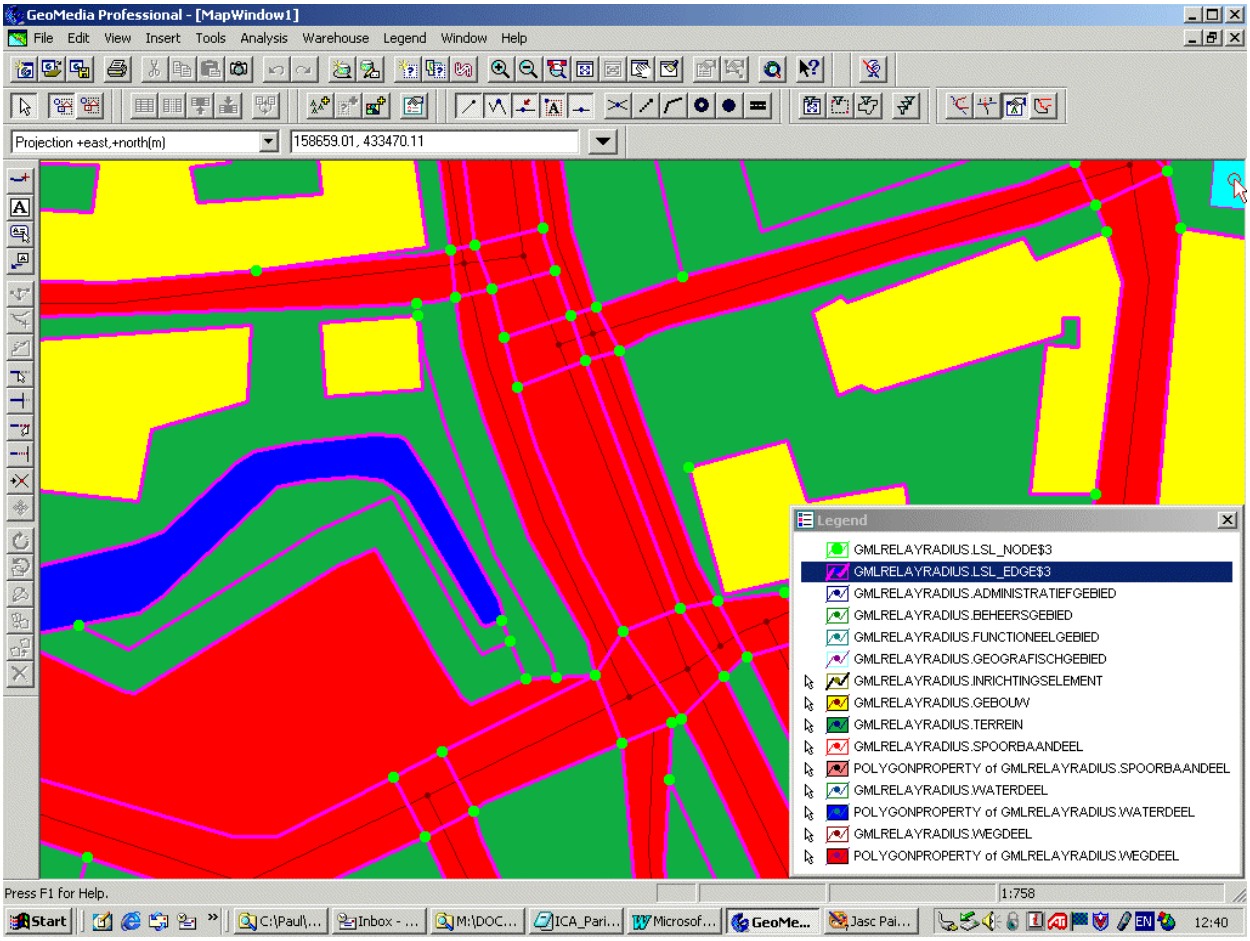


Fig. 5 - Topology present in map data - shared edges and joining nodes

4.1 Why topology?

Topology knowledge is essential to preserve and utilise the connectivity and adjacency relationships quickly and simply during generalisation processing. It is used for several reasons including the following:

