Spatial coincidence modelling, automated database updating and data consistency in vector GIS

Promotor: Dr. Ir. M. Molenaar
Hoogleraar in de Theorie van de Geografische
Informatie Systemen en de Remote Sensing

Co-promotor: Dr. T. Bouloucos
Universitair Docent
International Institute for Aerospace Survey and Earth Sciences (ITC),
Enschede

Olajide Kufoniyi

Spatial coincidence modelling, automated database updating and data consistency in vector GIS

Proefschrift
ter verkrijging van de graad van
doctor in de landbouw- en milieuwetenschappen
op gezag van de rector magnificus,
Dr. C.M. Karssen,
in het openbaar te verdedigen
op dinsdag 23 mei 1995
des namiddags te half twee uur in de aula
van de Landbouwuniversiteit te Wageningen

ISN 907964

ISBN 90 6164 105 5

International Institute for Aerospace Survey and Earth Sciences (ITC)
PO Box 6, 7500 AA Enschede, The Netherlands

Olajide Kufoniyi Spatial coincidence modelling, automated database updating and data consistency in vector GIS

PROPOSITIONS

(1) The progress being made towards the ultimate goal of automated change detection and feature extraction should be complemented by an automation of database updating and consistency enforcement.
- *This thesis*

(2) Terrain objects are often spatially coincident; this spatial coincidence should be explicitly represented in a spatial data model to have richer information content and faster query realisation.
- *This thesis*

(3) The current practice of delayed reconstruction of topology (after database updating) should be replaced with an automatic reconstruction (during updating) to facilitate on-line updating of a spatial database from a (remote) data acquisition centre.
- *This thesis*

(4) An information system loses its reliability once information retrieved from it is identified as being inconsistent; thus the reliability of the system should be enhanced through the provision of automated procedures for monitoring and enforcing data consistency.
- *This thesis*

(5) The subject of automated techniques for map revision and database updating requires serious attention now that more and more organizations have completed database construction and are faced with the task of maintaining these databases with current information.
- *1994 ISPRS Annual Report*

(6) A GIS will be successful only if it can present the user with an accurate, consistent and current view of the world as required for his application.

(7) When developing a generic spatial data model, the complexity of a terrain object (in terms of shape) cannot be predefined (except for application-specific models); thus the most feasible approach is to model elementary objects which can then be used as building blocks for complex objects.

(8) An individual without information cannot take responsibility; an individual with information cannot help but take responsibility.
- Jan Carlson, CEO, Scandinavian Air Systems

(9) Stable democracy cannot be realised until the society is ready to invent its own form of democracy that is rooted in its own realities.

(10) Only creative, rather than emulative, socio-political theories can lead to progress of developing nations.

(11) The less developed world will not become developed through the goodwill or generosity of the developed powers; it can only become developed through a struggle against the external forces which have a vested interest in keeping it undeveloped.
- Kwame Nkrumah, First Post-independence President of Ghana (1957 - 1966).

(12) The man dies in him who keeps quiet in the face of tyranny.
- Wole Soyinka, 1986 Nobel Laureatte for Literature, in "The Man Died"

(13) You don't need a good memory if you always speak the truth.

To Taiwo, Mayowa, Dunni and Ope

ABSTRACT

Kufoniyi, O., 1995. Spatial coincidence modelling, automated database updating and data consistency in vector GIS. PhD Thesis, Department of Surveying and Remote Sensing, Wageningen Agricultural University, The Netherlands, 206 pp

This thesis presents formal approaches for automated database updating and consistency control in vector-structured spatial databases. To serve as a framework, a conceptual data model is formalized for the representation of geo-data from multiple map layers in which a map layer denotes a set of terrain objects of the same mapping context, e.g., cadastral, soil mapping, etc. The necessity for a generalised model arises from the frequent requirement in spatial analysis and planning for a geometric integration of several different views of the world, whereas most existing data models were designed from the perspective of a "single application", leading to ad hoc and repeated overlay computations (during query processing) when dealing with an integrated analysis. An alternative model is therefore proposed in this thesis for the geometric integration of geo-data from multiple map layers. The proposed model is an object-based, query-oriented 2.5D data model for multi-valued vector maps (DMMVM).

A multi-valued vector map refers to the vector-based representation of terrain objects from multiple map layers whereby two objects of the same geometric type may be spatially coincident. Two objects of the same type are spatially coincident if they (partially) overlap in space. In this model, positions of objects are defined in a 3D metric space but embedded in 2D topologic space. The model is based on the 2D formal data structure (FDS) for single-valued vector maps.

Terrain objects play a central role in the terrain description; each object has a thematic component and a geometric component. In the thematic domain, the objects can be grouped into thematic classes in which each class has a specific attribute structure, and in the geometric domain the object types (points, lines and areas) are distinguished for a 2D or 2.5D terrain description.

A geometric data type -- the m-dimensional container, or simply m-container, where $m \in \{0,1,2\}$ -- is then introduced to model spatial coincidence among objects of the same geometric type. By introducing the container data type, overlapping sections across the layers are uniquely identified such that they have their own individual geometric data and non-spatial data, apart from those inherited from the overlapping objects; they can then be maintained and manipulated by the DBMS just like single objects. Using graph theory as a mathematical tool, the three container types are then represented by the topologic primitives arc and node. A node defines one 0-container and/or the beginning or end of an arc, while an arc defines (part of) one 1-container and/or (part of) the boundary of a 2-container. The arc is defined by one start node and one end node, and a node is defined by a coordinate triplet X,Y,Z. A flexible integration of the model with a DTM is also presented in the thesis, using an edge-based TIN. Two primitives of the edge-based TIN (edge and vertex) are added to the data types of the DMMVM to define the integrated model.

Research and development on the updating of geo-information have been confined mainly to the aspects of data collection and change detection, with little emphasis on the corresponding

automated propagation of the updating in the database in a consistent manner. To address the latter aspect, procedures are formulated in the thesis for a consistent automated updating of a vector-structured database, using the DMMVM as a framework. Algorithms are provided for the automated update propagation such that topology is automatically updated by the system, while maintaining structural and semantic consistency. This will improve on the current practice in operational systems, which usually requires a delayed reconstruction of topology whenever there is a geometric change in the database. Algorithms are developed for the insertion, deletion or modification of each of the eight data types (area, line, point, 2-container, 1-container, 0-container, arc and node) in the DMMVM. The human operator interacts with the system at the object-level, while the system propagates the update. The topology of the database is updated dynamically by the system by evaluating, using computational geometry, the topologic relationship between the new primitive (arc or node) of an object and the existing primitives in the database. The type of relationship detected will then activate the relevant consistency rule (including update propagation) to validate the topology and consistency of the database. The system alerts the human operator if it is not possible for it to resolve the inconsistency.

To enforce data consistency during geometric updating of the database, consistency rules are defined to ensure structural consistency, while a monitoring strategy is formulated for semantic (application-dependent, topologic) constraints. In both cases, topology plays the central role as an "alerter" of constraint violations. Thus the possible topologic relationships among the three elementary object types (area, line and point), and among the geometric primitives (arc and node) in the DMMVM are formalised and algorithms are defined for detecting the occurrence of any of the elementary relationships for any object combination. Then the consistency constraints can be translated to topologic relationships and stored in the database as *events*, and the corresponding responses of the system to enforce consistency can be defined as *actions*, thus giving a rule-based procedure (using the **if *event* then *action*** convention) for the management of data consistency in spatial databases.

The DMMVM was translated into a relational database structure and an object-oriented database structure to facilitate implementation in a variety of systems. The object-oriented data structure and the consistency rules and algorithms were tested experimentally using Postgres, an extended relational database management system. Data were acquired using the Kork digital mapping system, on a Planicomp C120 photogrammetric stereoplotter equipped with a Zeiss Videomap and a Calcomp drawing board digitizer.

The thesis concludes with an evaluation of the proposed model and an outline of areas requiring further investigations.

Keywords: geographic information system, spatial coincidence, geometric data integration, data modelling, topologic relationship, consistency rule, database updating.

PREFACE

This thesis is the outcome of a cooperative PhD research project involving the Department of Geoinformatics, International Institute for Aerospace Survey and Earth Sciences (ITC), Enschede, and the Department of Land Surveying and Remote Sensing, Wageningen Agricultural University (WAU), The Netherlands, with Professor Martien Molenaar as the supervisor (called "promotor" in Dutch) and Dr. Theo Bouloucos as the co-promoter. The research started in January 1992 and was funded by ITC.

The thesis addresses the issues of spatial database updating and data consistency, including spatial data modelling. It is a continuation and an elaboration of my MSc research (1987/89) on the topologic editing of vector-structured geo-data which was also supervised by both Martien Molenaar and Theo Bouloucos.

During the PhD research, I participated in the supervision of some MSc research projects and gave some lectures and presentations. These activities, together with my various presentations at international conferences and symposia, were found useful as media for the dissemination of the concepts developed in the thesis. They provided a kind of external moderation of the research work and thus contributed to the realization of the project.

Many people contributed one way or another during the research project. I hereby express my sincere appreciation to all of them. However, I will still want to mention a few of them. First, I am particularly grateful to my promoter, Dr. Martien Molenaar for his scientific contributions and support, right from the problem formulation stage of the research. His meticulous and critical reviews of the thesis and the stimulating discussions on the conceptual aspects of the research are most appreciated.

My special thanks go to Dr. Theo Bouloucos, my co-promoter, for his comments and cooperation, and especially for the valuable logistic support. I thank Dr M.M. Radwan for the discussion we had on the object-oriented modelling aspect of this thesis.

During the implementation phase of the project, I received assistance from a number of people at ITC and I wish to acknowledge some of them. Martin Blankestijn helped in installing the Postgres DBMS. Mr. J.L.A. Tariel assisted in procuring the soil map from France and Professor J.A. Zinck helped to translate the soil classes (from French to English). Mr. Remy Ackermann assisted in the use of the Kork digital mapping system and Morakot Pilouk gave his visualization program. J.P. Bakx assisted in the reproduction of the colour prints.

The MSc projects of Samson Okoth Ayugi, Raja Ram Chhatkuli and El Hossein Essayah were also of great help in the experimental implementation of the research project. I also appreciate the contacts and discussions with my friends and PhD colleagues at ITC and the Department of Land Surveying and Remote Sensing, WAU, especially Luisa Pereira and her husband Jacob Keizer, Hoanh Chu Thai, Nana Suryana, Sylvia de Hoop, Lukman Aziz and Wanning Peng.

I want to express my special gratitude to Ms Ann Stewart, the editor of the ITC Journal, for

sparing the time to edit this thesis. I also appreciate the contributions of members of the ITC support staff, especially the Centrale Afdeling Informatisering (CAI) and the secretariat of the Geoinformatics Department.

Finally, I want to express my profound appreciation to my wife, Taiwo, and children, Mayowa, Dunni and Ope for their moral support, patience, love and understanding, but most especially for their understanding in being deprived of my attention and care during the period of this research. It was a very difficult period of my life and I missed them a great deal.

23 May 1995 Olajide Kufoniyi

CONTENTS

FIGURES

TABLES

APPENDICES

1

INTRODUCTION

1.1 Information System and Geoinformation Production

Information systems (IS), in general, have been widely accepted as efficient tools for the collection, storage and analyses of various kinds of data and for decision making.
The general concept of the system can be schematically described by Figure 1.1 (Wintraecken, 1985; Molenaar, 1991b).

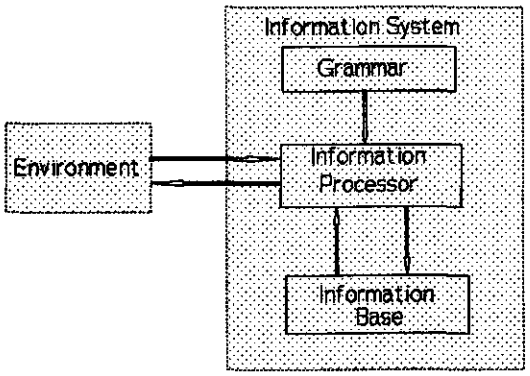


Figure 1.1 A Conceptualized description of information system

In the diagram, "environment" represents the users (persons, machines and other information systems) and the real world processes that interact with the system. The "information processor" serves as an interface between the environment and the information base. Through it, the system communicates with the environment to receive data to be stored in the information base as well as the requests of users for information from the system. In response to these requests, it retrieves requested information from the base and supplies it to the user. The "grammar" gives the rules for the allowable states of the information base and its state transitions. It therefore guides the information processor in its processes and behaviour.

Relating this general concept of an information system to geographic application, which is then referred to as a geographic information system (GIS), in the "information base" will be a structured digital spatial database, the "grammar" will be the rules and algorithms to guide data input and update and information retrieval from the database while the "environment" represents the users in the wider sense of it (persons and other GISs). The "information processor" is, in general, a collection of four interrelated software subsystems respectively performing the following four functions, namely data collection and input, data storage and retrieval, data manipulation and analysis, and visualization and reporting. In other words, the information processor contains the database management system (DBMS) plus the other application software. The spatial database represents the real world objects as seen by an application. Its design often evolves through the hierarchic processes of conceptualization of reality in a data model, the structuring of this model in a computer-representable format, and the design of a file structure for the storage of the structured data. The information content of a database depends therefore on the data model. Obviously, information that has not been represented in the model, either explicitly or implicitly, cannot be retrieved from the database.

Consequently, database designers have always placed much emphasis on data modelling. A common trend, however, is that the data models are often developed for specific tasks even within a discipline.

A high percentage of the cost of operationalizing GIS for geoinformation production is attributed to data collection; hence it must be ensured that the quality of the data meets the technical specifications of the project. The database reflects the "reality" of an application at a specific time, e.g., at the time of aerial photography if acquisition is done by photogrammetric method of data collection. However, the database is supposed to be reusable for spatial planning and decision making; thus it must be up-to-date and consistent. Unfortunately, the terrain objects to be represented in the database are generally not static in time. Therefore the database should be made to efficiently respond to object dynamics through updating. And because the updating may disturb data consistency, rules should be provided to guide the system and human operator during the updating. Some of the outstanding problems in the acquisition and modelling of spatial information that will be addressed in this thesis are discussed in the following section.

1.2 Need for the Study

Geographic information systems provide the means for a variety of users to handle spatial data in a wide range of applications. As mentioned above, at the heart of the system should be a structured spatial database representing terrain objects of interest. Because different applications normally view terrain situations differently, the data model on which the database design is based is usually tailored to an application. At the same time, the system offers the opportunity to bring together hitherto separate disciplines, thus facilitating integrated analysis of spatial data. For example, it is possible to integrate cadastral information, land use data and soil data in the database. Conventionally, each of them would be regarded as separate mapping themes and produced as such because of the limitations of traditional map-making technology. This traditional spatial reasoning, otherwise called the "layer approach", has been carried into the digital era, in which most operational GI systems operate on the principles of layers.

To integrate the three themes mentioned above, each theme would be modelled as a layer and the three layers would be intersected by overlay computation during query processing. Apart from the high overhead cost necessitated by the ad-hoc overlay computations, a lot of redundant data are collected and stored (e.g., common geometry will be collected and stored in each layer). Furthermore, one of the advantages of defining a data model for database design is that the information content of the database system built on that model can be formalized a priori and a spatial query language developed for the retrieval of such information. However, if the model is designed for a single layer, the above will hold only for that layer.

In other words, the information content of an integrated database can be predetermined only by first defining a data model that can handle spatial coincidence among terrain objects (i.e., that is capable of representing multiple layers). In this thesis, two objects are said to be spatially coincident if they partly or fully share the same location in space. "Normal" spatial coincidence among objects, such as two objects sharing the same boundary or a line object

passing through an area object (in the vector domain), are normally taken into consideration even in the layer models. But they normally exclude the overlapping of two or more objects of the same geometric type, e.g., two collinear line objects, which unfortunately do exist in reality. This deficiency in the presently available spatial data models should be eliminated to derive more benefits from the capabilities of GIS.

Apart from the modelling aspect, maintaining the quality and currency of a spatial database is also very important. The maintenance includes how to keep the database up-to-date and consistent. With the advances in research and development in the fields of pattern recognition and automatic feature extraction in digital photogrammetry and image processing, and in digital field survey equipment, the total process of data collection for geo-information production may be automated by also providing a dynamic update propagation facility with a "topology builder" and the means for enforcing consistency rules. For instance, when a new object is digitized, using for example on-screen digitizing, it should not be necessary for the operator to construct topology off-line (by activating a separate program); rather the system should be able to reconstruct topology on-line and to propagate the update while enforcing data consistency. With the assistance of computational geometry, building topology on-line should not pose much of a problem if the consistent topologic relationships among objects represented in the data model are formalized beforehand, something which is generally lacking in many existing spatial data models.

Maintaining data consistency in a spatial database requires major attention because the database loses its reliability once inconsistent information is retrieved from it, e.g., getting different values for the length of the same road. At the moment, no operational system warns the user when (for example) two land parcels erroneously overlap in a cadastral database; instead the system, where there is provision for topology building, simply decomposes the two overlapping parcels into three. The reliability and performance of the system will increase through the provision of a strategy for the on-line monitoring of such (in)consistencies. It will even minimize the cost of data acquisition because it will be possible to warn the operator immediately that a consistency violation occurs such that remeasurement can be done in the case of erroneous digitizing. Take, for example, a situation in which a national cadastral database is set up on a main server at the headquarters, and field offices are established for data collection by field surveyors for the updating of the database. With the availability of electronic equipment such as the "Total station" (for measurement), and the "Modem" (for transmission), the field surveyor can transmit the measured data, in real time, to the main server. If an automated update propagation facility with consistency rules is provided on the main server, the transmitted data will be entered into the database and the system can automatically propagate the update with the possibility of a feed-back to the field officer if there is any violation of consistency (such as a new parcel overlapping an existing one). Should the violation be caused by incorrect measurement, the surveyor can immediately repeat the measurements while still on the site, thereby saving the cost of a revisit to the site which may be in a remote place. Of course, this example holds for other data acquisition methods as well.

The problems identified above do not constitute an exhaustive research overview, but rather the presentation of some of the critical areas in vector-based GIS that will be given priority in this research project. These are presented as the objectives of this thesis in the next section.

1.3 Objectives of the Thesis

The objectives of this research are:

- To formalize a 2.5D vector data model that is capable of handling spatial coincidence among terrain objects, i.e., for the integrated representation of multi-layer geo-data.

- To formalize the basic set of consistency rules for the developed model for the creation and updating of its database.

- Since terrain objects are not static in time, there is a need to analyze and model the effects of object dynamics on the database; hence the third objective is to analyze terrain object dynamics, define their effects on the database structure and consequently propose rule-based algorithms for the dynamic updating of the database in response to changes in physical terrain (focusing mainly on the geometric domain).

- To translate the integrated model to a prototype database structure for implementations.

1.4 Research Method (see Figure 1.2)

To fulfil the aforementioned objectives, the research tasks are divided into four phases as described below:

Phase 1: Conceptual Data Modelling

With the availability of many existing vector data models, it is not necessary to start from the scratch. Thus a review of some of the existing models is made in order to select one that can be extended to cover representation of a multi-layer terrain situation. In addition, the mathematical theory that forms the framework of the selected model is reviewed. By analyzing the basic data types in the selected model, including the functional links among these data types, we can generalise the model probably by adding extra data types. This results in a 2.5D data model for vector GIS in which terrain objects from more than one map layer can be represented. This model is referred to as the data model for multi-valued vector map (DMMVM) to indicate representation of multi-layer geo-data. Intuitively, the formal data structure (FDS) for single-valued vector maps (SVVM) (Molenaar, 1989) seems to be the best candidate for the generalisation.

Because of the importance of DTM in geoinformation production and with the absence of full 3D GIS, coupled with the fact that the positions of objects are already defined in 3D in the above-mentioned 2.5D model, a compatible digital elevation data structure is selected and integrated with the proposed data model.

In preparation for the second and third phases, the topologic relationships among the basic data types in the proposed data model are formalized. The relationships are derived at two levels, namely at the object level i.e., among the elementary object types and at the level of the geometric primitives i.e., among the geometric primitives. These relationships, apart from their use in the formulation of spatial query language, are used as tools for maintaining data

consistency and for update propagation in this thesis.

Phase 2 Formalization of Consistency Constraints and Rules

The second phase of the research deals with the identification of the basic static consistency constraints that are associated with the proposed data model to ensure the correctness of the database (after initial set-up and after updating), e.g., ensuring that two nodes do not have the same coordinates, that each node appearing in the boundary of a closed area object has exactly two incident arcs of the object (i.e, has degree 2 in graph theory), etc.

The identified constraints are then translated to formal consistency rules making use of the formalized topologic relationships. The monitoring and enforcement of the rules are based on the conventional **if *event* then *action*** procedure where the *event* represents the occurrence of an inconsistent topologic relationship and *action* represents the operation to be activated by the system to ensure consistency.

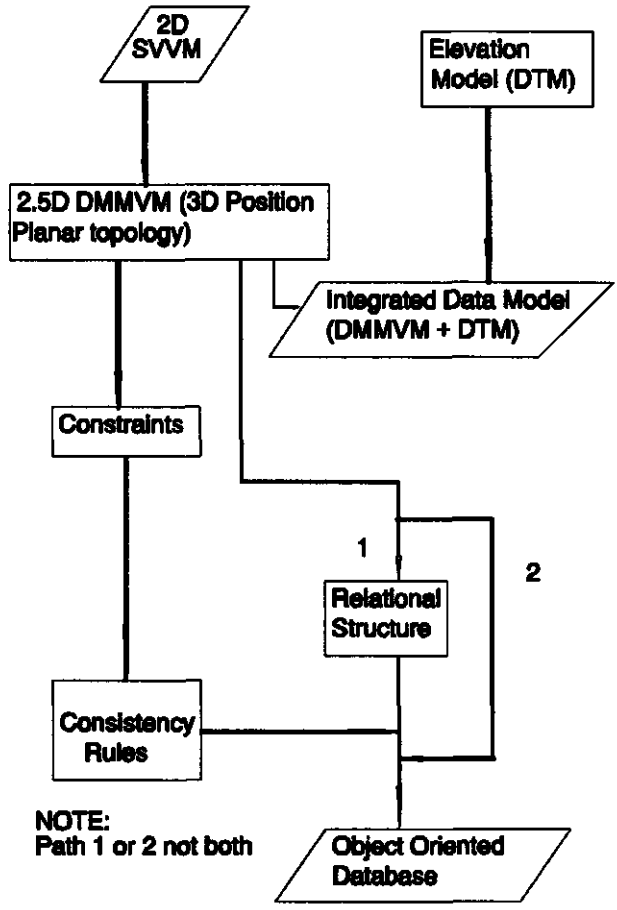


Figure 1.2 Diagram showing the framework of the research

Phase 3: Object Dynamics and Database Updating

This phase deals with the analysis and modelling of terrain object dynamics and formalization of algorithms for database updating. The main concentration is on the geometric structure of objects. Procedures are provided for the propagation of the insertion, deletion or modification of each of the basic data types in the proposed model. The procedures are arranged in a hierarchic manner with the procedures for updating of the geometric primitives of the model serving as the basic set on which the updating of the other data types is based. The aim is that the user has to operate at only the object level with the system taking care of the database re-structuring in real-time.

Phase 4: Database Structure and Implementation

Before the data model can be implemented, it has to be translated to a database structure. Various data structures are available for handling data in a database, e.g., network, hierarchic, relational and object-oriented data structures. The relational structure is the most popular but it has shortcomings in the handling of spatial data (Date, 1990). The object-oriented structure, on the other hand, has been recommended for spatial data handling (Alagic, 1989; Hughes, 1991) but operational systems that use this structure are still few. The gap-bridging solution is the so-called evolutionary approach (Beech, 1988) in which object-oriented concepts are added on top of a relational model, e.g., Postgres (Postgres, 1994). Thus an object-oriented implementation can be done in one of two ways (see Figure 1.2): (1) by using an extended relational system (e.g., Postgres and Iris), popularly called "evolutionary approach" or (2) by using a "pure" object-oriented system (e.g., Ontos) or building one from the scratch by programming using an object-oriented programming language, e.g., C++ and Smalltalk; this is popularly referred to as the "revolutionary approach".

The implementation of the proposed data model is based on the evolutionary approach to object-oriented database design. Consequently, the data model is translated to a fully-normalized relational database structure using Smith's normalization method (Smith, 1985), such that interested users can implement the prototype in either a relational system or an extended relational system. The data model is then mapped into an object-oriented data structure with appropriate definition of classes, including the identification of the properties and operations (or methods) of each class.

For the experimental implementation, a selection is made of the data collection subsystem and the database management subsystem. The main guiding criteria are (1) a stereo-photogrammetric data acquisition method is preferred because it still accounts for the most accurate and fastest data collection for high- and medium-resolution spatial databases (akin to large- and medium-scale mapping) and because of the background of the author; (2) an extended relational (evolutionary object-oriented) database management system is preferred because of the availability of a standard query language, thus minimizing programming tasks required in the pure object-oriented approach. For data collection, the Planicomp C120 with Videomap superimposition facility is selected, using the Kork Digital Mapping Software version 8.0 for the digital compilation of terrain objects. And for database management, the Postgres DBMS version 4.2 is used.

The specific tasks performed during implementation include
- mapping of the object-oriented data structure to the Postgres data model
- preparation for data collection (selection of the data sources (e.g., aerial photographs), preparation of system configuration (hardware and software), etc)
- data collection and input
- consistency checks on the created database
- sample database updating.

The experiment is then evaluated and conclusions made.

1.5 Limitations of the Study

As reflected in the thesis title, this study is generally limited to vector representation of terrain objects. Although spatial coincidence modelling implies data integration, only the geometric aspect of the integration is covered; however, the thematic attributes of the new object resulting from the spatial coincidence of two or more objects can be derived by propagation from the attributes of the coinciding objects.

Although updating of geo-data includes change detection, data collection and database updating, the main focus of the thesis is on the last aspect, covering the propagation of an update resulting from the insertion, deletion or modification of objects in the database.

In order to formalize rules for data consistency in this study, the integrity constraints are analyzed into two groups: static and dynamic constraints. The former relates more to the structure and semantics of the embedding data model while the dynamic integrity constraints deal mainly with the allowable transitions from one database state to another (i.e., transaction management, etc). We focus more on the static constraints in this research because the dynamic constraints are more related to the system environment.

1.6 Organization of the Thesis

This thesis is arranged as follows. A review of some of the concepts and literature related to this research is made in chapter 2. The formalization of the data model for representing multi-layer geo-data is described in chapter 3. It also includes the description of an approach for integrating the formalized model with the edge-based triangulated irregular network (TIN) structure to enable a flexible integration of a digital terrain model (DTM) with other geo-data. Because topologic spatial relationships play important roles in the procedures proposed in this thesis for maintaining data consistency and for database updating, chapter 4 is devoted to the modelling of topologic relationships in vector maps, including the formulation of an algebra for detecting the existing relationships between any two elementary objects in the database. In chapter 5, consistency rules are formalized for ensuring the integrity of a vector-structured database. These consistency rules are used in chapter 6 as part of the formalized algorithms for update propagation during database updating. For implementation purposes, the proposed data model is translated to both relational and object-oriented data structures in chapter 7. Two examples of the experimental implementation of the relational structure are given in this chapter. The implementation of the object-oriented data structure in Postgres, using photogrammetric data acquisition method, is described in chapter 8. Examples of database updating with automatic enforcement of data consistency are also given in this chapter. The thesis ends with some conclusions in chapter 9.

2

SPATIAL DATA MODELS AND STRUCTURES

The organization of spatial data in computer systems has received a considerable attention in the field of GIS. In a simplified form, we are interested in knowing *what* is *where* and *when*, as depicted in Figure 2.1a, using a field-based or an object-based concept of the "real world". The field-based approach conceptualizes "reality" as a "non-empty" space composed of a tiling of area units in which thematic data are recorded for each unit, while the object-based concept views "reality" as an "empty" space filled by individual terrain objects (Ehlers et al, 1989; Goodchild, 1992). On the basis of the object-based concept, we are thus concerned with an abstraction process involving the handling of *terrain objects* (what), *location* (where) and *time* (when), and the relationships among them (see Figure 2.1b). To simplify the abstraction process, one of the three domains is usually kept fixed, one is pre-defined and the third is measured or observed. In the mapping disciplines, time is often kept constant (except in spatio-temporal modelling where it is an important variable), an assumption that will be adhered to in this thesis.

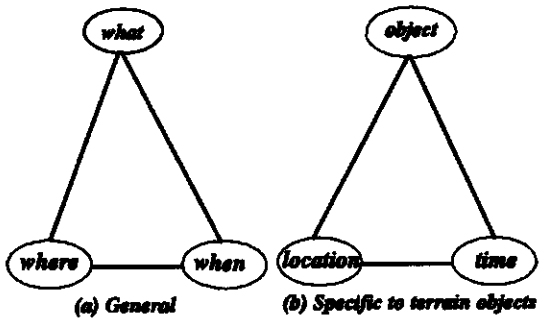


Figure 2.1 Domains of spatial data modelling

The decision as to which of the other two is pre-defined and which is to be measured then depends on the preferred mode of geometric representation, whether "tessellation", in which location is pre-defined by partitioning the space into regular or irregular units and observing and recording the entity occurring in each unit, or "vector" in which the entity is predefined and its location measured. Thus before we can organize spatial data in a computer system, we first have to identify and formalize the elements of the above-mentioned domains, usually with respect to one or more applications. In order to achieve this task, different levels of data abstraction (or data modelling) are often recognized. We will adopt the four levels proposed by Peuquet (1984), namely:

(1) Reality - phenomena as they actually exist, including all aspects which may or may not be perceived by individuals;

(2) (Conceptual) Data model - an abstraction of the real world which incorporates only those properties thought to be relevant to the application or applications at hand, usually a human conceptualization of reality;

(3) Data structure - a representation of the data model often expressed in terms of diagrams, lists and arrays designed to reflect the recording of the data in computer code;

(4) File structure - the representation of the data in storage hardware.

The last three constitute the major steps involved in database design and implementation; thus they have received much attention in the field of GIS in order to define appropriate representations of terrain objects in the spatial database.

Comprehensive treatments of these modelling steps, including examples of the data models and structures, can be found in Peucker and Chrisman (1975), Nagy and Wagle (1979), Ahuja (1983), Peuquet (1984), Burrough (1986), Samet (1989) and Oosterom (1990). An overview of some of these data models and structures is the focus of this chapter. Special attention is given to the vector representation, being the choice for geometric representation in this thesis. In §2.1, an overview of geometric models of spatial data is presented, while §2.2 focuses on conventional database structures. In §2.3, a new modelling approach in spatial applications, object-oriented data modelling, is reviewed. The object-oriented approach has been recommended for the modelling and management of spatial data (Date, 1990; Hughes, 1991; Egenhofer, 1992). Despite the acclaimed suitability of the object-oriented approach for spatial data handling, the method also has some deficiencies, especially the absence of a mature standard query language in which the relational structure has gained its popularity, maturity and reliability. Thus a database system which combines the benefits of the relational and the object-oriented structures is now being considered as more suitable than either of the two. This approach, called an object-relational database system (Stonebraker, 1994), is briefly discussed in §2.4, where some implementation issues concerning the object-oriented structure are treated. Concluding remarks are given in §2.5.

2.1 Geometric Models of Spatial Data

A spatial data model is a human conceptualization of reality (a given geographic space including all the entities embedded in that space, otherwise called "universe of discourse" UOD). The formalization is normally done without consideration of hardware and other implementation conventions (Peuquet, 1984; Egenhofer and Herring, 1991). A commonly employed approach in spatial data modelling is to separate the data into geometric and non-geometric data. In other words, having assumed time to be constant, the terrain objects to be represented in a data model are characterised by their geometric data and their (non-spatial) attribute data, as shown in Figure 2.2 (Molenaar, 1989). The geometric data are further classified into locational data, spatial relationships and the shape and size of the object (see Figure 2.3).

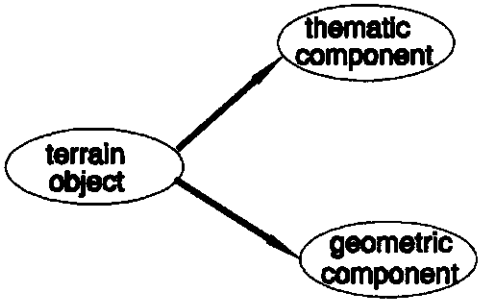


Figure 2.2 Basic structure of terrain object in a GIS

Traditionally, the thematic and geometric components of terrain objects are modelled and presented for analysis by means of 2D analog models, called maps. In the digital era, the geometric data are formalized using a tessellation or vector approach; thus the geometric models of spatial data are often grouped into tessellation data models and vector data models. Examples from the two groups are summarised in the following sub-sections with more emphasis on vector data models.

2.1.1 Tessellation Data Models

In tessellation models, the basic data unit is a unit of space for which entity information is explicitly recorded in digital form. They can be broadly classified into regular and irregular tessellations.

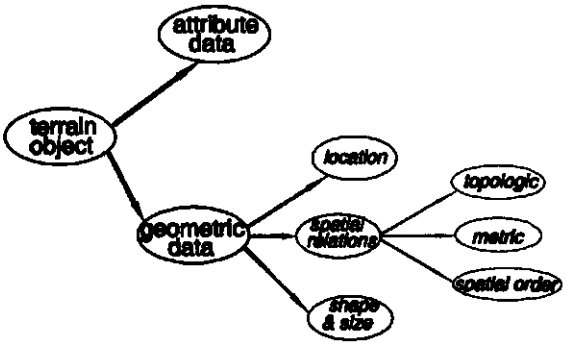


Figure 2.3 Components of terrain objects in a GIS

(a) Regular tessellations

In a regular tessellation, the geographic space is partitioned into regular cells and each cell is characterised by: (1) the area it covers and (2) one or several values describing non-spatial properties of the cell. In surface modelling (i.e., 2D topologic space), the three common types of cells are square (or regular grid), triangular and hexagonal, with the square type, popularly called "raster", being the most commonly used. Regular tessellations come close to the perception of spatial objects when data are collected by digital photogrammetry, remote sensing or other scanning devices. A regular tessellation can be subdivided into smaller cells of the same shape to have a nested tessellation model, as in a quadtree which is based on the recursive decomposition of a grid.

(b) Irregular tessellations

These result from the decomposition of the geographic space in irregularly sized cells. An irregular tessellation therefore gives a variable resolution because the size and density of the cells vary over space. The basic advantage of the model is that it reduces data redundancy and it can be tailored to the areal distribution of the data. Three examples are the irregular grid, triangulated irregular network and thiessen polygons.

2.1.2 Vector Data Models

In vector models, the individual entity is the basic data unit for which spatial information is explicitly recorded. Thus they are object-based geometric data models. They are usually classified into two main types, namely spaghetti (unstructured) and topologic models.

(a) Spaghetti model

Analog maps, as a basic data model, hold all their information in the form of graphical representations, sometimes with different colours used to differentiate the objects. This traditional view of geographic data formed the basis for many of the earlier GIS which were actually designed for the purpose of mapping (computer-assisted mapping) rather than spatial analysis. The process of digitizing from maps was imitated directly in the digital domain, leading to the unstructured vector model popularly called the *spaghetti* data model, by analogy

with a plate of disconnected, but intertwined and intertangled pasta. In other words, it is a direct line-for-line translation of the paper map in which geometry of each line-string is represented as sequences of straight line segments. Thus a line-string is represented as a sequence of n connected straight line segments with two end points and n-1 breakpoints (vertices). The position of each point is defined by a pair or triplet of coordinates. Since the model is a direct imitation of the graphical map, it is very efficient for graphical purposes, as in computer-assisted mapping. However, because the model does not incorporate topology, it is not efficient for GIS applications.

(b) Topologic model

In this model, the basic logical entity is a line segment. A line segment begins or ends at the intersection with another line or at a bend in the line. Each line segment is recorded with the coordinates of its end points, called nodes, as well as the identifier of the polygon on each side (in 2D topologic space). The more elementary spatial relationships are thus explicitly stored. With the incorporation of topologic relationships, a more "intelligent" spatial analysis can be performed and the geometric definitions of objects are represented in a non-redundant manner (e.g., the common boundary of two adjacent polygons is represented only once). In addition, the model can serve as a very useful start in automating map generalisation (Haywood, 1988). Furthermore, it eliminates the double digitization of common boundaries. The model has served as the basis of many successful proprietary vector GIS, e.g., Arc/Info. Examples of topologic data models include the Geographic Base File/Dual Independent Map Encoding (GBF/DIME) model (U.S. Census Bureau, 1970), the Polygon Converter (Polyvrt) model (Peucker and Chrisman, 1975), the TIGRIS data model (Herring, 1987), the formal data structure (Molenaar, 1989) and the Authoritative Topographic-Cartographic Information System (ATKIS) data model (Hesse and Leahy, 1990).

Many of the existing topologic data models are simplified such that two objects of the same type (e.g., two area objects) do not partially or fully coincide in space. But many GIS users are concerned with the management of overlapping objects of interest in a single structure of a vector-based GIS. In other words, it is often of interest to accommodate different views, or layers, of the same geographic space, as in questions such as "which parcels (represented as distinct area objects) have a soil type Y (with soil units also represented as distinct area objects)?" This *spatial coincidence* problem is usually handled by an overlay computation after organizing each of the two layers separately using any topologic data model.

When the frequency of the multi-layer analysis is limited, this solution may be acceptable. However, in proprietary systems, it has not been practical to manage overlap by using different layers, often because of the frequency of overlap and the irregular way in which it occurs (Herring and Pullar, 1993). Alternative methods of handling the problem are (see also Hoop et al, 1993): (i) to separate the metric information of the layers and precompute (by overlay) and store topologic relationships among the objects within and across the layers, as in the "geographic region" data type in Arc/Info (Roessel and Pullar, 1993), (ii) to share metric information and separate topologic relationships (i.e., topology is explicitly recorded per layer) as in the ATKIS data model and (iii) to combine all the layers in a single topologic model, as proposed in this thesis, in order to derive the benefits of an integrated model (see chapter 3).

Most proprietary systems still handle this spatial coincidence problem by overlay computations. The solution of Arc/Info to the problem is still limited to overlapping two-dimensional object types (see Roessel and Pullar, 1993). A new feature type, called geographic region, has been defined in the extended model of the system (as reported in the above-mentioned reference) to handle overlapping polygons as well as disjoint polygons with identical attributes as a single region. At present, this solution has not been extended to cover overlapping point and line object types, which are the two other geometric object types commonly defined in a 2D topologic data model. In addition, the geometric data redundancy is not solved. The ATKIS data model provides a mid-way solution for all the three object types by sharing metric information, but topologic relationships among overlapping objects have to be derived computationally. This model is summarised in the following section.

The ATKIS data model

ATKIS is the result of a multi-stage research and development project of the Federal Republic of Germany's State Survey Working Committee (Adv) in the fields of state survey, cartography and automation (Hesse and Leahy, 1990). The ATKIS data model describes the data elements and their relationships within the ATKIS database as required for the description of the landscape and their representation in maps. It consists of a digital landscape model (DLM) and a digital cartographic model (DKM). The former represents topographic objects and the surface of the landscape; it is thus the primary model of the system. The DKM is a secondary model consisting of objects derived from the DLM, together with the cartographic generalisation and symbolisation rules for graphical output from the system. The DLM is therefore of more interest in this review.

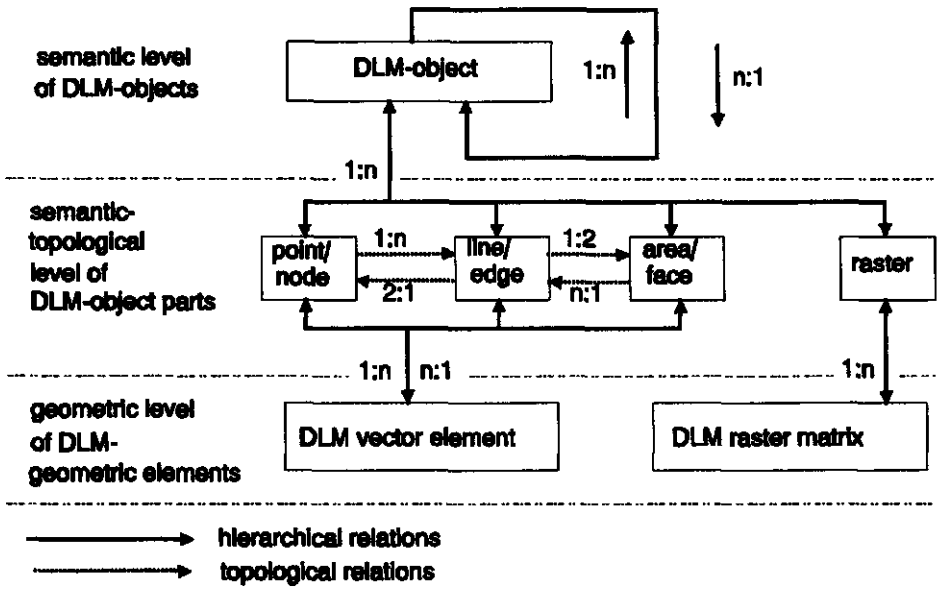


Figure 2.4 The ATKIS DLM Data Model (Hesse and Leahy, 1990)

The structure of the ATKIS DLM data model (see Figure 2.4) has three levels of data elements describing three important information aspects (Hesse and Leahy, 1990):
- the semantic aspect of the DLM objects,
- the topologic information aspect of the DLM object parts and
- the geometric information aspect of the DLM geometric elements.

The possible relationships among these data elements are indicated in Figure 2.4. The model represents real world objects by means of DLM complex objects, objects, object parts, and vector elements/raster matrices. Being essentially a 2D data model (elevation may however be incorporated, usually in a separate structure), the DLM objects are geometrically classified into object types point, line and area (for vector representation) and raster (for raster representation). Complex objects can then be constructed by means of references to the DLM objects constituting them. The DLM object parts are formed according to semantic and/or topologic criteria. They can be of the geometric types area/face, line/edge, point/node or raster and can have attributes, like a DLM object.