- To find the objects connected to an object or a group of objects that are generalised, that did not take part in the generalisation process, for example:
    i. If a lake area object is collapsed to a point, the area previously covered by the lake must now be covered by the adjacent objects, so topology is used to find the adjacent area objects
    ii. If road is displaced from its original position, the nodes underlying the road must remain connected to the road network, so topology is used to find the connected road objects.
    iii. The specification for the generalised product states that minor rural side roads must be elided, unless they have farms or similar buildings at the end. Both the 'in rural area' and 'has something at the end' tests are topological queries, and hence fast and accurate.
- To determine whether the results of a generalisation process have broken the topological network. For example
    i. If a pylon is a small distance from a road to the left, the road generalisation could result in the road being on the other side of the pylon - this case can be detected using face maintenance
    ii. If road generalisation resulted in a road crossing another road that it did not previously cross - this case can be detected by following the topological references
- To ensure that when a real-world object is modified during a generalisation process, the objects that share the topology of this real-world object are also modified appropriately. This is achieved by either modifying the topology directly or using the topological references to identify the connected objects and modifying them appropriately. For example,
    i. If the geometry of a road is simplified, the linework of an administrative boundary (line object) and a forest (area object) that share the underlying topology of this road will also be simplified.
    ii. If the geometry of a road is displaced, the bridge (point object) that lay on the road will remain on the road.

5 JAVA & XML

Two technologies which have swept across the IT industry over the past few years are Java and XML. They have underpinned the explosion in Internet and e-commerce applications. We have taken the opportunity of the work that is creating ***Clarity***, to make use of Java and XML, and gather the benefits from them.

Java is introduced as an alternative to Lull, as the preferred language for implementing method behaviours, such as the constraints and algorithms for agent-based generalisation.

XML is introduced as the format for storage and transfer of generalisation specifications, such as the target map specification, and the definitions of the process sequences to be applied during generalisation.

5.1 Why Java?

Java is a portable, object-oriented programming language, created originally by Sun Microsystems (http://java.sun.com/), but now controlled by a community process (http://www.jcp.org/). Among the reasons for the selection of Java for ***Clarity*** were:

- Java is the language of choice for Object-Oriented programming
- It is the main teaching language at university, so trained staff readily available
- The language implementation (J2SE) is available free for download from Sun (and others).
- Powerful development tools are available (Eclipse, Borland JBuilder, etc)
- It is portable across platforms
- It has automatic memory management with garbage collection, thus avoiding many common programming faults.

5.2 Java for methods, constraints, measures and algorithms

Clarity continues to support Gothic's original method language (Lull), and its ability to call routines written in C. Hence, existing proven methods are not being rewritten in Java. However, new measures, constraints and algorithms for ***Clarity*** will be implemented in Java.

The following example (Fig 6) shows a Gothic method written in Java (not a generalisation method, but one to change the colour of the representation according to the number of points in the line). Note the use of exceptions (try), which avoids all the tests of status codes needed in the Lull equivalent.

```
package gothic.user;

/**
 * An example of how to write a Java class that defines Java methods for use as Gothic method behaviours.
 */

import gothic.descriptor.Plot;
import gothic.descriptor.Geometry;
import gothic.descriptor.Version;
import gothic.main.GothicException;
import gothic.main.UnexpectedNullException;
import gothic.main.UnexpectedTypeException;

public class Area extends gothic.main.GothicObject {
    /**
     * Our display method. This assumes that the Gothic "Area" class using this display method inherits from "graphic"
     *  or one of its relations, which provide a "Default" display method.
     */
    public Boolean Default(Plot plotId, int overlayIndex, int colourModel, boolean draw)
        throws GothicException, UnexpectedNullException, UnexpectedTypeException
    {
        int      result;
        Geometry geomId;
        boolean  success = false;

        geomId = (Geometry)getValue("geometry");

        try {
            // Plot in a colour according to our number of vertices
            plotId.areaFilled(overlayIndex,colourModel,draw,geomId,getIntValue("number_of_vertices"));
            success = true;
        }
        finally {
            geomId.destroy();
        }
        return new Boolean(success);
    }
}
```

Fig. 6 - Java Example Gothic Method

5.3 Java client for generalisation definition

The Java-based interface for defining generalisation has been implemented as a plug-in to Laser-Scan's Gothic JADE (Java Application Development Environment). Java interfaces will be provided to setup the parameters for generalisation processes. This will include the setting up the constraints, as shown in Figure 7.