Using object parts makes it possible to represent objects (e.g., a stream) by means of object parts (e.g., sections along the length of the stream) having different values of certain attributes (e.g., width). The object parts constituting an object can differ in their geometric representation. The DLM geometric elements that describe the geometry of a DLM object part are bound to the object part by references, while every object part is specified by means of a reference as being owned by a certain DLM object. The actual bearers of the geometric information are the two geometric elements DLM vector element and DLM raster matrix. The vector elements describe lines in terms of the coordinates of points linked by different, selectable interpolation types. In addition to its horizontal coordinates, the elevation of a point may also be specified. Geometric data redundancy is reduced because any geometric element can belong to a number of object parts (i.e., 1:n); a geometric element is defined once. It is thus possible to represent spatial coincidence among objects of the same type with this model at the geometric level.

Note that one object part defines only one DLM object, while one DLM object is defined by one or more object parts (i.e., one-to-many relationship between object and object parts). Thus to represent spatial coincidence, each of the overlapping objects will have its own unique object parts even if the object parts are defined by the same geometric element. This means that if this model is used to represent terrain objects from two or more map layers, the vertical topologic relationship (i.e., among objects from different layers) can be retrieved only by computational means (e.g., by first selecting the object parts of the objects and comparing their geometric elements). In other words, spatial coincidence is implicitly modelled and the benefit of explicit representation whereby the overlapping parts can also be treated as new objects with separate additional attributes cannot be realised with the model. In addition, the hierarchic many-to-many relationship between geometric elements and object parts can introduce complexity during updating operations especially with respect to integrity maintenance.

The intention in this research is to select a generalised topologic model (that is capable of handling spatial coincidence among objects of the same geometric type) as the basis for formalizing automated database updating and consistency management in vector-based GIS. Mainly for the reasons stated above, the ATKIS data model cannot be selected. Another existing topologic model which has a rich syntax and semantic structure (Webster and Omare,

1991; Hoop and Oosterom, 1992) is the formal data structure (FDS) for single-valued vector maps (Molenaar, 1989). The model does not support spatial coincidence but it can be generalised to be able to handle this. A review and extension of this model is presented in chapter 3. Another variant of the extension of the FDS to accommodate a multi-valued terrain description, proposed in Hoop et al (1993) will also be reviewed in the next chapter.

2.2 Database Structures

An appropriate database structure, or logical data model, is required to organize the data for storage and retrieval in the computer after developing (or selecting) the most appropriate data model. Until the recent advent of the object-oriented data structure, three main kinds of database structures were commonly recognised (Burrough, 1986; Date, 1990). These are the network, hierarchic and relational database structures. The three types are reviewed in the following subsections, with emphasis on the relational data structure. More details can be found in, for example, Date (1990) and Burrough (1986). The new addition to the common database structures, the earlier mentioned object-oriented data structure, is separately reviewed in §2.3.

2.2.1 Network Data Structure

The network structure consists of two sets (Date, 1990), a set of records and a set of links, i.e, a set of multiple occurrences of each of several types of record, together with a set of multiple occurrences of each of several types of link. Each link type involves two record types: a "parent" record type and a "child" record type. Each occurrence of a given link type consists of a single occurrence of the parent record type, together with an ordered set of multiple occurrences of the child record type. For a particular link type L with parent record type P and child record type C, each occurrence of P is the parent in exactly one occurrence of L, and each occurrence of C is a child in at most one occurrence of L. In other words, in the network structure, a given record may have any number of immediate superiors (parents) as well as any number of immediate dependents (children), thus allowing for a direct representation of a many-to-many link type among data types.

The structure is more flexible than the relational and hierarchical structures since it can accommodate many-to-many relationship which often occurs in applications. However, it is more complicated and the database is enlarged by the large number of pointers, which must also be updated every time the database is updated, thereby increasing the overhead cost (Burrough, 1986).

2.2.2 Hierarchic Data Structure

This is a special case of the network structure. In it, the data are represented by a simple tree structure where one entity is "superior" (parent) and others "dependent" (children), giving a one-to-many relationship. The record type at the top of the tree is usually known as the "root" and a root may have any number of dependents, each of which may have any number of lower-level dependents. In other words, it is an ordered set consisting of multiple *occurrences* of a single *type* of tree (Date, 1990). A *tree type* consists of a single "root" record type, together with an ordered set of zero or more dependent (lower-level) subtree types and the

subtree type in turn consists of a single record type (the root of the subtree type) together with an ordered set of zero or more lower-level dependent subtree types, etc. Each *tree occurrence* consists of a single root record occurrence, together with an ordered set of zero or more occurrences of each of the subtree types immediately dependent on the root record type, and each of the subtree occurrences in turn also consists of a single record occurrence (the root of the subtree occurrence) together with an ordered set of zero or more occurrences of each of the subtree types immediately dependent on that root record type, etc. Naturally, no child is allowed to exist in the structure without its parent.

The structure naturally models truly hierarchic spatial objects and it is easy to understand and expand. However, it has some disadvantages, which include lack of flexibility to accommodate retrieval requests involving different trees, maintenance of large index files, data redundancy caused by the fact that certain attribute values may have to be repeated many times, and updating problems caused by the dependent occurrence of data.

2.2.3 Relational Data Structure

The relational model of representation has been most widely accepted because of its simplicity and the availability of a standard language (the structured query language (SQL)) for the manipulation of the database. Various commercial relational systems are available in the market, e.g., DB2, Oracle, Ingres, etc. In the structure, the data are organized in a single uniform manner: in the form of relations. It is characterised by such formal terms as relation, tuple, attribute, cardinality, degree, primary key and domain (the terms are schematically described in Figure 2.5) which respectively mean -informally- table, row or record, column or field, number of rows, number of columns, unique identifier, and a pool of legal values. Each relation consists of a number of tuples and a fixed number of attributes with each attribute having a unique domain of values. In other words, a relation is a collection of tuples (rows), each of which contains values for a fixed number of attributes (columns). Thus given a set of domains S_1, S_2, ..., S_n (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples, or simply tuples, each of which has its first element from S_1, its second element from S_2, etc., i.e., R is a subset of the cartesian product $S_1 * S_2 * ... * S_n$. This gives a relation of degree n, called n-ary relation (for example, a relation of degree 1 is called unary relation and that of degree 2 binary relation).

A relational structure has certain characteristics which distinguish it from traditional computer files (Freiling, 1982; Date, 1990); these are:
(a) duplicate tuples are not permitted
(b) each tuple/attribute (row/column) intersection within each relation must contain a single data field, i.e., multiple values are not allowed
(c) no ordering of tuples within a relation is assumed
(d) no ordering of columns within a relation is assumed
(e) no component of a primary key may be null.

Any relation that has the above-mentioned characteristics is said to be normalized (actually, in the first normal form). This means that each relation in the structure must be normalized. Thus normalization (informally, the process of converting a table with repeating attribute values in one or more columns to one with atomic attribute values per column) is one of the important considerations for the structure. If the relations are not normalized, the structure

will be vulnerable to major problems, i.e., addition, deletion and update anomalies as well as unnecessary data redundancy. Inherent benefits have in fact been claimed for relational structures that are fully normalized (Kent, 1983; Smith, 1985; Roessel, 1986). However, a fully normalized structure, apart from slowing down retrieval, also introduces referential integrity problems which the designer of the database must provide rules to control. Data concerning a single object are spread over many tables, thus making the structure liable to integrity violations.

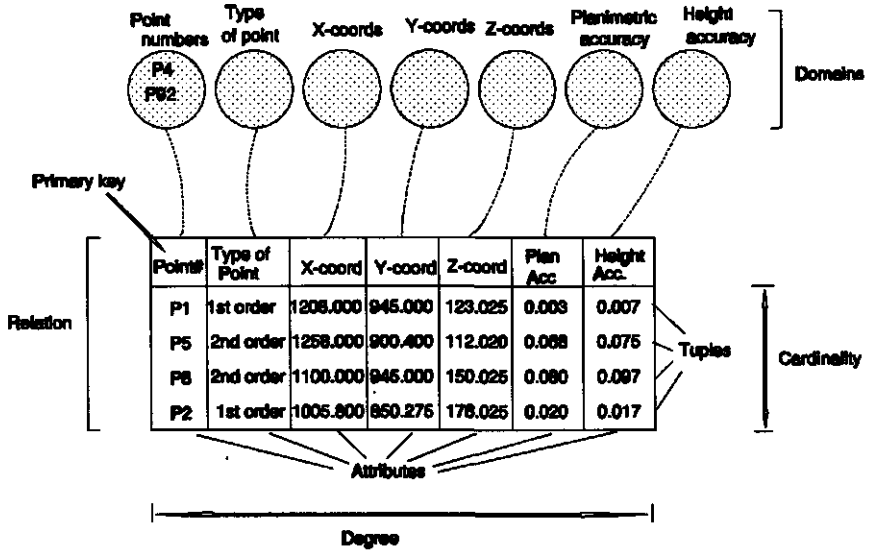


Figure 2.5 Key Words in Relational Data Structure

Five normal forms are usually defined for a relational structure and a relation that is in the fifth normal form is said to be fully-normalized. A description of the five normal forms can be found in Kent (1983) and Date (1990). The most common method of composing fully normalized relations is the non-loss decomposition procedure (Date, 1990), which progresses successively from the first normal form (1NF) to the highest normal form (usually the fifth normal form (5NF)) but satisfying the condition that if join operations are performed, the preceding normal form can be derived without loss of data. The first normal form is usually constructed from an entity relationship diagram (Chen, 1985) of the application.

Another method for composing fully normalized relations was developed by Smith (1985). It has been acknowledged that Smith's approach is less tedious than a non-loss decomposition and yields a fully normalized structure in a rather straightforward manner (Roessel, 1985). This approach will be used in chapter 7 of this thesis to design a prototype relational structure for vector maps and is therefore summarised in the following subsection.

Smith's Normalization Procedure

This method is based on two important concepts of single-valued and multi-valued dependencies among data types (these two terms should not be confused with the terms as used to describe vector maps where they indicate the relationship between a geometric primitive and an object, see chapter 3). A single-valued dependence from P to Q (P and Q are either single or composite data fields) exists if, at any time, a fact about P determines one fact about Q. Each value of P must then be non-null and unique, but Q may contain null or duplicate values. A multi-valued dependence exists if, at any time, a fact about P determines a set of facts about Q. P and Q must both be non-null, and each combination of P and Q must be unique. A single-valued dependence is diagrammed by a single headed arrow from the prime-key (e.g., P) to the target field (e.g., Q) while a multi-valued dependence is diagrammed with a double headed arrow.
Four major steps are involved in this approach (Smith, 1985), namely
(1) identification of the data fields to be represented in the structure
(2) listing of the dependencies among these data fields as dependency statements
(3) construction of a rigorous dependency diagram from the dependency statements
(4) composing relations from the completed dependency diagram.

Identification of data fields: In this step, all the data types (including attributes) and the relationships among them are identified. This can be described using an entity-relationship diagramming convention, for example. However, if a conceptual data model has been predefined, the data types and relationships defined in the model, as well as the attributes of the data types, become the data fields.

Guidelines for listing the dependencies: This step consists of the careful identification and listing of the functional relationships between a data type and all other data types that depend on it. Single-valued and multi-valued dependencies among the data fields must be well defined in the statements. Each data field in each statement must be atomic or non-decomposable to satisfy the first normal form automatically, and the name assigned to the field should be chosen such that it satisfies the rules of the computer software that will implement the database. In addition, an identification number should be assigned in ascending sequence to each dependency statement. Examples of dependency statements can be found in §7.1.2.

Guidelines for constructing the dependency diagram: As each dependency statement is written, field names are posted to a sheet of paper, enclosed inside bubbles (e.g., ellipses), and arrows are used to diagram the dependencies among the data fields. The arrowheads are shown only on one end of a line connecting two fields to show clearly which data field (prime-key bubble) determines the other (target bubble). Each field should appear only once in a connected diagram (interlinked bubbles) and the corresponding dependency statement number should be noted against the relevant arrow. As successive statements are written and diagrammed, earlier statements and diagrams may be revised, if necessary, as the overall data requirements of the application become clearer. Double or more bubbles are used to enclose a data field that participates in two or more chains. Transitive dependency occurring in a diagram must be corrected after correcting the corresponding statement. Figure 7.1 is an example of a dependency diagram.

Composing relations from the completed dependency diagram: Fully-normalized relations are composed from the completed dependency diagram by (a) working the single-valued dependencies into relations where the prime-key bubble(s) of a chain of linked bubbles becomes the primary key(s) of a relation, while all the target bubbles of that chain become other fields in that relation; the fields within the target bubbles become foreign keys of the relation if they also function as prime-keys of another chain or are tagged with a domain flag (a domain flag is used on the dependency diagram to identify data fields that share a common domain); (b) working the multi-valued dependencies into relations where each multi-valued dependency is composed into separate relations and fields within the end-key bubble, the prime-key bubble, and any/all uplink-key bubble in the same chain become the primary key(s) of one relation (an uplink-key field is a field on the dependency diagram that has a double-headed arrow pointing from it to a prime-key field, while the prime-key field points to other fields, and an end-key is a field having a double-headed arrow pointing to it while no arrow emanates from it); and (c) working the isolated bubbles into relations where fields within an isolated bubble become the primary key of one relation (an isolated bubble exists if one or more fields with a multi-valued dependency to one another are enclosed within one bubble and that bubble has no arrows pointing to or from it).

These four guidelines are applied in chapter 7 to design a prototype relational data structure for multi-valued vector maps. To take care of the referential integrity problems caused by the spreading of data concerning a single object over many tables during normalization, rules must be provided to control data consistency and database updating. This means that the conceptual data model on which the relational structure is based must have a well-defined set of consistency rules to enforce data consistency especially under updating operations. These rules are often called enterprise rules in the commercial database systems' environments (e.g., banking) where the relational DBMS first made its impact. Chapter 5 of this thesis provides such rules in the spatial domain, specifically in a vector-based GIS. And with the incorporation of the rules in the database updating algorithms proposed in chapter 6, a sound framework is thus provided for the implementation of a spatial relational DBMS.

2.3 Object-Oriented Data Modelling

The spatial data models reviewed in §2.1 concentrate mainly on the geometric aspects of spatial data. The non-spatial aspects are often modelled separately using a semantic data modelling approach and the two components linked by some kind of (object) identifier. The non-spatial component is often implemented in a relational structure, while the geometric component is handled by another structure (e.g, as in Arc/Info). The need for a unified representation of all the components of an object and the shortcomings of the traditional data structures, including relational, in spatial data handling (Alagic, 1989; Oxborrow and Kemp, 1989; Date, 1990; Hughes, 1991; Egenhofer, 1992), have led to the search for a more appropriate data structure for spatial applications. The reported shortcomings of the conventional data structures, especially relational, include (a) unacceptable performance when the database is populated with large quantity of data, (b) inadequate support for the treatment of complex objects such as spatial objects in GIS/LIS, and (c) absence of appropriate mechanisms for data structuring such that data concerning a single spatial object are not spread over different parts of the structure as in a relational structure.

With the development of object-oriented programming languages, popularly called "4th Generation Languages" (4GL) in computer science, implementation tools are now available for the implementation and enrichment of the abstraction mechanisms offered by semantic modelling, thus bridging the gap between the conceptual data model and the data structure and facilitating a unified representation of geometric and attribute data in a single structure. This new approach, called *object-oriented (OO)* data modelling, has been advocated for spatial applications whereby real world entities and their properties are modelled as objects in order to support the treatment of complex (geometric) entities. In this approach, the word *object* is used (in a wider sense than in topographic science where it is used as synonym for a terrain feature) for a single occurrence (instantiation) of data describing something that has some individuality and some observable behaviour (Egenhofer and Frank, 1989). In this section, some of the principal terms and concepts of the OO approach are introduced. These can be grouped into: (a) the modelling constructs which include the object/object identity and the four abstraction mechanisms: classification, generalisation/specialization, aggregation and association and (b) the implementation constructs which include inheritance, propagation, encapsulation, persistence, abstract data type, polymorphism and overloading.

Object/Object Identity:

In object oriented modelling, all conceptual entities are modelled as objects (Worboys, 1992). An object has a state and a behaviour. The state of an object is implemented through properties or attributes, but unlike a relational structure, such properties are not restricted to non-decomposable data types and may in fact be objects themselves. The behaviour of an object is implemented as a set of procedures (also called methods or operations) that are encapsulated with the properties within the object. Objects can be as simple or as complex as the application demands; more complex objects can be constructed from combinations of existing objects which can, in turn, be simple or complex objects. Every object has a unique identity which persists through time, although the properties of the object may change.

Classification:

Classification can be defined as the mapping of several objects (instances) to a common class (Egenhofer and Frank, 1989). The process of classification is central to the object oriented approach whereby all objects with similar properties and behaviour are grouped into object classes. In other words, all objects that belong to the same class are described by the same properties and have the same behaviour. Thus, instead of describing individual objects, the OO approach concentrates on the patterns of both state and behaviour that are common to an entire class of objects (Hughes, 1991). The class structure, encompassing both properties and behaviour, is therefore the natural unit of abstraction in OO systems and may be used to model both entity objects and relationship objects. This differentiates the OO approach from the extended entity relationship (EER) model in which entities are classified according to their structure, with no regard to their behaviour, and a separate concept, called the relationship type, is used to model relationships among the entities (Hughes, 1991).

Classification is often referred to as the *instance of* relationship because the individuals are *instances of* the corresponding class. For example, a GIS model of the city of Enschede will include the classes *building, street, park,* etc. Each of these classes will have a set of properties whose values will be evaluated for all instances of the class, in addition to a set

of behaviour (e.g., create, modify, destroy, etc.). For example, the properties of the class *building* may include address, owner, built-date, etc. A single building with the address 350 Boulevard 1945 is an object, i.e., an instance, of the class *building*; the values of other properties of the class will be recorded for this instance and all the methods defined for the class will operate on this object.

Generalisation/Specialisation:

Generalisation as an abstraction mechanism provides views of the same geographic space in different levels of details. Several classes of objects which have some properties and behaviour in common are grouped together to a more general class, called *superclass*. Thus the terms *subclass*, the converse relation of superclass, and *superclass* characterise generalization hierarchy in which objects are linked by **is a** relationship. Subclasses are object types which share all of the properties and behaviour of another class (the superclass) but which also possess more specific properties and behaviour not shared by the superclass; they therefore describe a *specialisation* of the superclass.

A generalisation hierarchy may have an arbitrary number of levels in which a subclass has the role of a superclass for another more specific class. The terms superclass and subclass are abstractions for the same object, and do not describe two different objects (Egenhofer and Frank, 1989). For example, all hotels in the city of Enschede may be grouped into the class *hotel* because they have some other common behaviour and properties (e.g., standard, usually expressed as number of stars) that distinguish them from other buildings. And because they do have all the properties and behaviour of a building, they are a subclass of *building*. Thus hotel **is a** building.

Aggregation:

This is an abstraction mechanism used for modelling composed objects whereby several objects are combined to form a semantically higher-level object. Each constituent object of the aggregation has it own properties and operations, and the operations of the aggregate are usually not compatible with the operations of the parts. The properties of the aggregate are derived by *propagation* from the properties of the constituent objects. Thus the aggregation abstraction is used to build complex objects from elementary objects in a bottom-up fashion (Molenaar, 1993) i.e., starting from the elementary objects, composite objects of increasing complexity are constructed in an upward direction.

The aggregation hierarchy is often expressed as a **part of** relationship because the constituent objects are part of the aggregate. The inverse relationship is often called **consists of**, i.e., the aggregate consists of some constituent objects. In the hierarchy, a constituent object can be part of more than one aggregation hierarchy. For example, in a model of a city, a complex object *residential district* may be defined as an aggregation of objects *houses, roads,* and *parks* which are selected according to some rules. Thus, as expressed by Molenaar (1993), the generic model for aggregate objects will consist of two components: (a) indication of the classes of objects that can be aggregated into a composite object and (b) rules to select the objects from the classes (indicated in (a)) for a particular aggregate object.

Association:

This is a form of abstraction whereby a relationship among two or more independent objects is considered as another object. The term **set** is often used to describe the abstraction and the associated objects are said to be **members of** the set. The details of a member object are suppressed and properties of the set object are emphasised at that level of abstraction. Unlike aggregation and generalisation, association does not build hierarchies and does not follow strict rules (Molenaar, 1993); it only indicates a set of objects that have something in common. For example, the International Institute for Aerospace Survey and Earth Sciences (ITC) and the University of Twente are associated by the relationship *inside Enschede*. In terms of terrain objects in general, topologic relationships among the objects are examples of association.

Inheritance:

In generalisation hierarchies, the properties and methods (operations) of the subclasses depend on the properties and structure of the superclasses. Properties that are common to a superclass and its subclasses are defined only once at the superclass level; the properties are then transmitted to all the objects of the subclasses. This transitive transmission of properties from one superclass to all related subclasses, and to their subclasses, etc., is termed **inheritance**. It is a powerful concept in an object-oriented system because it reduces data redundancy (Woelk, 1987). It supports modularity and helps in maintaining integrity since essential properties of an object are defined once and are inherited at other lower levels in which it (the object) takes part (Egenhofer and Frank, 1989).

Operations of the superclass are applicable to all objects of the subclass because each object of the subclass is also an object of the superclass; but operations which are specifically defined for a subclass are not compatible with superclass objects. The inheritance relation can be single or multiple. In **single inheritance**, a strict generalisation hierarchy is defined whereby each class has at most a single immediate superclass. **Multiple inheritance**, on the other hand, permits one subclass to have more than one distinct immediate superclass.

Propagation:

This is the mechanism used in aggregation hierarchies and association to derive values respectively for complex objects (aggregates) and associated objects (set) from the constituent objects. It supports complex objects which do not own independent data and is based on the concept that values are stored only once, for the properties of the component objects which are then propagated to the properties of the aggregates or associated objects when required. For example, the population of a city is the sum of the populations of all the districts that are part of the city.

Apart from the propagated values, the composite object can have property values which are specifically owned by it and distinct from those of its components. Propagation works in a bottom-up manner as against the top-down transmission in inheritance. It helps to maintain consistency because the dependent values of the aggregate are derived and need not be updated every time a change is made to any of the components - only the constituent objects need be changed.

Encapsulation:

The concept of encapsulation is very important in the context of object-oriented programming languages (Oxborrow and Kemp, 1989). Support for encapsulation of data and operations in objects enables objects to be defined completely, both in terms of their properties and in terms of their behaviour. Thus, for example, if certain actions have to be taken, or rules applied, or constraints checked, these actions, rules or constraints can be built (encapsulated) into the definition of the object. The encapsulation concept, when fully applied, means that the only operations that can be performed on an object of a certain class are those which are declared in the definition of that class or inherited from its superclass(es).

Persistence:

This means the permanent storage and maintenance of objects which have been created. It is a concept that has been added to the object-oriented programming language to distinguish it from conventional programming languages (in which the data created by a program exist only during the execution of that program) and thus have some functionalities of a conventional DBMS. Persistence is a normal feature in conventional DBMS which permanently store created data in the database.

Polymorphism and Overloading:

These are terms used by computer language specialists in describing some important and powerful aspects of computer languages, which have consequently been introduced in the implementation of object-oriented systems. Both relate to the number of ways a name can be used to represent an object or function. **Polymorphism** means that a name (variable) may represent at different times different classes of object. **Overloading** is a related concept which refers to the multiple functions that a function or operator may represent, depending on the types of the operands. For example, the function invoked by the "+" operator depends on whether the types of the operands are floating point, integer, complex, etc.

2.4 Some Implementation Issues Concerning Object-Oriented Data Structure

Despite the acclaimed suitability of the OO approach for spatial applications, most proprietary systems that claim to be object-oriented at the moment are built on a relational structure because of the lack of a standard object-oriented query language. This means that the relational structure will continue to play a significant role in database management either in its conventional form (as a main DBMS) or as a base for an object-oriented system. Stonebraker (1994) emphasised the need to harmonise the benefits of the relational and the object-oriented approaches to have a powerful database system. According to him, the object-oriented DBMS vendors have focused on tactics that will best support the applications having complex data but requiring no query capability, just as the relational DBMS vendors concentrated on query capability for simple data. The data are said to be simple if they can be described using the base data types of the SQL, such as integer, character, date and floating point; they are complex if they cannot be directly described individually by any of the base data types unless by artificial decomposition.

To have the query capability in an object-oriented system that is comparable to the SQL, in addition to the ability to handle complex data, the object-relational DBMS has been recommended. Stonebraker (1994) defined an object-relational DBMS as one which adds the following concepts (which include OO concepts) to SQL:
- unique identifiers (i.e., implementation of object identity);
- user-defined types
- user-defined operators
- user-defined access methods
- complex objects
- user-defined functions
- overloading
- dynamic extendibility
- inheritance of both data and functions
- arrays

These concepts are very useful in GIS applications. For instance, the user-defined types can be used to define such types as point, line and polygon in a spatial DBMS; the user-defined operators will be useful to provide metric spatial operators, such as area, distance and direction, and topologic operators for deriving topologic relationships between two spatial objects and the support for user-defined access methods is essential for efficient searching and data retrieval in the usually large database. Also, the complex objects will be useful in GIS for the representation of terrain objects of different shapes and sizes, while the inheritance of both data and functions concept will be useful in the implementation of a generalisation hierarchy.

An example of this DBMS is the (public domain) Postgres DBMS (Postgres, 1994) whose commercial version is called Illustra (Illustra, 1994). The Postgres DBMS is used to test the prototype object-oriented data structure proposed in this thesis (see chapter 8).

2.5 Summary

This chapter has focused on the review of spatial data models and structures for GIS. The geometric models of spatial data are classified into two groups: the tessellation models and the vector models. Examples from the two groups are given, with more emphasis on the vector data models, being the choice in this thesis. The vector approach was chosen because it is closely related to the more accurate data acquisition methods (e.g., land surveying and photogrammetry) for large- and medium-resolution spatial databases which are very vital in many GIS applications (e.g., cadastral applications and urban planning). In the domain of conventional database structures, the common ones, including relational, and a procedure for designing the relational structure were also described. The object-oriented data modelling approach was also reviewed, including aspects regarding its implementation which have given rise to the call for the use of an object-relational system instead of a "pure" object-oriented system.

The review made in this chapter is of course not exhaustive; it was limited to aspects that are more relevant to this thesis and the review of some aspects are also deferred to be treated under the relevant chapters. The review of the spatial data models indicated that a lot has been done in the field of spatial data modelling but it also shows the need for a generalised

spatial data model that can incorporate the explicit representation of spatial coincidence among terrain objects for an integrated spatial analysis. The database structures defined the logical representation of data types and relationships defined in a conceptual data model and provide concepts for minimizing inconsistencies (e.g., by using the propagation concept in the object-oriented structure, and by normalization in relational) but, being generic database structures, they cannot indicate the constraints to be enforced and how to enforce them. This means that the conceptual data model on which the database structure is based must have a well-defined set of consistency rules to enforce data consistency especially under updating operations. (These rules are often called enterprise rules in the commercial database systems' environments (e.g., banking) where the relational DBMS first made its impact.) Chapter 5 of this thesis provides such rules in the spatial domain, specifically in a vector-based GIS. And with the incorporation of the rules in the database updating algorithms proposed in chapter 6, a sound framework is thus provided for the implementation of a spatial DBMS using any database structure.

In the next chapter, one of the existing vector data models is selected and reviewed, and then extended to be able to handle geometric integration of spatial data belonging to one or more mapping themes. The object-oriented data modelling concepts described in §2.3 are applied in chapter 7 to translate the extended model to a prototype object-oriented data structure. In the same chapter, a prototype relational structure for the same model is designed using the normalization method summarised in §2.2.3.

3

CONCEPTUAL DATA MODEL FOR VECTOR MAPS

In mapping sciences and GIS, different applications normally view terrain situations differently, thereby extracting only those terrain phenomena which play definite roles within those applications. For example, a cadastral surveyor will partition a given region into land parcels with each parcel having its unique attribute values. The same region will be partitioned by a soil scientist into different soil units. This implies a "layered" view of a terrain situation. However, in spatial analyses and planning it is often necessary to integrate different views of the world, in which case (part of) a terrain object may play more than one distinct role in the same database. In other words, two or more objects of the same type (point, line or area) from different mapping contexts or map layers may spatially coincide. Here we use the term *map layer*, or simply layer, to denote a geographic dataset describing a certain aspect of the real world (Hoop et al, 1993) i.e., the set of objects belonging to the same mapping context (cadastral, soil mapping, etc). A vector structure which incorporates spatial data from a single map layer (e.g., cadastral mapping, soil mapping or land use mapping) can be described as a single-valued vector map. When the structure incorporates data from one or more layers, it is described as multi-valued vector map.

In this chapter, a conceptual data model for multi-valued vector maps is defined by extending the formal data structure for single-valued vector maps. The mathematical tool for the modelling is provided by graph theory. The relevant elements of the theory are summarised in §3.1, where aspects of simplicial complexes are also given because of their similarity to some geometric data types used in the conceptual model.

The choice of the FDS for the conceptual modelling instead of other topologic models was based on the preference for an object-based model which is generic, flexible and extendible. In the 2D FDS, terrain objects play a central role in the terrain description; each object has a thematic component and a geometric component (see Figure 2.2). In the thematic domain, the objects can be grouped into thematic classes in which each class has a specific attribute structure. The geometric component of a terrain object is clearly distinguished into three independent aspects, namely topology, shape and size, and position (see Figure 2.3). The clear distinction of the geometric aspects of terrain objects not only facilitates the construction of a semantically-rich, query-oriented spatial database, it also leads to an extendible and flexible data model. For example, having distinguished the semantic characteristics of terrain objects into thematic and geometric, it follows that the geometry of the same terrain situation can be represented either by vector elements (arc and node) or by raster elements (see Molenaar and Fritsch, 1991; Molenaar and Janssen, 1992) leading to flexibility in the choice of system configuration for its implementation and in data exchange.

The clear articulation of the geometric aspects of an object also facilitates key modelling decisions. (1) It becomes possible to decide on the dimension of the metric space (whether 2D or 3D) independent of the dimension of the topologic space (which also helps our appreciation of what a 3D spatial data model should be). (2) It also helps in deciding at which level to integrate geo-data from multiple map layers, i.e., at geometric level or at thematic level. At the geometric level (the choice in this thesis), it becomes possible to distinguish four

different approaches to the geometric integration by considering how metric (positional) data and topologic data of the different layers are handled. The four possibilities are (i) to structure each layer separately, i.e., combining metric and topology per layer and perform an overlay of the layers when necessary; (ii) to structure the geometric data such that all layers share the metric dataset while topology is kept per layer; vertical topologic query will then be done by overlay computation or by comparison of metric data; (iii) to structure the geometric data such that all layers share a common topology, while the metric information is structured per layer; and (iv) to define a model in which both metric data and topology are shared by all layers as proposed in this chapter.

The FDS is summarised in §3.2, followed by a discussion in §3.3 of the different approaches for modelling a multi-valued terrain situation and a review of related work on the modelling of multi-valued vector maps. In §3.4 the FDS is extended to incorporate a multi-valued terrain situation. In the last section of the chapter, the integration of the proposed data model and a DTM is described. The integrated model can be used for the establishment of a multipurpose vector GIS incorporating multi-layer object data and terrain relief information with minimum redundancy. Here a vector map refers to a database representation of terrain objects as points, lines, surfaces (areas), and bodies (volumetric objects) in which positional data of the objects are given in form of coordinates. In a 2D representation, only the first three types are present. The fourth exists as an additional type in 3D representation.

3.1 Graph Theory and Simplicial Complexes

3.1.1 Elements of Graph Theory[1]

A graph is defined abstractly as a pair (V,E) in which V is the non-empty finite set of *vertices* of the graph and E is a finite family (permitting the existence of repeated elements) of unordered pairs of (not necessarily distinct) elements of V, called *edges*. The graph is said to be *directed* if V is a non-empty finite set and E is a finite family of ordered pairs of elements of V.

Hereafter, only the directed graph, or simply graph, will be referred to and an *edge* will be called *arc* and a *vertex* will be called *node*. Also, it is assumed that E contains distinct elements and the two elements of V that define an element of E are not equal. Thus a graph is a collection of two sets:
a set of nodes $N = \{n_1, n_2, ...\}$, and
a set of arcs $A = \{a_1, a_2, ...\}$
in which for each $a_j \in A$ is $a_j = \{n_p, n_q\}$ where n_p and $n_q \in N$ and $n_p \neq n_q$.
Figure 3.1a is an example of a graph with N = {p,q,r,s} and A = {(p,q), (q,r), (r,s), (s,p), (p,r)}.

Let G(N,A) be a graph. A graph $G_b(N_b,A_b)$ is a subgraph of G if A_b is a subset of A and N_b is a subset of N such that the arcs in A_b are incident only with the nodes in N_b.
Figure 3.1.b shows a subgraph of the graph in Figure 3.1.a

[1] The materials used in this section are taken from Wilson (1990) and Liu (1986).

Two nodes of graph G are said to be *adjacent* if there is an arc joining them; the nodes are then said to be *incident* to that arc. Also, two arcs of G are *adjacent* if they have one node in common. The *degree* of a node n is the number of arcs incident to n. A node of degree zero is called an *isolated node* and a node of degree one is an *end-node*. In the graph of Figure 3.1.(b), for example, the degree of node q is two and that of node p is one (therefore p is an end-node).

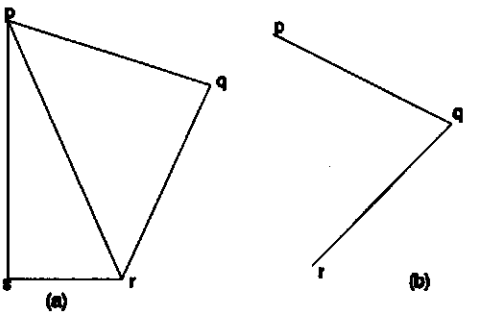


Figure 3.1 Examples of a graph

In a graph, a *path* or *walk* is a sequence of arcs $(a_{i1}, a_{i2},....,a_{ik})$ such that the terminal node of a_{ij} coincides with the initial node of $a_{i(j+1)}$ for $1 \leq j \leq k-1$. The path is said to be *simple* if it does not include the same arc twice. It is *elementary* if it does not meet the same node twice (i.e., no two arcs in the sequence have the same terminal node). The *length* of a path is the number of arcs it contains. A graph is said to be *connected* if there is a path between every pair of nodes, i.e., if there is a path from any node to any other node; otherwise it is said to be *disconnected*. A graph whose set of arcs is empty is called a *null graph*. The graph is said to be *planar* if it can be drawn on a plane so that no arc crosses another (i.e., the two crossing arcs are decomposed into four arcs meeting at a common node).

In a planar graph, the arcs of the graph divide the plane into planar segments called *faces*; thus a face is an area of the plane that is bounded by arcs and is not further divided into subareas. The relationship between the number of nodes (n), arcs (a) and faces (f) - including the outerface - that must exist for a graph to qualify as planar is given by Euler's formula as follows:

$$n - a + f = 2$$

This formula holds for only connected planar graphs but it can be extended to include disconnected graphs (Wilson, 1990) to have

$$n - a + f = k + 1$$

where k = number of component graphs. Also, in a planar graph, the sum of the degrees of all nodes must be an even number and twice the number of arcs.

The concepts summarised above are applied in the definition of the FDS and the data model for multi-valued vector maps proposed in this chapter in which arc and node serve as primitives in the geometric representation of terrain objects. The use of the third element of a planar graph, i.e., face, as a geometric primitive is not necessary in 2D terrain description because, since planarity is enforced, every arc will have a face on either side, and since a face will always represent a 2D terrain object, the arc can reference the object directly (which is not so with arc and node: an arc may represent a linear object or just the boundary of an area object and a node may represent a zero-dimensional object or just the end-point of an arc).

3.1.2 Simplexes and Simplicial Complexes

In §3.4, a geometric data type, called *m-container*, will be introduced for the explicit modelling of spatial coincidence among objects in a vector map. In a mathematical sense, the m-containers have some similarities to n-simplicial complexes, although the latter terms are not used in the model because of some semantic difference between the two, as will be explained in §3.4.2. The definitions of simplicial complexes (Giblin, 1977; Worboys, 1992) are therefore relevant and are given here. The definitions are given for the case of spatial objects embedded in the Euclidean plane, i.e., for cases of up to two dimensions. *Simplicial complexes* are amalgamations of basic building blocks, called *simplexes* (Worboys, 1992) in which an *n-simplex*, $n \in \{0,1,2\}$ is defined as follows:

0-simplex: A set consisting of a single point in the Euclidean plane.
1-simplex: A set consisting of all the points on a straight line between two distinct points in the Euclidean plane, including the end points.
2-simplex: A set consisting of all points on the boundary and in the interior of a triangle whose vertices are three non-collinear points.

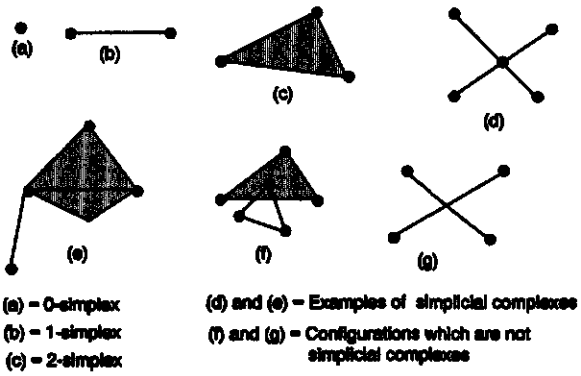


Figure 3.2 Simplexes and simplicial complexes (Worboys, 1992)

Thus an *n-simplex* is the convex hull of a set S of n+1 linearly independent points, $p_0,...,p_n$, say. Such a simplex is denoted $\langle p_0,...,p_n \rangle$ or <S>. Furthermore, the integer n is defined to be the dimension of <S>. Given $T \subseteq S$, then the convex hull of T is itself a simplex contained in <S> and called a *face* (not to be confused with the face in graph theory) of <S>; this is expressed as $\langle T \rangle \leq \langle S \rangle$. Simplexes serve as building blocks for larger structures.

A *simplicial complex*, C, is then defined as a finite set of simplexes satisfying the following properties:

1. A face of a simplex in C is also in C.
2. The intersection of two simplexes in C is either empty or is a face of both simplexes.

The dimension of a simplicial complex is the maximum dimension of its constituent simplexes. In a 2D space, we have *0-*, *1-* and *2-simplicial complexes* which are defined as follows:

- A *0-simplicial complex* is a complex of dimension zero.
- A *1-simplicial complex* is a complex each of whose maximal simplicial components is a 1-simplex.

- A *2-simplicial complex* is a complex each of whose maximal simplicial components is a 2-simplex.

Examples of simplexes and simplicial complexes are given in Figure 3.2. They are often used for the geometric modelling of spatial objects (see for example Worboys, 1992) in which a 0-simplicial complex corresponds to a point, a 1-simplicial complex corresponds to an arc or edge and a 2-simplicial complex corresponds to a polygon, after imposing some constraints.

3.2 Formal Data Structure (FDS) for Single-Valued Vector Maps

The FDS is an object-based terrain description in which the analysis of the terrain is based on distinct objects such as areas with well defined boundaries, linear terrain structures, and individual point objects. Only a summary of the model will be given here; comprehensive descriptions have been published elsewhere (see Molenaar, 1989; 1991c; 1993).

3.2.1 Summary of the Model

The fundamental information structure of the FDS is shown in Figure 2.2. It shows that distinct terrain objects, represented by object identifiers, have two semantic characteristics, namely (1) the geometric characteristics and (2) the thematic (non-spatial) characteristics. The former comprise topology, shape and size, and position (see Figure 2.3) while the latter indicate the non-spatial attributes of the objects.

To model the thematic component of objects, the terrain objects occurring in a vector map are grouped into several distinct classes according to their thematic characteristics. A list of attributes is connected to each class and the class is identified by a class label or name. The attribute list of a class gives the names of the attributes. The objects belonging to the same class will have a common attribute structure inherited from the class (see Figure 3.3a). Thus each object of that class will have a list containing a value for each attribute in the class attribute list; the values are taken from the value domains of the individual attributes.

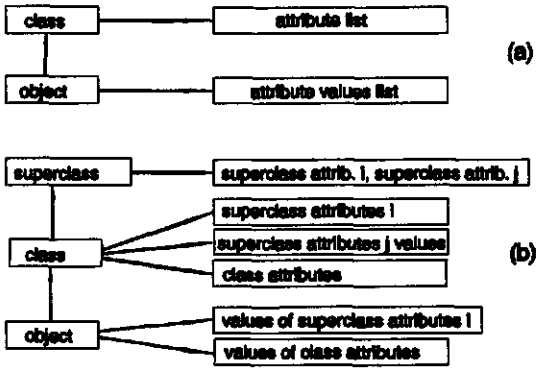


Figure 3.3 Class and superclass structure of objects

The FDS supports a class hierarchy in which classes that have some common attributes are grouped into a superclass, while superclasses having some common attributes are also grouped into a higher superclass, and so on (see Figure 3.3b). Each (super)class will have its own list of attributes. At the highest level we have the superclasses with their lists of superclass attributes which can be split as in Figure 3.3b, because some of these attributes (superclass attributes j) are evaluated at the next lower level, the class level, and some are evaluated at

the lowest level, the object level. At the class level, new attributes that are specific to that class will be defined as class attributes (in addition to the inherited superclass attributes). At the object level no new attributes are introduced, but all attributes introduced at the higher levels will be evaluated as far as they have not been evaluated at class level. If an attribute is evaluated at class level, it means that all objects belonging to the class do have the same value for that attribute. The class definition should be mutually exclusive, with each terrain object belonging to exactly one of the classes, i.e., the classes should be defined so that the classification system is disjoint and complete. Thus in the FDS, an object can belong to only one class and, for simplicity, all objects that belong to one class must be of the same type.

In the geometric domain, the object types points, lines and areas are distinguished for a 2D or 2.5D terrain description. In the model, the geometry of a terrain object is clearly distinguished in three independent aspects, namely topology, shape and size, and position (see Figure 2.3). This geometric dataset has been carefully structured in the FDS, leading to a semantically-rich, query-oriented and extendible data model in which information on topology, shape and size, and position can be retrieved. The three terrain object types are geometrically described by their linear characteristics using the two elementary geometric primitives: arc and node.

The mathematical framework for the geometric description of objects in a vector map is provided by graph theory (see §3.1.1). Thus the geometry of a vector map is represented by a planar graph $G(N, A)$ in which
$N = \{...,n_i,...\}$ is the set of nodes of the map, and
$A = \{...,a_j,...\}$ is the set of arcs of the map in which for each $a_j \in A$ is $a_j = (n_p,n_q)$ with n_p, $n_q \in N$; n_p and n_q are respectively the beginning and end nodes of a_j.
Thus each arc is directed and is a subset of N. Furthermore, each arc has an area object on its left side and one area object on its right side and may represent one line object. The geometry of each object occurring in the map is thus a subgraph $G_o(N_o,A_o)$ of G where $N_o \subset N$ and $A_o \subset A$.

Figure 3.4 represents the formal data structure for single-valued vector maps. In the figure, the classes are represented only by class labels; the class labels will serve as a link to the thematic descriptions when the thematic classes are identified during implementation.

3.2.2. Elementary Links Among Objects, Geometric Components and Thematic Components

Four categories of elementary links (functional relationships) among the three groups of information represented in Figure 3.4 have been identified and defined in the FDS:
(1) object to class links (*oc-links*)
(2) object to object links (*oo-links*)
(3) geometry to object links (*go-links*)
(4) geometry to geometry links (*gg-links*)
The four categories of link type will be explicitly represented in the database to facilitate unambiguous application, analysis of spatial relationships and other query operations in an FDS database. They facilitate the clear interpretation of the links represented in Figure 3.4

Object - Class Links (oc-links)

(1) Each point object belongs to a point object class.
(2) Each line object belongs to a line object class.
(3) Each area object belongs to an area object class.

Object - Object Links (oo-links)

(1) A point object may lie inside an area object.
(The explicit representation of this link is necessary because the relationship cannot be derived without the use of coordinate information.)
(2) A line object crosses another line object. (This is required in 2D topologic space to take care of linear objects crossing each other, e.g., a road crossing a river.)

Geometry - Object Links (go-links)

(1) One node may represent at most one point object.
(2) One arc may represent at most (part of) one line object.
(3) An arc has only one area object on its left side and only one area object on its right side.

The concept of the single-valued vector maps is emphasised by the three geometry-object links. The three links clearly indicate that there should be only one occurrence of a link between a geometric primitive and a terrain object. The first link shows that one node can represent only one point object but it does not mean that every node must represent a point object, i.e., this link may be empty for some nodes. The same explanation holds for the second link. The third link emphasises the fact that planarity must be enforced such that an arc will always have one area object on its left side and one area object on its right side. This link cannot be empty.

Geometry - Geometry Links (gg-links)

(1) Each arc starts from a node.
(2) Each arc ends at a node.
(3) Each arc has a shape (represented here as a straight line).
(4) Each node has a position defined by a pair of X and Y coordinates.
The first two links indicate that a node should not contain loops.

The FDS is indicated in Figure 3.4, in which the ellipses represent the elementary data types and the labelled arrows represent the elementary link types among them. Headed arrows represent a many-to-one link type in the direction of the arrow, e.g., many area objects may belong to the same area class, while non-headed arrows represent one-to-one relationships, e.g., one node defines only one point object. Each data type represents a set. The expression "elementary" means that these data types and link types cannot be further decomposed. In the diagram, the thematic data are represented by the class labels. The objects (identifiers) are represented by the object types, and the geometric data are the arcs, nodes, shapes and positions (coordinates).

3.2.3 Conventions of the FDS

The following seven conventions have been defined to complement Figure 3.4 in ensuring the consistency of an FDS database by defining rules for their monitoring and enforcement (see chapter 5).

(1) The object classes are mutually exclusive, i.e., each object belongs to exactly one class.
(2) Each class contains objects of only one geometric type.
(3) When a vector map is analyzed as a graph, all points that are used to describe the geometry will be treated as nodes.
(4) The arcs of this graph are geometrically represented by straight line segments.
(5) For each pair of nodes there is at most one arc connecting them directly; each of them may be connected also by one or more chains consisting of two or more arcs.
(6) Two arcs should not intersect; if they do intersect, they should be replaced by four arcs joining at a node.
(7) A node may represent at most one point object. An arc may be part of at most one line object. An arc has exactly one area object at its right-hand side and exactly one at its left-hand side.

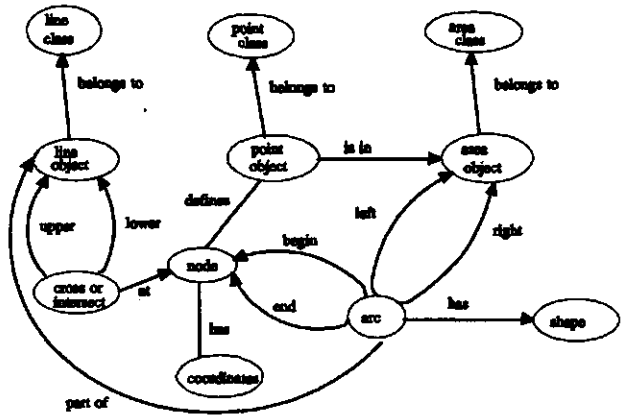


Figure 3.4 Diagram representing the FDS for vector maps

A map which has the FDS as shown in Figure 3.4 and fulfils these seven conventions is a single-valued vector map (Molenaar, 1989).

3.3 Handling Multi-valued Terrain Description

From convention 7 (§3.2.3), it is clear that two objects of the same type are not permitted to spatially coincide (overlap), and in order to generalise the model to accommodate representation of objects which are spatially coincident, the model should be extended. The clear distinction in the FDS of the geometric aspects of terrain objects into topology, shape

and size, and position gives room for the necessary extension. It facilitates key modelling decisions. (1) It becomes possible to decide on the dimensions of the metric space (whether 2D or 3D) independent of the dimensions of the topologic space. Here, the extended model is based on 3D coordinate space (position) and 2D topologic space, i.e., a 2.5D data model (see §3.4). (2) It also helps in deciding at which level to integrate geo-data from multiple map layers, i.e., at geometric level or at thematic level.

At the geometric level (the choice in this thesis), it becomes possible to distinguish four different approaches to the geometric integration by considering how metric (positional) data and topologic data of the different layers are handled. The four possibilities and their pros and cons are outlined below (see Hoop et al, 1993 for more details). Here, it is assumed that the layer is structured according to the FDS.

(a) The first, and conventional approach, is to structure each layer separately, i.e., combining metric and topology per layer and perform an overlay of the layers when necessary. This results in fast single-layer topologic query and the geometry of objects in a single layer can be reconstructed more easily. Updating is more straightforward and it is easier to manage data consistency within an individual layer. It is also easier to implement in existing vector-based systems.

However, the approach gives room for much data redundancy because common geometric elements are stored in every layer in which they occur. In addition, vertical topologic queries (queries involving objects from different layers) can be derived only by on-line map overlay computations (with the attendant editing problems arising from pseudo polygons and sliver lines), thus increasing costs.

(b) A second option is to structure the geometric data such that all layers share the metric dataset while topology is retained per layer. This eliminates metric data redundancy as well as the problems of sliver lines and pseudo polygons during overlay computations that may be required to answer vertical topologic queries. However, vertical topologic query can be realised only by on-line overlay computation or by comparison of metric data, and the reconstruction of the geometry of individual objects is slower. In addition, it is necessary to provide rules that will guarantee consistency of the metric data during updating (note that consistency rules are also required even in the single-layer approach but at a lower scale).

(c) Another option is to structure the geometric data such that all layers share a common topology, while the metric information is structured per layer. This leads to faster multi-layer topologic queries and faster single-layer metric queries. On the other hand, it leads to metric data redundancy and requires additional rules to guarantee consistency of the topologic data during updating. Moreover, topologic queries related to a single layer are slower to retrieve.

(d) The fourth approach is to define a model in which both metric data and topology are shared by all layers as proposed in the next section. This approach is mainly beneficial when frequent vertical topologic queries are envisaged. In addition, the reconstruction of the geometry of an object is slower. Because all layers are intersected, overlay computation will be performed even in areas that may not be required for multi-layer analysis. Moreover, updating procedure becomes more complex and consistency considerations are very critical.

But the approach has many advantages, which include the following.
(1) Elimination of redundant data: A single geometric dataset (positional and topologic) is kept for all layers.
(2) Reduction in overhead cost because map overlay is computed once during database creation; thus problems of spurious polygons, etc., are handled once.
(3) Faster multi-layer queries since it will not be necessary to compute an overlay before answering such queries.
(4) Higher information content, the knowledge of which is known a-priori; thus a query language can be predefined for retrieval of geo-information from the multi-valued database.
(5) Spatial consistency can be maintained at system level since only one data structure is used and only one set of geometric data is kept.
(6) If overlapping sections across the layers are uniquely identified, as proposed in §3.4, they can have their own geometric and non-spatial datasets. Therefore, they can be maintained and manipulated by the GIS functions just like single objects. Thus it is easier to include them in aggregation and association abstractions.

Since all options have their individual pros and cons, every solution will be a compromise. The choice of approach will therefore depend largely on the intended application. The fourth approach (d) is used in §3.4 to develop a data model for multi-valued vector maps by extending the FDS. This can then be used in applications that require frequent analysis of multi-layer geo-data in which there is a frequent occurrence of spatial coincidence among objects of the same type. An example of the second approach (b) is the ATKIS model which was reviewed in §2.1.2.

In the work of Hoop et al (1993), cited above, an example of the extension of the FDS to accommodate multi-valued vector maps was also given. The extension allows the geometric primitives arcs and nodes to have many-to-many links with the terrain objects so that objects from one or more layers can be represented in a single structure. The links are defined as follows. A node may represent one or more coinciding point features (in this thesis the term object is used instead). An arc may represent (part of) one or more (partly) coinciding line features. With n merged structure layers, an arc has n coinciding or overlapping area features on each side. A point feature may lie in n area features. A line feature consists of one or more arcs. An area feature consists of at least three arcs and may additionally contain several point features in its interior. The structure was designed for implementation in the Postgres database management system (see chapter 8 for information on Postgres); thus four groups of Postgres classes were defined for the implementation of a multi-valued vector map as follows (the classes defined for single-valued vector maps are excluded here):

Data Structure for Multi-valued Vector Maps as Defined by Hoop et al (1993):

```
/* PART 1: LAYERS */
        structure_layer (
                slayer_id = int4, /* primary key */
                single_valued = bool /* single or multi-valued */
                description = char[] )
        thematic_layer (
                tlayer_id = int4,   /* primary key */
                slayer_id = int4,   /* foreign key structure_layer */
                description = char[] )
```

```
/* PART 2: FEATURE CLASSES */
        area_feature (
                aid = int4,         /* primary key */
                tlayer = int4,      /* foreign key thematic_layer */
                area_class = char[] /* thematic info */ )
        line_feature (
                lid = int4,         /* primary key */
                tlayer = int4,      /* foreign key thematic_layer */
                line_class = char[] /* thematic info */ )
/* PART 3B: TOPOLOGICAL CLASSES, MULTI-VALUED */
        point_feature_multi (    /* also FEATURE CLASS */
                pid = int4,         /* primary key */
                tlayer = int4,      /* foreign key thematic layer */
                point_class = char[] /* thematic info */
                node_id = int4,     /* foreign key node */
                areas = int4[]      /* foreign keys area_features */ )
        arc_multi (              /* primary key: from_node & to_node */
                from_node = int4,   /* foreign key node */
                to_node = int4,     /* foreign key node */
                left_areas = int4[], /* foreign keys area_features */
                right_areas = int4[], /* foreign keys area_features */
                lines = int4[]      /* foreign keys line_features */ )
/* PART 4: METRIC INFORMATION */
        node (
                node_id = int4,     /* primary key */
                location = point    /* x,y-coordinates */ )
```

Like the ATKIS model, this structure does not explicitly model spatial coincidence among the overlapping objects so as to derive the sixth advantage listed under the fourth approach (d above). The many-to-many relationships that exist between the geometric primitives and the objects may also introduce integrity control problems when implemented in a pure relational system. In the next section, an alternative model is defined which explicitly models spatial coincidence among objects.

3.4 Data Model for Multi-valued Vector Maps

In this section, an extension of the FDS is proposed to represent spatial data from one or more map layers, i.e., multi-valued vector maps. The proposed model is a 2.5D (3D position, 2D topology) model that integrates both metric and topologic data of one or more map layers from the same geographic space. The extended model will be called *data model for multi-valued vector maps (DMMVM)*. Since positions of objects are defined in a 3D metric space but embedded in planar topology, only surfaces of objects are represented. The DMMVM is still based on the fundamental structure shown in Figure 2.2. How the thematic and geometric components of objects are handled in the DMMVM is shown in the following sections.

3.4.1 Thematic Component

The representation of the thematic aspects of objects remains essentially the same as in the FDS (see Figure 3.3); only a slight extension is required to accommodate multi-valued situations. In the FDS, each object can belong to only one class. This convention has been adopted here with respect to individual map layers. Normally, each layer has a distinct

domain of objects but sometimes an object may occur in more than one layer. In such situations, the object can be classified in more than one layer in the DMMVM. At a higher level of abstraction, it is also possible to further identify classes with some common attributes and by that generalise them into superclasses with each superclass having its own set of attributes, i.e., a hierarchic classification of objects. Like objects, a class can belong to only one superclass at the next higher level in the hierarchy.

Consequent upon the strict classification hierarchy adopted in this model, the following rules must be observed:
- Cyclic classification is not permitted, i.e., the classification must be a directed acyclic graph; for example, given three distinct classes A, B and C, C is not allowed to be a subclass of A if it is a superclass of B when B is a superclass of A.
- The classification must be complete, i.e., all objects must be classified.
- Each object must belong to only one class in each map layer but the object can be classified in more than one layer.

In the DMMVM, the thematic data will be represented by only class labels because this thesis concentrates on the geometric component of vector maps. But the model does not preclude the representation of thematic data. During implementation (when thematic attributes of objects are identified) the thematic data can be arranged in a hierarchic manner as described above.

3.4.2 Geometric Component

As in the FDS, the geometry of each object is represented by topologic primitives, arcs and nodes (see §3.2.1). An arc is defined by a pair of connected nodes while a node is defined by a set of X, Y and Z coordinates. The fact that the DMMVM integrates objects from one or more map layers implies that two or more objects of the same geometric type can overlap in space. This means that each geometric primitive may represent more than one object in the model, i.e.,
- a node may represent J point objects from J map layers
- an arc represents (part of) K line objects from K layers and has L area objects from L layers on each side (left and right). Objects may therefore have many-to-many (M:N) relationships with the geometric primitives.

For example, (part of) many line objects may be represented by the same arc while many arcs may define a single line object. This is a many-to-many relationship between arcs and line objects (see Figure 3.5 (a)). It is desirable to decompose the M:N relationship between objects and the geometric primitives into components of M:1 relationships to simplify the structure for easy manipulation, especially with respect to integrity controls. This will also allow the use of a single set of consistency rules for geometric primitives in both single- and multi-valued vector maps. Therefore, an additional data type, *m-container*, $m \in \{0,1,2\}$, is introduced such that one m-container will represent many objects of corresponding dimension while a geometric primitive represents only one m-container (see Figure 3.6). In the example above, many line objects will be represented by a single 1-container (i.e., M:1 relationship), while an arc defines (part of) only one 1-container; since one 1-container can be described by many arcs, we have a 1:N relationship between a 1-container and arcs (see Figure 3.5 (b)). Note, however, that there is still an M:N relationship between an m-container and an object

with respect to line and area objects. But this is now more at a semantic level than a geometric level. Thus (for example) an arc will still have only one two-dimensional entity on either side as in the FDS so that the same consistency operation defined for an arc in this model can also be used for the FDS arcs. By introducing the m-container data type, overlapping sections across the layers are uniquely identified such that they have their own individual geometric data and non-spatial data apart from those inherited from the overlapping objects; they can then be maintained and manipulated by the DBMS just like single objects. Thus it is easier to include them in aggregation and association abstractions, thereby improving spatial analyses in GIS.

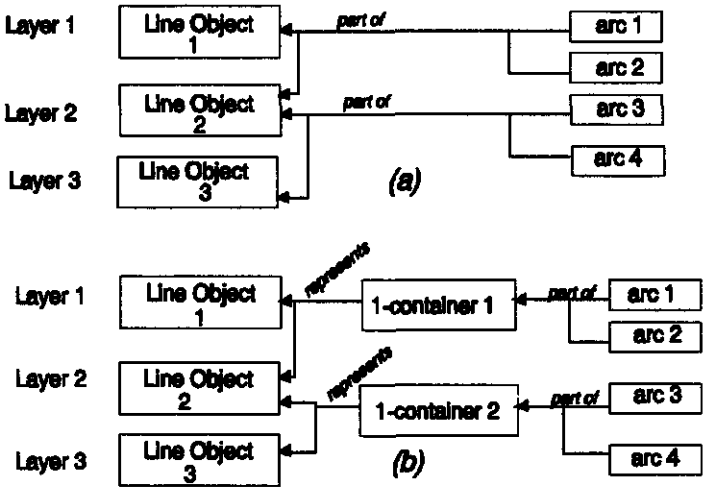


Figure 3.5 Decomposing M:N relationships between arcs and line objects (*(a)*) to 1:M relationships using 1-container (*(b)*)

To formally define the m-container, let the metric space covered by terrain objects in the multi-valued vector map (which is assumed to be a closed region) be denoted by E^n where n = 3, i.e., a 3D metric space. Also, the allowable entity (a bounded portion of space in E^3) in the map is denoted by R^m where $m \in \{0,1,2\}$, i.e., a 2.5D terrain representation. Thus $R^{m,n}$ denotes an entity in E^n where m = dimensionality of R and n = dimensionality of the space E in which R is located, and $m \leq n$. The allowable entities in our 2.5D representation are:
(a) $R^{0,3}$: 0D entity in E^3 space having a position but no spatial extent;
(b) $R^{1,3}$: 1D entity in E^3 space having shape and position but only length as a measurable spatial attribute;
(c) $R^{2,3}$: 2D entity in E^3 space having a 2D spatial extent with shape, size and position. Its measurable spatial attributes are area, perimeter and centroid.
$R^{3,3}$, a volumetric or solid object, does not occur in a 2.5D representation.

The $R^{m,3}$ corresponds to the m-container where $m \in \{0,1,2\}$ such that
- one 0-container represents J point objects from J map layers,

- one 1-container represents (part of) K line objects from K layers, and
- one 2-container represents (part of) L area objects from L layers,

where L is the maximum number of layers and J and K may each be less than or equal to L.

The m-containers $\forall$ m $\in$ {0,1,2} are then structured as a 2.5D graph using the two geometric primitives, arc and node. A node represents one 0-container and an arc represents (part of) one 1-container and/or boundary of a 2-container. The node is defined by a coordinate triplet X, Y and Z with respect to some coordinate system, while an arc is defined by a pair of adjacent nodes. The arc has a shape which is defined here as a straight line.
Thus the geometry of the multi-valued vector map is represented by a planar graph *G(N, A)* as defined in §3.2.1. The geometry of each m-container E occurring in the map is thus a subgraph $G_e(N_e,A_e)$ of G where $N_e \subset N$ and $A_e \subset A$. Note that for 0-container, $A_e = \varnothing$.

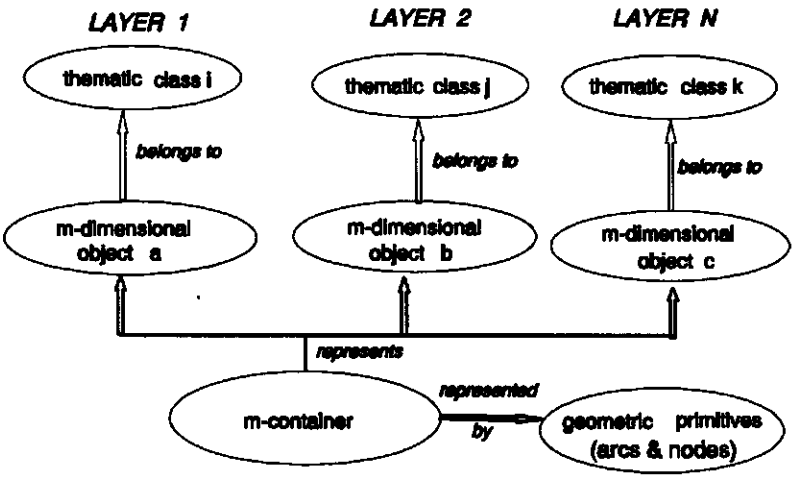


Figure 3.6 Representation of geometric primitives, m-containers, objects and classes in the DMMVM

Apparently, the m-container, m $\in$ {0,1,2}, is similar to the n-simplicial complex, n $\in$ {0,1,2} described in §3.1.2, when m equals n. However, while a 0-container is related to a 0-simplicial complex on a one-to-one basis, the semantic aspect of 1- and 2-containers introduces some differences between an m-container, m $\in$ {1,2}, and an n-simplicial complex, n $\in$ {1,2} whereby

one n-simplicial complex $\subseteq$ *one m-container* $\forall$ $m = n$; $m,n \in \{1,2\}$

In other words, a 2-container is a contiguous (connected) set of 2-complexes representing the same set of elementary objects of corresponding dimension (the same goes for 1-container vis-a-vis 1-complex).
Figure 3.7 illustrates this semantic difference. As shown in the figure, suppose we have two overlapping area objects P and Q (from two layers) with P = {1,8,2,3,4,10,5,6,7} and Q = {2,3,4,10,5,11,12,6,9}; a line object L = {8,9,10} intersects P and Q. The situation in the

figure will be represented by one 1-container (L) and three 2-containers R, S, T (where R = {1,8,2,9,6,7}, S = {2,3,4,10,5,6,9}, and T = {6,5,11,12}) whereas, in strict mathematical sense, we have five 2-complexes (each of the five closed non-overlapping polygons).

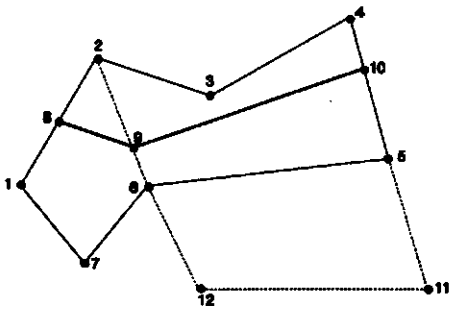


Figure 3.7 Simplicial complex and m-container (see text)

The DMMVM is shown in Figure 3.8 in which headed arrows represent many-to-one relationships in the direction of the arrow (e.g., many arcs may begin from the same node) while non-headed arrows represent one-to-one relationships (e.g., one node defines only one 0-container). Data types are represented by ellipses, each of which represents a set.

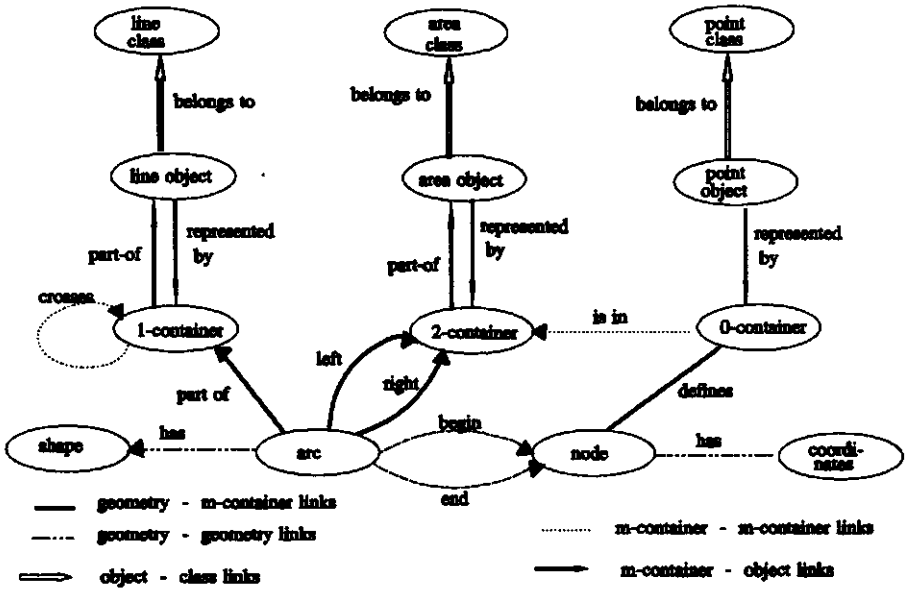


Figure 3.8 Data model for multi-valued vector maps (DMMVM)

3.4.3. Elementary Links Among Data Types in the DMMVM

There are four groups of data in the DMMVM, namely objects' thematic data as represented by the thematic classes, the objects (identifiers), the m-containers representing the objects, and the geometric data of the m-containers. Six categories of elementary links among the four groups of data are presented below. The links indicate the semantic constraints for the model.

Object - Class Links (oc-links)

- Each point object belongs to only one point object class in a given map layer but the object may be classified in more than one layer.
- Each line object belongs to only one line object class in a given map layer but the object may be classified in more than one layer.
- Each area object belongs to only one area object class in a given map layer but the object may be classified in more than one layer.

Object - Object Links (oo-links)

These will be realised through the combination of the following mo-links and mm-links.

m-Container - Object Links (mo-links)

- One m-container, m $\in$ {0,1,2}, is part of K m-dimensional objects from K different map layers.

m-Container - m-Container Links (mm-links)

- One or more 0-containers may lie in one 2-container.
- One 1-container may cross one or more other 1-containers at one or more points.

Geometry - m-Container Links (gm-links)

- A node may represent only one 0-container.
- An arc may represent (part of) only one 1-container.
- An arc always has just one 2-container on its left side and just one 2-container on its right side.

Note that the two geometric primitives, arc and node, retain the same structure as in the FDS (see go-links in §3.2.2) except that they are linked to the m-containers and not the terrain objects directly.

Geometry - Geometry Links (gg-links)

- Each arc has a beginning node.
- Each arc has an end node.
- Each arc has a shape, defined here as a straight line.
- Each node has a position given by coordinate triplet X, Y and Z.

Note that the four links are the same as defined in the FDS (see gg-links in §3.2.2).

3.4.4 Conventions of the DMMVM

The seven conventions of the FDS should be modified to fit multi-valued vector maps. The modified conventions and their relationships with the FDS conventions are as follows:

(1) *The object classes are mutually exclusive in each layer, i.e., each object belongs to exactly*

one class in one layer but the object can be classified in more than one layer. This convention is the same with the FDS convention with respect to individual map layers but with an added provision to allow an object to be classified in another layer if it plays a role in that layer.

(2) *One m-container, m ∈ {0,1,2}, represents K m-dimensional objects from K different map layers.* This is a new convention for the added data type m-container that has been introduced to model spatial coincidence among objects of the same type.

(3) *When a vector map is analyzed as a graph, all points that are used to describe the geometry will be treated as nodes* (same as in the FDS).

(4) *The arcs of this graph are geometrically represented by straight line segments* (same as in the FDS).

(5) *For each pair of nodes there is at most one arc connecting them directly; each of them may be connected also by one or more chains consisting of two or more arcs* (same as in the FDS).

(6) *Two arcs should not intersect; if they do intersect, they should be replaced by four arcs joining at a node* (same as in the FDS).

(7) *A node may represent at most one 0-container.*
An arc may be part of at most one 1-container.
An arc has exactly one 2-container at its right side and exactly one at its left side.
This is structurally the same as in the FDS except that the geometric primitives are linked to the m-containers and not directly to the terrain objects as in the FDS. Here, the primitives are linked to the objects through the m-containers (see convention 2 above).

A vector map which is modelled according to Figure 3.8 and satisfies the conventions above is termed a multi-valued vector map.

3.4.5 Elementary Objects

The complexity of terrain objects in terms of shape and size will often vary from one mapping context to another, which creates a problem in the definition of a generic model that can be used in different applications. A feasible solution is to define a limited set of elementary (non-decomposable) objects which can then be used as building blocks for complex objects in different applications (by providing some rules for the aggregation of the elementary objects). This philosophy can be related to the provision in the English language of 26 letters (A - Z) which can then be grouped together using defined rules (grammar) to formulate different words, e.g., the letter Q should always be followed by the letter U in any word. Thus in the DMMVM, the elementary terrain object types (area, line and point) are defined and constrained as follows. The mapping of terrain objects to these elementary types can then be one-to-one or one-to-many.

Elementary Area Object Type

An instance of this type is defined and constrained as follows:
- A two-dimensional object, F, geometrically represented by the subgraph $G_F(N_F, A_F)$ where $A_F = \{a_1, a_2, ..., a_j\}$ is the set of arcs defining the boundary of the object and $j \geq 3$, i.e., a minimum of three arcs will form a closed polygon since an arc is assumed to be a straight line segment; and $N_F = \{n_1, n_2, ..., n_k\}$ is the set of nodes defining $A_F \ni k = j$.
- $\forall n_i \in N_F$, $degree_F(n_i) = 2$. This implies that
  * The start-node = end-node.
  * A *path* must exist between the start-node and the end-node with only one occurrence of $a_i \in A_F$ in the *path*, i.e., the path must be simple.
- The interior (see Table 4.1) must be fully connected.

Elementary Line Object Type

Each instance of a simple line object type is defined and constrained as follows:
- A 1D spatial object, L, geometrically defined by the subgraph $G_L(N_L, A_L)$ where $A_L = \{a_1, a_2, ..., a_j\}$ is the set of arcs defining the geometry of the object L and $j \geq 1$; and $N_L = \{n_1, n_2, ..., n_k\}$ is the set of nodes defining $A_L \ni k = j + 1$.
- Exactly two of the nodes N_L have $degree_L(n) = 1$
- All other nodes in the set N_L have $degree_L(n) = 2$. These imply that
  * A simple line object can neither intersect nor close back on itself.
  * A *path* must exist between the start-node and the end-node of G_L, with only one occurrence of $a_i \in A_L$ in the *path*, i.e., the path must be elementary and simple (see §3.1.1).

Elementary Point Object Type

An instance of this type is always a primitive and is geometrically represented by a single node.

3.5 Integrating the DMMVM with DTM

The DMMVM can easily be extended to cover representation of other spatial information in vector mode. This section shows its extendibility by incorporating a digital terrain model. Digital terrain model (DTM) is used here to mean a dataset representing the elevation of a given terrain. Development of digital terrain modelling techniques started more than three decades ago (Ebner and Eder, 1992), and program packages are today available covering all phases from preparation to derivation of DTM products.

With the increasing popularity and use of GIS, attention is being focused on the total integration of DTM and related packages into GIS such that DTM-specific information can be derived from the same database, just like any spatial information under a single DBMS. Recent works on the integration of DTM into GIS include Sandgaard (1988), Fritsch (1991), Radwan (1991), Ebner and Eder (1992), Höhle (1991, 1992), and Pilouk and Tempfli (1993).

A DTM contains two complementary sub-sets (Makarovic, 1988): the skeleton and the filling information, with the skeleton being largely contained in various terrain objects such as lakes,

rivers, etc. Therefore, many planimetric objects serve as characteristic objects (breaklines, etc) and primary data in digital elevation modelling. Thus, in developing this integrated model, the terrain relief is regarded as a mapping theme such that it (terrain relief) is classified into geometric classes like other terrain objects. To realise this, the DTM class to which an object belongs can be made a mandatory property of each terrain object in the database. Table 3.1 gives the DTM classification scheme for terrain objects.

Various models have been developed for the digital representation of terrain relief. These include:
- regular grid,
- irregular grid,
- isolines (digital contours),
- triangulated irregular network (TIN)

The TIN is an appropriate structure that can be integrated with vector structures since characteristic points and lines, which form a logical part of the TIN, are often objects in the vector structure (e.g., rivers, lakes). In other words, TIN, though a tessellation model, can be seen as a vector topologic structure for representing polygon networks (Burrough, 1986). The TIN-DTM was therefore chosen for integration with the DMMVM.

Table 3.1. Classification model for DTM objects (modified from Höhle (1992)).

ENTITY TYPE	DTM CLASSES	DEFINITION
Point	Regular	Any regular point with xyz coord
	Spot	Local high and low points. No assumptions are made on the slope of the surrounding terrain
	Peak	Specific local high z-value
	Pit	Specific local low z-value
Linear	Break	A line which defines a change in slope or a surface discontinuity
	Drain	A specific form of the breakline, it is assumed that the surface on either side of the line object has an increasing slope
	Ridge	A specific form of the breakline, it assumes a decreasing slope on either side of the object
	Contour	Equal z values along the line
	Regular	Any line entity which is not a breakline

Area	Double-line drain	A drainage object which at the map scale is large enough to be represented as an area object. Heights decrease uniformly in one direction.
	Water body	A hydrographic area object with assumed constant z-value in its interior
	Obscure	A dead area. Any area object which obstructs the measurement of heights in its interior, the values of which cannot be assumed, e.g., dense forest cover
	Regular	Any area object in a relief-homogeneous region

The primitive topologic entities of a TIN are vertices, edges and triangles. The internal storage structure of a TIN is therefore usually based on one or a combination of the three primitives. In a vertex-based TIN, the primary entity is the vertex; for each vertex, the vertex number is stored with the list of pointers to connected vertices and edges. In triangle-based TIN, it is possible to store the triangle number with its three vertices and its three neighbours. For edge-based TIN, a record will comprise the edge number, the two adjoining triangles and the two vertices that define the edge.

Thus in an edge-based structure, the TIN will be fully described by two geometric primitives, edges and vertices. This is consistent with the use of arcs (edges) and nodes (vertices) as topologic primitives in multi-valued vector maps. Thus the proposed integrated model uses the edge-based TIN structure by adding the data types TIN edge and TIN vertex to the data types defined in the DMMVM. The integrated data model is shown in Figure 3.9 in which part B indicates the edge-based TIN data types and their links and part A is the DMMVM of Figure 3.8. This gives a flexible structure which allows separation of the two parts into two subsystems in the same database such that geo-information that does not require DTM input can be retrieved without involving the DTM part. And because they are integrated, objects in the object-base can contribute to the generation of a DTM with high fidelity, while the DTM supports the object-base, e.g., when updating via mono-plotting techniques, to provide elevation information for objects whose Z values could not be determined during the data collection phase, and to provide relief information in general.

With reference to Figure 3.9, the **part of** link between arc and TIN_edge and **is a** link between node and TIN_vertex provide the link between the map base and the DTM at geometric level. It would appear that the arc and node structure, being similar to the edge and vertex of the TIN, should be sufficient to handle the geometric information of the joint model without having a separate TIN structure. We prefer not to do this because a single arc in the map base may be decomposed into more than one edge after triangulation, as illustrated in Figures 3.10a and b. For example, using a relational structure, the ARC table in the map base (Figure 3.10a) will contain a record for arc number 5 in the following form:

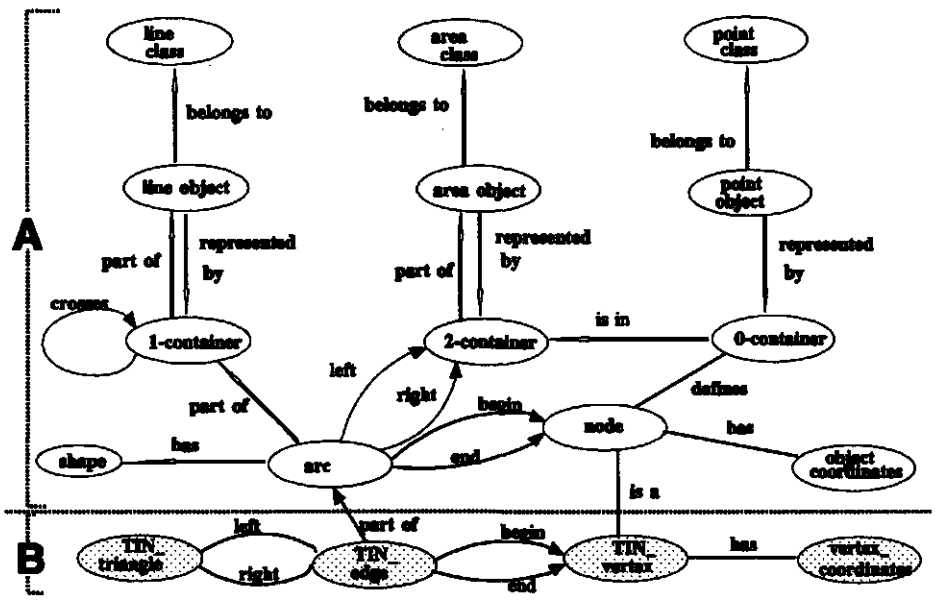


Figure 3.9 Integrated data model for multi-valued vector maps and DTM

Arc_id	beg	end	lft_aid	rgt_aid	alid
5	1	4	11	12	10

where 10, 11 and 12 are the m-container identifiers of river1, cultivatedLand1 and forest1, respectively. After triangulation (Figure 3.10b), the TIN-EDGE table will include two records, related to arc number 5, in the following form:

begVertex	endVertex	leftTriangle	rightTriangle	arc_id
1	8	1	5	5
8	4	3	8	5

Figure 3.11 shows the block diagram of the procedure for implementing the integrated data model. First, the data acquisition for the multi-valued vector map can be done either in multi-valued mode (i.e., from a combined multi-valued data source such as photogrammetric superimposition of two layers) through screen digitizing, land surveying, etc., or by data acquisition per layer followed by overlay computation. During data acquisition, objects would also be classified into DTM classes (apart from their thematic classes). The dataset is then reformatted and checked for consistency (see chapter 5). The result can then be structured according to a relational data structure, for example.

To generate the DTM subsystem, the 0-, 1- and 2-containers which belong to DTM object classes are extracted to form part of the DTM skeleton. In other words, the arcs of such 1- and 2-containers will form part of the TIN edges and the nodes of those 0-containers will be vertices in the TIN. Further sampling is carried out to collect filling data for the interior parts of 2-containers and for more skeleton data for characteristic lines which have not been

captured as terrain objects during the initial data acquisition. The coordinate data of other objects in the database which belong to the DTM class "regular" (see Table 3.1) are extracted as filling data. The skeleton and the filling data are then triangulated and structured as an edge-based TIN. Heights of points defining the boundaries of 2-containers belonging to DTM class "obscure" are then derived by interpolation. Finally, the relational tables describing the DTM subsystem (e.g., Tinedge, Tinvertex and Triangle) are filled.

For queries on objects requiring DTM input, the part-of link between arc and Tin_edge, and is-a link between node and Tin_vertex (Figure 3.9) would be used for navigation. With this flexible set-up, non-DTM-related information will be more efficiently retrieved. For example, topologic queries at the object level will not have to search through DTM data. DTM-related queries involving line and point objects can be easily handled through the part-of link between arc and Tin-edge and the is-a link between node and Tin-vertex. Deriving DTM-related information for area objects is, however, not as straightforward as in the other two object types, since the DTM points in the interior of the area object are not directly linked to the object. To facilitate retrieval of DTM points for any area object, a routine can be provided as an integral part of the database. This routine may make use of a "point-in-polygon" subroutine (in 2D mode) to retrieve all vertices located in the interior of a certain area object and its output will then serve as input into the derivation of DTM information concerning area objects.

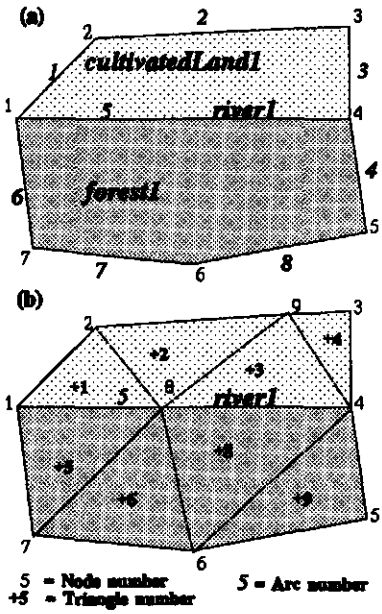


Figure 3.10 A simple map in the map-base (a) and DTM-base after (fictitious) triangulation (b)

3.6 Summary

In this chapter, a vector data model has been proposed to represent multi-valued terrain abstraction, especially when frequent spatial analyses across many map layers are envisaged, i.e., in applications involving frequent analysis of multi-layer geo-information. The proposed conceptual data model is an object-based 2.5D data model for multi-valued vector maps (DMMVM). A multi-valued vector map refers to the vector-based representation of terrain objects from multiple map layers whereby two objects of the same geometric type may be spatially coincident. Two objects of the same type are said to be spatially coincident if they (partially) overlap in space. In the model, positions of objects are defined in a 3D metric space but embedded in planar topology, i.e., a 2.5D model. This means that only surfaces of objects are represented such that a pair of X and Y coordinates must have a single Z value, thus a single-elevation model. The model was based on the formal data structure (FDS) for

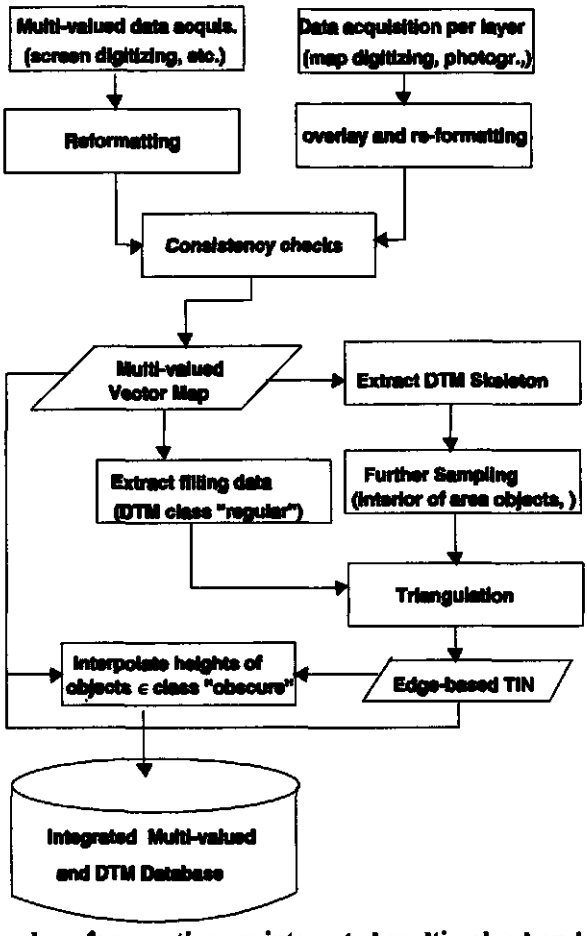


Figure 3.11 Procedure for creating an integrated multi-valued and DTM database

single-valued vector maps. In the 2D FDS (see §3.2), terrain objects play a central role in the terrain description; each object has a thematic component and a geometric component. In the thematic domain, the objects can be grouped into thematic classes in which each class has a specific attribute structure; in the geometric domain, the object types points, lines and areas are distinguished for a 2D or 2.5D terrain description, subject to a constraint that two objects of the same type may not be spatially coincident. The three object types are then completely described by a common set of two types of geometric elements: arc and node, using graph theory as the mathematical framework. In the FDS, the geometry of a terrain object is clearly distinguished into three independent aspects, namely topology, shape and size, and position. This geometric dataset has been carefully structured in the FDS, leading to a semantically-rich, query-oriented and extendible data model in which information on topology, shape and size, and position can be retrieved.