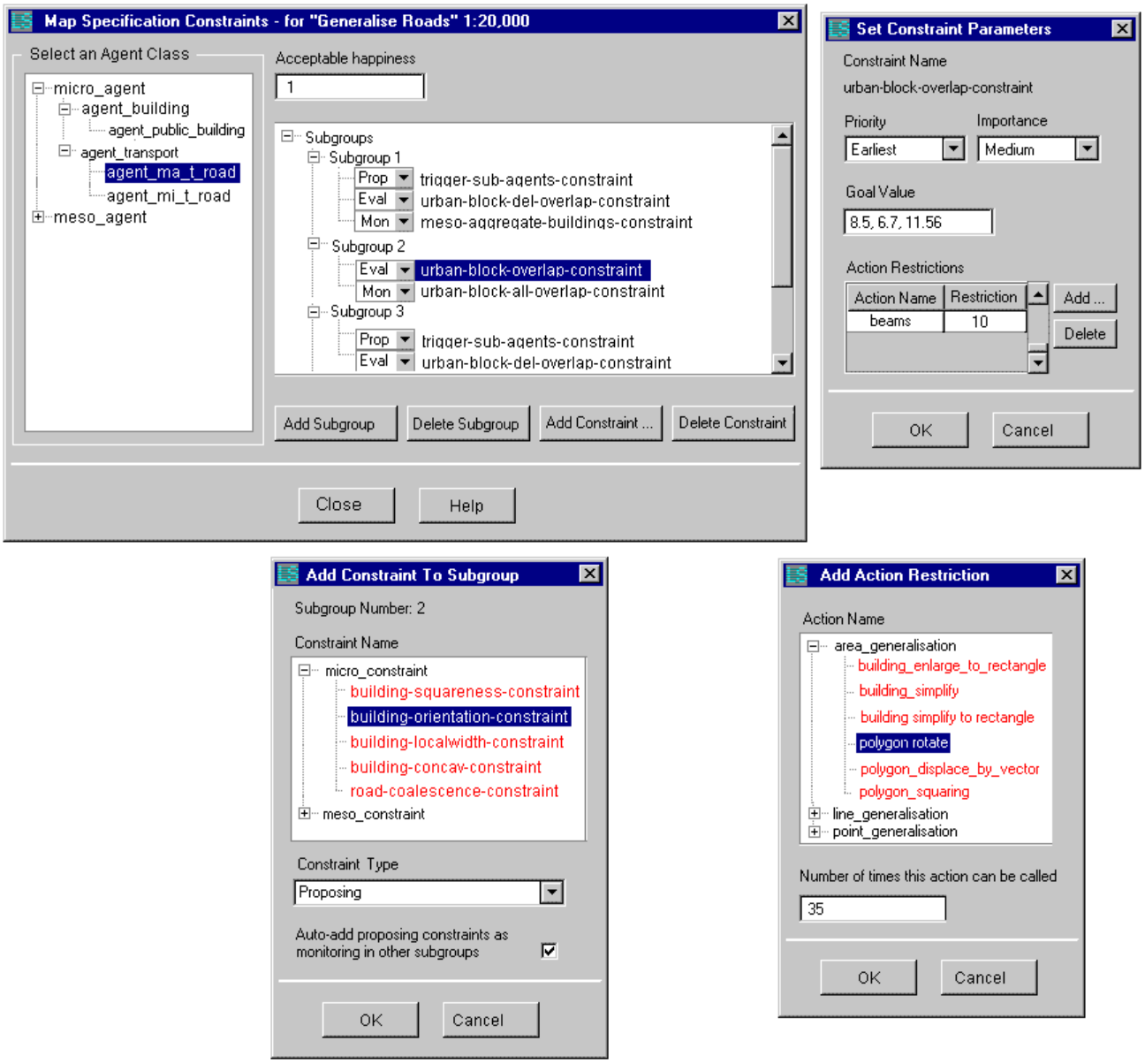


Fig. 7 - Java interface for generalisation definition

The Java-based interface also includes the setting of global and class specific parameters for the agent methods (algorithms and constraints), see Figure 8. The interfaces will be able to save and load the parameters to and from XML files, as described in section 5.7.