The FDS was extended in this thesis to allow objects of the same type to be spatially coincident, thus facilitating the use of a single structure for the representation of multi-layer

geo-data. A geometric data type, m-dimensional container, or simply m-container, where m $\in$ {0,1,2} was introduced to model spatial coincidence among objects of the same geometric type. Thus a 0-container represents spatially coinciding J point objects from J layers, a 1-container represents (part of) K line objects from K layers and a 2-container represents (part of) L area objects from L layers, where L is the maximum number of layers and J and K may each be less than or equal to L.

By introducing the container data type, overlapping sections across the layers are uniquely identified such that they have their own individual geometric data and non-spatial data apart from those inherited from the overlapping objects; they can then be maintained and manipulated by a DBMS just like single objects. Thus it is easier to include them in aggregation and association abstractions, thereby improving spatial analyses in GIS.

Using graph theory as a mathematical tool, the three container types are then represented by topologic primitives arc and node. A node defines one 0-container and/or beginning or end of an arc while an arc defines (part of) one 1-container and/or (part of) boundary of a 2-container. The arc is defined by one start node and one end node, and a node is defined by a coordinate triplet X,Y,Z.

Thus eight basic geometric data types are defined to represent geo-data from multiple map layers, namely area, line, point, 2-container, 1-container, 0-container, arc, and node. Each data type plays some specific roles in the model. The area, line and point data types abstractly represent terrain objects, whereby each terrain object in the application is mapped into one of the three types during implementation. The mapping can be one-to-one or one-to-many, depending on the complexity (shape) of the terrain object, e.g., a two-dimensional object with a connected boundary and interior will be mapped to one elementary area object type while a two-dimensional object with disconnected boundaries and interiors will be mapped into two or more elementary area objects. These related elementary objects will then be aggregated to reconstruct the parent (original) object during query.

One of the attributes of each of the three object types should be the thematic class of the object. Although the thematic aspects of objects were given less attention here, the model does not preclude the representation of thematic data. During implementation (when thematic attributes of objects are identified) the thematic data can be arranged in a hierarchic manner as proposed in §3.2.1 and §3.4.1

The m-container, m $\in$ {0,1,2}, models spatial coincidence among elementary objects of corresponding spatial dimension as explained above. Apart from the attribute values inherited from the spatially coinciding objects, an m-container data type can have additional attributes as required by the user. Arc and node, as stated above, play the roles of geometric descriptors in the model.

An integration of the DMMVM with the TIN-DTM was described in §3.5. The integrated model provides a unified representation of multi-valued terrain object data and terrain relief information in a flexible manner. The position of an object can thus be given in 2D or 3D; when defined in 2D, the height value will be interpolated from the DTM subsystem.

The philosophy behind this integrated approach to spatial data modelling is based on two

main considerations:
(1) Although data acquisition is usually layer-oriented, subsequent analyses often require integration of data from more than one layer, leading to ad hoc overlay computations. The model presented here can be used to organize the result of an initial overlay of all relevant layers which will subsequently be used for future single-valued or multi-valued queries.

(2) Most of the skeleton of a DTM is usually contained in terrain objects such as rivers, roads, lakes, etc.; with the importance of DTM in spatial analyses, it is apparently more efficient to integrate planimetric and elevation models. Thus objects in the object-base will contribute to the generation of a DTM with high fidelity while the DTM supports the object-base, e.g., when updating via mono-plotting techniques, to provide height information for objects whose Z values could not be determined during the data collection phase, and to provide relief information in general.

It has been shown here that the DMMVM can be extended to handle DTM information, but this will not be elaborated further in this thesis. Additional information on the integration of the DMMVM and DTM can be found in Kufoniyi et al (1994), Kufoniyi and Pilouk (1994), Pilouk and Kufoniyi (1994), and Kufoniyi and Bouloucos (1994). Subsequent chapters will focus only on the DMMVM. The translations of the DMMVM to prototype relational and object-oriented database structures are given in chapter 7. The topologic spatial relationships supported by the DMMVM are formally derived in the next chapter; they will be used as tools in chapters 5 and 6 for the provision of consistency rules to guide updating operations.

4

MODELLING TOPOLOGIC RELATIONSHIPS IN VECTOR MAPS

In chapter three, a conceptual data model for multi-valued vector maps was proposed. The model described how the geometric and attribute data of objects from different map layers can be organized in a structure. Part of an object's geometric data are the spatial relationships between the object and other objects. They often serve as the main tool for intelligent analyses and processing in GIS. The relationships can be grouped into three main types (Kainz, 1990):

(1) Topologic spatial relationships: these are the relationships which remain invariant under certain topologic transformations such as rotation, shift and scaling. Examples are neighbourhood and connectivity.
(2) Spatial order relationships: these concern the representation of the concepts of inclusion and containment of spatial objects using partially ordered sets and lattices based on mathematical order theory.
(3) Metric spatial relationships which cover the concepts of distance and direction.

The main focus in this thesis is the first group: topologic spatial relationships. They give more detailed spatial relationships than the spatial order set (Kainz, 1990). The metric relationships are normally computed from the database using the coordinates of objects. For example, the metric relationship **distance** (euclidean) between two point objects A and B whose positions are respectively defined in a 3D cartesian space as X_A, Y_A, Z_A and X_B, Y_B, Z_B will be computed using the formula

$$\textbf{distance}(A,B) = \text{sqrt}((X_B-X_A)^2 + (Y_B-Y_A)^2 + (Z_B-Z_A)^2)$$

All topologic relationships among objects can normally be derived through the use of coordinate information and analysis, but this approach will slow down operations in the system.

In topologic data structures, most of the topologic relationships are implicitly or explicitly represented and can be derived by queries. However, it is useful to formalise the elementary (basic) set supported by a certain data model in order to know a-priori the information content of the database. The elementary set can then be translated into basic topologic operators as a fundamental step towards the establishment of an active (dynamic) spatial database. In general, the topologic operators will be useful for the following operations in an active vector GIS:
(1) On-line building of topology such that if the geometry of an object changes, topology is automatically updated. Also, complex topologic relationships and other implicitly represented relationships can be dynamically derived.
(2) Dynamic checking of spatial consistencies as a step towards consistency enforcement.
(3) Dynamic building of a consistent multi-valued vector map from two or more single-valued vector maps.
(4) As tools for constructing complex objects from elementary ones.

This chapter describes the formal derivation of the topologic relationships in vector maps based on the data model proposed in the preceding chapter. The "9-intersection" formalism

(Egenhofer and Herring, 1992) (see §4.1) was used for the derivation. The relationships are defined at two levels: first, among the elementary objects (object topology), and second, among the topologic primitives, arcs and nodes (geometric-primitive topology). The algorithms for detecting the particular relationship (from the possible set) that exists between any pair of objects are also formalised. The application of the derived topologic relationships in automated consistency validation and updating of the geometric structure of vector maps is proposed in the next chapter.

The importance of topologic relationships in GIS has made it an active research topic. For example, Molenaar (1991a) defined the topologic relationships among data types in the FDS at three levels. The first comprises a set of semantically defined relationships among the three elementary objects (point, line and area) in single-valued vector maps (see Figure 4.1). The second level consists of the links provided by the graph structure of vector maps between the geometric primitives and the objects. The third consists of the connectivity among the geometric primitives (arc and node).

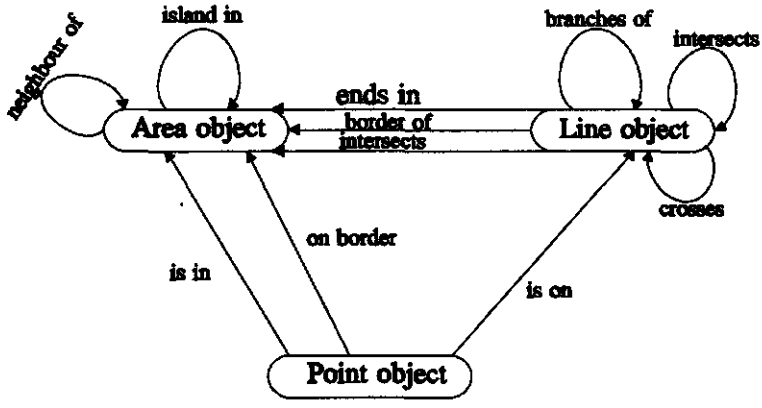


Figure 4.1 Topologic relationships among elementary objects (Molenaar, 1991a)

Also, Pullar and Egenhofer (1988) applied the point-set topology (Vaidyanathaswamy, 1960) to define six minimal set of binary relationships (disjoint, meet, overlap, concur, common-bounds and equals) between two one-dimensional intervals (line segments).

The point-set approach has been formalised as a mathematical framework for the derivation of topologic relationships between n-dimensional objects by using either the "4-intersection" model (Egenhofer and Herring, 1990) or the "9-intersection" model (Egenhofer and Herring, 1992). The former distinguishes two parts of an object, the interior and boundary, and evaluates the set intersection between each part of one object and each part of a second object. The 9-intersection extends this by including the exterior component of an object. This approach has become popular among researchers for deriving topologic relationships, e.g., Egenhofer and Franzosa (1991), Hadzilacos and Tryfona (1992), Hoop and Oosterom (1992), Egenhofer and Al-Tahar (1992), Pigot (1991), Clementini et al (1993), Kufoniyi et al (1993, 1994) and Molenaar et al (1994). The 9-intersection model is used in this thesis to analyze the topologic relationships in the DMMVM.

4.1 The 9-Intersection Formalism for Modelling Topologic Relationships

Egenhofer and Herring (1992) proposed the 9-intersection model for formalising binary topologic relationships between two arbitrary objects. The model was based on point-set theory of algebraic topology. In the model, an object, O_i, is represented as a point-set consisting of the following three components (subsets): the boundary set of O_i represented by ∂O_i, the interior set of O_i represented by $^\circ O_i$ and the exterior set of O_i represented by $^- O_i$

With the three components of a point-set identified, the modelling proceeds with the following three main operations:
Step One: Evaluate the set intersections ($\cap$) between the boundary, interior and exterior of one point-set O_1 and the boundary, interior and exterior of the second point-set O_2. This gives a 9-intersection configuration as shown in Figure 4.2

$$\begin{array}{lll} \partial O_1 \cap \partial O_2 & \partial O_1 \cap {}^\circ O_2 & \partial O_1 \cap {}^- O_2 \\ {}^\circ O_1 \cap \partial O_2 & {}^\circ O_1 \cap {}^\circ O_2 & {}^\circ O_1 \cap {}^- O_2 \\ {}^- O_1 \cap \partial O_2 & {}^- O_1 \cap {}^\circ O_2 & {}^- O_1 \cap {}^- O_2 \end{array}$$

Figure 4.2 The 9-Intersection configuration

Each element in the nine-element tuple is evaluated as empty, denoted $\varnothing$, or non-empty, denoted $\neg\varnothing$. This gives a total of 512 (2^9) mutually exclusive candidate binary topologic relationships between the two arbitrary objects.
Step Two: Eliminate from the 512 the topologically impossible relationships. Rules are defined for the elimination, based on the definition of object types in the embedding data model.
Step Three: Combine the topologically similar relationships in the result of step two. Two relations are topologically similar if they share the same boundary-boundary, interior-interior and exterior-exterior specifications, but have opposite boundary-interior and interior-boundary, and/or boundary-exterior and exterior-boundary, and/or interior-exterior and exterior-interior specifications.
The third step will yield the topologically consistent set of mutually exclusive candidate binary relationships between the two objects.

This formalism is applied here to derive the topologic relationships between objects and between geometric primitives in the DMMVM.

4.2 Topologic Relationships in Vector Maps

4.2.1 Relationships among the elements of a planar graph

In chapter three, vector maps are modelled as a planar graph. The relationships among the elements of this graph, i.e., nodes, arcs and faces, can be defined with a limited number of relational functions (Molenaar et al, 1994) as follows.

Nodes and arcs

The following relationships can be defined between nodes and arcs:

- *Arc a_i has node n_j as the begin node* $\rightarrow Begin[a_i, n_j] = 1$
- *Arc a_i has node n_k as the end node* $\rightarrow End[a_i, n_k] = 1$

If loops are not allowed:

$Begin[a_i, n_j] = 1 \rightarrow End[a_i, n_j] = 0$
and $End[a_i, n_k] = 1 \rightarrow Begin[a_i, n_k] = 0$

So arcs will have distinct begin and end nodes. Whether n_j is a node of arc a_i can be investigated by the function:

$$N[a_i, n_j] = Begin[a_i, n_j] + End[a_i, n_j]$$

If $N[a_i, n_j] = 1$ then n_j is a node of a_i
If $N[a_i, n_j] = 0$ then this is not the case.

- The degree of a node can be found through:
  $Degree(n_j) = \sum_i (N[a_i, n_j])$

Arcs and faces

Each arc will always have one face at its lefthand side and one at its righthand side. These relationships will be expressed by the following functions:

- Arc a_i has face F_a at its left-hand side $\rightarrow$ $Le[a_i, F_a] = 1$

For any $F_b \neq F_a$ we get then $Le[a_i, F_b] = 0$

- Arc a_j has face F_a at its right-hand side $\rightarrow Ri[a_j, F_a] = 1$

and again for $F_b \neq F_a$ we get then $Ri[a_j, F_b] = 0$

If an arc a_i is part of the border of F_a then only one of the functions Ri and Le is equal to 1 for a_i, but not both. So if we define the function:

$$B[a_i, F_a] = Le[a_i, F_a] + Ri[a_i, F_a]$$

then when a_i is part of the boundary of F_a we find $B[a_i, F_a] = 1$.
The boundary of F_a is:

$\partial F_a = \{N_a, A_a\}$ *with* $A_a = \{a. \mid B[a., F_a] = 1\}$ *and*
N_a *contains the nodes of the arcs of* A_a
"." *stands for an unspecified index value.*

The arcs that make up the border of a face form a *polygon* (i.e., a closed chain of arcs). So for any arc a_i that is part of a polygon in a planar graph there are only two faces, so that $B[a_i, F.] = 1$, for all other faces $B[a_i, F.] = 0$.

If an arc a_i is part of a polygon then it is not possible that there is any face $F.$ for which $B[a_i, F.] = 2$. If an arc a_i does not belong to a polygon then there must be a face $F.$ for which $B[a_i, F.] = 2$. In this case there are two possible relationships between an arc $a_i = \{n_b, n_e\}$ and ∂F_a, see figure 4.3:

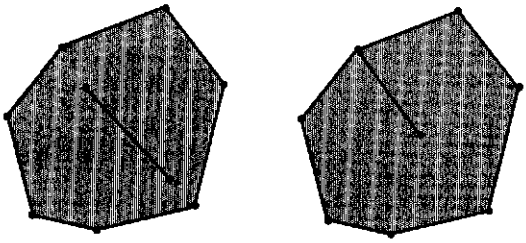


Figure 4.3: Relationships between an arc on a face and the boundary.

- the arc is not directly connected to the boundary:
  $n_b, n_e \notin N_a$
- the arc is connected to the boundary through one node:
  $n_b \in N_a$ *or* $n_e \in N_a$

Nodes and faces

If the degree of a node n_i is Degree[n_i] = 0, then it is not related to an arc. In that case it must be contained inside a face F_a; this relationship will be expressed by:

$ISIN[n_i, F_a] = 1$

If Degree[n_i] $\neq 0$, then there is some arc a_j for which $N[a_j, n_i] \neq 0$. This arc will be related to some face through one of the relationships explained above; the relationship between the node and that face is then established through the arc.

Face to face

For face F_1 with a boundary $\partial F_1 = \{N_1, A_1\}$ as defined above, and F_2 with $\partial F_2 = \{N_2, A_2\}$, there are four possibilities for the intersection of these subgraphs, see Figure 4.4:

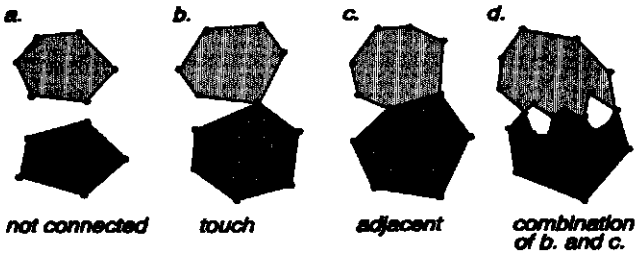


Figure 4.4: Geometric connections between two faces

- there is no geometric connection if there are no common nodes or arcs; thus:
  $N_1 \cap N_2 = \emptyset$ *and* $A_1 \cap A_2 = \emptyset$

- the objects *touch* if there are one or more common nodes but no common arcs; thus: $N_1 \cap N_2 \neq \emptyset$ *and* $A_1 \cap A_2 = \emptyset$
- the objects are *adjacent* if there are one or more common arcs, but no common nodes that do not belong to the common arcs; in that case the objects also have the nodes of these arcs in common, so it is sufficient to state that: $A_1 \cap A_2 \neq \emptyset$
- the objects have a combination of the *touch* and *adjacency* relationships; that means there are common nodes that do not belong to the common arcs.

4.2.2 Topologic Relationships at Object Level

4.2.2.1 Elementary Objects in the data model.

The 9-intersection model requires the definition of the boundary, and interior and exterior components of an object. To define these components for the three object types (point, line and area) in vector maps, the definitions of the three types (see §3.4.5) are reformulated in terms of the relational functions of §4.2.1 as follows.

Elementary Area Objects

The geometry of an object of this type is represented by one or more adjacent faces. If a face F_a is part of an area object AO_f, this will be represented by:

$$Partof[F_a, AO_f] = 1$$

The relationships between arcs and area objects can be found via their relationship with the faces by:

$$Le[a_i, AO_f] = Max_{F_a}(Le[a_i, F_a]*Partof[F_a, AO_f])$$

$$Ri[a_i, AO_f] = Max_{F_a}(Ri[a_j, F_a]*Partof[F_a, AO_f])$$

and

$$B[a_i, AO_f] = Le[a_i, AO_f] + Ri[a_i, AO_f]$$

The boundary of AO_f represented by ∂AO_f, is defined by the subgraph (N_f, A_f) where

$A_f = \{a_i / B[a_i, AO_f] = 1\}$ and
$N_f = \{n_1, n_2, ..., n_k\}$ contains the nodes of these arcs.

The interior of an area object $^{\circ}AO_f$ is defined by its faces minus ∂AO_f. For an elementary area object these should be adjacent so that if we eliminate the arcs a_j with $B[a_j, AO_f] = 2$ from the graph, then the object would be represented by exactly one face.

Elementary Line Objects

When an arc a_i is part of a line object LO_l, this can be represented by:

Partof[a_i, LO_l] = 1; if the arc is not a part of the line object this function will have the value = 0.

An elementary line object type can then be pragmatically defined as any 1D spatial object, LO_l, geometrically defined by the subgraph $G_l = \{N_l, A_l\}$ where

A_l = *{a_i/Partof[a_i, LO_l] = 1}* is the set of arcs defining the geometry of LO_l and N_l = *{n_1, n_2, ..., n_k}* contains the nodes of these arcs.

- Exactly two of the nodes of N_l have $Degree_l(n) = 1$; these are the endpoints and they constitute the boundary, ∂LO_l, of the line object
- All other nodes in the set N_l have $Degree_l(n) = 2$; with the arcs in A_l they form the interior of the line object.

Elementary Point Objects

An instance of this type is always elementary and is any point object PO_p geometrically represented by a single node n_i where $n_i \in N$; this will be expressed as:

Repr[n_i, PO_p] = 1; if the node does not represent this object the value of the function will be = 0.

A point object has no interior; n_i is the boundary of the point.

Boundary, Interior and Exterior of Elementary Objects

Based on these definitions, the boundary, interior and exterior of each of the three elementary object types are defined in Table 4.1 . The universal set of points defining the map is represented by $\mho$.

Table 4.1. Boundary, interior and exterior of elementary objects

Object Type	Boundary (∂)	Interior ($^\circ$)	Exterior ($^-$)
Point (P)	$\partial P = n_p \mid Repr[n_p, P]=1$	No interior point (undefined)	$^-P = \mho - \partial P$
Line (L)	$\partial L = \{n_i \in N_L \mid Degree_L(n_i)=1\}$	$^\circ L = G_L - \partial L = \{(N_L - \partial L), A_L\}$	$^-L = \mho - (\partial L \cup {}^\circ L)$
Area (R)	$\partial R = G_r(N_r, A_r)$ $A_r = \{a_i \mid B[a_i, R]=1\}$ $N_r = \{n_j \mid N[a_i, n_j]=1, a_i \in A_r\}$	$^\circ R = \{F_i \mid Partof[F_i, R] \approx 1\} - \partial R$	$^-R = \mho - (\partial R \cup {}^\circ R)$

4.2.2.2 Deriving the Relationships

For n number of elementary objects, there are n^2 groups of binary topologic spatial relationships among them, each group consisting of 512 mutually exclusive candidates. It is assumed that the order of the objects in the n by n combinations will not be relevant, so because of their symmetry the number of groups reduces to (n(n+1))/2. For the three elementary objects, there are thus six groups of candidate relationships, namely Area/Area, Area/Line, Area/Point, Line/Line, Line/Point and Point/Point.

To eliminate the topologically impossible relationships from the 512 candidates of each object combination, rules were defined as illustrated with the following three examples. First, two assumptions were made: (1) all objects are embedded in the same closed geographic space (similar to a project area in mapping); (2) a situation where only two complementary objects occupy the whole region does not exist.

Rule 1: Based on the two assumptions above, the exteriors of two arbitrary objects must always intersect.

Rule 2: Considering a point object (p) and a line object (l),
$\partial p \cap \partial l = \neg\varnothing \Rightarrow \partial p \cap {}^{\circ}l = \varnothing$ and $\partial p \cap {}^{-}l = \varnothing$

Rule 3: Since the interior of a point object is undefined, all intersections between the point's interior and any of the three components of the second object must be empty.

Furthermore, the topologic relationships which are not consistent with the DMMVM, although topologically possible under a different object definition, are eliminated. There are two cases in this category. The first case deals with single-valued vector maps while the second deals with multi-valued vector maps. After eliminating the topologically impossible set (using the defined rules) and those that are not consistent with the embedding data model (DMMVM), the resulting consistent relationships and their intersection configuration for the six object combinations are presented in Table 4.2.

The graphic representations of the relationships are given in Figures 4.7 to 4.10. No attempt at linguistic definition of the relationships is made; rather, each of them is coded as the decimal conversion of the binary number corresponding to its relationship. This is because relationships with the same topologic structure (9-intersection configuration) will have different names in different contexts. For example, while the relationship coded as r220 can be interpreted as overlap for area/area and line/line, the topologic overlap for point/point is defined by r272. This difficulty has also been acknowledged by Mark and Egenhofer (1994). For the distinct coding, empty intersection is interpreted as bit value 0 and non-empty as bit value 1. Thus, relation r511 (111111111 in binary digits) represents a tuple where all nine intersections are non-empty. Likewise, r000 (000000000) represents the relationship in which all nine intersections are empty. The structure of the nine-field tuple is in the following form:

$$\partial O_1 \cap \partial O_2 \;\; {}^{\circ}O_1 \cap {}^{\circ}O_2 \;\; {}^{\circ}O_1 \cap \partial O_2 \;\; \partial O_1 \cap {}^{\circ}O_2 \;\; {}^{-}O_1 \cap {}^{-}O_2 \;\; \partial O_1 \cap {}^{-}O_2 \;\; {}^{\circ}O_1 \cap {}^{-}O_2 \;\; {}^{-}O_1 \cap \partial O_2 \;\; {}^{-}O_1 \cap {}^{\circ}O_2$$

The number of relationships for each object combination can be further reduced by combining the topologically similar ones (e.g., r179 and r220 between two area objects) in Table 4.2.

The result of this combination is the limited number of topologic relationships in vector maps shown in Table 4.3.

Note that the 9-intersection model improves on the 4-intersection model (using only boundary and interior parts) (Egenhofer and Franzosa 1991), because relationships which cannot be differentiated in the latter model can now be distinguished, e.g., relationships r252, r253 and r255 between an area and a line object will be the same relationships in the 4-intersection model. However, the 9-intersection model can still not eliminate all ambiguities in the types of relationships. For example, Figure 4.5 shows two (semantically) different spatial relationships, but they are topologically equal in the 9-intersection model. Both cases are defined by the same relationship r287. Thus additional measures, such as the dimension of intersection, must be applied to resolve this type of ambiguity. But by basing the realization of the relationships on the elementary set and the relational functions defined for the elements of a planar graph (§4.2.1), this ambiguity can be taken care of. All simple and complex object-level topologic relationships can then be derived from this elementary set.

However, the use of the elementary relationships between the elements of a planar graph assumes that the geometry of the map is consistent, i.e., all the conventions relating to geometry (see §3.2.3 and §3.4.4) are observed. An inconsistent geometry can be detected by checking the topologic relationship between the geometric primitives (arc and node), e.g., by checking whether two arcs intersect without a node at the intersection point or whether two nodes overlap. Therefore, it is beneficial to also formalize the relationships between the geometric primitives. Rules can then be made to resolve any inconsistency that may arise. The relationships among the geometric primitives are presented in the next subsection.

Table 4.2: Topologic relationships between two simple objects O_1 and O_2 in vector maps

Relation	Intersection	A/A	A/L	A/P	L/L	L/P	P/P
	∂∂ ∘∘ ∘∂ ∂∘ — ∂− ∘− −∂ −∘						
r026	0 0 0 0 1 1 0 1 0	-	-	-	-	-	Yb
r030	0 0 0 0 1 1 1 1 0	-	-	Yb	-	Yb	-
r031	0 0 0 0 1 1 1 1 1	Yb	Yb	-	Yb	-	-
r063	0 0 0 1 1 1 1 1 1	-	Yb	-	Yb	-	-
r092	0 0 1 0 1 1 1 0 0	-	-	Yb	-	Yb	-
r095	0 0 1 0 1 1 1 1 1	-	-	-	Yb	-	-
r127	0 0 1 1 1 1 1 1 1	-	-	-	Yb	-	-
r159	0 1 0 0 1 1 1 1 1	-	-	-	Yb	-	-
r179	0 1 0 1 1 0 0 1 1	Ym	-	-	Ym	-	-
r191	0 1 0 1 1 1 1 1 1	-	Yb	-	Yb	-	-
r220	0 1 1 0 1 1 1 0 0	Ym	Yb	-	Ym	-	-

r223	0 1 1 0 1 1 1 1 1	-	-	-	Yb	-	-
r252	0 1 1 1 1 1 1 0 0	-	Yb	-	-	-	-
r253	0 1 1 1 1 1 1 0 1	-	Yb	-	-	-	-
r255	0 1 1 1 1 1 1 1 1	-	Yb	-	Ym	-	-
r272	1 0 0 0 1 0 0 0 0	-	-	-	-	-	Ym
r277	1 0 0 0 1 0 1 0 1	-	-	-	Yb	-	-
r279	1 0 0 0 1 0 1 1 1	Yb	-	-	-	-	
r284	1 0 0 0 1 1 1 0 0	-	-	Yb	-	Yb	-
r285	1 0 0 0 1 1 1 0 1	Yb	Yb	-	-	-	-
r287	1 0 0 0 1 1 1 1 1	Yb	Yb	-	Yb	-	-
r311	1 0 0 1 1 0 1 1 1	-	-	-	Yb	-	-
r316	1 0 0 1 1 1 1 0 0	-	Yb	-	-	-	-
r317	1 0 0 1 1 1 1 0 1	-	Yb	-	-	-	-
r319	1 0 0 1 1 1 1 1 1	-	Yb	-	-	-	-
r349	1 0 1 0 1 1 1 0 1	-	-	-	Yb	-	-
r373	1 0 1 1 1 0 1 0 1	-	-	-	Yb	-	-
r400	1 1 0 0 1 0 0 0 0	Ym	-	-	Ym	-	-
r412	1 1 0 0 1 1 1 0 0	-	Yb	-	-	-	-
r415	1 1 0 0 1 1 1 1 1	-	-	-	Yb	-	-
r435	1 1 0 1 1 0 0 1 1	Ym	-	-	Ym	-	-
r439	1 1 0 1 1 0 1 1 1	-	-	-	Yb	-	-
r444	1 1 0 1 1 1 1 0 0	-	Yb	-	-	-	-
r445	1 1 0 1 1 1 1 0 1	-	Yb	-	-	-	-
r447	1 1 0 1 1 1 1 1 1	-	Yb	-	-	-	-
r476	1 1 1 0 1 1 1 0 0	Ym	Yb	-	Ym	-	-
r477	1 1 1 0 1 1 1 0 1	-	-	-	Yb	-	-
r501	1 1 1 1 1 0 1 0 1	-	-	-	Yb	-	-
r508	1 1 1 1 1 1 1 0 0	-	Yb	-	-	-	-
r509	1 1 1 1 1 1 1 0 1	-	Yb	-	-	-	-
r511	1 1 1 1 1 1 1 1 1	Yb	-	-	-	-	-

The symbols in Table 4.2 are as follows

- (dash) = not applicable
Yb = allowed in both single-valued and multi-valued vector maps
Ym = allowed only in multi-valued vector maps

4.2.3 Relationships Among Geometric Primitives

Two geometric primitives are used in the DMMVM: arc and node. The third primitive, face, was not required for the modelling in 2D topologic space because, since planarity is enforced, every arc will have a face on either side, and since a face will always represent a 2D terrain object, the arc can reference the object directly (see §3.1.1). Thus there are three groups of binary topologic relationships among the primitives, namely arc/arc, arc/node and node/node.

The definitions of the boundary, interior and exterior components of a node are similar to those of a point object defined above. The boundary of the arc, denoted ∂a, is defined by its two end-nodes. The interior, denoted $^{\circ}a$, consists of the set of points of the arc, excluding the boundary points. Since the arc is defined here as a straight line segment, the interior points are not explicitly represented but can be derived by interpolation. The exterior of the arc, denoted ^{-}a, $= \mathfrak{U} - (\partial a \cup {}^{\circ}a)$.

Applying the same procedure as in object-level topology, the topologic relationships in Figure 4.11 for the two topologic primitives are derived. The "r..." (e.g., r031) represents the code of a relationship and the binary value in parenthesis (e.g., 000011111) is the 9-intersection value of the relationship. The components of the 9-intersection are ordered as in object-level topology (see §4.2.2.2). The asterisked relationships in the figure are not consistent with the DMMVM and can therefore not be detected by simple query. Actually, they can occur only before a consistent vector map, based on the DMMVM, is constructed i.e., during the process of database creation and updating. Thus they can be detected only by computational geometry, using coordinate information. They are necessary for two purposes: (1) to detect geometric inconsistency in the database and (2) for on-line geometric updating of the database. These are further explained in the next chapter. Note that at this level, the same situation holds for both single-valued and multi-valued vector maps because each primitive (arc or node) must have a one-to-one link with an m-container in the DMMVM (e.g., one arc represents (part of) just one 1-container).

Table 4.3 Number of allowed object level topologic relationships in vector maps

Object Combination	Total allowed in SVVM	Total allowed in MVVM
Area/Area (A/A)	3	7
Area/Line (A/L)	19	19
Area/Point (A/P)	3	3
Line/Line (L/L)	12	16
Line/Point (L/P)	3	3
Point/Point (P/P)	1	2

4.3 Algorithms for Detecting the Existing Topologic Relationship

Only one of the candidate relationships derived in §4.2.2.2 can exist between two objects at any one time. This section describes how to detect the existing relationship between two objects in a vector map. Generally, topologic relationships between objects stored in a spatial database can be derived in two ways:
(1) through computational geometry, for non-topologic data structures and during the process of creating or updating a topologic database, and
(2) through query, for topologic data structures.

In topologic data structures such as the FDS and the DMMVM, most of the topologic relationships supported will be implicitly represented. Those that would require the use of computational geometry are usually represented explicitly so that they can be retrieved without the use of coordinate information (e.g., point-in-polygon). Deriving the implicit relationships in a spatial database using conventional database management system (DBMS) may require some programming effort by the user because the query language of the DBMS often lacks spatial operators. Provision of fundamental spatial operators in a spatial query language to detect the elementary topologic relationships will improve the performance of the information system. This section outlines how the topologic relationships among objects formalized in §4.2.2 can be realised in a graph-structured vector map. The outlines can be formulated into topologic relationship operators as a software module in an operational GIS.

The relationships between any two elements of a graph were described in §4.2.1. By analyzing these relationships and the set intersection between the subgraphs $G_P(N_P, A_P)$ and $G_Q(N_Q, A_Q)$ of two arbitrary elementary objects P and Q, the existing topologic relationship between the two objects can be detected. The analyses which can be translated (programmed) into an overloaded spatial operator *Relation(P,Q)* (where P and Q are related to the relevant object type at run-time, i.e., late binding) consist of the following six main operations.

(1) *Evaluate* $A_P \cap A_Q$
Set intersection, i.e., evaluate whether there are common and/or different elements between A_P and A_Q; this will indicate the possible types of relationships to further examine, e.g., considering P and Q as area objects, an empty set reduces the possible types to r031, r179 and r220 see Figure 4.7

(2) *Evaluate* $N_P \cap N_Q$
Set intersection: to check whether some nodes are common to both P and Q. If P and Q have some arcs in common, it implies that they have the nodes of those arcs in common. But they can also have some nodes in common irrespective of whether or not they have common arcs (e.g., see Figures 4.4b, c and d). This is useful, for example, to determine whether P and Q intersect, overlap with common boundaries or touch.

(3) *Determine Degree*(n_i) $\forall$ $n_i \in N_P \cap N_Q$ *with respect to subgraph* $G_P \cup G_Q$
(This helps to detect differences among similar relationships, e.g., a line object that branches off another line object will have a common node of degree 3, while intersecting (crossing) line objects will have a common node of degree 4.)

(4a) *Evaluate whether* $\exists\, a_i \in A_P \ni B[a_i,Q]=0$
(i.e., whether any arc of P lies in the exterior of Q.)

(b) *Evaluate whether* $\exists\, a_j \in A_P \ni B[a_j,Q]=1$
(i.e., whether any arc of p is part of Q's boundary.)

(c) *Evaluate whether* $\exists\, a_k \in A_P \ni B[a_k,Q]=2$
(i.e., whether any arc of P lies in Q's interior.)
The three operations give further indication of the type of relationship, e.g., if each of the sets derived in a, b and c is not empty with respect to two area objects, it implies a variant of the relationship r511 (see Figure 4.7). They are necessary for only area/area and area/line combinations.

(5a) *Evaluate whether* $\exists\, a_l \in A_Q \ni B[a_l,P]=0$
(i.e., whether any arc of Q lies in the exterior of P.)

(b) *Evaluate whether* $\exists\, a_m \in A_Q \ni B[a_m,P]=1$
(i.e., whether any arc of Q is part of P's boundary.)

(c) *Evaluate whether* $\exists\, a_n \in A_Q \ni B[a_n,P]=2$
(i.e., whether any arc of Q lies in P's interior.)
The operations performed for P in 4a, b and c are repeated for Q when dealing with two area objects. Operation 4 alone will suffice in the case of the area/line object combination.

(6) *Evaluate* $^{\circ}P \cap N_Q$
(i.e., whether a node of Q lies in the interior part of P; required for only area/point combination)

The third operation will return a list of integer values indicating the degrees of node n_i, the sixth operation will return true (T) or false (F), while other operations are evaluated as empty ($\varnothing$) or not empty, with the elements of the set being kept for further analysis if necessary. Not all the six operations will be required for each object combination. In some cases, the occurrence of an operation may already be implied by a previous one. For instance, $^{\circ}P \cap N_Q = T \Rightarrow B[a_j,Q]=2$ where P = area object and Q = line object, and $A_P \cap A_Q \neq \varnothing \Rightarrow N_P \cap N_Q \neq \varnothing$. Also, for any point object Q, $A_Q = \varnothing$; hence operations 1, 4 and 5 will not be applicable for combinations involving point objects, i.e., Area/Point, Line/Point and Point/Point. In addition, operation 6 is required only for Area/Point combination to determine if a point object is topologically inside an area object. This relationship can be expressed by the following function, where P = area object and Q = point object:

$N_Q \cap {}^{\circ}P = \text{Repr}[n_q,Q]*\text{ISIN}[n_q,F_a]*\text{Partof}[F_a,P]$ where F_a = face a.
$N_Q \cap {}^{\circ}P = 1 \Rightarrow$ point object is inside area object, and $N_Q \cap {}^{\circ}P = 0$ means it is not.

The input into the procedure Relation(P,Q) will be the geometric primitives of objects P and Q and the return value will be the existing topologic relationship between P and Q.

The relevant operations for each object combination are elaborated in the following subsections.

4.3.1 Detecting Topologic Relationship Between Two Area Objects

The following expressions should be determined in order to detect the existing relationship between two area objects in our model. In the expressions, Vi represents the return value of the expression while the other terms are as defined above.

(1) $V1 = A_P \cap A_Q$
i.e., V1 is a set containing the set of arcs which are common to both P and Q
(2) $V2 = A_P - V1$
i.e., V2 is a set containing the set of arcs which define geometry of P only
(3) $V3 = A_Q - V1$
i.e., V3 is a set containing the set of arcs which define geometry of Q only
(4) $V4 = N_P \cap N_Q$
i.e., V4 is a set containing the set of nodes which are common to both P and Q
(5) $V5 = \{a_i \in A_P \ni B[a_i,Q] = 0\}$
i.e., the set of arcs of P which intersect the exterior part of Q.
(6) $V6 = \{a_j \in A_P \ni B[a_j,Q] = 1\}$
i.e., the set of arcs of P which are part of boundary of Q.
(7) $V7 = \{a_k \in A_P \ni B[a_k,Q] = 2\}$
i.e., the set of arcs of P which intersect interior part of Q (this is possible only in a multi-valued vector map; a non-empty set in a single-valued map indicates inconsistency).
(8) $V8 = \{a_l \in A_Q \ni B[a_l,P] = 0\}$
i.e., set of arcs of Q which intersect the exterior part of P.
(9) $V9 = \{a_m \in A_Q \ni B[a_m,P] = 1\}$
i.e., the set of arcs of Q which are part of boundary of P.
(10) $V10 = \{a_n \in A_Q \ni B[a_n,P] = 2\}$
i.e., set of arcs of Q which intersect interior part of P (this is possible only in a multi-valued vector map; a non-empty set in a single-valued map indicates inconsistency).

Each expression is evaluated as empty (∅) or non-empty (¬∅). The combined values of Vi, i = 1,10, will be unique for each of the ten elementary topologic relationships, as indicated by the examples in Table 4.4 (relate the examples with Figure 4.7).

Table 4.4 Values for some relationships between two area objects

Relation	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
r031	∅	¬∅	¬∅	∅	¬∅	∅	∅	¬∅	∅	∅
r179	∅	¬∅	¬∅	∅	∅	∅	¬∅	¬∅	∅	∅
r287 (touch as in Figure 4.5a)	∅	¬∅	¬∅	¬∅	¬∅	∅	∅	¬∅	∅	∅
r287 (touch as in Figure 4.5b)	¬∅	¬∅	¬∅	¬∅	¬∅	¬∅	∅	¬∅	¬∅	∅

"If *condition* then *action*" rules can then be formulated in the system based on the values of

Vi for each relationship. Then, when it is required to find an existing relationship, the values of Vi will be computed for the object pair and compared with the predefined configuration to ascertain which of the relationships holds.

The procedure can be further optimised by analyzing the implication of one result for the next operation. For instance, if V1 ≠ ∅ and V2 = ∅ and V3 = ∅, then Relation(P,Q) = r400 (i.e., equal) thus it is not necessary to evaluate the remaining expressions.

4.3.2 Detecting the Topologic Relationship Between an Area Object (P) and a Line Object (Q)

To detect the existing relationship between an area object P and a line object Q in our model, the following expressions should be determined where V1, V2, V3 and V4 are as defined above. The eight expressions would be analyzed conjunctively to determine the elementary relationship that exists between the area object P and the line object Q. These can be programmed as spatial operators in the database.