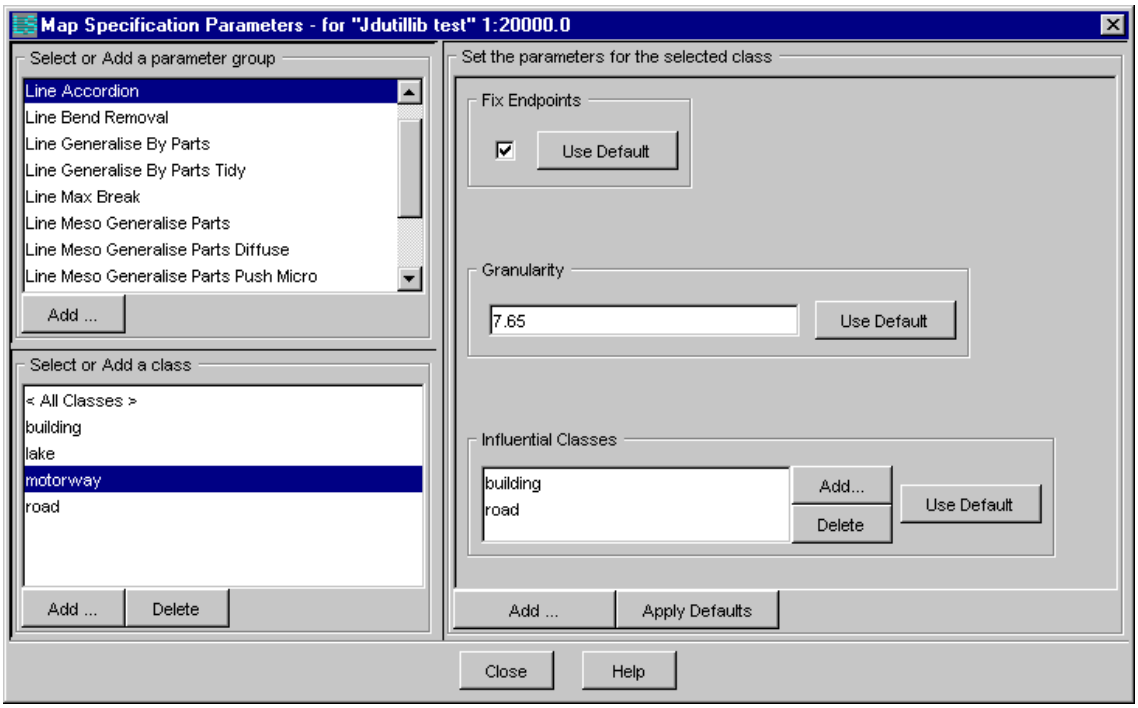


Fig. 8 - Java interface for generalisation definition

A Java interface is also being developed to set up the data model for generalisation. This will allow the real-world classes to be made to inherit from the appropriate generalisation baseclasses, and will also be used to define constraints and algorithms. Further phases will include extension to cover other aspects of data model, such as defining process methods.

5.4 Java clients for generalisation execution

Two interfaces (see Figure 2) are provided regarding execution of generalisation processes.

- The Agent Process Diagnostic Interface is an interactive tool for setting up, manipulating and inspecting the agent stack. It is of use primarily for flowline developers or researchers.
- The Process Job Interface is for running sequences of process methods with parameters, including object selection criteria. Invocation of agents is done by a particular process method, taking the map specification as a parameter.

5.5 Java client for interactive completion

The JADE environment can also be used for visualisation of generalisation results and for interactive completion of generalisation for the cases where human intervention is needed. The following screenshot (Fig 9) shows the JADE user interface, displaying the results of some experiments in Agent-based generalisation of buildings.