(1) $V1 = A_P \cap A_Q$
(2) $V2 = A_P - V1$
(3) $V3 = A_Q - V1$
(4) $V4 = N_P \cap N_Q$
(5) $V5 = \{a_i \in A_Q \ni B[a_i,P] = 0\}$
i.e., set of arcs of Q which intersect the exterior part of P.
(6) $V6 = [\text{Degree}(n_e) \mid \{N[a_i,n_e] = 1\}]$, $a_i \in V5$ and $n_e \in N_Q$;
(degree computed with respect to the subgraph $G_P \cup G_Q$)
V6 is an array containing the degree of each node n_e. The minimum degree will be 1 and the maximum will be 4 with two intermediate values 2 and 3. These should be analyzed further to determine the location of the endpoints of the line object, e.g., two occurrences of the minimum degree means that the two endpoints of Q are located in the exterior of P; only one with minimum and one with degree 3 will indicate that one endpoint of Q is outside of P and one is probably on the border (confirmed to be on border if V7 is empty); one minimum and one maximum implies one endpoint outside and one probably inside of P (confirmed by analysis of V8); two nodes with degree 3 and others with degree 2 implies two endpoints on border of P (confirmed if V7 is empty); one node with degree 3 and one with maximum degree with others having degree 2 implies one endpoint on border and one inside P (confirmed if V7 is not empty); two nodes with maximum degree and others with degree 2 means the two

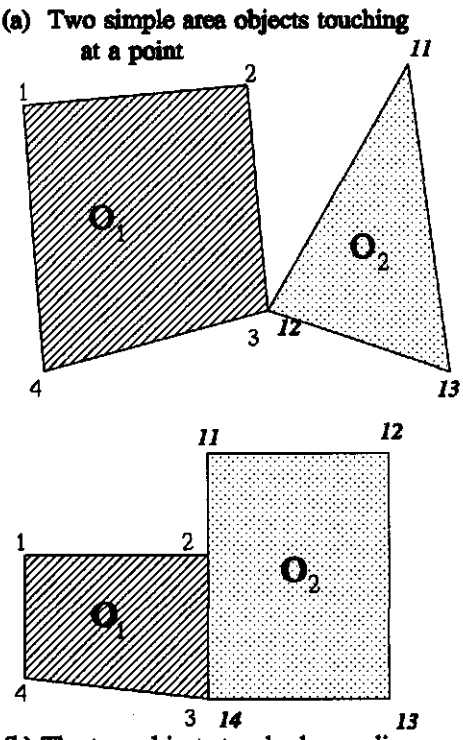


Figure 4.5 Topologic relationship r287 (touch) between two simple area objects

endpoints are inside P.

(7) V7 = $\{a_k \in A_Q \ni B[a_k,P] = 2\}$
i.e., arcs of Q which intersect interior part of P

(8) V8 = [Degree(n_e) | $\{N[a_k,n_e] = 1\}$], $a_k \in$ V7 and $n_e \in N_Q$
V8 is an array containing the degree of each node n_e. Like V6, V8 should be analyzed further to determine the location of the endpoints of the line object, e.g., two nodes with minimum degree means that the two endpoints of Q are located in the interior of P; only one minimum and one node with degree 3 means one endpoint of Q inside P and one is probably on the border of P (confirmed to be on border if V5 is empty); one minimum and one with maximum degree implies one endpoint inside and one outside P; two nodes with degree 3 and others with degree 2 implies two endpoints on border of P (confirmed if V5 is empty); one node with degree 3 and one with maximum degree with others having degree 2 implies one endpoint on border and one outside P (confirmed if V5 is not empty); two nodes with maximum and others with degree 2 means the two endpoints are outside P.

For example, let us assume that Figure 4.6 is part of a consistent vector map (with nodes, say n1 and n2, created at the two points where the road intersects the boundary of soil a1: n1 introduced to decompose arcs (1,4) and (11,12) and n2 to decompose arcs (2,3) and (12,13)). The relationship between the soil unit (area object P) and the road (line object Q) which is similar to the query " find the area object P through which the line object Q passes", i.e., relationship r191 (see Figure 4.8) gives V1 = ∅, V2 = ¬∅, V3 = ¬∅, V4 = {n1,n2} (i.e.,¬∅), V5 = {(11,n1), (n2,13)} (i.e., ¬∅), V6 (for only nodes 11, n1, n2, 13 respectively) = [1, 4, 4, 1], i.e., two nodes with minimum degree hence endpoints outside, V7 = {(n1,12), (12,n2)}, V8 = [4, 2, 4] (confirms that the two endpoints are outside). Compare this result with relationship r447 (as represented in Figure 4.8) which is almost similar to r191. For r447, V1 = ∅, V2 = ¬∅, V3 = ¬∅, V4 = ¬∅, V5 = ¬∅, V6 = [4, 1] (confirms that one endpoint is outside and one probably inside), V7 = ¬∅, V8 = [3, 2, .., 2, 4] (ie., one node with degree 3, one with maximum degree and others with degree 2, hence one endpoint on border and one outside. Note that for both relationships, V1, V2, V3, V4, V5, and V7 are similar, the difference being realised through the results and analyses of V6 and V8. Thus when translating the eight steps into a spatial operator for area/line topology, the analyses must be carefully incorporated in order to distinguish between seemingly similar relationships.

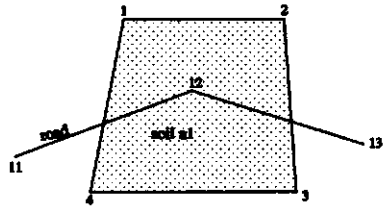


Figure 4.6. A new line object (road) passing through existing area object (soil)

4.3.3. Detecting Topologic Relationship Between Area Object and Point Object

To determine the existing relationship between an area object P and a point object Q, the following expressions, which are a subset of the steps in §4.3, will suffice.

(1) V1 = $N_P \cap N_Q$
i.e., check if node of Q is an element of the set of nodes of P. Since N_Q contains only

one element, V1 will also contain one element if it is not empty. If V1 is not empty, it means that the relationship r284 (see Figure 4.10a) exists and the next step is not required. If V1 is empty, perform the next operation.

(2) $V2 = {}^{\circ}P \cap N_Q$

If V2 is true, it means the existence of r092 (see Figure 4.10a), while Boolean value false means the existence of r030 (Figure 4.10a). Note that V2 can be determined only through computational geometry (e.g., using point-in-polygon algorithm) unless the relationship is explicitly represented.

4.3.4. Detecting Topologic Relationship between Two Line Objects

The existing relationship between two line objects P and Q can be detected with the following algorithm consisting of six operations to be determined and analyzed. Vi, i =1,4 is as defined in §4.3.1.

(1) $V1 = A_P \cap A_Q$
(2) $V2 = A_P - V1$
(3) $V3 = A_Q - V1$
(4) $V4 = N_P \cap N_Q$
(5) $V5 = [Degree_P(n_j),\ n_j \in V4]$

V5 is an array containing the degree of each node n_j in G_P

(6) $V6 = [Degree_Q(n_j),\ n_j \in V4]$

V6 is an array containing the degree of each node n_j in G_Q

The maximum degree of n_j in V5 and V6 will be 2 (indicating middle of line) and the minimum will be 1 (indicating endpoint). By comparing the degree of n_j in V5 and V6, it should be possible to distinguish an intersection at the end of a line object or somewhere in its middle (like a road branching off another compared with two crossing each other). The combination of this with the values of Vi, i =1,4, can then be applied to detect the topologic relationship between two elementary line objects.

For example, compare the values for the five relationships in Table 4.5 (relate with Figure 4.9)

Table 4.5 Values for some relationships between two line objects

Relation (see Figure 4.9)	V1	V2	V3	V4	V5	V6
r031	∅	¬∅	¬∅	∅	[ ]	[ ]
r095	∅	¬∅	¬∅	¬∅	[2]	[1]
r220	¬∅	¬∅	∅	¬∅	[2,2]	[1,1]
r400	¬∅	∅	∅	¬∅	[1,...,1]	[1,...,1]
r435	¬∅	∅	¬∅	¬∅	[1,1]	[1,2]

4.3.5. Detecting Topologic Relationship Between a Line Object and a Point Object

The following two operations will suffice to determine the existing relationship between a line object P and a point object Q:

(1) $V1 = N_P \cap N_Q$
(2) $V2 = Degree_P(n_j)$, $n_j \in V1$

V2 is computed only if V1 is not empty and it indicates whether the point lies in the middle of the line object, wherein V2 = 2 (relationship r092, Figure 4.10b), or at one of its endpoints, wherein V2 = 1 (relationship r284). If V1 is empty, it indicates relationship r030.

4.3.6. Detecting Topologic Relationship Between Two Point Objects

Only one operation is required to determine the existing relationship between two point objects, viz.

Determine $V1 = N_P \cap N_Q$

If V1 is empty, it indicates relationship r026 and non-empty set (with one element) indicates r272 (see Figure 4.10c).

4.4 Summary

In this chapter, an extensive set of object-level topologic relationships in vector maps has been derived using 9-intersection point-set algebraic topology. The topologic relationships between geometric primitives were also derived; these can be combined with the graph structure of the model and used as the elementary set from which the object-level relationships are derived. Algorithms are then defined for detecting the occurrence of any of the elementary relationships for any object combination. The algorithms can be translated to topologic operators and used for topologic queries, as well as a tool for detecting violation of and enforcing geometric constraints. In the latter case, the topologic operators will serve as detectors of inconsistencies. The return value of an operator will trigger the relevant rule that will enforce consistency if violation occurs. The rules that will enforce the geometric consistency are presented in the next chapter.

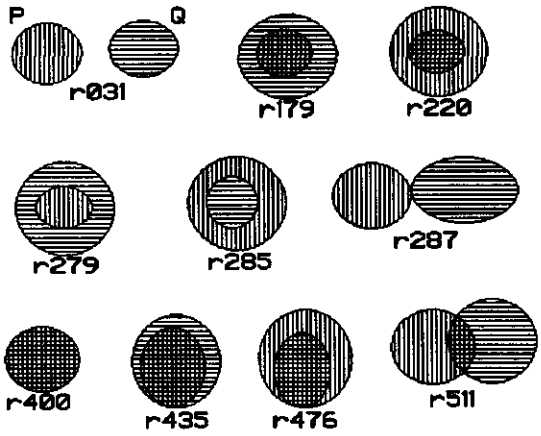


Figure 4.7 Topologic relationships between two area objects

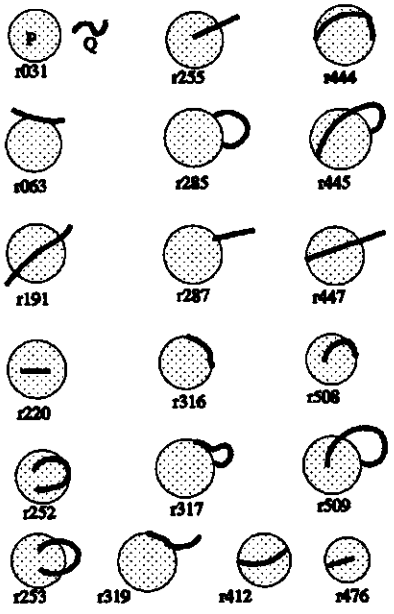


Figure 4.8 Topologic relationships between an area object and a line object

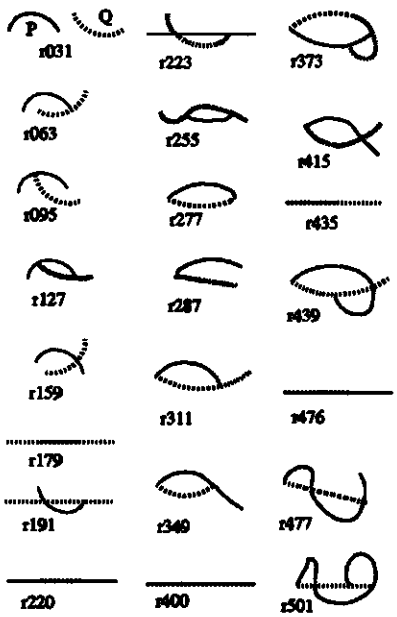


Figure 4.9 Topologic relationships between two line objects

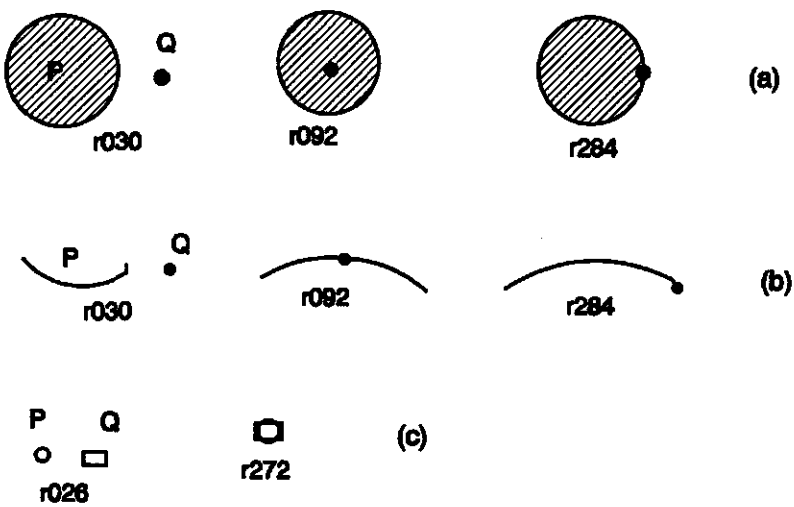
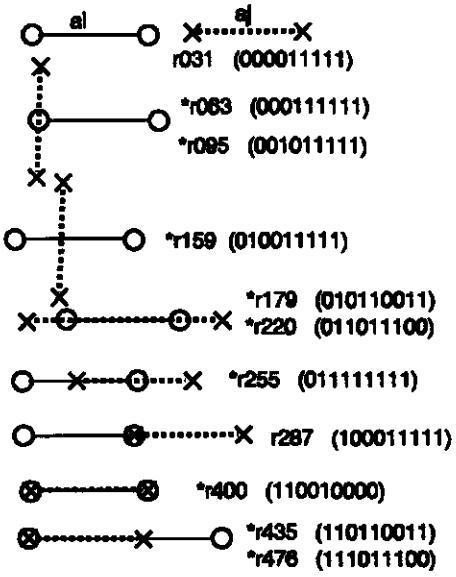
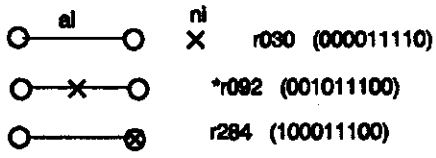


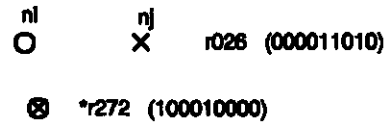
Figure 4.10 Topologic relationships between (a) an area object and a point object, (b) a line object and a point object, (c) two point objects



(A) Between two arcs ai and aj



(B) Between arc ai and node ni



(C) Between two nodes ni and nj

Figure 4.11 Topologic relationships between the geometric primitives

5

MONITORING AND ENFORCING INTEGRITY CONSTRAINTS IN VECTOR MAPS

For every large data collection intended for repeated production of information, as in GIS, controls must be installed to check that all incoming data or changes made to existing data follow some pre-defined rules and do not contradict existing data in the database. Internal contradictions in the data will cause retrieval of inconsistent information from the database, which leads to users' lack of confidence in the data (Frank, 1984). The logical consistency of data is in fact one of the main components of data quality in GIS, a constituent of reliability. The other components of data quality are positional accuracy, attribute accuracy, completeness and lineage (DCDSTF, 1988; Aronof, 1989). Like other components, it deserves serious attention in spatial databases, considering the fact that the major cost of setting up a GIS arises from data acquisition. The logical consistency of the data contained in a database can be ensured by enforcing integrity constraints. Integrity refers to the (logical) accuracy or validity of data (Date, 1990). Consistency or integrity constraints are statements that must always be true for data items in the database (Zdonic and Maier, 1990), i.e, they can be regarded as conditions that a correct state of the database is required to satisfy.

Integrity constraints may exist in different forms, from simple (e.g., specifying that all object identifiers must be represented by a four-byte positive integer), to more complex constraints (e.g., the boundary polygon of a land parcel must be closed and must abut a road). While the former can be enforced at run-time by the system, the latter has to be monitored and enforced by some kind of user-defined consistency rules. Lack of data integrity can arise from many different sources (Hughes, 1991), including data entry errors, logical errors in the application program, errors in system software which result in data corruption, and topologic errors. Unfortunately, most commercial GIS software does not adequately support integrity constraints - except for some constraints on the domain of object identifiers, key constraints, or referential constraints (Date, 1990; Hughes, 1991; Nassif et al, 1991; Kemppainen, 1992) - and most integrity checking is still done by user-written procedural code. It is preferable to specify integrity constraints in a more declarative fashion so that the system can do the checking. Furthermore, in spatial databases (where terrain objects are the main focus), because the objects do change with time, it is necessary to update the database, whereas updating operations always imply the risk of disturbing data consistency. To prevent this, the information system should be able to monitor its own consistency and to take measures to preserve it (Molenaar, 1991c). Thus integrity enforcement is an important aspect of GIS development.

In general, two types of integrity constraints can be distinguished (Nassif et al, 1991; Hadzilacos and Tryfona, 1992):
(1) Static integrity constraints, which define the valid state of the database.
(2) Dynamic integrity constraints, which are the conditions on the allowable transitions from one database state to another.

The static constraints are closely related to the structure and semantics of the embedding data model, while the dynamic constraints relate mainly to the implementation domain, e.g., how

a multi-user database is managed during an updating session. The focus in this thesis is on the static consistency constraints.

In a spatial database, an important consideration for determining whether a database state is correct or not is topology. For example, two neighbouring countries must have a common border, two land parcels cannot overlap, every parcel must have access to a road, etc, are all topologic constraints. However, not all constraints are topologic. Some are specific to the database model used for the implementation of a conceptual data model. For example, the three types of integrity constraints in a relational database model, namely (1) domain integrity which specifies the range of legal values for each field in a relation, (2) intra-relation integrity which relates to the correctness of relationships among attributes of the same relation and to the preservation of key uniqueness, and (3) referential integrity constraints, which assert that a reference in one data item indeed leads to another data item (e.g., when an arc is part of a road, then a data item for that road must actually exist). Some DBMSs provide triggers to help enforce some of these database model-related constraints, although mainly for the domain integrity constraint. In general, to enforce consistency, actions can be initiated on access to particular data items, either to check that the stated constraints hold or to perform additional updates to bring the database to a state of consistency.

This thesis focuses on the provision of consistency rules for monitoring and enforcing static geometric integrity constraints under updating operations in vector-structured spatial databases, using the DMMVM as a framework. The rules should ensure that the database state after an update is a correct mirror of the reality that it models. This implies that the database is assumed to be consistent before an update operation. This assumption will hold where initial database creation is also regarded as updating, in this case from "zero-level" with enforcement of consistency during data input. This approach has been adopted in this thesis.
The proposed consistency rules can be grouped into two classes, namely
(1) the consistency rules for the geometric structure of the data model, i.e., structural integrity rules,
(2) consistency rules related to the application making use of the model, i.e., application-dependent semantic integrity rules.

The simple constraints in the two groups -- such as uniqueness of identifiers, unique occurrence of instances of data types -- can easily be checked and enforced during implementation. These are related to the definition of acceptable values (domain) for the identifiers, as well as rules for installation-wide uniqueness and existence of the identifiers. Many systems do provide automatic immutable identification of objects (e.g., Postgres and Arc/Info) together with indexing mechanisms to eliminate repeated identifiers where the identifier is defined by the user. The user-defined identifiers are often mapped to one of the system's base types, which are already provided with operators to reject non-valid values. Thus these simple constraints will not be given attention in this thesis.

In the following section, constraints in the first group will be analyzed with the corresponding rules for monitoring and enforcing the constraints. The second group will be treated in §5.2. An approach for their implementation is outlined in §5.3, with a summary in §5.4. The topologic relationships formalized in the preceding chapter will play a major role in the monitoring of the two groups of constraints.

5.1 Structural Consistency Rules

From the basic structure of terrain objects in figure 3.2, the three components of each object, namely the object identifier and its two semantic characteristics (thematic and geometric), represented in the database do have their individual integrity constraints. Because objects from different layers can overlap in space, the data type m-container was introduced to uniquely identify overlapping sections of objects; the geometric description of an object is therefore composed of the geometry of its component m-containers. Each m-container is geometrically described by the two (geometric) primitives, arc and node. These m-containers and the geometric primitives also have integrity constraints which must be monitored and maintained. In addition, there are consistency rules for the functional relationships among these data types. Therefore, the structural consistency constraints for a vector map that is based on the DMMVM can be analyzed in different levels of complexity as follows.

(1) Consistency rules for the geometric primitives arc and node.
(2) Consistency rules for the geometric structure of m-container types.
(3) Consistency rule for the planarity of the map (Euler constant)
(4) Consistency rules for the structure of object types
    - rules related to the thematic component of objects,
    - rules for the geometric descriptions of objects.
(5) Consistency rules for the functional relationships among the data types (node, arc, m-container, and object).

5.1.1 Consistency Rules for the Geometric Primitives

When the geometry of a terrain object changes or when a new one has to be inserted in the database, it implies insertion, modification or deletion of some arcs and nodes. This means that updating operations have to be performed on the set of arcs and nodes related to the object in a manner that maintains the geometric consistency of the database, i.e., in a manner that preserves the topology of geometric primitives. This is the lowest level of consistency rules for vector maps. It is the lowest level because updating of the geometric primitives depends on the behaviour of m-containers whose updating depends on the dynamics of terrain objects. At this level, the system should check that conventions 3 to 6 of §3.4.4 are fulfilled for each arc or node introduced into the database.

Conventions 3 and 4 can be easily enforced during the data collection phase, i.e., each point introduced should be treated as a node and each straight line segment should be treated as an arc. Conventions 5 and 6 are topologic in nature and can therefore be translated broadly into the following topologic constraints, with an additional one for nodes.

Geometric Primitive Constraint #1 (GPC_1): Nodes must not overlap

Geometric Primitive Constraint #2 (GPC_2): Arcs must not intersect

Geometric Primitive Constraint #3 (GPC_3): Arcs must not overlap

Geometric Primitive Constraint #4 (GPC_4): An arc must be defined by only two adjacent node.

The GPC_1 arises from the fact that there is a single-valued link between a node (geometric primitive) and a 0-container. In addition, because geometric primitives can be shared by objects, a node that defines a 0-container can also be a node of an arc, but with the conditions that only a single set of coordinates are kept for the node and the identifier must be unique. Hence we still have the condition that two nodes may not overlap. This condition is violated if the position of a new point coincides with that of an existing point. GPC_2 is a planarity condition in graph theory which formed the basis of our model (see §3.1.1). GPC_3 is akin to GPC_1 because an arc must have a single-valued link to a 1-container and/or boundary of a 2-container. The fourth arises because we treat every point as a node (see convention 3 in §3.2.3) hence, when a new node (representing a new point object for instance) falls on an existing arc (but not on its nodes) the existing arc must be decomposed.

Violation of any of the four conditions can be interpreted as an inconsistent topologic relationship between two geometric primitives. To ensure that the constraints are not violated during database updating, an automated consistency rule for monitoring and enforcing them must be provided. The strategy being proposed here is to use topologic relationships among geometric primitives as "alerters" and to define the consistency operations that should be performed by the system in response to each inconsistent topologic relationship in order to maintain geometric consistency. In other words, when a geometric primitive is inserted into the database, the consequent automated updating of the database involves two integrated operations: (1) evaluation of the topologic relationship between the primitive and each of the other existing primitives, and (2) the result of the evaluation (type of relationship) triggers the necessary consistency rules.

When GPC_1 is converted to a topologic relationship between geometric primitives (see Figure 4.11), its violation implies one occurrence of the following set of relationships: {r272(node,node), r284(arc,node), r287(arc,arc), r400(arc,arc), r435(arc,arc), r476(arc,arc)}. Violation of GPC_2 means the occurrence of one of the following set of topologic relationships: {r063(arc,arc), r095(arc,arc), r159(arc,arc)} while a violation of GPC_3 implies the existence of a member of the topologic relationship set {r179(arc,arc), r220(arc,arc), r255(arc,arc), r400(arc,arc), r435(arc,arc), r476(arc,arc)}. The fourth, GPC_4 is violated if there is occurrence of a relationship $\in$ {r092(arc, node), r063(arc, arc), r095(arc, arc), r179-(arc, arc), r220(arc, arc), r255(arc, arc), r435(arc, arc), r476(arc, arc)}. The four topologic constraints can therefore be represented by a negation (using $\neg$) of the applicable relationship, e.g., $\neg$r159(arc1,arc2) means "two arcs must not intersect".
To detect these inconsistent topologic relationships, computational geometry must be used. For instance, an algorithm similar to line intersection will be used to check whether two arcs intersect.

The consistency rules for monitoring and enforcing the three constraints as defined in Kufoniyi et al (1993 and 1994) are described in the following section.
The rules can be translated into the *IF condition THEN action* convention as follows, using relation r272(node, node) as an example.

If *Relation(NewNode, OldNode) = r272*
Then
do *GP_Rule_1*

The mandatory time to enforce the rules is during insertion of a geometric primitive. They

can also be checked, optionally, at some defined intervals to validate the reliability of the database.

Rules for enforcing consistency of the geometric primitives

The consistency operations that must be performed to maintain the four constraints, GPC_1, GPC_2, GPC_3, and GPC_4, during (geometric) updating are defined in Table 5.1. The following notations are used in the table.
Let the universal set of arcs of the map be represented by A where ai $\in$ A. Let A be further subdivided into two subsets P and Q where P = set of arcs already existing in the database and Q = set of new arcs to be inserted, i.e., A = P $\cup$ Q. Note that P and Q must be mutually exclusive $\ni$ P $\cap$ Q = $\varnothing$ hence ai $\in$ P $\Rightarrow$ ai $\notin$ Q. In other words, if a new arc already has a counterpart in the database, the new arc should not be accepted by the system; rather the affected properties of the existing arc (brought by the new arc) should be only updated to reflect the changes.
Let the universal set of nodes of the map = N $\ni$ n $\in$ N. Let N be further subdivided into p and q where p = set of nodes existing in the database and q = set of new nodes to be inserted. Hence, as in arcs, N = p $\cup$ q and n $\in$ p $\Rightarrow$ n $\notin$ q.
One existing arc is then defined by P1 = (p1,p2) where p1 and p2 are respectively the starting and ending nodes of the arc, while a new arc is represented by Q1 = (q1,q2) where q1 and q2 are respectively the starting and ending nodes of the arc.
The fact that a node of one arc, say q1 of Q1, intersects the interior (°) of another arc, say a1, is represented as q1 $\cap$ °a1.

Table 5.1 Consistency rules for geometric primitives

Topologic Constraint	Intersection Rules	Violation response	Name of Rule
¬r272(n1, n2)	n1 $\in$ p, n2 $\in$ q or v.v.	store the more accurate coord., assign number of existing node to new.	GP_Rule_1
¬r092(ai, n)	ai$\in$Q, n$\in$p	decompose ai into (q1,n) & (n,q2)	GP_Rule_2a
	ai$\in$P, n$\in$q	decompose ai into (p1,n) & (n,p2)	GP_Rule_2b
¬r063(a1, a2)	a1$\in$P, a2$\in$Q	do GP_Rule_2b $\ni$ n $\in$ {q1,q2}	GP_Rule_3
¬r095(a1, a2)	a1$\in$Q, a2$\in$P	do GP_Rule_2a $\ni$ n $\in$ {p1,p2}	GP_Rule_4
¬r159(a1, a2)	a1$\in$P, a2$\in$Q or v.v.	replace a1 and a2 by four arcs joining at new node, compute and insert coord. of new node.	GP_Rule_5
¬r179(a1, a2)	a1$\in$Q, a2$\in$P	split a2 into (p1,q1), (q1,q2), (q2,p2); update attributes of a1	GP_Rule_6
¬r220(a1, a2)	a1$\in$P, a2$\in$Q	split a2 into (q1,p1), (p1,p2), (p2,q2); update attributes of a1	GP_Rule_7

¬r255(a1, a2)	a1∈Q, a2∈P or v.v.	If (q2∩°a2∧p1∩°a1) then replace a1 by (q1,p1), do GP_Rule_2b ∋ n = q2, update attributes of (p1,q2), delete a2 ElseIf (q1∩°a2∧p1∩°a1) then replace a1 by (p1,q2), do GP_Rule_2b ∋ n = q1, update attributes of (p1,q1), delete a2 ElseIf (q1∩°a2∧p2∩°a1) then replace a1 by (p2,q2), do GP_Rule_2b ∋ n = q1, update attributes of (q1,p2), delete a2 ElseIf (q2∩°a2∧p2∩°a1) then replace a1 by (q1,p2), do GP_Rule_2b ∋ n = q2, update attributes of (q2,p2), delete a2	GP_Rule_8
¬r287(a1, a2)	a1∈P, a2∈Q or v.v.	do GP_Rule_1	GP_Rule_9
¬r400(a1, a2)	a1∈P, a2∈Q or v.v.	do 2x GP_Rule_1, update attributes of a1 if necessary	GP_Rule_10
¬r435(a1, a2)	a1∈Q, a2∈P	do GP_Rule_1, update attributes of a2 replace a1 by (n1,n2) ∋ n1 ∈ {p1,p2-}:n1∩°a1, n2 ∈ {q1,q2}:n2 ∉ {p1-,p2}	GP_Rule_11
¬r476(a1, a2)	a1∈P, a2∈Q	do GP_Rule_1, do GP_Rule_2b ∋ n ∈ {q1,q2}, update component of a1 which = a2, delete a1	GP_Rule_12

The 12 consistency operations for the geometric primitives can then serve as elementary operations on which updating of the geometric aspects of objects can be decomposed, as shown in the following section.

Application of the Geometric Consistency Rules

The following two examples will illustrate the use of the consistency rules defined in Table 5.1 during updating of vector maps. A more intensive illustration of the application of these rules during updating will be given in the next chapter.

(a) With reference to Figure 4.5b, suppose the area object O_1, defined by the set of arcs {(1,2), (2,3), (3,4), (4,1)}, is being newly inserted into the database, and O_2, defined by {(11,12), (12,13), (13,14), (14,11)}, already exists in the database. This situation requires geometric updating to make the database consistent. This can be effected by the system without interference of the human operator. First, the system evaluates the topologic relationship between each arc of O_1 and each arc of O_2 using the procedure Relation(arc1,arc2). The result of this procedure should then trigger the necessary updating rule. In this particular example, the system enforces geometric consistency in the following ways:
(i) Relation(arc1,arc2) where arc1 = (1,2) and arc2 = (14,11) will return r063 which then triggers the GP_Rule_3, and
(ii) Relation(arc1,arc2) where arc1 = (2,3) and arc2 = (14,2) will return r400 which triggers the GP_Rule_10.

(b) Suppose in Figure 4.6 that the road, defined by {(11,12), (12,13)}, has been inserted into a static database (i.e., without dynamic updating facility) and soil unit a1, defined by {(1,2), (2,3), (3,4) & (4,1)}, exists in the database before the insertion of the road. The situation in the figure violates the planar graph constraint of our data model because arcs (4,1) and (11,12) as well as (12,13) and (2,3) intersect without creating nodes at the points of intersections. As in the previous example, the system evaluates the topologic relationship between each arc of the road and each arc of soil unit a1. Relation(arc1,arc2) where arc1 = (11,12) and arc2 = (1,4) should detect the topologic relationship r159 which then triggers the GP_Rule_5. Similar operations will be performed for the two arcs (12,13) and (2,3).

5.1.2 Consistency Rules for the Geometric Structure of M-container Types

In this subsection, consistency rules for the geometric structure of the m-container where m $\in$ {0,1,2} are defined. The rules are at a higher level than the preceding ones since the behaviour of an m-container affects the geometric primitives that describe the m-container. They are, however, on a lower level in comparison with the consistency rules for terrain objects because the dynamics of an m-container depends on the behaviour of the terrain object(s) which the m-container is part of. If the DMMVM is used for a single layer, there will be a 1:1 relationship between the m-container and the terrain object; hence the constraints defined for the m-container become the constraints for the terrain object of equivalent dimension.

0-Container

Geometrically, a 0-container is represented by a single node with X, Y and Z coordinates. In this respect, what needs to be checked is that the node defining the geometry of the 0-container exists. This can be done by a simple query.

1-Container

The geometry of an instance of this type is defined by the subgraph $G_L(N_L,A_L)$ (see §3.4.2). The following constraints must hold for each instance, the violation of which will have the consequence of retrieving the wrong metric and topologic information for the line objects of which the affected 1-container is a part.

1-Container Constraint #1 (1CC_1): For each 1-container, a simple and elementary path must exist (see §3.1.1).

1-Container Constraint #2 (1CC_2): The length of the path in 1CC_1 must be ≥ 1.

From the geometric definition of a 1-container L, 1CC_1 translates to the fact that exactly two elements of N_L must have $degree_L(n) = 1$, while all other elements of N_L must have $degree_L(n) = 2$. In other words, a 1-container must not close back on or intersect itself. The two 1-degree nodes are the end-nodes of the 1-container.

This constraint can be checked after the overlay of two or more layers, i.e., after the creation of the multivalued spatial database, by retrieving the beginning and end nodes of each element of A_L for the subject 1-container and counting the number of times each node occurs, i.e., the degree of each node, in the list.

Violation of the constraint may arise from a gap (undershoot), hanging arc or sliver line in the chain of arcs of the 1-container. These will be manifested in the degrees of the elements of N_L. If a gap exists, there will be a minimum of four elements of N_L having degree 1. An overshoot will give rise to three nodes with degree 1 and at least one node with degree 3. A sliver line will lead to two connected nodes having degree 3. They may result from a digitizing error, e.g., a gap may be caused by a digitizing error whereby a node connecting two adjacent arcs of the 1-container is digitized twice (for each arc) with the coordinate difference between the two being greater than the defined threshold for snapping two nodes. The undershoot error will arise only if the digitizing is done in multivalued mode, i.e., direct digitizing of a multi-valued vector map from primary sources (photographs, images, etc) or from a hardcopy map. It can also be caused by incorrect coding during manual input whereby one of the arcs of the 1-container is mistakenly omitted. The errors can also be caused by a failure to georeference all layers on the same system. These types of gross error or blunder can be eliminated only by remeasuring the 1-container, or recomputation of overlay with new snapping tolerance, or proper coordinate transformation.

1CC_2 means that set A_L must contain at least one element. This implies that $A_L \neq \emptyset$. If $A_L = \emptyset$, it implies that $N_L = \emptyset$ since each element of A_L is defined by a subset of N_L. Violation of this constraint therefore means that the geometry of the 1-container is not defined at all and can be checked first within the consistency rule for 1CC_1.

Consistency Rule for 1-container

The following consistency rule, called 1CC_Rule_1, can be defined to monitor 1CC_1 and 1CC_2. The checking will occur during insertion of a 1-container or after an overlay computation.

Let Lset be the set of 1-containers with $L \in$ Lset defined by subgraph $G_L(N_L,A_L)$.
For each L

> Count number of distinct $n_i \in N_L$
> If $n_i < 2$ notify user "1-container ill-defined: length of path = 0 or is defined by only one point"

```
            Exit to data input
            Else continue
        Count occurrence of a_i ∈ A_L
        If Count(a_i) > 1 /* i.e., if a_i occurs more than once */
            Display error message: "path not simple"
            Display L for interactive editing
        continue
        For each n_i ∈ N_L
            compute degree_L(n_i)
            let j ∈ {n_i | degree_L(n_i) = 1}
               k ∈ {n_i | degree_L(n_i) = 2}
               m ∈ {n_i | degree_L(n_i) > 2}
            If ∑j = 2 and ∑k = N_L - ∑j  → consistent
                goto next L
            Elseif ∑j > 2 and ∑m = 0
                display error message: "no path: gap in 1- container"
                display L for interactive editing
            Elseif ∑j < 2 or m >= 1
                display error message: "path not elementary, loop or overshoot in 1-
                container"
                display L for interactive editing
            Endif
        Next n_i
Next L
```

2-Containers

The geometry of a 2-container is defined by the subgraph $G_E(N_E,A_E)$ (see §3.4.2).
The following constraint must hold for each instance of this type to satisfy the geometric definition.

2-Container constraint 1 (2CC_1): For all $n_i \in N_L$ $degree_L(n_i)= 2$
The implications of this constraint are
(a) The start-node and end-node of the 2-container must be equal.
(b) A simple path must exist between its start-node and end-node.
(c) The length of the path must not be less than 3 (because an arc is defined as a straight line segment here).
(d) The length of the path must be equal to the total number of nodes.

Thus satisfying 2CC_1 will automatically enforce the four implied constraints.
A violation of 2CC_1, as in 1CC_1, occurs if there is a gap (undershoot) or an overshoot, or sliver polygon. They can be caused by digitizing error or omission of an arc during manual input, or by incorrect geo-referencing of the layers on the same system. The constraint must be checked when inserting or modifying a 2-container, and the system should warn the operator of a violation so he can perform immediate interactive editing. If acquisition is made in multivalued mode, a snapping distance can be defined to automatically ensure that the 2-container is defined by a closed polygon. The consistency rule, called 2CC_Rule_1, for monitoring this constraint is as follows.

Consistency Rule for 2-Container

For each 2-container E
    For each distinct $n_i \in N_E$
        compute $degree_E(n_i)$
        if $\exists\ n_i \in N_E \ni degree_E(n_i) < 2$
            display error message: "gap in 2-container"
            display E for interactive editing
        else
        if $\exists\ n_i \in N_E \ni degree_E(n_i) > 2$
            display error message: "overshoot in 2-container"
            display E for interactive editing
        endif
    next n_i
next E

5.1.3 Consistency Rule for the Planarity of the Map

To ensure that the overall geometry of the vector map is consistent as a planar graph, the Euler constant must be checked. The constants for connected and disconnected planar graphs are given in §3.1.1. Although it can be assumed that if the constraints at the two lower levels are enforced, the Euler constant should also follow. Nonetheless, it is preferable to check the constant at some intervals to ascertain that the overall geometry is consistent. A violation of the Euler constant definitely implies that one of the constraints at the lower levels has been violated.

It is assumed that the preceding constraints have been fulfilled. If this assumption holds, the Euler constant (see §3.1.1) can then be checked for a subset of the map, consisting of the planar graph in which all arcs having $B[a_i, AO_f] = 2$ are isolated (these individual 1-containers can be checked separately as described above). Here, a_i = arc i and AO_f = 2-container f. The procedure for monitoring the constant will then be as follows.

Let the map be represented by the planar graph G(M,A) where M is the set of nodes and A is the set of arcs.

- Subtract $\{n_i \in M | \text{degree}(n_i) = 0\}$ from M /* eliminate isolated nodes */
- Select $\{Temp2\} = \{a_i \in A | B[a_i,AO_f] = 1\}$ /* select arcs that define boundaries of 2-containers */
- Let N2 = number of arcs in {Temp2} /* count number of arcs in Temp2 */
- Select $\{Temp3\} = \{a_i \in A | B[a_i,AO_f] = 2\}$ /* select arcs that define only 1-containers */
- Let N3 = number of arcs in {Temp3} /* count number of arcs in Temp3 */
- Let $\{Temp1\} = \{n_i | (N[a_j,n_i]=1 \wedge N[a_k,n_i]=0,\ a_j \in Temp3,\ a_k \in Temp2,\ k = 1,N3)\}$ /* select the nodes that define only 1-containers */
- Let {M} = {M} - {Temp1} (i.e., subtract {Temp1} from {M})
- Let N4 = number of nodes in {Temp1}
- Let N5 = number of (remaining) nodes in {M}
- Let N6 = number of distinct 2-containers
- Compute E (general Euler constant) as

E = N5-N2+N6-1 (i.e., v-e+f-1)

- Analyze value of E:
  E = 1 ⇒ consistent and connected map
  E > 1: investigate presence of subgraphs. Number of subgraphs should equal E else map is not geometrically consistent.

As an additional check, the total number of arcs must equal N2 + N3.

5.1.4 Consistency Rules for Elementary Object Types

At the next higher level of complexity are the consistency rules for terrain object types. The semantics of the objects are given by their geometric type and their thematic classes (Molenaar, 1991c); thus two sets of consistency rules can be defined for each object. These are:
(1) Rules for the geometric description of individual elementary objects.
(2) Rules related to the thematic component of the object.

Rules for the Geometric Description of Elementary Objects

The geometric definitions of elementary objects in vector maps have been given in §3.4.5 (see also Molenaar, 1991c). The geometry of each object in a vector map must be consistent with these definitions, i.e., the topology of individual objects must be consistent and must be preserved. If the consistency rules at the three lower levels have been fulfilled, the rules for the geometric structure of an object can easily be checked. There are two aspects of these rules. First, the geometric description of the object must be consistent (i.e., properly defined) during data capture before it is accepted into the database. The data input and editing routines of most data acquisition subsystems can handle this. Second, the geometry of the object as provided by the m-containers representing it must be consistent after insertion. This has to be done by specified rules as outlined below.

Point Object

For an individual point object, it should be verified that the 0-container defining the object has been created.

Line Object

For an elementary line object, a simple and elementary path (see §3.1.1) must exist between the beginning and end of the chain of 1-containers defining the object, i.e., the chain of 1-containers representing the object must be fully connected. In other words, let L_{ij} represents 1-container j of line object L_i; for the object's geometry to be consistent after the creation of its 1-containers, the terminal node of $L_{i[j-1]}$ must be the initial node of L_{ij}, the terminal node of L_{ij} must be the initial node of $L_{i[j+1]}$, etc. (Note that if the DMMVM is used for a single layer, each object will be represented by a single equivalent m-container.) This constraint can be monitored by evaluating the topologic relationship between pairs of the 1-containers representing the line object, as illustrated in the following rule LO_Rule_1:

Consistency Rule for Line Object (LO_Rule_1)

Let $\{...,L_{ij},...\}$ be the chain of 1-containers representing line object L_i.
If Relation(L_{ij},$L_{i[j+1]}$) $\neq$ r287 then
display error message
display $\{...,L_{ij},...\}$ for editing.

If the geometry of the object has been correctly measured with accurate georeferencing during acquisition, then the error can be due only to gap(s) or overshoot(s) during creation of the 1-containers (e.g., by overlay computation), which can be caused by incorrect tolerance values for snapping. By displaying the chain, the erroneous segment can be corrected.

Area Object

For an individual area object, the 2-containers representing the object must be connected and non-overlapping. This constraint can be monitored by evaluating the topologic relationship between pairs of the 2-containers as follows:

Consistency Rule for Area Object (AO_Rule_1)

Let $\{...,F_{ij},...\}$ be the set of 2-containers representing area object F_i.
If Relation(F_{ij},$F_{i[j+1]}$) $\notin$ {r279, r285, r287} then
display error message
display $\{...,F_{ij},...\}$ for editing.

As in the case of line object, if the geometry of the object has been correctly measured with proper georeferencing during acquisition, then the error can be caused only during creation of the 2-containers (e.g., by overlay computation), which may be due to incorrect tolerance values for snapping. By displaying the set of 2-containers, the erroneous part can be corrected.