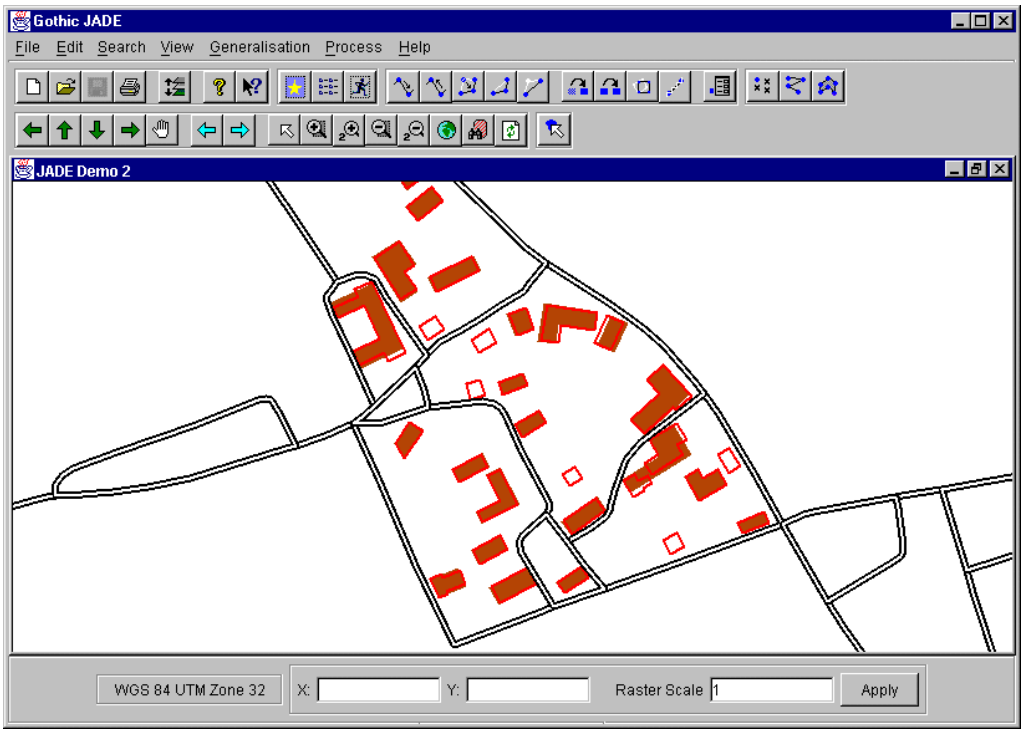


Fig. 9 - Java interface for interactive completion

5.6 Why XML?

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. XML was created by the World Wide Web Consortium (W3C), and standardised in 1998 (http://www.w3.org/XML/).

Among the reasons for the selection of XML for ***Clarity*** were:

- XML is an IT standard across the enterprise
- It is highly extensible and flexible
- Many third-party are XML tools available, not just for GIS (xslt etc).
- Third-party editors are available (e.g. XML-Spy), which dynamically adapt to XSD schema definitions.
- Free and robust software routines are available to parse and write XML
- Data is self-describing - less documentation needed.
- Data is highly reusable
- XML is totally portable across platforms
- It is human readable
- XML can easily be read in to Java. Castor (http://castor.exolab.org/) and JAXB (http://java.sun.com/xml/jaxb/) are two approaches that provide mapping between XML documents and Java objects.
- Generalisation is a field of active research. New methods and algorithms are developed all the time. An XML file is extensible which allows it to accommodate new algorithms without the need to change the software dealing with the existing algorithms.

5.7 XML for map specifications and process sequences

Generalisation historically required a large number of user-definable parameters. In the agent-based scenario, many of these parameters are superseded by a 'map specification', defining what is to be achieved, coupled with a set of 'constraints' which put limits on allowable change. In either case, there is a need to define and store a complex set of definitions.

XML is the format of choice because it allows these definitions to be structured in a logical way. It can be read by humans, and displayed and edited in a large number of applications, which makes it easy to check that no accidental errors are introduced. This makes XML an ideal depository for parameters that are required by process methods. An example of the resultant XML is shown in Figure 10.