The other components of geometric aspects of an object are the shape and size of the object. These are constraints such as: a house must be rectangular, the two sides of a road must be parallel, etc. These constraints can be related to the thematic classes of the object as suggested by Molenaar (1991c).

Rules for Thematic Component of Objects

The thematic data of individual objects are application-dependent. However, apart from the constraints that will be defined for these data, the constraints described in §3.4.1. must be enforced for each object to conform with the strict classification hierarchy of the DMMVM. These mandatory constraints are:
(a) Cyclic classification hierarchy is not permitted.
(b) The classification must be complete, i.e., all objects must be classified.
(c) The classes must be mutually exclusive, i.e., each object must belong to only one class in each layer, but if one object appears in more than one layer, the object can be classified in the other layer as well.

5.1.5 Consistency Rules for Functional Relationships Among Data Types

The functional relationships among the eight data types of the DMMVM, namely node, arc, 0-container, 1-container, 2-container, point, line and object, are described by the elementary links of §3.4.3 These relationships should be checked for consistency.
In the preceding sections, consistency rules have been defined for individual data types and between arcs and nodes. The functional relationships among different data types can be verified for consistency by using the relational functions described in §4.2.

5.2 Semantic Consistency

In general, semantic constraints are application-dependent and they are both spatial (e.g., two adjacent parcels must share a common border) and non-spatial (e.g., a lessee cannot transfer his right on a leasehold parcel beyond the period of the lease). In the spatial domain, they are mostly a group of forbidden relationships between pairs of objects. Here the focus will be on the geometry-related semantic constraints. These constraints are also important in spatial databases because the database may be geometrically consistent (satisfying the constraints in the preceding sections) but semantically inconsistent. For instance, it is topologically in order for a line object to cross an area object insofar as the geometric constraints are fulfilled. But this relationship may or may not be consistent, depending on the *meaning* of the two objects, i.e., the application for which the data model is being used.

This makes it difficult to provide a generic algorithm for resolving semantic constraints. But a monitoring procedure can be formulated for those that are topologic in nature, especially between pairs of objects. The monitoring strategy being proposed here is to use topologic relationships as alerters. The strategy is for only semantic constraints that are geometric in nature, and it is based on the assumption that the database is structurally consistent. Some of these constraints are for individual objects, e.g., size and shape constraint. These can be monitored and enforced by the editing routines of the data acquisition system or with functions defined as part of the topologic editor.

A scheme for monitoring the constraints between object pairs is as follows (see Figure 5.3).

- Translate the constraint to a set of expected topologic relationships ({E}) which will maintain the consistency of the map. In practice, this can be done for all semantic constraints (topologic) in the application and stored in the database.
- On-line derivation of the actual relationship (A) by the system during insertion using the topologic relationship operator (implementation of the scheme in §4.3)
- System checks for membership of A in the set E
- If $A \notin \{E\}$ then violation has occurred and the system should warn the user. The user can then interactively resolve the violation either by using pre-defined rules (if applicable) or by taking any other action.

This strategy will be illustrated with some examples of semantic constraints in cadastral application.

Examples of Constraints in Cadastral Mapping

Constraint #1: A river may not flow into or through a cadastral parcel.
Assuming that a river, represented as line object L, and a parcel, represented as area object A, exists as depicted in Figure 5.1

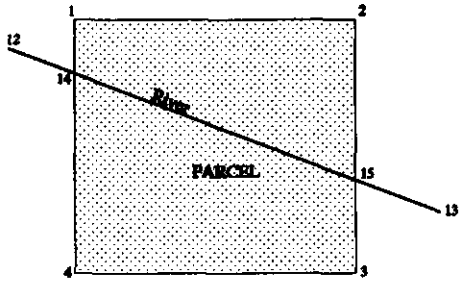


Figure 5.1 A river crossing a cadastral parcel

Translating the constraint to the expected topologic relationships {E} gives {E} = {r031, r063, r285, r287, r316, r317, r319} (see Figure 4.8). The actual relationship A (to be derived by the procedure Relation(A, L)) = r191. Testing A for membership of {E} indicates that A ∉ {E} thus a violation has occurred. The system must then warn the user about this inconsistency or execute the relevant rule to enforce consistency if such a rule has been defined. Possible causes of this may be incorrect geo-referencing or incorrect digitizing (by field survey or any other means) of either of the two objects. Assuming that the geo-referencing is accurate and the digitizing is within tolerance, then the parcel must be partitioned into two and re-allotment carried out if necessary.

A similar constraint to this is that a (residential) building (represented as an area object) must not be crossed by a (underground) gas pipeline (represented as a line object). If the actual relationship (A) is not an element of the above set E, then violation has occurred and the building plan must be disapproved (assuming the digitizing of the two is accurate and geo-referencing is accurate).

Constraint #2: A parcel must have access to a road.

This is an important constraint in modern cadastral layout, a topologic constraint concerning the connectivity of a land parcel and a road. Let a parcel be represented as an instance A of area object type and a road an instance L of line object type. The constraint can be monitored for each parcel being inserted by the system with the following algorithm:

If Relation(A,L) ∉ {r063, r285, r287, r316, r317, r319} then display error message: "parcel A does not abut a road".

Here, Relation(A,L) will return the actual relationship (A in Figure 5.3) while the relationships inside braces are the expected relationships (E) (see Figure 4.8).

If an inconsistency is detected and it is certain that the survey was done accurately, then a buffer can be defined around the parcel and the parcel together with the objects inside the buffer displayed to assist the human operator in interactive editing.

Constraint #3: Two parcels must not overlap.

Given the situation in Figure 5.2 between new parcel O1 and an existing parcel O2. The expected relationships that will not violate the constraint, i.e., {E} = {r031, r279, r285, r287} (see Figure 4.7). Note that although r279 and r285 do not actually violate this particular constraint, occurrence of any of them will not be permitted because of constraint #2 above. Thus {E} reduces to {r031, r287}. From Figure 5.2, the actual relationship A = r511. Since A ∉ {E}, the constraint is violated and the system should warn the human operator.

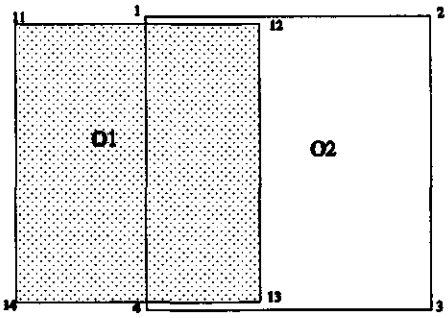


Figure 5.2 Two overlapping parcels

The possible actions to be taken by the operator include an adjudication process if the overlap is caused by what is commonly referred to as "land-in-dispute" in cadastral surveying, re-measurement of one or both of the parcels, splitting the two parcels into three, or reduction of the size(s) of one or both of them. This example clearly indicates that human intervention will still be required to resolve some inconsistencies.

Constraint #4: A building must be contained inside a parcel.

This constraint is normally checked before a building plan is approved. Assuming that buildings and parcels are represented as area objects, the expected relationships are {E} = {r179, r220, r400, r435, r476} (see Figure 4.7). The actual relationship (A) between a given building and the parcel on which it is to be located (assumed to exist in the database) should be derived by the procedure Relation(O1, O2). If A ∉ {E} then violation has occurred and the building plan will not be approved pending the resolution of the conflict.

5.3 Implementation Approach

Figure 5.3 shows the flow-chart of the scheme for spatial consistency management in a vector GIS. At the core of the GIS is a spatial database which is assumed to be structured according to the DMMVM. The topologic relationship operators will consist of the automated procedure to dynamically derive the existing topologic relationships between objects and between geometric primitives using the algorithms in §4.3 and computational geometry where necessary. The topologic editor will comprise the automated procedures to carry out topologic editing of the database in a consistent manner as described in §5.1.1. User-defined operations will consist of the operations that must be performed when there is a violation of semantic constraints. The operations may include the use of the topologic editor functions.

The set of expected topologic relationships {E} between geometric primitives that will not result in constraint violation may be stored as a kind of look-up table (LUT). This should also be done for the semantic constraints (geometric) between objects. To monitor topologic consistency, e.g., between two objects, the system will then evaluate the actual relationship

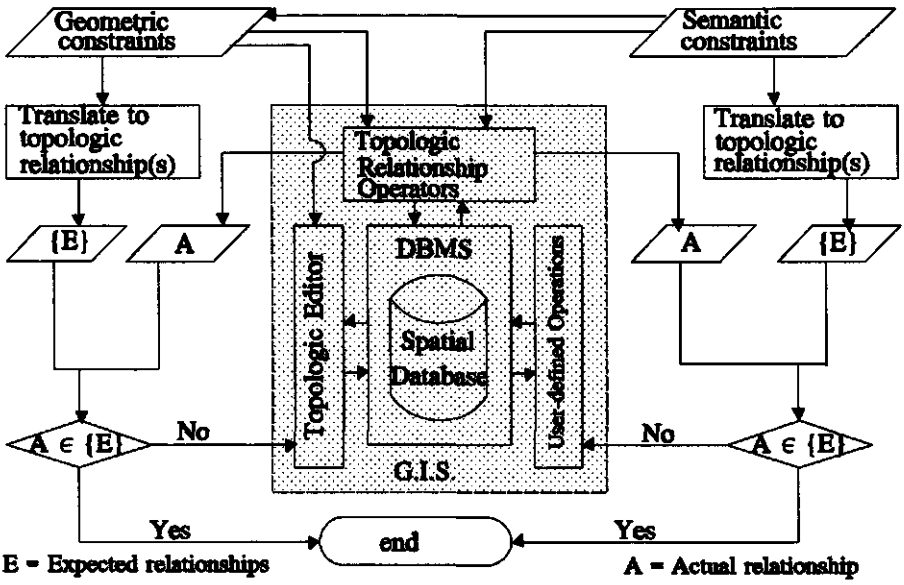


Figure 5.3 Scheme for consistency operations in vector maps

(A) between the two objects, check for its occurrence in the set {E} for that constraint, give a message if violation occurs, resolve the violation if the operation is provided or call for the decision of the human operator.

5.4 Summary

A large proportion of the cost of setting up a database for spatial information production is attributed to data acquisition. To offset the cost and derive profit, the information produced by that system must be reliable, i.e., the quality of the data from which the information is derived must be trustworthy. This has made the issue of data quality an important aspect in GIS. Data consistency is a component of data quality because consistency is essential for the database's reliability. This chapter has focused on this issue.

Consistency rules have been formulated to ensure structural constraints, while a monitoring strategy was proposed for semantic constraints. In both cases, topologic relationships play the central role as alerters of constraint violations. The next chapter will focus on the handling of object dynamics in the database in a manner that does not disturb its structural consistency.

6

OBJECT DYNAMICS AND UPDATING IN VECTOR MAPS

In geo-information production, currency of the data plays a very important role, together with data quality (accuracy, completeness, consistency), in the reliability of the information. In the preceding chapter, procedures were defined for monitoring and enforcing one aspect of data quality: data consistency. This chapter focuses on the maintenance of data currency, i.e., updating. Database updating is an important aspect of GIS development because the terrain objects represented in the database are generally not static in time; thus the database should also respond to such object dynamics through "consistent" updating. By consistent, we mean that the structural and semantic constraints of the database (see chapter 5) must be enforced after each update.

In the mapping disciplines, in reaction to the evolution of mapping methods and processes (from conventional analogue mapping through computer-assisted mapping to digital mapping and then to GIS), updating of geo-information has evolved from simple graphic map revision using analogue methods and equipment through digital map revision with the aid of computer hardware and software but using a spaghetti model and with intensive involvement of the human operator. The goal is to be able to update a structured database in a GIS with a high degree of automation, but this last stage is still very much confined to the research and development domain; this thesis aims at contributing towards achieving that goal.

In principle, it is simple to update a database: remove the outdated data and replace them with the new data. This used to be largely true when updating was carried out on the (plastic) master copy of the hardcopy map. The major problem then was on how to detect changes and what policy to follow for the updating, i.e., whether cyclic, selective or continuous; the actual change involved peeling off the old data on the master copy and scribing in the new information. Still, it is an expensive and complex operation which leaves many maps un-updated, especially in developing countries.

In this digital era when many processes are being automated to take advantage of technologic developments, and where the interrelationships among terrain objects are also modelled to make the database more useful, updating has become a more complex operation and researchers are focusing attention on the complete automation of updating procedures starting with change detection through the extraction of changed data, to database updating including editing and quality assessment. The general procedure for the updating of geo-information is represented in Figure 6.1.

For change detection, superimposition techniques and image processing and analysis are powerful tools. Information about changes in the terrain can also be obtained from local knowledge, i.e., from people and agencies involved in building and construction. The changes can be extracted manually or automatically by means of photogrammetry (mono or stereo), image processing, head-up (screen) digitizing, field survey, etc. These aspects (change detection and data collection) are not addressed in this thesis. Rather, the focus is on the database updating aspect, addressing the issues of update propagation, topologic editing and consistency enforcement in vector-structured spatial databases. In this context, database

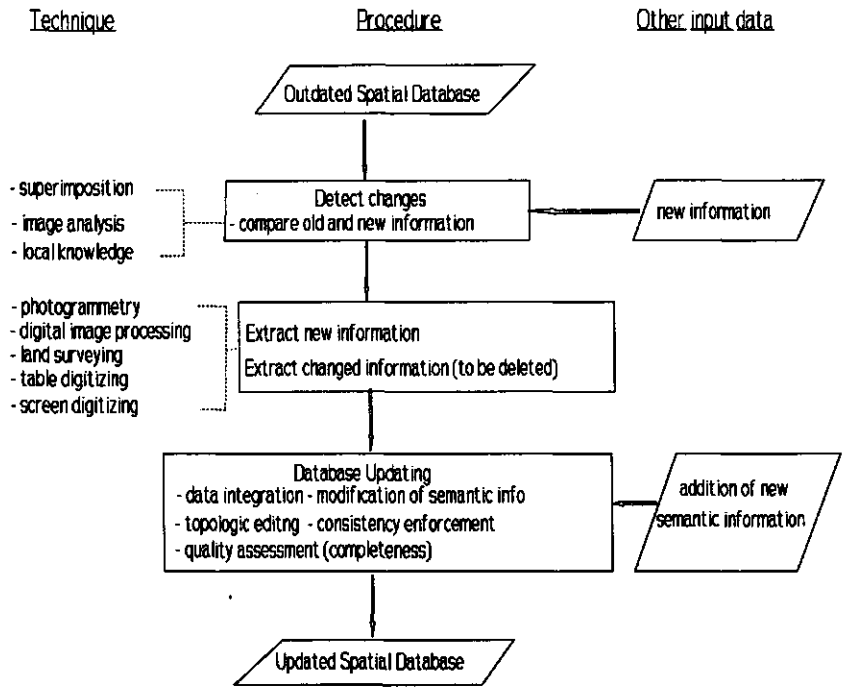


Figure 6.1 General procedure for geo-information updating

updating denotes an operation that leads to (a) the insertion of new data into the database, (b) the modification of some existing data, and (c) the removal (deletion) of obsolete data from the database. Because the DMMVM is a topologic model, the representation of terrain objects includes their interrelationships; it is thus possible that an update on one object will effect another object(s), causing an inconsistent state. To forestall such inconsistency, an update "propagation" must be effected when any of the three updating operations is carried out. Intuitively, update propagation means the system should identify all objects that are affected by a single update and modify them (or give warning to the human operator) to ensure consistency.

In the DMMVM, three generic types of spatial object are represented: point, line and area objects. Changes in the database will therefore result from the dynamics of these objects. In Figure 2.2, an object (O) is uniquely defined by its geometry (G) and the set of its thematic attributes (T), i.e., O = f(G, T). The dynamics of an object can thus be grouped into two basic aspects: thematic changes and geometric changes. In addition there can also be a change in

the aggregation structure of objects (see Molenaar, 1991). These changes can occur singly or jointly.

The aggregation aspect of terrain objects indicates how a complex terrain object can be composed from smaller objects. Since the aggregation structure of objects is not explicitly represented in the DMMVM (they can be derived through queries), this aspect will not be considered as a basic updating operation in the database.

The updating of the other two aspects are analyzed in the following sections with emphasis on the geometric aspects. The procedures for propagating the geometric updates are also presented in pseudo codes.

6.1 Updating Thematic Data of an Object

Here, the thematic attributes of an object are restricted to its thematic class label in order to focus on the more complex geometric updating, but the other attributes can be handled as well. Thus only the class label will require updating, as in the reclassification of an object (e.g., an area object "parcel" changing from class "vacant" to class "built-up"). It is assumed that the DBMS of the implementation platform will have facilities (e.g., insert, update, and delete in RDBMS) for simple updating in which propagation is not required. When a new object comes into existence, simple insertion of its class label (and other thematic attributes if any) can easily be done by the DBMS tools. The same holds when an object comes to the end of its life-span, in which case its thematic information is simply deleted.

However, if hierarchic classification is implemented, then certain rules must be observed when updating the thematic classes. For class creation, the superclass has to be created first because its attribute structure is expected to be inherited by its subclasses. The consistency rules defined for thematic data (see §5.1.4) should also be monitored and enforced. The insertion of the thematic attribute values of the individual objects belonging to the class can then be effected by the DBMS commands. When it is necessary to remove a class, an essential rule is that a superclass cannot be deleted unless all its subclasses have been deleted. And in general, no class should be deleted unless all its instances have been deleted. Hereafter, the analyses will focus on the updating of geometric aspects of objects.

6.2 Updating of Geometric Components

Changes in the geometric aspects of an object might include a change of position, or size or shape, or a combination. These changes may also lead to changes in the topologic relationships among the objects in the database. Thus care has to be taken to ensure integrity of the database during geometric updating of objects. The main focus of the updating procedure is therefore on this aspect. In this research, it is assumed that the boundaries of objects are well-defined (crisp dataset). The objects are also assumed to be correctly geo-referenced, using the same coordinate reference system and the same resolution.

In the DMMVM, the geometric characteristics of objects are defined by the data types 2-container, 1-container and 0-container, which are represented by the two geometric primitives: arcs and nodes. Thus geometric changes directly imply changes in the five data types. The

updating can be analyzed at three levels. The lowest level concerns the geometric primitives whose updating is triggered by an updating request from higher level data types by propagation. The next higher level concerns the m-containers, $m \in \{0,1,2\}$, whose updating will also be triggered by propagation from the updating at the highest (i.e., object) level. At the highest (and user) level are the individual terrain objects of types point, line and area, which actually trigger the database updating.

The update propagation path is depicted by Figure 6.2, indicating that the updating must always be initiated by the need to update (insert, modify, delete) terrain objects. The expectation in automated database updating is therefore that the system receives an updating request at the highest (i.e., object) level, together with the necessary input data, and performs the propagation from the object level to the level of geometric primitives. In addition, for the database to be geometrically consistent, all the constraints defined in chapter 5 must be satisfied after each updating. Thus (automated) procedures must be provided to propagate the update while checking and enforcing the constraints. In the following sections, the update propagation involving the basic data types (area, line, point, 2-container, 1-container, 0-container, arc and node) are analyzed at their respective levels, starting from the lowest level (node and arc), with each level serving as "building block" for the next higher level.

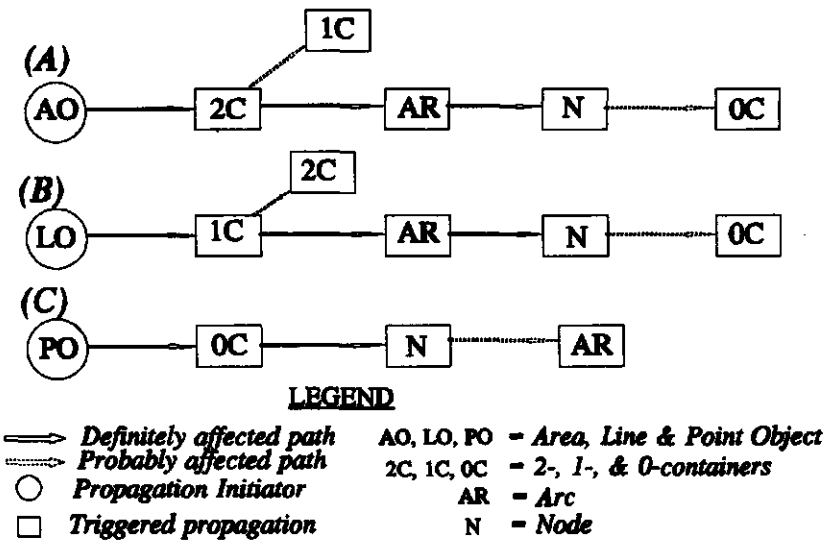


Figure 6.2 Update Propagation Path in the DMMVM; (A) = Area Object's Path, (B) = Line Object's Path, (C) = Point Object's Path

6.2.1 Notations

We will recall the functions presented in chapter 4 (see §4.2.1 and §4.2.2) for representing the functional relationships between data types, and add other ones to completely represent

the functional relationships among the data types in the DMMVM (see Figure 3.8) by similar notations. The functional relationships indicate the navigation route among the data types; thus the functional notations described here will be used in the pseudo codes for the update propagation procedure described in this chapter.

Link between thematic class and terrain object

The link **belongs-to** between a terrain object O_i and the thematic class TC_j can be represented by:

$$Belongs\text{-}to[TC_j,O_i] \in \{0,1\}$$

When the function has the value 1, the object belongs to that class, and when it has value 0, the object does not belong to that class.

Link between elementary terrain object and m-container

The links between an elementary object O_i ($O_i \in$ {area, line, point}) and m-container ($m \in \{2,1,0\}$) can be represented as follows.

The fact that a 2-container$_j$ is **part-of** an area object AO_i can be represented by

$$Partof[AO_i,2\text{-}container_j] = 1$$

And if the 2-container is not part of the area object, then

$$Partof[AO_i,2\text{-}container_j] = 0$$

If a 1-container$_j$ is **part-of** a line object LO_i, it will be represented by

$$Partof[LO_i,1\text{-}container_j] = 1$$

And if the 1-container is not part of the line object, then

$$Partof[LO_i,1\text{-}container_j] = 0$$

Also, the link **represented by** between a point object PO_i and a 0-container$_j$ can be indicated by the function

$$Repreby[PO_i,0\text{-}container_j] = 1$$

If the 0-container does not represent the point object, then

$$Repreby[PO_i,0\text{-}container_j] = 0$$

Link between m-container and geometric primitives (arc and node)

In the DMMVM, planar enforcement must be satisfied. Thus an arc must always have one 2-container on its left side and one 2-container on its right side. The relationship between an arc a_j and a 2-container$_i$ can then be depicted as follows.

The existence of the relationship **left** between arc a_j and 2-container$_i$ can be represented as

$$Left[2\text{-}container_i,a_j] = 1;$$

if the 2-container is not on the left of the arc, then

$$Left[2\text{-}container_i,a_j] = 0.$$

And the fact that arc a_j has 2-container$_i$ on its right side can be represented by

$$Right[2\text{-}container_i,a_j] = 1;$$

if the 2-container is not on the right of the arc, then

$$Right[2\text{-}container_i,a_j] = 0.$$

A boundary arc a_j of 2-container$_i$ can then be found from the function
Boundary[2-container$_i$,a_j] = Left[2-container$_i$,a_j]+Right[2-container$_i$,a_j]
If this function = 1, the arc is part of the boundary of the 2-container, and if = 0 or 2 it is not.

The fact that arc a_j is part of 1-container$_i$ can be represented by
Partof[1-container$_i$,a_j] = 1;
if the arc is not part of the 1-container, then
Partof[1-container$_i$,a_j] = 0.

The fact that node n_j represents 0-container$_i$ can be represented by
Repr[0-container$_i$,n_j] = 1
If the node does not represent the 0-container, then
Repr[0-container$_i$,n_j] = 0.

Link between arcs and nodes

See §4.2.1.

Other Links

The link **crosses** between 1-container$_i$ and 1-container$_j$ can be represented by
Crosses[1-container$_i$,1-container$_j$] = 1 when j crosses i, and
Crosses[1-container$_i$,1-container$_j$] = 0 if they do not cross.

Also, the fact that 2-container$_i$ contains 0-container$_j$ can be depicted by
Contains[2-container$_i$,0-container$_j$] = 1,
and if it does not contain it, then
Contains[2-container$_i$,0-container$_j$] = 0.

From the basic functions above, other transitive links can also be derived, e.g., whether or not 1-container$_k$ is part of the border of 2-container$_i$ can be found through
*Border[2-container$_i$,1-container$_k$] = (Left[2-container$_i$,a_j]+Right[2-container$_i$,a_j])*Partof[1-container$_k$,a_j]*
If the value of the function = 1, then the 1-container lies on the border of the 2-container; if the value is 0 or 2 it does not.

Whether or not arc a_j is part of area object AO_i can be established from
Partof[AO_i,a_j] = Partof[AO_i,2-container$_k$](Left[2-container$_k$,a_j]+Right[2-container$_k$,a_j])*
If the value of the function = 1, the arc is part of the area object's geometry; if = 0 or 2, the arc is not part of the geometry of the area object.

6.2.2 Updating of Geometric Primitives

At the lowest level of the update propagation path are the two geometric primitives. Their updating will normally be an indirect operation triggered by geometric updating at the next

higher level. The consistency rules defined for the geometric primitives in chapter 5 (see Table 5.1) are intended to propagate updating of the primitives while maintaining geometric consistency. The updating operations (resulting from insertion, deletion or modification) involving the two primitives are analyzed below. They will serve as the elementary operations into which the updating of the higher level data types can be decomposed.

Inserting a new node

When inserting a new node in the database, the topologic relationships between the new node and the existing arcs and nodes should be checked for inconsistent relationships (r272 and r092 in Table 5.1); if they occur, the necessary consistency operation is performed by the system; otherwise, the node (and coordinate information) can be inserted. The algorithm for inserting a node is as follows. The block diagram of the algorithm is given in Appendix 1.1.1.

Algorithm Insert_Node:

```
begin
    get nodeid (n1) and its properties (x, y and z coord and accuracy data)
    do while exists n_j ∈ N!Degree(n_j)=0 /* N = existing nodes */
        determine Relation(n1,n_j) /* by comparing coordinates */
        if Relation(n1,n_j) = r272
            do GP_Rule_1
            goto end
        endif
    end do while
    do while exists a_i ∈ A /* A = existing arcs */
        determine Relation(a_i,n1) /* using coordinate geometry */
        if Relation(a_i,n1) = r092
            do GP_Rule_2
            goto end
        else if Relation(a_i,n1) = r284 /* by comparing coord. see Figure 4.11b
        */
            do GP_Rule_1
            goto end
        endif
    end do while
    insert n1 (and its properties)
end
```

Deleting an existing node

Before a propagator causes the deletion of a node during update propagation, the system should check that the node does not define other arc(s) and/or a 0-container.

The following is the procedure Delete_Node for deleting a node n_i, in pseudo code (see Appendix 1.1.2 for the block diagram):

```
begin
    get identifier of n_i from propagator
```

```
        check for each 0-container 0c_j
                if ∃ 0c_j ∋ Repr[0c_j,n_i] = 1
                        goto end /* i.e., do not delete /*
                else next 0c_j
                endif
        check for each arc a_j
                if ∃ a_j ∋ (Beg[a_j,n_i] = 1 ∨ End[[a_j,n_i] = 1)
                        goto end /* i.e., do not delete /*
                else delete node (and its attributes)
                endif
end
```

Modifying an existing node

The modification request for a node may arise when the value(s) of one (or more) of its coordinates is (are) to be changed, e.g., because of more accurate measurement. This modification request is not as simple as it appears because the node's existing topologic relationships with some arcs (if any) may have resulted from the previous coordinate values. It may also be defining a 0-container which represents more than one point object. It is therefore necessary to ascertain that this node does not define the position of other arcs or point objects (same 0-container) before the modification. The most appropriate solution is to insert the affected node as new (with its new coordinate values) using Insert_Node procedure and delete the old using the Delete_Node procedure. However, if only the accuracy property has changed, this can be effected by simple updating facilities of the DBMS. The algorithm for effecting the modification will therefore be as follows (see Appendix 1.1.3 for the block diagram):

Algorithm Modify_Node

```
begin
        get nodeid and modified properties
        select the node and current properties /* as a kind of view */
        modify requested values and assign new id
        do Insert_Node /* with its modified values and unchanged properties */
        do Delete_Node /* for obsolete node */
end
```

Inserting a new arc

To maintain the integrity of the database when a new arc is being inserted, the geometric primitive constraints GPC_i, i ∈ {1,2,3,4} should be enforced through the consistency rules defined in Table 5.1., i.e., when a new arc is inserted, the system must evaluate the arc's topologic relationship with each of the existing primitives in the database and apply the corresponding consistency rule for any relationship that violates geometric consistency.
The following is the algorithm Insert_Arc for inserting a new arc. The block diagram of the algorithm is given in Appendix 1.1.4.

Algorithm Insert_Arc:

```
begin
    get new arc a_i (and its properties: id, coords, left, right, 1-container)
    assign node numbers for its start and end nodes
    do while exists n_j ∈ NlDegree(n_j)=0 /* N = existing nodes */
        determine Relation(a_i,n_j) /* by coordinate geometry */
        if Relation(a_i,n_j) = r092
            do GP_Rule_2
        endif
        if Relation(a_i,n_j) = r284 /* by comparing coord. see Figure 4.11b */
            do GP_Rule_1
    end do while
    do while exists a_j ∈ A /* A = existing arcs */
        determine Relation(a_i,a_j) /* using coordinate geometry */
        Case r063 do GP_Rule_3
        Case r095 do GP_Rule_4
        Case r159 do GP_Rule_5
        Case r179 do GP_Rule_6
        Case r220 do GP_Rule_7
        Case r255 do GP_Rule_8
        Case r287 do GP_Rule_9
        Case r400 do GP_Rule_10
        Case r435 do GP_Rule_11
        Case r476 do GP_Rule_12
        goto end
    end do while
    determine new values for left and right relationship
    store a_i
    insert start and end nodes of a_i using Insert_Node
end
```

Deleting an existing arc

When a delete request is propagated to an arc, the system should first verify that the arc is not linked to other primitives. An existing arc will have one of the following combinations of properties at one time:

(1) Left[2-container$_i$,a$_j$] = 1 $\wedge$ Right[2-container$_i$,a$_j$] = 1 $\wedge$ Partof[1-container$_k$,a$_j$] = 1 ($\wedge$ = logical and), i.e., the arc has the same 2-container on each side and represents part of a 1-container. In this case, the arc defines only (part of) a 1-container and can be deleted only if the 1-container is being deleted or modified.

(2) Left[2-container$_i$,a$_j$] = 1 $\wedge$ Right[2-container$_i$,a$_j$] = 0 (or vice versa) $\wedge$ Partof[1-container$_k$,a$_j$] = 0, which means that the arc demarcates two 2-containers and can be deleted only if a preceding updating operation would have made left = right.

(3) Left[2-container$_i$,a$_j$] = 1 $\wedge$ Right[2-container$_i$,a$_j$] = 0 (or vice versa) $\wedge$ Partof[1-con-

tainer$_k$,a_j] = 1, in which case the arc represents a 1-container and demarcates two 2-containers. It can be deleted only if the updating operation requires that the 1-container which the arc represents be deleted, and left and right are first updated to have equal values.

(4) Left[2-container$_i$,a_j] = 1 $\wedge$ Right[2-container$_i$,a_j] = 1 $\wedge$ Partof[1-container$_k$,a_j] = 0. This is a 'dangling' arc which should be deleted from the database.

Thus if an existing arc was deleted, the preceding operations in the update propagation chain that triggers it would have led to the fourth situation; otherwise the arc would have been only modified. The following algorithm Delete_Arc will do the necessary checking and delete the arc if it is now redundant (see Appendix 1.1.5 for the block diagram):

Algorithm Delete_Arc:

```
begin
        get arc id → a_j
        if Left[2-container_i,a_j] = 1 ∧ Right[2-container_i,a_j] = 1 ∧ Partof[1-container_k,a_j]
        = 1  goto end /* do not delete */
        if Left[2-container_i,a_j] = 1 ∧ Right[2-container_n,a_j] = 1 (where i ≠ n) ∧
        Partof[1-container_k,a_j] = 0 goto end
        if Left[2-container_i,a_j] = 1 ∧ Right[2-container_n,a_j] = 1 (where i ≠ n) ∧
        Partof[1-container_k,a_j] = 1 goto end
        if Left[2-container_i,a_j] = 1 ∧ Right[2-container_n,a_j] = 1 (where i = n) ∧
        Partof[1-container_k,a_j] = 0
                then
                select n1| Beg[a_j,n1] = 1 and n2| End[a_j,n2] = 1 /* beg & end nodes */
                delete a_j  /* simple DBMS command to remove record of a_j */
                do Delete_Node(n1)
                do Delete_Node(n2)
        endif
end
```

Modifying an existing arc

During update propagation, the modification of an existing arc may be necessitated by any of the following situations:
- When the value(s) of the Left and/or Right function(s) of the arc change(s), which may be caused by the creation or modification of a 2-container.
- If there is a change in the value of the relationship Partof between the arc and a 1-container, which may be caused by an insertion of a new line object (thus modification or creation of a 1-container) or deletion of a 1-container.
- When there is a change in the coordinates of one or both of its two nodes. This may arise from more accurate measurement of the position of the node(s) and, because this may affect the existing topology, the affected arc should be inserted as a new arc using the Insert_Arc procedure, thereby recreating the topology (this can be optimised by forcing the system to use the primitives associated with the former arc and those in that vicinity) and the old arc deleted using Delete_Arc procedure.
- When a new node or a new arc is inserted and necessitates the execution of any of the

consistency rules GP_Rule_i, $2 \leq i \leq 12$ (see Table 5.1). The necessary modification will be performed within the propagation chain that triggers the insertion of the new primitive. The algorithm for modifying an existing arc is given below; the block diagram is shown in Appendix 1.1.6.

Algorithm Modify_Arc

```
begin
        get arcid and modified properties
        if ∃ locational change
                select the arc and current properties /* as a kind of view */
                modify requested values and assign new id to arc and its nodes
                do Insert_Arc /* with its modified values and unchanged properties */
                do Delete_Arc /* for obsolete arc */
        else
                modify affected property /* simple DBMS command */
        endif
end
```

6.2.3 Updating of the m-Containers

At the next level of the update propagation path are the three m-containers: 2-container, 1-container and 0-container. Their updating will normally be an indirect operation triggered by updating at the highest level, i.e., of individual terrain objects. Like the geometric primitives, the updating of the m-containers should be automated as much as possible while enforcing the consistency rules defined for them (see chapter 5). The updating operations (resulting from insertion, deletion or modification of a terrain object) involving the three types of m-container are analyzed below.

Inserting a new 2-Container

To satisfy completeness of incidence in the DMMVM as a planar vector map, all 2D segments (faces) must be classified, i.e., every closed polygon must be (part of) a 2-container; thus the addition of a new 2-container implies modification or deletion of an existing one. This means that insertion of a 2-container involves a combination of insert, modify and/or delete operations. It is thus a complex operation which may still require the use of semantic information as well as human intervention. For example, in a single-valued vector map, the new 2-container will be spatially coincident with one or more existing 2-containers. If after analyzing the semantic information of those 2-containers (through the Partof[area-object, 2-container] function) it is found that they are part of area objects classified as "vacant" (or "unclassified"), then the update propagation can proceed; but if any of those 2-containers is part of a "real" terrain object then the human operator's decision is required as to whether that existing object should be deleted or modified. Also, in the multi-valued situation, it must be determined first if the overlapping 2-containers are class-compatible, e.g., a cadastral parcel cannot overlap a lake. Class-incompatibility can be predefined in a look-up table (LUT), for example, such that the system consults the table when a new 2-container is being inserted and, in general, during overlay computation to ascertain if the overlapping 2-containers

represent compatible objects. The same situation holds for 1-containers and 0-containers.

Detecting and forestalling such incompatibilities can be handled in two ways: (1) at the beginning of the update propagation (or overlay computation to derive a multi-valued vector map), or (2) immediately after the update/overlay computation. The first approach implies that the checking routine should be part of the algorithm being used for the computation, but it will require more interaction with the human operator (except if an LUT is provided) during the updating. The second approach allows the use of any available overlay computation algorithm, after which the consistency module can check for compatibility. The algorithm provided here uses the first approach, but this part can be transferred to the end during implementation if the user so decides.

If the insertion is allowed, the existing 0-containers topologically contained by the 2-containers should be identified (computationally) in order to make this relationship explicit. The system should continue the update propagation by inserting one arc of the 2-container at a time using the Insert_Arc algorithm (defined in the previous section). Some existing arcs may need to be deleted, while some may require modification. For example, Figure 6.3a shows a small vector map which has been structured according to the DMMVM; 2-container A5 is to be inserted, as shown in Figure 6.3b.

From the figures, it is obvious that the new 2-container's position has affected the geometry of 2-containers A1, A2, A3, and A4, and also the topology of the 1-container representing the railroad. When the algorithm for inserting an area object is used, the incorporated algorithm for inserting an arc will create extra nodes (11, 12, 13, 14, 15 and 16) and split arcs (9,5) into (9,16) and (16,5) and (5,4) into (5,13) and (13,4), respectively; the algorithm for updating an arc will update arcs (16,5) and (5,13), while the algorithm for deleting an arc will delete arcs (12,5), (11,5), (15,5) and (14,5) (arcs having Left[A5,a_i]=1 and Right[A5,a_i]=1 and Partof[1-container$_{\text{rail-line}}$,a_i]=0 where a_i is an existing arc). It is therefore obvious that having provided the basic updating operations for arcs, any complex situation can be decomposed into a set of elementary updating operations. The generalised algorithm for inserting a new container is defined below (see Appendix 1.1.7 for the block diagram):

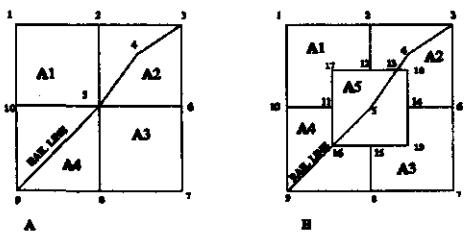


Figure 6.3 Insertion of a new 2-container. A = original situation; B = 2-container A5 is inserted

Algorithm for inserting a 2-container: Insert_2-container

```
begin
        do for each new 2-container c2
        get G_c2(N_c2,A_c2)
        do 2CC_Rule_1 /* consistency rule for 2-container */
        select all existing 2-containers OC2 for which oc2 ∈ OC2| Relation(c2, oc2)
```

```
        ∈ {r179, r220, r400, r435, r476, r511} /* using computational geometry e.g.,
        any polygon intersection algorithm */
          for each oc2
                select area object AO_j for which Partof[AO_j,oc2]=1
                if map is single-valued
                        determine thematic class of AO_j
                        if class ∉ {"vacant", "unclassified"}
                        notify user /* decision ∈ {interactive editing?, next oc2} */
                        else next oc2
                        endif
                else
                  if map is multi-valued
                        determine compatibility between AO_j and new area object
                        represented by c2 /* LUT or user decision */
                        if class incompatible
                        notify user /* decision ∈ {interactive editing?, next oc2} */
                        else next oc2
                        endif
                  endif
        display all 2-containers EC2 ∋ ∀ ec2 ∈ EC2 is Relation(ec2,c2) ∈ {r179,
        r220, r400, r435, r476, r511} /* see Figure 4.7 */
        perform interactive updating /modify neighbouring 2-containers which are
                affected by the new one and insert arcs and nodes of c2; using update
                algorithms of lower level data types */
        for each ec2 ∈ EC2 where Contains[ec2,c0]=1 /* c0 = an existing 0-container
        */
                determine the new 2-container nc2 ∋ Contains[nc2,c0]=1
                if nc2 exists, store relation as property of nc2 and property of c0
        next ec2
next c2
end
```

Deleting a 2-container

Deleting a 2-container may involve a combination of delete, modify and insert operations. The delete request may be triggered because an area object represented by the 2-container is to be deleted or its location is to be shared among neighbouring 2-containers. In both cases, modification of the geometry of neighbouring 2-containers will be required by inserting extra arcs and deleting some obsolete arcs. Moreover, the 0-containers inside the affected 2-container may have a new topology: now inside or on the boundary of a new 2-container. To handle the operations leading to a consistent database, the procedure for deleting the 2-container should comprise the following main steps (see the block diagram in Appendix 1.1.8).

Algorithm Delete_2-container:

```
begin
        get id of the 2-container → c2
```

select the 0-container(s) c0 for which Contains[c2,c0] = 1
select all existing 2-containers O_i for which $o_k \in O_i$| Relation(c2, o_k) ∈
{r179, r220, r279, r285, r287, r400, r435, r476, r511} /* figure 4.7 refers */
modify $o_k \in O_i$ e.g., by inserting new arcs to modify its geometry /* has to be done interactively but using combinations of Insert_Arc, Delete_Arc and Modify_Arc */
delete any arc a_j for which Left[2-container$_i$,a_j]=1 ∧ Right[2-container$_m$,a_j]=1 ∧ Partof[1-container$_k$,a_j]=0 where i = m
determine 2-container nc2 for which Contains[nc2,c0]=1 /* using computational geometry e.g., point-in-polygon algorithm */
if True, store as property of nc2 and property of c0

end

For example, 2-container A5 is to be deleted from Figure 6.3b and its former space apportioned among adjoining 2-containers as shown in Figure 6.3a. This operation involves
- geometric update of 2-containers A1, A2, A3, A4 (by inserting arcs (11,5), (12,5), (14,5), and (15,5) or new arcs as defined by user's sharing criterion, which is simply by defining the position of the common node)
- update of arcs (17,12), (12,13), (13,18), (18,14), (14,19), (19,15), (15,16), (16,11), (11,17), (16,5) and (5,13)
- after the immediate step above, the system will delete arcs (17,12), (12,13), (13,18), (18,14), (14,19), (19,15), (15,16), (16,11) and (11,17) using the algorithm for deleting an arc since, in each case, the values of the left and right relationships are equal and the arc is not part of any 1-container.