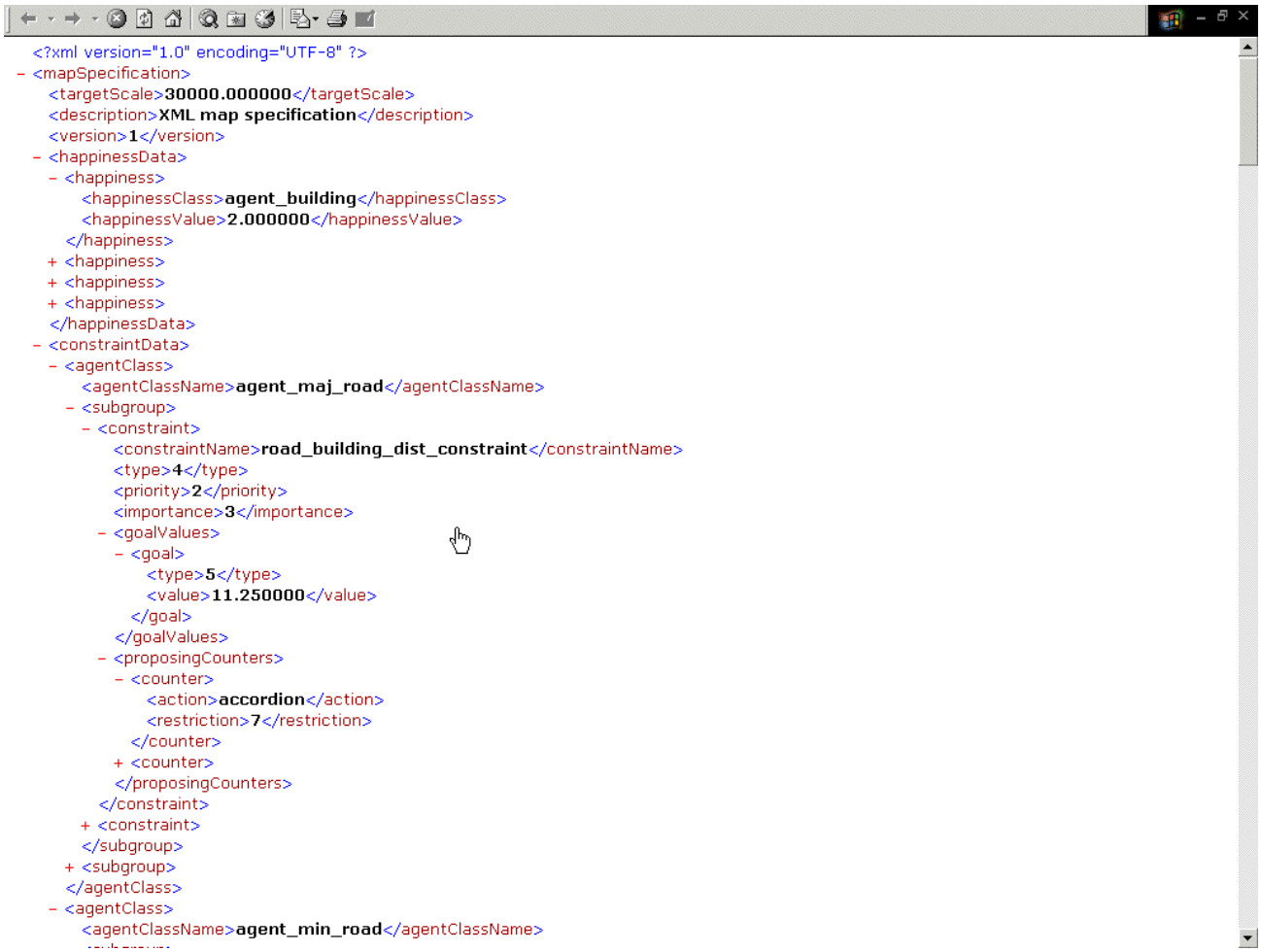
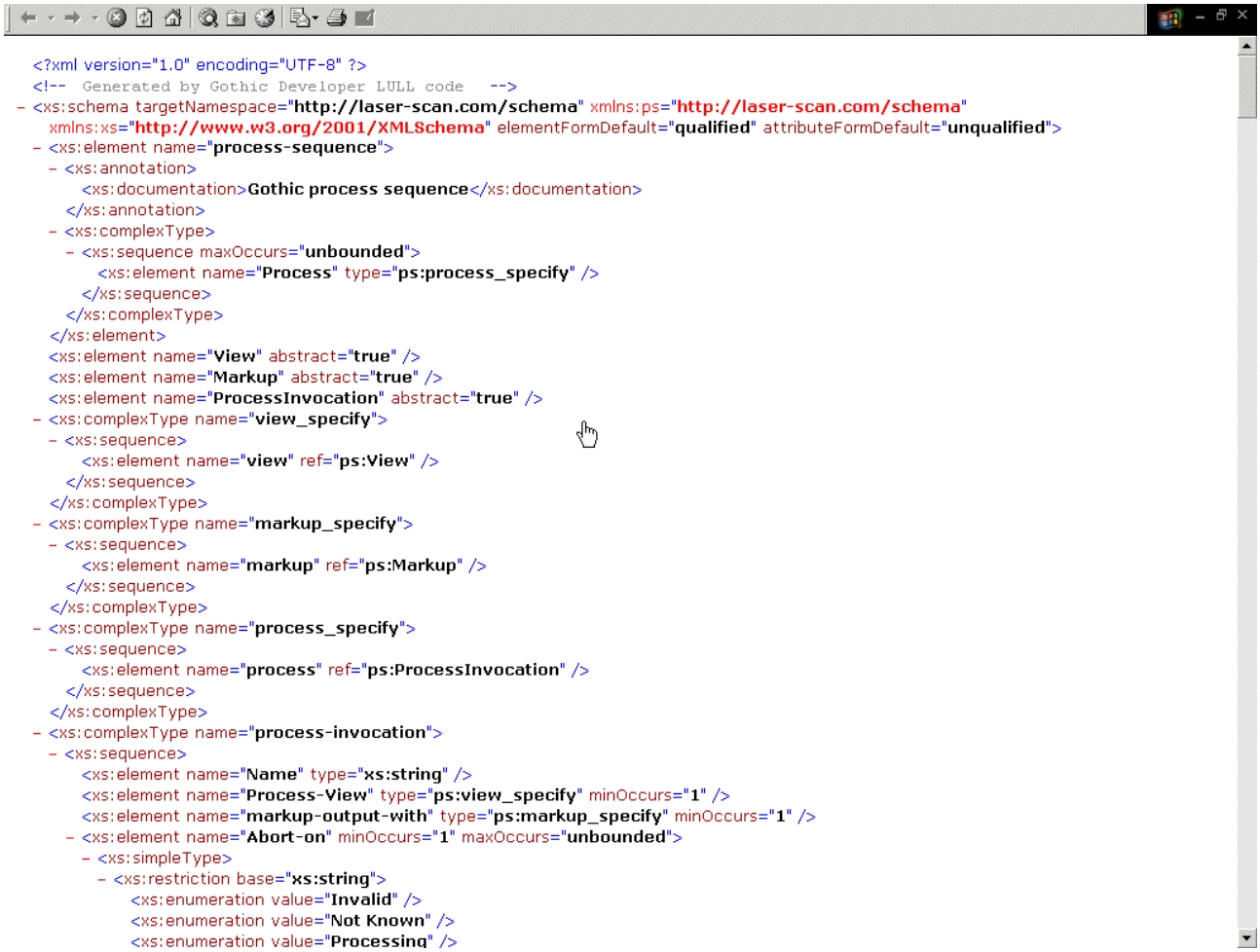