Modifying an existing 2-container

Modification of an existing 2-container can be triggered in three ways: (1) when a new 0-container falls in the interior of the 2-container, thereby necessitating the modification of the value(s) of the relationship Contains[2-container,0-container]; this modification, being a secondary update, will be triggered by the Insert_0-container algorithm; (2) as a secondary modification triggered by an insertion of a new area object, or deletion or (geometric) modification of another existing one causing the modification of *this* 2-container; this modification will therefore be triggered by the primary update; or (3) as a primary update triggered by the need to modify the geometry of one of the area objects which the 2-container is part of (e.g., caused by the expansion of the size of an area object "parcel" from 36 m by 18 m to 36 m by 36 m), or an additional area object (as a result of overlay) is to be represented by the 2-container, or the area object(s) which the 2-container is part of is (are) to be deleted.

In the third case, if the modification involves changing only the value(s) of the relationship Part-of (between area object and the 2-container), this can be effected by simple "add", "remove" or modify commands of the DBMS. If geometry is involved and if the change is substantial, the strategy is to use the algorithm for inserting a 2-container to insert the modified version and use the algorithm for deleting a 2-container to delete the obsolete one. If only a few arcs are involved, the affected arcs are isolated from the (retrieved) arc list of the 2-container, the new arcs (to replace the affected ones) are added and the new geometry of

the 2-container is inserted using the algorithm for inserting a 2-container while the isolated arcs are deleted using the algorithm for deleting an arc. This can be summarised in the following algorithm for (primary) modification of a 2-container (see Appendix 1.1.9 for the block diagram).

Algorithm Modify_2-container:

```
begin
    get 2-container_id → c2
    if much geometric change /* subjective decision of user! */
    then
        insert modified 2-container with Insert_2-container
        delete obsolete 2-container with Delete_2-container
    else
        retrieve G_c2(N_c2,A_c2)
        isolate affected arcs (A_k) from A_c2
        add new arcs (A_n) ∋ A_c2 = (A_c2 - A_k) + A_n
        (re)insert G_c2(N_c2,A_c2) using Insert_2-container
        delete a_i ∈ A_k using Delete_Arc
    endif
end
```

Inserting a 1-container

The insertion of a new 1-container is triggered when a new line object comes into existence or part of an existing one has a geometric change. As part of the required update propagation, the class compatibility of the line objects represented by the 1-container should be ascertained; where they are not class-compatible, the insertion should be rejected. For example, a river and a road cannot be represented by the same 1-container. In addition, the consistency rule 1CC_Rule_1 (see §5.1.2) should be enforced by the system. If the line objects which the 1-container is part of are class compatible and the consistency rule has been enforced, the update propagation can then continue by first determining if there is any existing 1-container having the topologic relationship r159 (see Figure 4.9), i.e., crossing, which is then stored as an instance of the explicit relationship Crosses (see Figure 3.8). Then each arc of the 1-container is inserted using the Insert_Arc algorithm. The algorithm Insert_1-container, in pseudo-code, is presented below, and its block diagram is shown in Appendix 1.1.10.

Algorithm Insert_1-container:

```
begin
    do for each new 1-container c1
      get G_c1(N_c1,A_c1)
      do 1CC_Rule_1 /* consistency rule for 1-container */
      select all existing 1-containers C_i for which c_k ∈ C_i| Relation(c1, c_k) ∈
                        {r159, r179, r191, r220, r223, r255, r400, r415, r435,
                        r439, r476, r477, r501} /* see Figure 4.9; using com-
                        putational geometry */
```

```
        for each c_k
            select line object LO_j for which Partof[LO_j,c_k]=1
            if map is single-valued
                notify user /* error, requires decision ∈ {interrupt, next c_k} */
            else
              if map is multi-valued
                determine compatibility between LO_j and new line object
                represented by c_k /* LUT or user decision */
                if class incompatible
                notify user /* error, requires decision ∈ {interrupt, next c_k} */
                else next c_k
                endif
              endif
            endif
        determine existing 1-container, ec1 for which Relation(c1,ec1)=r159
            /* using computational geometry (see Figure 4.9 for r159) */
        if ec1 exists, store Crosses[c1,ec1]=1 as properties of c1 and ec1
        for each a_i ∈ A_c1
            do Insert_Arc(a_i)
        next a_i
      next c1
end
```

Deleting a 1-container

The triggered request to delete a 1-container may arise if the line object(s) represented by the 1-container no longer exist or because of substantial locational changes of the objects. When an existing 1-container c1 is to be deleted, the following operations must be performed:
- check if there is any 1-container ec1 for which Crosses[c1,ec1]=1; if so modify ec1 by setting Crosses[c1,ec1]=0.
- delete the arcs of the 1-container using the algorithm for deleting an arc. An arc that exists only because of the 1-container will have equal values for its left and right relationships; such an arc should be deleted. Where these values are not equal, the value of the Partof relationship between the 1-container and the arc should be changed to zero or null because the arc still demarcates two area objects.

These operations can be translated into the following algorithm Delete_1-container for deleting a 1-container (the block diagram is given in Appendix 1.1.11).

Algorithm Delete_1-container:

```
begin
    get id of the 1-container → c1
    select the 1-container ec1 for which Crosses[c1,ec1] = 1
        if exists, set value to zero
    for each arc a_j ∈ A_c1 having Partof[c1,a_j]=1 set value of Partof = 0
    for each arc a_i ∈ A_c1 /* arcs of the 1-container c1 */
        do Delete_Arc(a_i)
end
```

Modifying a 1-container

Modification of an existing 1-container can be triggered in three ways: (1) when a new 1-container crosses a 1-container, thereby necessitating the modification of the value(s) of the relationship Crosses[1-container1,1-container2]; this modification, being a secondary update, will be triggered by the Insert_1-container algorithm; (2) as a secondary modification triggered by an insertion of a new line object, or deletion or (geometric) modification of another existing one causing the modification of *this* 1-container; this modification will therefore be triggered by the primary update; or (3) as a primary update triggered by the need to modify the geometry of one of the line objects which the 1-container is part of (e.g., Figure 6.4), or to delete some of the line objects which the 1-container is part-of.

Figure 6.4 Modification of a 1-container

In the latter case, if the modification involves changing only the value(s) of the relationship Part-of (between line object and the 1-container), this can be effected by simple "add", "remove" or "modify" commands of the DBMS. If geometry is involved and if the change is substantial, the strategy is to use the algorithm for inserting a 1-container to insert the modified version and use the algorithm for deleting a 1-container to delete the obsolete one. If only a few arcs are involved, the new arcs (to replace the affected ones) of the 1-container are inserted using the algorithm for inserting an arc, while the affected arcs are deleted using the algorithm for deleting arc.

For small changes, it must also be verified whether the 1-container crosses/intersects any other 1-container within the changed segment (in which case the two 1-containers will have a common node in this segment). Three new possibilities can then be identified:
- the two still intersect/cross at the same point despite the geometric change
- they are now disjoint
- they have a new intersection/crossing point and situation (upper/lower).

The first possibility requires no action. In the second, the value of the relationship Crosses for the two 1-containers should be set to zero. In the third, the new cross-point of the affected 1-container(s) should be determined and value of Crosses updated to reflect the new situation. These can be summarised in the following algorithm for (primary) modification of 1-container. The block diagram of the algorithm is presented in Appendix 1.1.12.

Algorithm Modify_1-container:

```
begin
        get 1-container_id → c1
        if much geometric change /* decision of user */
        then
                insert modified 1-container with Insert_1-container
                delete obsolete 1-container with Delete_1-container
        else
                retrieve G_c1(N_c1,A_c1)
```

```
                determine existing 1-container, ec1 for which Relation(c1,ec1)=r159 /*
                reference Figure 4.9 for r159) */
                if ec1 exists set Crosses[c1,ec1]=0
                select affected elements of A_c1 (say A_k)
                for each a_i ∈ A_n /* A_n = the replacement arcs */
                        do Insert_Arc(a_i)
                next a_i
                for each a_j ∈ A_k
                        do Delete_Arc(a_j)
                next a_j
        endif
        determine existing 1-container, ec1 for which Relation(c1,ec1)=r159
        if ec1 exists set Crosses[c1,ec1]=1
end
```

Inserting a 0-container

Before propagating the triggered insertion of a 0-container, it is necessary to verify that, where the 0-container represents more than one point object, the objects are class-compatible (see discussion on inserting 2-container). If the insertion should be propagated, the system should determine (by using a computational algorithm) if there is any existing 2-container topologically containing the 0-container and register the occurrence. The node defining the geometry of the 0-container should then be inserted using the Insert_Node algorithm.

These operations can be translated into the following algorithm Insert_0-container (see the block diagram in Appendix 1.1.13).

```
Algorithm Insert_0-container:
    begin
        do for each new 0-container c0
          get G_c0(N_c0,A_c0) /* note: A_c0 = ∅ */
          retrieve existing 0-container ec0 ∋ Relation(ec0,c0)=r272
              if exists and map is single-valued
                    notify user /* error, requires operator decision */
              endif
              else
                if exists and map is multi-valued
                 select point object(s) PO_j for which Repreby[PO_j,ec0]=1
                      determine compatibility between (each) PO_j and new point
                      object represented by c0 /* LUT or user decision */
                      if class incompatible
                      notify user /* error, requires operator decision */
                      endif
                endif
              endif
          determine existing 2-container, ec2 for which Contains[ec2,c0] = 1 /* using
          computational geometry */
          if ec1 exists, store Contains[ec2,c0]=1 as properties of c0 and ec2
```

```
            do Insert_Node(N_c0)
        next c0
end
```

Deleting a 0-container

The triggered deletion of a 0-container c0 may be required if the point object(s) represented by the 0-container no longer exist or because of changes in its/their location(s). Before effecting the delete operation, the existence of any 2-container ec2 for which Contains[ec2,c0]=1 should be checked; if so, modify the property of ec2 by setting Contains[ec2,c0]=0. Then the 0-container can be deleted and its node deleted using Delete_Node algorithm. These operations can be translated into the following algorithm Delete_0-container for deleting a 0-container (see the block diagram in Appendix 1.1.14).

Algorithm Delete_0-container:

```
begin
        get id of the 0-container → c0
        select n_i for which Repr[c0,n_i] = 1
        select the 2-container ec2 for which Contains[ec2,c0] = 1
                if exists, set value to zero
        delete c0
        do Delete_Node(n_i)
end
```

Modifying an existing 0-container

There are usually three reasons for a triggered modification of a 0-container:
(1) When one of the point objects represented by the 0-container requires deletion or positional shifting while the other point objects remain at the same location; this is a secondary request which involves a simple operation of modifying the value of the Represent-by relationship between the 0-container and the point object that triggers the update.
(2) Change in relationship Contains[2-container$_i$, 0-container$_j$] caused by a change in the status of the 2-container$_i$; this requires only a setting of the proper value for the relationship and will be propagated by the affected 2-container.
(3) Change in its position (this may also involve topologic changes).
In the last case, the best strategy is to treat the modified 0-container as new, thereby inserting it with the Insert_0-container algorithm. The obsolete version of it is then deleted using the Delete_0-container algorithm.
These can be translated into the following algorithm Modify_0-container. The block diagram of the algorithm is given in Appendix 1.1.15.

Algorithm Modify_0-container:

```
begin
        get 0-container_id → c0
        if ∃ positional change
        then
```

```
                insert modified 0-container with Insert_0-container
                delete obsolete 0-container with Delete_0-container
        else

                modify affected property of 0-container /* simple query */
        endif
end
```

6.2.4 Updating the Elementary Objects

We will now analyze the required updating operations when inserting, deleting, or modifying a point, line, or area object. The thematic aspects of the objects, such as its layer and thematic class, can be handled as discussed in §6.1. Here the emphasis will be on geometric aspects. The geometric update propagation initiated by the updating of a terrain object can be seen as a message carried by the object concerned into the database; the type of object and type of update (insert, delete or modify) will signify the message type and thus the actions to be performed by the system. This can be depicted by the following expression 6.1 (see also Figure 6.2):

$$UP(OT, UT) \rightarrow \{PR_i : LPUT_i\} \oplus \{SR_k : LPUT_k\} \qquad (7.1)$$

where UP = Update propagation message
OT = Object type ($\in$ {point, line, area})
UT = Update type ($\in$ {Insert (I), Delete (D), Modify (M)})
$\rightarrow$ = Triggers
PR_i = Primary receiver (affected) data type (i.e., mandatory and definitely affected)
$LPUT_i$ = List of (alerted) propagated update types ($\in$ {I,D,M}) for data type i
SR_k = Secondary receiver (consulted/probably affected) data type k
$\oplus$ = Possibly affected path

Updating Point Objects

As indicated in Figure 6.2c, any geometric updating of point objects will also trigger updating of 0-container and node data types. It will necessarily involve consultation of the arc data type, probably leading to the updating of some existing arcs. Thus the algorithms defined for the affected data types will be part of the update propagation chain for point objects. The propagation of insertion, modification and deletion of individual point objects in the DMMVM are discussed below.

Inserting a point object

The expression (7.1) with respect to insertion of a point object is:

$$UP(PO, I) \rightarrow \{PO: I; 0C: I,M; N: I, M\} \oplus \{AR: M, 2C: M\} \qquad (7.1.1)$$

This implies that the insertion message for a new point object PO triggers the insertion of a new object of point type, insertion of a new or modification of an existing instance of 0-container (0C) type, the insertion of a new or deletion of an existing instance of node (N)

type. The propagation will also consult the arc data (AR) and 2-containers (2C) for possible modification, e.g., if the position of the point object coincides with a point in the interior of an arc, thereby necessitating decomposition of the arc (GP_Rule_2b).
After the insertion of the object's thematic data, the geometric update propagation proceeds by inserting first the object as an instance of point type. To continue the propagation, the point object is assigned a temporary 0-container identifier and the Insert_0-container algorithm triggered. This will then trigger any necessary updating that may be required in the chain such as the updating of node and possibly an arc. The algorithm for inserting a point object will therefore have the following structure (see the block diagram in Appendix 1.1.16):

Algorithm Insert_Point:

```
begin
        get properties of point object PO /* id, coords, thematic class, layer, */
        insert thematic data
        assign 0-container identifier → c0
        do Insert_0-container(c0)
end
```

Deleting a point object

This will be required when an existing point object comes to the end of its life span. Like an insert operation, the delete point object operation is an initiator of update propagation which will have the following interpretation of the expression (7.1):

$$UP(PO, D) \rightarrow \{PO: D; 0C: D, M; N: D\} \oplus \{AR: M; 2C: M\} \qquad (7.1.2)$$

Expression 7.1.2 implies that when a message to delete (D) a point type object (PO) is issued, the update propagation operation will delete the instance from the point type, trigger updating of its 0-container (to delete or modify the 0-container), which in turn triggers the updating of an instance of data type node (for possible deletion). The operation on node will trigger the consultation of arc before the node is deleted and the operation on 0-container may require modification of an instance of the 2-container if the instance contains the 0-container. The algorithm for deleting an object of the Point type is therefore as follows (see Appendix 1.1.17 for the block diagram):

Algorithm Delete_Point:

```
begin
        get id of object → po
        delete thematic data of po
        select 0-container c0 where Repreby[po,c0]=1
        select point object po1 where Repreby[po1,c0]=1∧po≠po1
        if po1 exists
                set Repreby[po,c0]=1 to Repreby[po,c0]=0
                goto end
        else
                do Delete_0-container(c0)
```

```
        endif
end
```

Modifying a Point Object

There are usually three reasons for modifying a point object:
(1) change in the object's thematic data, e.g., thematic class
(2) consequential change in topologic relationship with other objects brought about by locational changes of those objects
(3) change in the object's position (which may also lead to topologic changes).
The first involves a simple operation of replacing the old thematic data by the new. The second operation will be taken care of by the primary update (which caused the topologic change). In the third case, if the modification involves positional change, the strategy is to insert the object as new using the Insert_Point algorithm and to delete the obsolete one with the Delete_Point algorithm. The update propagation chain for the modification is depicted by expression 7.1.3, indicating the need for consistent update propagation.

$$UP(PO, M) \rightarrow \{PO: M; 0C: M,D,I; N: D,I\} \oplus \{AR: M; 2C: M\} \qquad (7.1.3)$$

The algorithm for the updating is as follows (see the block diagram in Appendix 1.1.18):

Algorithm Modify_Point

```
begin
        get id of object (po) and new (modified) data
        modify thematic data /* simple DBMS commands */
        if positional change
                select 0-container colRepreby[po,c0]=1
                assign new 0-container id → nc0
                do Insert_0-container(nc0)
                do Delete_0-container(c0)
end
```

Updating Line Objects

A line object, like the two other generic objects area and point, has three components:
(1) the object's identity
(2) the object's non-spatial (thematic) data which are restricted to the object's class in our case and can be handled by simple update commands of the DBMS during implementation
(3) the geometric data which can be further split into three:
- the positional data as represented by its 1-containers
- the shape information which is derived from the aggregation of the 1-containers that define the object
- the topology of the object, which is derived from the topology of its 1-containers.

Thus, as shown in Figure 6.2b, the geometric updating of line objects will trigger updating of 1-container, arc and node data types. It will necessarily involve consultation of the existing instances of 2-container data type to derive the left and right properties of the arcs defining

the line object's 1-container. Also, instances of 0-container will be chained by operations on nodes of the line object. Hence updating on line objects will propagate to the two lower levels. The propagation of insertion, modification and deletion of geometric aspects of individual line objects in the DMMVM are presented below.

Inserting a Line Object

The insertion of the line object's geometric data can be decomposed into insertion of the 1-container that defines the object and the algorithm for inserting 1-container used to insert this and propagate the insertion of its arcs and nodes. The propagation chain for inserting a line object is given in expression 7.1.4 below (the secondary path is not involved).

$$UP(LO, I) \rightarrow \{LO: I;\ 1C: I,D,M;\ AR: I,D,M;\ N: I, D\} \qquad (7.1.4)$$

After the insertion of the object as an instance of the line type and its thematic data, the geometric update propagation proceeds by assigning a temporary 1-container identifier to the object. The Insert_1-container algorithm is then triggered. This will in turn trigger any necessary updating that may be required in the chain such as the insertion of new instances and/or modification of existing instances of arc (AR) type. The algorithm for inserting a line object will therefore have the following structure (see Appendix 1.1.19 for the block diagram):

Algorithm Insert_Line:

```
begin
        get properties of line object LO /* {id, coords, layer, etc. */
        insert thematic data
        assign 1-container identifier → c1
        do Insert_1-container(c1)
        enforce LO_Rule_1
end
```

Deleting a Line Object

A line object that has come to the end of its life span should be deleted from the database. Also, an existing line object may have undergone such a substantial positional change that it is better to insert it as a new object and delete the obsolete one. The delete propagation chain for a line object is shown in expression 7.1.5

$$UP(LO, I) \rightarrow \{LO: D;\ 1C: D,M;\ AR: D,M;\ N: D\} \qquad (7.1.5)$$

The expression indicates that when a delete message is sent to an object of the line type, this triggers the deletion of the instance from the line type, and triggers the deletion or modification of its 1-containers. The triggered deletion of a 1-container will trigger the necessary operation on instances of arc AR which propagates to some instances of node N. These operations are indicated in the following algorithm for deleting a line object. The block diagram of the procedure is given in Appendix 1.1.20.

Algorithm Delete_Line:

```
begin
        get id of object → lo
        delete thematic data of lo
        select 1-containers C1 ∋ for each c1 ∈ C1 is Partof[lo,c1]=1
        for each c1
          select line object lo1 where Partof[lo1,c1]=1 ∧ lo≠lo1
          if lo1 exists
                set Partof[lo,c1]=0
          else
                do Delete_1-container(c1)
          endif
        next c1
end
```

Modifying a Line Object

The need to modify an existing line object may result from one, or a combination, of the following changes:
(1) thematic changes, e.g., the object's class; these can be handled by simple updating commands and require no propagation;
(2) geometric changes.
For geometric changes, since a line object is geometrically represented by 1-containers, it is possible to identify the elements of the set of its 1-containers in which the change has taken place. The changed segment(s) of the line object can then be inserted using the algorithm for inserting a 1-container while the obsolete 1-containers should be deleted using the algorithm for deleting a 1-container. When a major part of the object has changed, the strategy is to insert the new one using the algorithm for inserting a line and delete the old object using the algorithm for deleting a line. In either case, the update propagation chain will be as shown in expression 7.1.6.

$$UP(LO, M) \rightarrow \{LO: M;\ 1C: I,D,M;\ AR: I,D,M;\ N: I,D\} \qquad (7.1.6)$$

This is expressed in algorithmic form as follows (see block diagram in Appendix 1.1.21):

Algorithm Modify_Line

```
begin
        get id of object (lo) and the replacement data
        modify thematic data /* simple DBMS commands */
        if positional change
        select 1-containers C1 ∋ for each c1 ∈ C1 is Partof[lo,c1]=1
          if substantial positional change /* decision of human operator */
                assign new 1-container id → nc1 /* object treated as one 1-container
                */
                do Insert_1-container(nc1)
                for each c1
```

```
                do Delete_1-container(c1)
                next c1
          else
                identify C11 ⊂ C1 in which positional changes occur
                for each c_i ∈ C11
                do insert_1-container(c_i)
                next c_i
                for each c_j ∈ {C1 - C11}
                do delete_1-container(c_j)
          endif
        endif
end
```

Updating Area Objects

Like point and line objects, an area object has two basic properties: thematic and geometric. Updating of the thematic data, as in the other two types, can be taken care of by DBMS commands during implementation. In the geometric domain, however, inserting, deleting or modifying a single area object may lead to further operations on other existing neighbouring area objects; thus updating of a single area object is a complex operation. The propagation of updating operations of inserting, deleting and modifying the geometric data of a single area object are presented in the following subsections.

Inserting a single area object

The complete classification (completeness of incidence) constraint of the DMMVM implies that all two-dimensional objects must be classified (even if as class "unclassified"); thus inserting an area object will always involve a combination of insert operations (for the new object) and possibly deletion and/or modification of existing area objects which are spatially coincident with the new object. The insertion of the area object's geometric data can be decomposed into insertion of a 2-container and the algorithm for inserting a 2-container used to insert this and propagate the insertion of its arcs and nodes. The insert algorithm for the 2-container already has facilities for modifying neighbouring 2-containers if they are affected. The propagation chain for inserting an area object is given in expression 7.1.7 below.

$$UP(AO, I) \rightarrow \{AO: I,D,M;\ 2C: I,D,M;\ AR: I,D,M;\ N: I, D\} \oplus \{0C: M\} \qquad (7.1.7)$$

This indicates that apart from inserting the area object, other area objects may have to be deleted or modified; the insertion of the new area object will then trigger the insertion of new 2-containers (2C) with possible deletion or modification of some existing ones, and similarly for arc (AR) and node (N) in the propagation chain. Some existing instances of the 0-container type may have to be modified if they are topologically contained by the new 2-containers of the area object. The updating operations to be performed on the existing area objects which are affected by the insertion of the new object cannot be determined by the system (e.g., in a cadastral database, if a new parcel, represented as area object, overlaps with an existing one, the system can only notify the user while the user decides whether to reduce the existing one or the new one as one of the numerous possible decisions). This aspect has to be done interactively. The propagation of the insertion will proceed by regarding the area

object as a new single 2-container. The following algorithm is proposed for inserting an area object. The block diagram of the algorithm is presented in Appendix 1.1.22.

Algorithm Insert_Area:

```
begin
        get properties of area object AO /* id, coords, thematic class, layer, */
        insert thematic data
        assign 2-container identifier → c2
        do Insert_2-container(c2)
        enforce AO_Rule_1
end
```

Deleting an Area Object

Deleting a single area object will also involve a combination of delete, update and insert operations. The operation will be required if an area object has come to the end of its life span or if an existing one has undergone such a substantial positional change that it is better to insert it as new object and delete the obsolete one. The updating also requires interaction with the human operator because of its effects on neighbouring area objects (whether to allocate its space to only one or among all of its former neighbours).
The delete propagation chain for an area object is shown in expression 7.1.8

$$UP(AO, D) \rightarrow \{AO: D,M; 2C: D,M; AR: D,M; N: D\} \oplus \{0C: M\} \qquad (7.1.8)$$

The expression indicates that when a delete message is sent to an instance of the area type, it will require not only the deletion of the instance from the area type but also the modification of neighbouring ones. It will require the deletion or modification of some 2-containers, which will trigger deletion or modification of some instances of arc AR, which propagates to some instances of node N. Being an interactive operation, the system should select the 2-containers representing the area object as well as related 2-containers for the human operator to decide on the necessary modification of affected neighbours. The human operator will use any of the updating algorithms defined for the lower level data types (2-container, arc and node, especially the last two). The outline of the operations is indicated in the following algorithm for deleting an area object (see Appendix 1.1.23 for the block diagram).

Algorithm Delete_Area:

```
begin
        get id of object → ao
        delete thematic data of ao
        select 2-containers C2 ∋ ∀ c2 ∈ C2 is Partof[ao,c2]=1
        for each c2 ∈ C2
                display all 2-containers EC2 ∋ ∀ ec2 ∈ EC2 is Relation(ec2,c2) ∈
                {r179, r220, r400, r435, r476, r511}
                perform interactive updating /* using update algorithms of lower level
                data types */
```

```
        next c2
end
```

Modifying an Area Object

The need to modify an existing area object may result from one, or a combination, of the following changes:
- thematic changes, e.g., the object's class; these can be handled by simple updating commands and require no propagation;
- geometric changes.

Geometric changes will also affect neighbouring area objects; thus the same situation applies as in the insertion and deletion of area objects. The update propagation path that will guide in the interactive updating during geometric updating is shown in expression 7.1.9.

$$UP(AO, M) \rightarrow \{AO: I,D,M; 2C: I,D,M; AR: I,D,M; N: I,D\} \oplus \{0C: M\} \qquad (7.1.9)$$

This is expressed in algorithmic form as follows (see the block diagram in Appendix 1.1.24):

Algorithm Modify_Area

```
begin
        get id of object (ao) and the replacement data
        modify thematic data /* simple DBMS commands */
        if positional change
          select 2-containers C2 ∋ ∀ c2 ∈ C2 is Partof[ao,c2]=1
          display all 2-containers EC2 ∋ ∀ ec2 ∈ EC2 is Relation(ec2,c2) ∈ {r179,
          r220, r400, r435, r476, r511}
          perform interactive updating /* using update algorithms of lower level data
          types */
        endif
end
```

6.3 Handling Updating of Multiple Objects

The updating procedure described in the previous section covers the eight data types (the elementary object types area, line and point, the m-containers, m ∈ {0,1,2}, and the two geometric primitives arc and node) that have been used to model multi-valued vector maps. As stated in that section, users operate at the level of the elementary object types: area line and point; an update on any instance of the three is then dynamically propagated to the other data types. The procedure implies that terrain objects are being updated on individual basis, which is in line with the fact that a spatial database stores information concerning a group of individual terrain objects. It also conforms with the normal practice because the need to update a spatial database usually arises when (on individual basis), one or more terrain objects come into existence, require modifications, or become extinct. The collection of the new data (by whatever method: photogrammetry, land surveying, etc.) is therefore usually object-based, e.g., recognizing that a new road has been constructed and has to be added to the database.

Consequently, updating of multiple objects in the database can still be based on the same algorithms. The implementation of the algorithms can be optimized to efficiently accommodate the complexity introduced when many objects are to be updated at the same time, for example, through the use of efficient search trees and a contiguous storage of related data. Updating of multiple objects at one time can then be grouped into the following:
- addition, removal or modification of multiple objects of the same generic type (e.g., multiple point objects)
- addition and/or removal of multiple objects belonging to different generic types.

Updating multiple objects of the same generic type

In this case, the operation is handled by treating one object at a time, within a loop, using the appropriate algorithm.

Updating multiple objects belonging to different generic types

These involve one or more point objects and/or one or more line objects and/or one or more area objects. The following strategy can be used to handle the operation:
- insert/delete/modify all the point objects (if any), one at a time, using the relevant algorithm for point object
- insert/delete/modify all the line objects (if any) using the relevant algorithm for line object
- insert/delete/modify all the area objects (if any) using relevant algorithms.

By grouping objects of the same type together, the same operation (e.g., Insert_Area) can be performed in a loop instead of changing from one operation to another in an ad-hoc manner. However, to delete objects in a region (e.g., defined by some coordinates), since some of those objects may be partially inside the region, the best approach is for the system to retrieve all objects that are associated with (i.e, fully or partially in) the region for the user to select those that should be deleted or to even remodify the region to carve out or include more objects.

6.4 Summary

In geo-information production, the cost of data collection has been said to be about seven to 10 times more than the cost of the hardware and software needed to establish the database (Peled, 1994). Thus it is very important that the accuracy and currency of the data should be reliable, such that the purpose for setting up the database can be fulfilled with profitable cost recovery. This chapter aimed at contributing towards achieving this by providing algorithms for consistent automated update propagation.

Although geo-information updating includes change detection, data collection and database updating, the focus in the chapter was on automated database updating under the assumption that the necessary changes have been detected and captured in readiness for input into a DMMVM-structured database. Ideally, the defined algorithms should be translated into computer modules within an existing DBMS. However, since most of the operational DBMS are not capable of accepting user-defined rules and data types, the algorithms may have to

be programmed in a high-level language and then coupled with the DBMS during implementation.

It should be noted that, although the final goal is to automate spatial database updating, some aspects will still have to be done interactively (i.e., require human intervention), aided by graphic visualisation. This limitation is related mostly to the area objects with respect to the correct decision to take when modifying the neighbouring area objects (of the same layer) effected by an area object that is being updated.

After the update propagation, the consistency rule for planar enforcement (see §5.1.3) must be enforced. In addition, some of the other consistency rules (especially semantic) defined in the last chapter can be enforced immediately after the update and certainly at regular intervals.

For the purpose of updating, in order to improve performance, the geographic space can be partitioned into different levels of windows (from coarse to fine), for example, using quadtree, and the list of the geometric primitives within a window, together with the coordinates of the window, can then be stored, e.g., in a random access file. This information can then be used when inserting a new object so that the necessary topologic editing is localised instead of involving the entire geometric primitives in the database.

Some of the proposed algorithms have been experimented in automated update propagation in single-valued vector maps. The experiment was done by coupling Microsoft Fortran with an Oracle DBMS in a microcomputer environment at the Department of Land Surveying, Photogrammetry and Remote Sensing, Wageningen Agricultural University. The algorithms were translated into Fortran programs, while Oracle served as the RDBMS retrieving the necessary data from the database into the Fortran program, for updating and consistency operations, and returning the updated data back into the database. Details of this experiment can be found in Kufoniyi (1989) and Kufoniyi et al (1993). Further experimentation of the algorithms are presented in chapter 8 of this thesis, using the Postgres DBMS, an extended relational database management system.

The next chapter focuses on the translation of the DMMVM into two prototype database structures (relational and object oriented). If the relational prototype is used for an implementation, then the update propagation algorithms and consistency rules would have to be handled by a high-level programming language and coupled with the RDBMS as in the experiment cited above, since most operational RDBMSs are not capable of handling user-defined rules. If the object-oriented (OO) prototype is chosen, the algorithms have been defined as class methods, which should then be programmed using the programming language of the OO system.

7

DATABASE STRUCTURES FOR MULTI-VALUED VECTOR MAPS

A conceptual data model is normally developed without consideration of the type of system that will implement it. This also holds for the object-based conceptual data model developed in chapter 3. However, for implementation purposes, it is necessary to translate the model into a prototype database structure, based usually on a database model (e.g., relational, network, etc.). A review of the common database models was presented in chapter 2, with the relational model being the most popular. Most of the present-day operational GI systems are built on the relational model, noted for its simplicity and standard query language. Another database model which is fast gaining importance in GIS is the object-oriented model, which has been acclaimed to be more suitable for spatial applications than the relational model (see chapter 2). However, mature operational object-oriented systems are not yet common.

The approach in this thesis is therefore to translate the DMMVM into two prototype database structures, one relational and the second object-oriented. The relational structure can serve for immediate implementation while the object-oriented structure can be implemented with the availability of mature operational object-oriented systems. In addition, the relational system can be upgraded into an evolutionary object-oriented (object-relational) systems by building an object shell on top of the relational system or by implementing the structure in an extended relational system (see §7.2.3). §7.1 describes the prototype relational database structure for multi-valued vector maps while the object-oriented data structure is presented in §7.2.

7.1 Relational Database Structure for Multi-valued Vector Maps

In this section, the translation of the DMMVM into a prototype relational database structure is presented. Smith's method for relational database design is used for the translation. This method was reviewed in chapter 2 and comprises four main steps, namely:

(1) identification of the basic data types and relationships to be represented by the database structure;
(2) listing of the functional dependency statements among the data types, showing the single- and multi-valued dependencies;
(3) construction of the dependency diagram; and
(4) construction of relations (tables) from the dependency diagram.

7.1.1 Identification of Data Types

The data types in the DMMVM are indicated in Figure 3.8. Since the identifier of an object in one layer may be the same as that of another object in another layer, it is necessary to add the thematic layer from which an object originated and the geographic name of the object as additional data types. Also, the planimetric and vertical accuracies as well as the lineage (e.g., source and method of acquisition) of the point are added as optional data (see also Chhatkuli, 1993 and Bouloucos et al, 1994 for modelling of geometric data quality in multi-valued vector maps).

7.1.2 Dependency Statements

The identified data types together with the elementary links among the data types (see §3.4.3), are translated into the following dependency statements. The data types and the link types to be explicitly represented in the data structure are in bold characters. The total number of layers is represented by N_L.

(1) One 1-container, identified by a **OneC**, represents *l* **Lobj** line objects ($l \leq N_L$), each of which originates from a map **Layer**, has a name **Lname**, and belongs to one **Lclass** thematic class in the layer.

(2) One 2-container, identified by an identifier **TwoC** represents *k* **Aobj** area objects ($k = N_L$), each of which originates from a map **Layer**, has a name **Aname**, and belongs to one **Aclass** thematic class in the layer.

(3) One 0-container, identified by **ZeroC**, represents *j* **Pobj** point objects ($j \leq N_L$), each of which originates from a map **Layer**, has a name **Pname** and belongs to one **Pclass** thematic class in the layer.

(4) Each **ZeroC** is geometrically represented by one **Pnode** node number and may lie in one **TwoC** 2-container.

(5) An arc, identified by an **Arcnr**, is defined by one **Bnode** starting node and one **Enode** end node and has one **LftTwoC** left 2-container identifier and one **RgtTwoC** right 2-container identifier and represents one **AOneC** 1-container.

(6) Each **Nodenr** node has a position given by a triplet of **Xcoord**, **Ycoord**, and **Zcoord** coordinates, and optionally has a **Pacc** planimetric accuracy, a **Vacc** height accuracy, a **P_lng** planimetric lineage and a **V_lng** elevation lineage.

(7) Two 1-containers identified by **U1c** upper 1-container and **L1c** lower 1-container may cross each other at a **Crosspt** point (crosspt belongs to the domain of nodenr).

7.1.3 The Dependency Diagram.

The dependency diagram constructed from the above dependency statements is given in Figure 7.1. Each data type is represented in an ellipse (or double ellipses to facilitate the representation of links) and the link between two data types is numbered according to the dependency statement from which the link is taken. Links may have single or double-headed arrows depending on whether the link represents single-valued or multi-valued dependencies (not to be confused with the same terms in vector maps, see §2.2.3 and §3.0). Related fields may be combined in a single bubble (ellipse) by using the "+" symbol. The domain flags, triangles with numbers inside them, easily identify all fields with a common domain and facilitates referential integrity rules at the implementation stage. Optional data types are written in lower-case letters.

7.1.4 Composing Fully Normalised Relations from the Dependency Diagram

The following seven relations are composed directly from the dependency diagram in Figure 7.1. Table names are written in upper-case letters and primary keys are underlined. The fields in square brackets are optional. Figure 7.2 shows the relational structure and the links among the tables.

AREA (twoC, aobj, layer, aname, aclass)
LINE (oneC, lobj, layer, lname, lclass)
POINT (zeroC, pobj, layer, pname, pclass)
POINTNODE (zeroC, twoC, pnode)
ARC (arcnr, bnode, enode, lftTwoC, rgtTwoC, aOneC)
NODE (nodenr, xcoord, ycoord, zcoord [,pacc, vacc, p_lng, v_lng])
LINCROSS (u1c, l1c, crosspt)

7.1.5 Implementing the Relational Structure

The prototype relational structure developed above can then be implemented in any relational system by creating a database consisting of the seven base tables in which the columns of each table are mapped into the system's built-in data types (e.g., numeric, character, etc.). The database will then be manipulated with a RDBMS.

In order to incorporate additional thematic information for the objects, extra tables should be created, one for each thematic class with each attribute of the class becoming one column of the table. The object identifiers (aobj, lobj, pobj) and layer identifier (layer) should be included as columns in the thematic tables in order to link the object's geometric and non-spatial data sets.

In the first experimental implementation of this prototype (with 2D position), an integrated land use and soil database was created in a micro-computer environment using dBase-IV as RDBMS. In the experiment, the land use map and the soil map (hard copies) covering the same geographic space and at the same scale were manual overlaid and coded into a dBase-IV DBMS to test the information content of the model. Details of the experiment can be found in Ayugi (1992) and Bouloucos et al (1993).

Another experimental implementation of the prototype, with objects' position still defined in 2D metric and topologic space but with incorporation of geometric data quality parameters, was carried out on Arc/dBase-IV configuration in which PC Arc/Info was used as the data acquisition subsystem and dBase-IV was used for the database management. For this experiment, "segment" was introduced as an additional data type (defined as a straight line between two adjacent vertices) to define the shape of an arc (which has been assumed to be straight in this prototype and in the first experiment). After performing the data acquisition and the necessary overlays in Arc/Info, both the geometric and thematic data were transferred into the dBase-IV DBMS and structured according to our relational prototype (with the necessary modifications to accommodate quality parameters and the segments) and then used for single- and multi-layer queries. Thus both the geometric and thematic datasets were organized in the same structure and managed by a single DBMS.

The introduction of the segment improved the performance of topologic queries because a reduced number of geometric primitives (arcs and nodes) are searched in comparison with an implementation of arcs as straight lines. However, a lot of computational effort was required to achieve the integrated structure because of the layer approach of Arc/Info. Details of this implementation can be found in Chhatkuli (1993) and Bouloucos et al (1994). Also, an example of the design and implementation (using Oracle RDBMS) of a prototype relational

database structure for single-valued vector maps can be found in Kufoniyi (1989 and Bouloucos et al (1990).

A system that appears more promising for the implementation of this relational prototype is the System 9 which uses a different kind of layer-approach in which a geographic layer is built on top of a standard relational DBMS (Eck and Uffer, 1990). This facilitates the integrated storage of spatial and non-spatial data, thus allowing to take benefit of consistency and concurrency control mechanisms on both kinds of data (Boursier and Faiz, 1993). The system defines three basic geometric data types for the representation of simple geographic objects, from which more complex objects can also be derived. The three generic primitives are node, line and area, which are comparable to our elementary object types point, line and area respectively.

All the experiments indicated that a relational system can be used for the implementation of vector maps. However, such issues as graphic output of query results, update propagation and integrity maintenance must be handled by user-written (or any third party) procedures. Algorithms for the automated update propagation and consistency enforcement in a relational data structure for single-valued vector maps (which were tested by coupling editing procedures written in Fortran with Oracle DBMS) have been proposed by the author (see Kufoniyi, 1989 and Kufoniyi et al, 1993). Similar approach can be used to implement the consistency rules provided in chapter 5 and the updating procedure of chapter 6 for this prototype, i.e., by using any programming language (e.g., C, Pascal, Fortran, etc) that is compatible with the chosen RDBMS to program the consistency rules and updating algorithms and then couple these with the RDBMS.

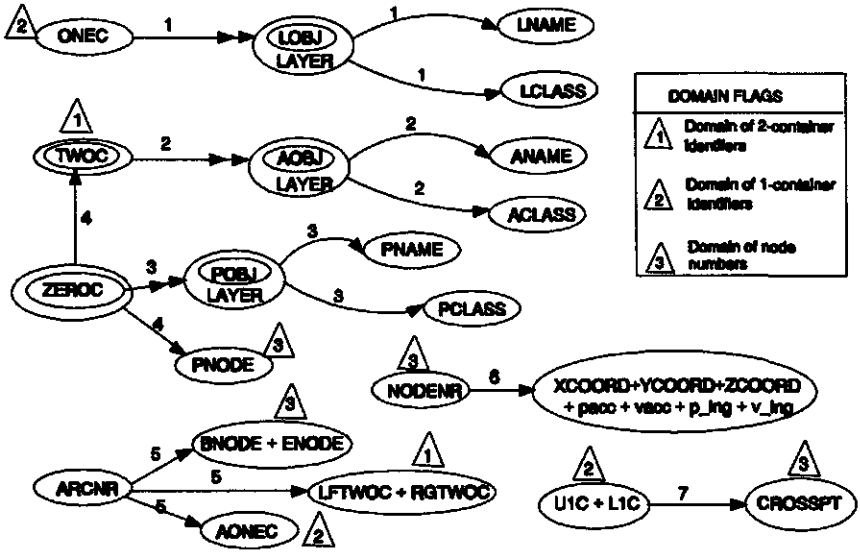


Figure 7.1 Dependency diagram for designing relational structure for the DMMVM

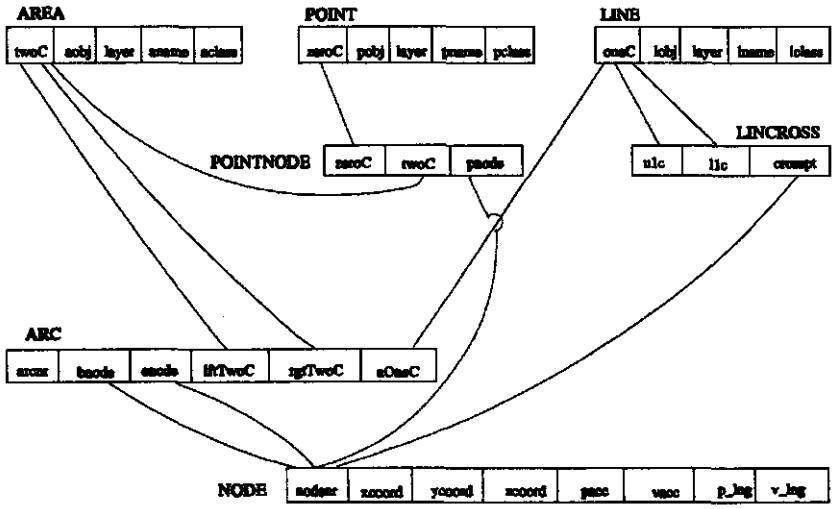


Figure 7.2 Relational database structure for multi-valued vector maps

7.2 Object-Oriented Data Structure for Multi-valued Vector Maps

In chapter 2, the main features of object-oriented data modelling were described. Some of the modelling constructs (classification, generalisation, aggregation and association), together with the concepts of inheritance and propagation, will be applied here to map the data model shown in Figure 3.8 to an object-oriented data structure.