Fig. 10 - Map Specification as XML

The structure of the XML for a Map Specification is defined by an XSD (XML schema definition). An example of such XSD is shown in Figure 11.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  Generated by Gothic Developer LULL code  -->
- <xs:schema targetNamespace="http://laser-scan.com/schema" xmlns:ps="http://laser-scan.com/schema"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  - <xs:element name="process-sequence">
    - <xs:annotation>
        <xs:documentation>Gothic process sequence</xs:documentation>
      </xs:annotation>
    - <xs:complexType>
      - <xs:sequence maxOccurs="unbounded">
          <xs:element name="Process" type="ps:process_specify" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="View" abstract="true" />
    <xs:element name="Markup" abstract="true" />
    <xs:element name="ProcessInvocation" abstract="true" />
  - <xs:complexType name="view_specify">
    - <xs:sequence>
        <xs:element name="view" ref="ps:View" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="markup_specify">
    - <xs:sequence>
        <xs:element name="markup" ref="ps:Markup" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="process_specify">
    - <xs:sequence>
        <xs:element name="process" ref="ps:ProcessInvocation" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="process-invocation">
    - <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Process-View" type="ps:view_specify" minOccurs="1" />
        <xs:element name="markup-output-with" type="ps:markup_specify" minOccurs="1" />
      - <xs:element name="Abort-on" minOccurs="1" maxOccurs="unbounded">
        - <xs:simpleType>
          - <xs:restriction base="xs:string">
              <xs:enumeration value="Invalid" />
              <xs:enumeration value="Not Known" />
              <xs:enumeration value="Processing" />
```

Fig. 11 - XSD Schema Definition for Map Specification

Process sequences are sets of process methods with parameters, each of which is run on their own selection of objects. They are defined using an XML editor (e.g. XML-Spy), based on a provided template and subject to a provided XSD schema, and are read from XML into Clarity. Once loaded, a Java interface in JADE is used to invoke the process sequence.

6 PRODUCT STRATEGY

6.1 Why *Clarity*?

The work to produce ***Clarity*** builds on the strengths of the Gothic object-oriented toolkit and spatial data access, but provides a modern, portable, extensible, user interface. Development of new constraints, measures, algorithms, sequences and map specifications will all become easier in the new environment. It pulls together a series of advances, which together make a new product:

- Java user interface
- Java programming of methods
- XML specifications
- Enhanced and re-engineered Agent core
- Chosen set of proven algorithms, measures, and constraints
- New documentation for user and researcher
- ClearText label placement engine (http://www.laser-scan.com/cleartext/)

6.2 What Next?

The initial release of ***Clarity*** will be focussed on the generalisation infrastructure. It will provide generalisation across the same scale bands (centred on 1:10K to 1:50K), and the same range of feature classes (centred on buildings and roads) as the Agent prototype. The feedback from the MAGNET group tells us that these are the highest priority topics.

Beyond the first release, the framework is designed for extensibility to other scale bands, and to other feature classes. The scale band of next greatest interest is that from 1:1K to 1:10K. Many organisations (Ordnance Survey in Great Britain and cadastral authorities across Europe) have basic scale data in the 1:500 to 1:2500 range. They want to derive 1:10K data, which then in turn would be used to derive 1:25K or 1:50K products. There is also interest in smaller scales (such as 1:250K), once the 50K dataset is available.

The likely feature classes to gain new benefit from the embedded intelligence of the agent approach are hydrography (rivers, lakes etc), and vegetation (forests, orchards, trees, gardens).

Other inputs will come from the parallel work on model generalisation and on incremental generalisation being done for the German Länder. This includes research on flowlines and algorithms based on the separation of model generalisation from subsequent cartographic generalisation.

In addition, there will be feedback and new algorithms from the early users, not only those doing commercial production, but also those involved in academic research.

6.3 Long-term Future Direction

As was discussed in section 3, there is a powerful industry drive towards use of relational database, and particularly Oracle, as the main enterprise datastore. Users of generalisation would like the generalisation tools to be directly interfaced to the Oracle datastore. However, the Oracle server environment is not yet up to the high demands of generalisation for irregular spatial data access, hence the ***Clarity*** approach of using an object-oriented datastore as a 'persistent cache'.

But, the relational database software manufacturers and third parties are not standing still, and Moore's Law continues to provide twice the compute power for the same cost every 18 months. Laser-Scan has recently launched Radius Topology (see http://radius.laser-scan.com) - the first in a family of products that add intelligence into the Oracle server. Work is underway on Radius Lifecycle, which will provide much of the object intelligence framework needed for active object and agent-based spatial processing.

With those trends in mind, the implementation of ***Clarity*** is being guided so as to be able to move smoothly in the medium future to operate in a Radius environment. The architecture of ***Clarity*** has the bulk of the work done in object methods, which could in future be executed in the context of an object-relational environment - either plugged directly in the Oracle server (as is done for Radius Topology), or in a middleware tier based on Oracle 9i Application Server. Either way, the future of generalisation lies with a mainstream relational datastore, active objects, intelligent agents, topology, Java, and XML.

7 ACKNOWLEDGEMENTS

Thanks are due for contributions and feedback from members of the AGENT project and from the early adopters, and to the members of the MAGNET consortium for content material and visual examples.

8 REFERENCES

Hardy P.G. and Haire, K., 2000, “Generalisation, Web Mapping and Data Delivery over the Internet”, ICA workshop, November 2000, Barcelona, Spain, or online at http://www.Laser-Scan.com/papers/barcelona2000pgh.pdf

Hardy P.G. 2000, “Multi-Scale Database Generalisation For Topographic Mapping, Hydrography And Web-Mapping, Using Active Object Techniques”, IAPRS, Vol. XXXIII, ISPRS Amsterdam, Netherlands, July 2000, or online at http://www.laser-scan.com/papers/isprs2000pgh_1436.pdf

Hardy P.G., 2001, “Active Objects And Dynamic Topology For Spatial Data Re-Engineering And Rich Data Modelling”, Dagstuhl Seminar 01191 on Computational Cartography and Spatial Modelling, May 2001, or online at http://www.laser-scan.com/papers/dagstuhl2001pgh.pdf

Lamy, S., Ruas, A., Demazeau, Y., Jackson, M., Mackaness, W.A., and Weibel, R., 1999, “The Application of Agents in Automated Map Generalisation”, 19th ICA/ACI Conference, Ottawa, 160-169.

Ormsby, D., and Mackaness, W. A., 1999, "The Development of Phenomenonological Generalisation within an Object Oriented Paradigm": Cartography and Geographical Information Systems, v. 26, p. 70-80.

Ruas A., 2000, “Project AGENT: Overview and Results of a European R&D Project in Map Generalisation”, ICA Workshop, November 2000, Barcelona, Spain.

Sheehan, R, “Generalisation at the Danish Mapping Agency - A Solution Based on the AGENT Research Project and Laser-Scan Object-oriented Technology” GIM International July 2001, Volume 15, Number 7

Van Oosterom, Stoter, Quak & Zlatanova, 2002, “The Balance Between Geometry and Topology”, Spatial Data Handling 2002, Ottawa, Canada, or online at http://www.radius.laser-scan.com/pdf/chpt16-147.pdf

[Original 2002-12-18, Revised 2003-04-11]