7.2.1 Class Definitions and Modelling

From the basic structure of spatial objects shown in Figure 2.2, two classification domains are distinguished, namely thematic domain and geometric domain. Thus each terrain object will be an instance of one of the thematic classes and an instance of one of the geometric classes as shown in Figures 7.3 (i.e., double inheritance). The mapping of the DMMVM (Figure 3.8) to an object-oriented data model is represented in Figure 7.4 in which the classes are represented by rectangles and the links among the classes by arrows. The ellipses show the separation between the two classification domains and emphasise the central focus of terrain objects in the model.

The thematic classification domain deals with the non-spatial characteristics of an object. In the scheme, objects with common non-spatial attributes are mapped into a common thematic class within one mapping layer (see Chapter 3) such that each object belongs to only one thematic class in that layer. Each class will have a list of properties (attributes) whose values are then evaluated during application for each object belonging to that class. Here, the classification may be hierarchical, such that at a higher level classes that share common properties are also generalised into a superclass.

The actual classification and instantiation in the semantic domain can be done only during application. Thus the thematic classification line will be restricted to the class labels of each object which are symbolically represented by Linthemclass, Arthemclass and Pntthemclass in Figure 7.4, and further attention will be focused on the geometric domain. During application, the three class labels can then be expanded and structured as described in §3.2.1 and §3.4.1

The geometric classification line deals with the geometric characteristics of an object. In this domain, objects are classified into three geometric classes in a 2D topologic space. The three classes, also referred to as object types, are:
- Pointtype: 0D objects having position but no spatial extension.
- Linetype: objects having shape, position and 1D spatial extension.
- Areatype: objects which extend over two spatial dimensions having position, shape and size.

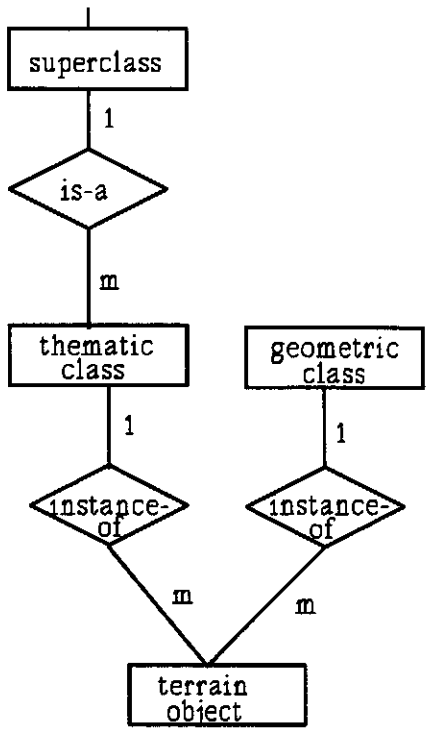


Figure 7.3 Classification structure for spatial objects

Each terrain object (indicated as Lineobj, Areaobj and Pointobj in Figure 7.4) will be mapped onto only one type (either on one-to-one basis or one-to-many) of these three geometric classes and the classification is not hierarchical. Note, however, that aggregation hierarchies can be defined to group objects of different types into a more complex object when the model is used to implement complex applications.

The three geometric classes are regarded as aggregated objects with five subordinate classes (2-container, 1-container, 0-container, arc and node) to define the geometric characteristics of each instance, as shown in Figure 7.4 and described below. In the figure, "instance-of" represents classification, "part-of" indicates aggregation, "member-of" indicates association and "is-a" represents generalisation. The constraints for each class are not repeated here since they have been treated under the consistency rules in §5.1. The major constraints to be monitored and enforced are translated into class methods (operations).

Areatype

An instance of this class is a complex object in the sense of geometric representation. It is composed of one or more 2-containers (see Figure 7.4). The geometric properties (positional information, metric area, etc.) of the object are derived by propagation from the geometric properties of the component 2-containers, e.g., the metric area of an instance of Areatype is

the sum of the metric area of each of its component 2-containers. Some properties will still require explicit representation. These properties are the object identifier of the instance, its thematic class label (instance-of), its layer, and the set of its component 2-containers. It is necessary to mention that if the structure is used to model a single-valued vector map, then an instance of the Areatype will have only one 2-container component (the same goes for Linetype and Pointtype vis-a-vis 1-container and 0-container).

An (explicit topologic) association (member-of) exists between two instances of the class if they have one or more common 2-containers. The topologic relationships (see §4.2.2.2) between an instance of the class and an instance of the same class or Linetype class or Pointtype class define other types of association. One of these is the explicit representation of topologic containment of an instance of Pointtype by an instance of this class as a property of the 0-container class. The other topologic relationships will be dynamically derived by predefined methods using the algorithms proposed in §4.3.
The properties (attributes) and the operations (methods) of this class are presented in §7.2.2.

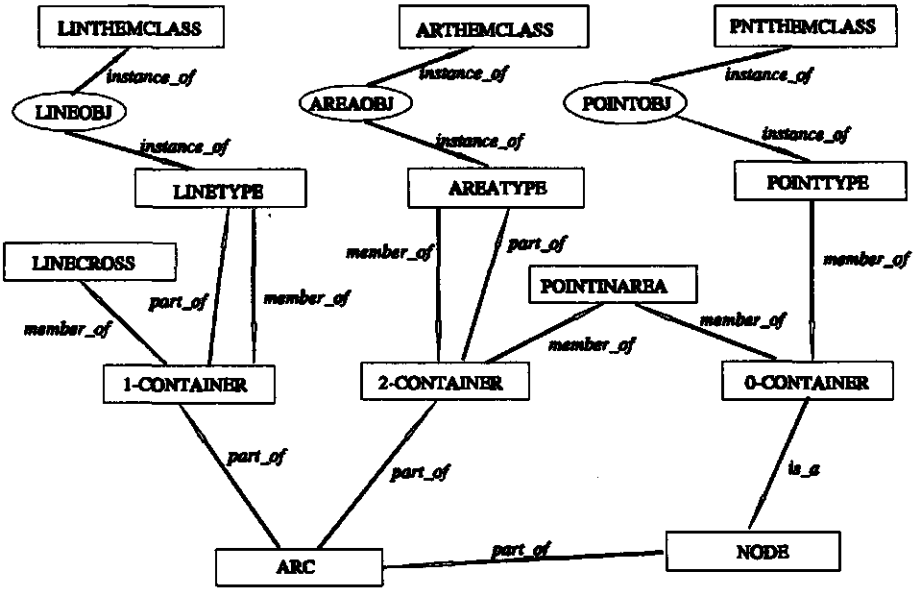


Figure 7.4 Object-oriented data model for multi-valued vector maps

Linetype

An instance of the Linetype is an aggregation of one or more instances of the 1-container class (see Figure 7.4). Its geometric properties are therefore derived by propagation from the geometric properties of the component 1-containers. The explicit properties of the class include the object identifier, its thematic class, its layer, and the set of its component 1-containers (see the database schema in §7.2.2). As in Areatype, an association exists between two instances of the class if they have one or more common 1-container. The topologic

relationships (see §4.2.2.2) between an instance of the class and an instance of the same class or Areatype class or Pointtype class define other types of association, one of which is made explicit as Linecross class at the level of 1-container (for explicit representation of the crossing of two instances of Linetype class). The other topologic relationships will be dynamically derived by predefined methods using the algorithms in §4.3.

Pointtype

This is the third geometric class to which a terrain object can belong. An instance of the class is a member of an association defined by an instance of the 0-container class. The corresponding 0-container defines an association between two or more instances of Pointtype if those instances are spatially coincident (i.e., overlap in space). The geometric properties (e.g., position) of the member Pointtype are then derived from the geometric properties of the corresponding 0-container. The explicit properties of the class include the object identifier, its thematic class, its layer, and the identifier of the 0-container of which it is a member (see the database schema in §7.2.2). As in the above classes, the topologic relationships (see §4.2.2.2) between an instance of the class and an instance of the same class or Areatype class or Linetype class define other types of association. One of these, the topologic containment of an instance of Pointtype by an instance of Areatype, is made explicit at geometric level as a property of the 0-container class. The other topologic relationships will then be dynamically derived by predefined methods using the algorithms proposed in §4.3.

2-Container

The 2-container class models a geometric association between spatially coinciding instances of Areatype. An instance of the class is part of one or more instances of the class Areatype. The introduction of this class is very useful in multi-valued analysis because the thematic properties of an instance are the elements of the set union of the thematic properties of the member instances of class Areatype (similarly for 1-container/Linetype and 0-container/Pointtype). Put differently, given instance A of class Areatype with properties' set P_A, an instance B of the same class with properties' set P_B, and an instance C of class 2-container modelling spatial coincidence between A and B, the properties P_C of $C \subseteq (P_A \cup P_B)$. Object C can then be manipulated as a unique object in the database. To the extent that a simple object of class Areatype is isomorphic to an instance of class 2-container (especially in single-valued vector map representation where there is a one-to-one mapping between an Areatype object and a 2-container object), some methods defined for the former may be applicable to the latter e.g., checking for the graph-closure of the object.

The class is at a lower level compared with the Areatype class because instances of the class exist only if instances of the Areatype exist. An instance is geometrically an aggregation of one or more instances of class Arc; the positional information as well as some geometric properties are therefore derived from the Arc class by propagation. The properties and methods of the class are presented in the database schema in §7.2.2.

1-Container

As in the 2-container class, the 1-container models a geometric association between spatially coinciding instances of Linetype and an instance of the class is part of one or more instances

of the class Linetype. The properties of the instance are the elements of the set union of the properties of the member instances of class Linetype. Also, some methods defined for the Linetype may be applicable to the 1-container class, e.g., checking that the object's graph is simple and elementary.

The class is at a lower level compared with the Linetype class since they come into existence only when instances of the Linetype exist. An instance is geometrically an aggregation of one or more instances of class Arc; thus the positional information as well as some geometric properties are derived from the Arc class by propagation. The properties and methods of the class are presented in §7.2.2.

0-Container

The 0-container class is defined as a subclass of the class Node; thus it inherits the properties of the Node class (i.e., the node-id and coordinate triplet). The class also models a geometric association between spatially coinciding instances of Pointtype. In addition to the properties inherited from the node class, the class derives its (thematic) properties from the set union of the thematic properties of its member Pointtype objects by propagation. It sends positional information to the member Pointtype objects.
The explicit properties of the class and its methods are presented in the database schema in §7.2.2.

Arc

This class participates in a double aggregation hierarchy (in the upward direction, see Figure 7.4). One aggregation hierarchy is the link between the Arc class and the 1-container class. In this scheme, an instance of class Arc is part of one 1-container object. The second scheme is the link between the Arc and the 2-container. Here, one instance of the class can be part of at most two instances of 2-container class (i.e., as a boundary of two adjacent 2-containers). The instances of the class are geometric primitives of instances of the 1-container and 2-container classes; thus an instance of the class will not exist unless a corresponding 1-container or 2-container exists. It therefore sends geometric properties to the 1-container and 2-container classes.

An instance of the class is itself, geometrically, an aggregate object, being composed of two instances of the Node class. The positional information of the instance is therefore derived from the Node class by propagation. The properties and methods of the class are given in §7.2.2.

Node

This is an atomic class whose properties are the coordinate information of the vector map. The planimetric accuracy, vertical accuracy, and planimetric and vertical lineage (e.g., source and method of acquisition) of the node can be added as additional properties if desired.
The class is treated as a superclass of 0-container and an instance of the class is (optionally) part of an instance of the Arc class. The properties and methods of the class are given in the schema below.

7.2.2 Object-Oriented Database Schema

[1] Class: Areatype

Properties:

- AreaObjId (OID, i.e., object's identity)
- instance-of ArThemClass (its thematic class)
- Layer (from which layer i.e., layer identifier)
- 2-containerList: setof member 2-containers (OIDs of its 2-containers)

Operations:

- Insert (Area) /* inserts a new instance of the class and enforces topologic & geometric (i.e., spatial) constraints */
- Retrieve (Area) /* retrieves an instance of the class */
- Delete (Area) /* deletes an instance of the class and enforces spatial constraints */
- Modify (Area) /* modifies instance of the class and enforces spatial constraints */
- Relation(P,Q) /* determines topologic relationship between two instances of the class, or an instance P of the class and an instance Q of Linetype or Pointtype */
- ComputeMetricArea /* computes metric area of an instance by propagation of metric area values of member instances of 2-container */
- ComputePerimeter /* computes and returns the perimeter of an instance of the class as aggregation of the lengths of arcs defining the boundary of the instance by propagation from member 2-containers */
- Retrievesubgraph (Area) /* retrieves the arcs and nodes defining the boundary of the instance by propagation from member 2-containers */
- graphCheck /* consistency check for the connectedness of boundary of an instance */

[2] Class: Linetype

Properties:

- LineObjId (OID)
- instance-of LinThemClass (object's thematic class)
- Layer (its layer)
- 1-containerList: setof member 1-containers (OIDs of its 1-container)

Operations:

- Insert (Line) /* inserts a new instance of the class and enforces spatial constraints */
- Retrieve (Line) /* retrieves an instance of the class */
- Delete (Line) /* deletes an instance of the class and enforces spatial constraints */
- Modify (Line) /* modifies instance of the class and enforces constraints */
- Relation(P,Q) /* determines topologic relationships between two instances of the class, or between instance P of Areatype and an instance Q of the class, or instance P of the class and an instance Q of Pointtype */
- ComputeLength /* computes metric length of the instance as aggregation of

lengths of member 1-containers */
- Retrievesubgraph (Line) /* retrieves the arcs and nodes defining the geometry of an instance of the instance by propagation from member 1-containers */
- graphCheck(Line) /* checking consistency of the graph (simple and elementary) of an instance */

[3] Class: Pointtype

Properties:
- PointObjId (OID)
- instance-of PntThemClass (object's thematic class)
- Layer (object's layer)
- 0-containerId (OID of its 0-container)

Operations:
- Insert (Point) /* inserts a new instance of the class and enforces constraints */
- Retrieve (Point) /* retrieves an instance of the class */
- Delete (Point) /* deletes an instance of the class and enforces constraints */
- Modify (Point) /* modifies instance of the class and enforces constraints if necessary */
- Relation(P,Q) /* determines topologic relationship between two instances of the class, or between instance P of Areatype or Linetype and instance Q of the class */
- ComputeDistance /* computes euclidean distance between the instance and another instance */

[4] Class: 2-container

Properties:
- 2-containerId (OID)
- RepresentAreatype: set of associated instances of class Areatype (OIDs of member Areatype objects)
- composedofArcs: RetrieveGeometry() (indicates that geometry is retrieved by a function)

Operations:
- CreateContainer (2-container) /* creates a new instance of the class and enforces constraints */
- DestroyContainer (2-container) /* destroys an instance of the class and enforce constraints */
- AmmendContainer (2-container) /* modifies instance of the class and enforces constraints */
- ComputeMetricArea /* computes metric area of an instance */
- RetrieveGeometry (2-container) /* retrieves the arcs and nodes defining the geometry of the instance */
- graphCheck /* consistency check for the connectedness of boundary of an instance */
- RetrieveContainer (2-container) /* retrieves an instance of the class */

[5] Class: 1-container

Properties:

- 1-containerId (OID)
- RepresentLinetype: set of associated instances of class Linetype (OIDs of member Linetype objects)
- composedofArcs: RetrieveGeometry() (geometry represented as a function)

Operations:

- CreateContainer (1-container) /* creates a new instance of the class and enforces constraints */
- DestroyContainer (1-container) /* destroys an instance of the class and enforce constraints */
- AmmendContainer (1-container) /* modifies instance of the class and enforces constraints */
- ComputeLength (1-container) /* computes metric length of the instance as aggregation of lengths of defining arcs*/
- RetrieveGeometry (1-container) /* retrieves the arcs and nodes defining the geometry of an instance */
- graphCheck /* consistency check for connectedness (simple and elementary path) */
- RetrieveContainer (1-container) /* retrieves an instance of the class */

[6] Class: 0-container

Properties:

- 0-containerId (OID)
- RepresentPointtype: set of associated instances of class Pointtype (OIDs of member Pointtype objects)
- isaNode: Nodenr (OID of node for positional information)
- inside2container: 2-containerId (OID of 2-container object in which it lies, or computational procedure to determine this, e.g. point-in-polygon routine)

Operations:

- CreateContainer (0-container) /* creates a new instance of the class and enforces constraints */
- DestroyContainer (0-container) /* destroys an instance of the class and enforce constraints */
- AmmendContainer (0-container) /* modifies instance of the class and enforces constraints */
- RetrieveContainer (0-container) /* retrieves an instance of the class */
- RetrieveGeometry (0-container) /* retrieves geometry (nodenr & coord) of an instance of the class */

[7] Class: Arc

Properties:

- ArcId (OID)
- NodePartofArc1: start nodenr (OID of start node)
- NodePartofArc2: end nodenr (OID of end node)

- Partof2container1: left2-containerId (OID of 2-container on its left)
- Partof2container2: right2-containerId (OID of 2-container on its right)
- PartOf1container: 1-containerId (OID of the 1-container which the instance is part-of)

Operations:

- Insert (Arc) /* inserts a new instance of the class and enforces spatial constraints */
- Retrieve (Arc) /* retrieves an instance of the class */
- Delete (Arc) /* deletes an instance of the class and enforces spatial constraints */
- Modify (Arc) /* modifies instance of the class and enforces constraints */
- Relation(P,Q) /* computationally determines topologic relationship between a new instance of the class and an existing instance of the class or an existing instance of class Node*/
- ComputeLength /* computes metric length of the instance */
- consistencyCheck (Arc) /* consistency check of an instance: no dangling or redundant arc, existence of node, etc. */

[8] Class: Node

Properties:

- Nodenr (OID)
- Xcoordinate (value of x coordinate)
- Ycoordinate (value of y coordinate)
- Zcoordinate (value of z coordinate)
- HasPlanAccuracy: PlanimetricAccuracy /* optional */
- HasHeightAccuracy: HeightAccuarcy /* optional */
- HasPlanLineage: PlanimetricLineage /* optional */
- HasHeightLineage: HeightLineage /* optional */

Operations:

- Insert (Node) /* inserts a new instance of the class and enforces constraints: uniqueness */
- Retrieve (Node) /* retrieves an instance of the class */
- Delete (Node) /* deletes an instance of the class */
- Modify (Node) /* modifies instance of the class */
- Relation(P,Q) /* computationally determines topologic relationship between an existing instance of class Arc or this class and a new instance of the class */
- consistencyCheck (Node) /* consistency check of an instance: no redundant node (single Z per pair of X & Y except at crossings, uniqueness of instance) */

[9] Class: LineCross /* optional, not required for 3D metric space */

Properties:

- Upper1containerId (OID of higher 1-container object)
- Lower1containerId (OID of lower 1-container object)
- Crosspoint: nodenr (OID of node where crossing occurs)

Operations:

- InsertLineCross /* inserts a new instance of the class */
- RetrieveLineCross /* retrieves an instance of the class */
- DeleteLineCross /* deletes an instance of the class */
- ModifyLineCross /* modifies instance of the class */

7.2.3 Implementing the Prototype Object-Oriented Structure

To implement the structure defined above, the classes would translate to abstract data types or modules or classes and the properties of each would be mapped onto the base types of the system to be used (e.g., Xcoordinate as **double** in a c++ based OODB system). If the system does not provide a complete set of base data types to cover all properties, the user can use the system's facility (if available) to provide user-defined types for the remaining properties, otherwise, some programming will be required to define them. The operations, as well as other operators and functions that will manipulate the user-defined types, must also be defined.

The object-oriented implementation can be done in one of two ways (see Figure 1.2): (1) By using an extended relational (or object-relational) system (e.g., Postgres and Iris), often called "evolutionary approach" in which object-oriented features are added to the SQL features of the relational database model (see §2.4). In its simplest form, this is often done by some kind of simulation in which an object-oriented shell is built on top of a relational system, i.e., the SQL itself is not extended to accommodate object-oriented features as described in §2.4, instead, the SQL commands are packaged together as macro-commands and encapsulated within the object definition in the shell; the underlying structure of the database is therefore relational. This "layered approach" has the advantage of using the standard relational query language but the geographic data must submit to relational database constraints e.g., normalization). In the case of truly extended relational systems (e.g., Postgres), the system allows the definition of abstract data types and functionalities. The system itself can also be extended. For example, Postgres has been used to develop a research-oriented GIS prototype GEO++ (Vijlbrief and van Oosterom, 1992). (2) By using a "pure" object-oriented system (e.g., Smallworld and Ontos) or building one from the scratch by programming using an object-oriented programming language, e.g., C++ and Smalltalk; this is popularly referred to as the revolutionary approach. As stated in §2.4, a standard query language comparable to the SQL is not yet available, which makes a true object-oriented implementation difficult at the moment. Moreover, the object-oriented database management systems still have to prove their efficiency over the relational DBMS (Boursier and Faiz, 1993).

Thus in this research, the structure will be tested in the extended relational database system **Postgres** which provides an evolutionary approach (with extended SQL) to object-oriented database implementation. In addition to other facilities, such as support for data of array type, the software supports some object-oriented modelling concepts to improve upon conventional relational DBMS. It supports object identity, multiple inheritance, operator overloading, user-defined types and functions, support of **set** type (acceptance of a set of values as an attribute in a relation), versioning, user-defined rules, etc. (see §8.1.2.2).

7.3 Summary

In this chapter, the conceptual data model for multi-valued vector maps (DMMVM) (see chapter 3) has been translated into two families of database structures, namely (1) relational database structure and (2) object-oriented database structure.

The prototype relational database structure for multi-valued vector maps was designed using Smith's method for relational database design (Smith, 1985). The method comprises four steps, namely: (1) identification of data types and relationships in the application; (2) listing the single-valued and multi-valued dependencies among the data types as dependency statements; (3) translating the dependency statements to dependency diagram; and (4) composing normalised relations from the diagram. Using this method, a total of seven base tables have been developed for multi-valued vector maps. Additional thematic data of the objects can be introduced by creating extra tables for the thematic classes. Some experimental implementations carried out with the prototype indicate its usability for multi-layer spatial data modelling. The relational prototype can then be implemented in a relational system, or in an extended relational system. When used in an extended relational system in which a shell is built on top of a relational database, the relations can be implemented directly as base tables. If the system supports direct object-oriented queries through the extension of SQL, the proposed relations will translate to classes in which the attributes of each relation become the static properties of its corresponding class.

For the object-oriented prototype, two classification lines (thus double inheritance) were distinguished for terrain objects, namely (1) thematic classification line for non-spatial aspects and (2) geometric classification line for geometric aspects of terrain objects. Attention was focused on the latter classification, which yielded a total of nine classes for the object-oriented data structure for multi-valued vector maps. The consistency rules and updating procedure proposed respectively in chapters 5 and 6 become operations (methods) of the classes. Also, the topologic relationships derived in chapter 4 are expected to be dynamically detected in the system by translating the proposed algorithms for detecting topologic relationship (also in chapter 4) into class methods.
An experimental implementation of the object-oriented database structure is described in the next chapter.

8

IMPLEMENTATION

The translation of the DMMVM into two families of database structures was described in the last chapter. This chapter focuses on the implementation of the object-oriented data structure (see §7.2) using an evolutionary object-oriented (extended relational) DBMS. The objectives of the implementation are:
(1) To test the use of the prototype database structure in the establishment of an integrated database using a combination of photogrammetric workstation (for data acquisition) and an object-relational (extended relational) system (for database management).
(2) To verify the algorithms for consistency rules.
(3) To verify the update propagation algorithms.
To meet the stated objectives, the implementation was done in two main phases:
(a) creation of an integrated database based on the object-oriented structure, including data acquisition by photogrammetric means, and verification of consistency rules,
(b) database updating with automatic update propagation and consistency enforcement.
The implementation platform and material resources used are outlined in the next section. The two parts of the experiment are described in §8.2 and §8.3 with an summary in §8.4

8.1 Materials and System Configuration used for the Implementation

8.1.1 Materials

(1) A soil map of Goult, Southern France at scale 1:50000, published in 1978.
(2) Near vertical aerial photographs of 1989 of the same region with a nominal focal length of 152mm and an approximate photo scale of 1:30000. The chosen test area is covered by two models.
(3) Ground control points (X, Y and Z).

8.1.2 System Configuration

The system (hardware and software) configuration for the implementation is shown in Figure 8.1. It consists of two parts: the data acquisition subsystem and the database management subsystem.

8.1.2.1 Data Acquisition Subsystem:

This consists of the following specific hardware and software.

Hardware
(1) Planicomp C120 analytical photogrammetric stereoplotter
(2) Zeiss VideoMap
(3) Calcomp drawing board
(4) Mirror stereoscope
(5) Graphics plotter
(6) Microcomputer with 80386 processor running on DOS 5.2

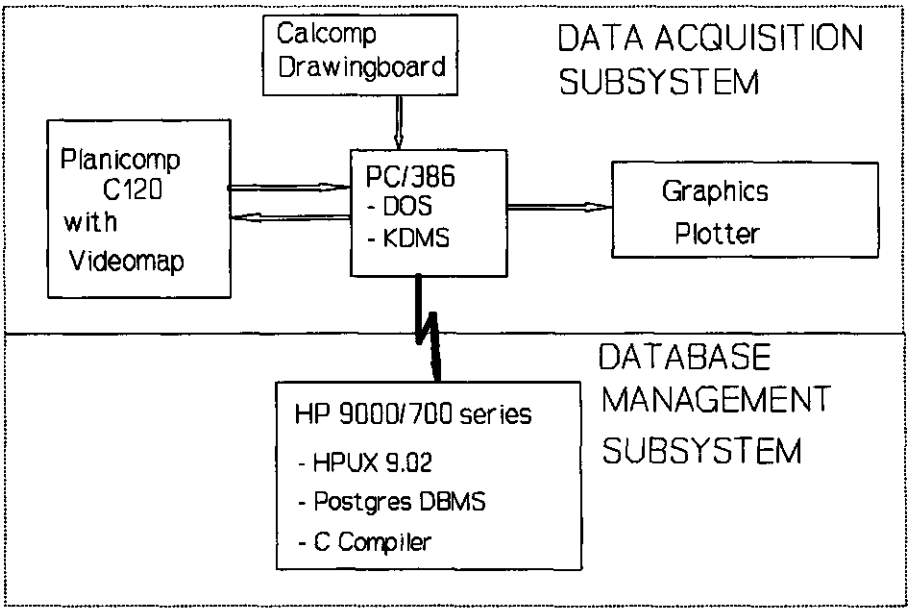


Figure 8.1 System configuration for the implementation

As stated in chapter 1, a stereo-photogrammetric data acquisition method was chosen because it accounts for the most accurate and fastest data collection for high- and medium-resolution spatial databases (akin to large- and medium-scale mapping) and because of the background of the author.

Software
(1) Planicomp C120 orientation (inner, relative and absolute) software
(2) Kork Digital Mapping System (KDMS) version 8.0

8.1.2.2 Database Management Subsystem

Hardware
Hewlett Packard (HP) model 9000 series 700 running on HPUX 9.02

Software
(1) Postgres DBMS version 4.2
(2) C compiler

The Postgres database management system was chosen because it supports object-oriented concepts and rule management; it also has a query language, thus minimizing programming tasks required in the pure object-oriented approach. It is public domain research software, so information about it may be limited. A summary of the essential features of the software is given in the following subsection.

The Postgres Database Management System

Postgres is a database research project from the University of California at Berkeley. It is an evolutionary object-oriented (extended relational) DBMS built on top of the Ingres relational DBMS. Its query language is called POSTQUEL. It runs on the following computers: Digital Equipment Corporation (DEC) computers based on MIPS R2000 and R3000 processors (under Utrix 4.2A and 4.3A), DEC based on Alpha AXP (DECchip 21064) processors (under OSF/1 1.3), Sun Microsystems based on SPARC processors (under SunOS 4.1.3), Hewlett-Packard (HP) model 9000 series 700 and 800 based on PA-RISC processors (under HPUX 9.0x ($x \geq 0$)), and International Business Machines (IBM) RS/6000 based on POWER processors (under AIX 3.2.5). Postgres has also been ported by users to many other architectures and operating systems, including NeXTSTEP, IRIX, Solaris 2.2, Linux and NetBSD.

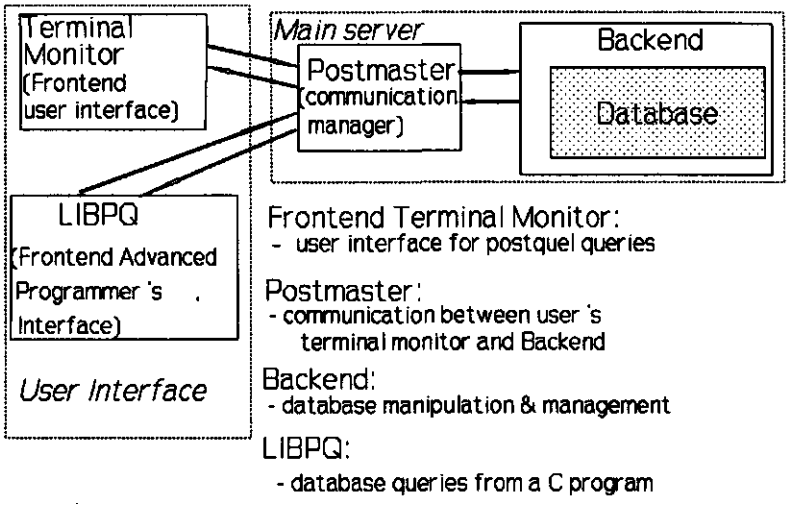


Figure 8.2 Postgres DBMS architecture

The system architecture of Postgres is illustrated with Figure 8.2. The architecture comprises three main parts:

(a) *The Postgres Backend* which runs on the main server where Postgres and the databases reside. It manages and manipulates the database. One backend can host one or more databases.

(b) *The Postmaster* which serves as the communication link between the front (user) end and the backend. Only one postmaster should run on one backend irrespective of the number of databases on that backend.

(c) *The Frontend Monitor and LIBPQ*: These are the user interfaces. The monitor is used for interactive queries with Postgres using the Postquel query language. The LIBPQ is the

interface for a user who wishes to interact with Postgres through C (or C++) programming language; Postquel commands can be directly executed in the user's program using this interface. One or more frontends (running on the same or different servers) can communicate with the same backend (through a common postmaster).

Postgres Data Model:

Postgres' data model is based on the relational database model but extended with two main capabilities (Rowe and Stonebraker, 1987; Postgres Group, 1994):
- object management capabilities
- rule management capabilities

Object management capabilities:

Four basic object-oriented constructs were added on top of the conventional relational constructs:

(a) Classes:

The fundamental notion of Postgres is that of a class. A class is a named collection of objects. Each object, or instance, of a class has the same collection of named attributes, and each attribute is of a specific type (base type or instances of other classes). Each instance has a unique system-defined object identifier (oid). Classes can inherit data and functions from one or more other classes.

(b) Inheritance

In Postgres, a class can inherit from zero or more other classes (multiple inheritance) and a query can reference either all instances of a class or all instances of a class plus all of its descendants. The inheritance hierarchy is a directed acyclic graph. Functions defined for the parent class are also inherited. If an inheritance conflict occurs (when the same attribute name is inherited from more than one parent), the inheritance of *that* attribute is disallowed. When a function (or procedure) inheritance conflict arises, the system uses the function defined for the first parent of the class in its inheritance precedence list (IPL). IPL is a list of all the parents of a class constructed by the system.

(c) Types

The data types supported by Postgres are classified into three groups:
- Base types which include int2, int4, float4, float8, bool, char, abstime, reltime, date, and postquel.
- Abstract data types (ADT): These are user-defined arbitrary base types. They can be defined by specifying the type name, the length of the internal representation in bytes, procedures (functions) for converting from an external to internal representation for a value and from an internal to external representation. The procedures are coded in a conventional programming language, such as C, and defined to the system using the **define procedure** command. The user must also define the **operators** on ADTs (see functions).
- Constructed types: These are the user-defined classes (classes are treated as data types also).

(d) Functions

Functions are classified into three groups in Postgres:
- Normal functions and procedures (programming language functions): Users can define an arbitrary collection of normal functions in a conventional programming language (e.g., C), apart from the predefined ones. Operands of a function can be any of the three Postgres types (i.e., base types, ADT's or classes).
- Operators: Users can define new unary or binary operators which operate on any Postgres type, especially ADT. The creator of the operator becomes its owner. Operator overloading is supported.
- Postquel functions (query language functions): A collection of commands in the POSTQUEL query language. A postquel function returns one or more instances of a class or one or more base types; thus Postquel functions are automatically constructed types (classes). They are useful for constructing composite types.

Rule management capabilities:

Postgres supports two rule systems:
- Instance-level rule system which uses "markers" placed in each instance in a class to "trigger" rules. It is more efficient if there are many rules on a single class, each covering a small subset of instances. This is the default system.
- Query rewrite rule system which modifies queries to take rules into consideration. It is more efficient when rules affect most of the instances in a class.

8.2 Creation of an Integrated Database using the DMMVM

A multi-valued spatial database, integrating a soil database (showing soil units and soil sample points) and a topographic database (showing major land uses and land cover types) was created based on the prototype object-oriented database structure described in §7.2. This aspect of the implementation consists of the following main phases:
- The translation of the proposed database structure to the Postgres DBMS's data model.
- The data acquisition phase using a photogrammetric workstation.
- The creation and instantiation of classes using Postgres.
- Checking data consistency.
- Database query

The second, third and fourth phases are illustrated with Figure 8.5

8.2.1 Translation of the Object-Oriented Database Structure to Postgres Data Model

For this implementation, the object-oriented database structure presented in §7.2 was mapped into the Postgres data model. The fact that the Postgres data model is based on the (extension of) relational model played a significant role in the translation process because some properties of a class can be derived through the relational join of two or more classes. Thus the 2-CONTAINER class was represented as the "twoC_id" property of the **area** (Areatype) class; the 1-CONTAINER class was represented as the "oneC_id" property of the **line** (Linetype) class and the 0-CONTAINER class was represented as the "zeroC_id" property of the **pointf** class (named pointf to avoid a conflict with the Postgres base type **point**). Three

new classes were created: **pointnode, accuracy** and **lineage**. The pointnode class was created to explicitly represent the POINTINAREA (see Figure 7.4) association between 0- and 2-containers and to store the node identifier of the 0-containers. The **accuracy** and **lineage** classes were created from the **node** class to reduce data redundancy because most of the nodes will have the same accuracy and lineage, having been digitized with the same instrument and by the same human operator.

The mapping produced the nine classes shown in Table 8.1. The properties of each class were mapped into base data types in Postgres and class methods were defined as Postquel or C functions (see Appendix 1.2).

Table 8.1 Creation of classes in Postgres

Class Name	Properties	Base type	Description of Property
area	twoC_id	int4	identifier of the area object's 2-container (nonnull)
	aobjid	int4	identifier of the area object (nonnull)
	layer	char16	name of the map layer of the object (nonnull)
	aname	char16	geographic name of the object
	aclass	text	the thematic attribute class of the object (nonnull)
line	oneC_id	int4	identifier of the line object's 1-container (nonnull)
	lobjid	int4	identifier of the line object (nonnull)
	layer	char16	name of the map layer of the object (nonnull)
	lname	char16	geographic name of the object
	lclass	text	thematic attribute class of the object (nonnull)
pointf	zeroC_id	int4	identifier of the point object's 0-container (nonnull)
	pobjid	int4	identifier of the point object (nonnull)
	layer	char16	name of the map layer of the object (nonnull)
	pname	char16	geographic name of the object
	pclass	text	thematic attribute class of the object (nonnull)
pointnode	zeroC	int4	identifier of each (unique) 0-container in pointf class (nonnull)
	twoC	int4	identifier of the 2-container that topologically contains the 0-container (value = 0 if none)
	pnode	int4	identifier of the 0-container's node (nonnull)
arc	arc_id	int4	identifier of an instance (nonnull)
	snode	int4	start node of the instance (nonnull)
	enode	int4	end node of the instance (nonnull)
	lftTwoC	int4	identifier of 2-container on its left (nonnull)
	rgtTwoC	int4	identifier of 2-container on its right (nonnull)
	aOneC	int4	identifier of 1-container which the arc is part of (null or 0 if it represents only boundary of area)
node	node_id	int4	identifier of a node (nonnull)
	xcoord	float8	x coordinate (nonnull)

	ycoord	float8	y coordinate (nonnull)
	zcoord	float8	z coordinate (nonnull)
	ac_id	int4	identifier of the accuracy of the object
	lin_id	int4	identifier of the object's lineage
linecross	upper1C	int4	identifier of upper 1-container (nonnull)
	lower1C	int4	identifier of lower 1-container (nonnull)
	crosspt	int4	node id of crossing point (nonnull)
	lower_H	float8	z coordinate of lower 1-container (interpolated from heights of the two adjacent nodes of the 1-container)
accuracy	ac_id	int4	identifier of the instance
	pl_acc	float4	value of planimetric accuracy
	ht_acc	float4	value of height accuracy
lineage	lin_id	int4	identifier of the instance
	pl_lin	text	lineage of planimetric coordinates
	ht_lin	text	lineage of height coordinates

Some of the class operations can be handled by simple queries (e.g., insertion and retrieval of instances of a class) using the Postquel query language without necessarily defining them as functions. However, the operations that are constantly required and the complex ones (e.g., geometric consistency enforcement) were defined as functions (see the list and descriptions of the functions in Appendix 1.2). The functions include those required for monitoring and enforcing data consistency and those required for update propagation; they were based on the algorithms presented in chapters 5 and 6.

8.2.2 Data Acquisition

Data acquisition for a multi-valued database can be made in one of two ways, as shown in Figure 8.3. The first procedure is to digitize each map layer separately, in the format of the mapping software, and then perform map overlay of all the layers. The output of the map overlay can then be structured in the format of the DMMVM (see Chhatkuli, 1993 for a detailed example using Arc/Info). This approach requires less preparation compared with the second method and is easier to implement in any existing system. However, it will require more editing (e.g., removal of spurious polygons and sliver lines) and consistency checks (e.g., checking for compatibility among objects that are spatially coincident; a lake and a football field may not overlap for instance) during and after the overlay operation. Moreover, if the mapping software has no

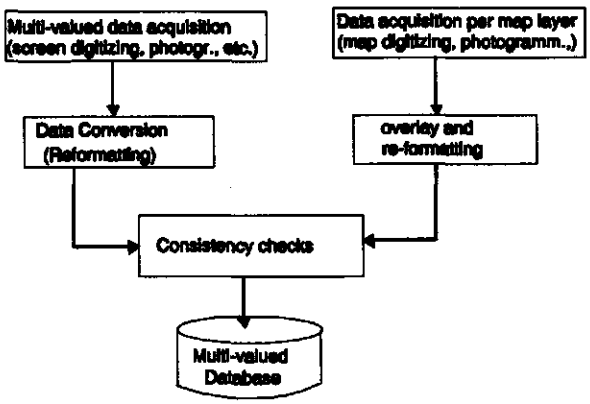


Figure 8.3 General procedure for multi-valued data collection

map overlay facility, then an extra third-party software (acquired or developed by the user) is required, thereby increasing the overhead cost.

In the second approach, data acquisition is made in multi-valued mode. This requires more extensive planning than the separate digitizing approach. A compilation guide has to be prepared during the planning phase, incorporating all the map layers including the annotation and identification (coding) of all m-containers ($m \in \{0,1,2\}$). The compilation of the multi-valued map can then be done by cartographic digitizing, screen (head-up) digitizing (if all boundary lines of objects can be identified) and stereo-photogrammetry, using "optical overlay" by superimposition if necessary. This method minimizes the editing task because problems of sliver lines and spurious polygons are dealt with during data collection. In addition, no extra overlay software is required since map overlay is not involved. On the basis of our experience during the data collection phase, it is believed that this method will become more labour-intensive and less cost-effective as the number of map layers increases, especially in a large project because of the extensive preparation work required.

The choice of approach will therefore depend on the size of project area, number of layers to be integrated and the system configuration (hardware and software) that is available for data acquisition. For a large mapping project or many map layers or both, it appears that the first approach will be more feasible; the structural and semantic consistency of the overlay result can then be checked as described in chapter 5. Conversely, for a limited number of map layers and not too large project area, especially in the absence of a system with map overlay capability, the second approach can be used. Obviously, there is still the need to properly investigate the cost-benefits of the two approaches to guide in the selection of the appropriate one.

In database updating, the data can be collected per layer and the objects input into the database using the updating procedures proposed in chapter 6.

The second approach was chosen for this implementation mainly to test the approach (the first being the common practice though the overlay result is usually not structured).
The data acquisition phases for this implementation are summarised in the following sections. More details can be found in the work of Essayah (1994).

8.2.2.1 Preparation

Two map layers - a topographic layer showing major land use and land cover types to be derived from aerial photographs and a soil layer to be derived from a soil map - were selected for this experimental implementation. A test area of approximately 5 km by 5 km, covered by two stereo models, was chosen in the Goult region of southern France on the basis of the availability of raw data (photographs with ground control points and a soil map). The following tasks were performed in preparation for data collection:

- The table digitizer (Calcomp drawing board) to be used for initial digitizing of the soil map and the stereoplotter (Planicomp C120 with Zeiss Videomap superimposition system) were coupled for the actual 3D compilation of both map layers. The Kork-KDMS was already installed on a PC and linked with the Planicomp.

- Checking of the Planicomp C120 using the system's calibration program.

- Preparation of the photographic materials and control data of the chosen area, including marking the control and tie points on the photographs.

- Preparation of the compilation guide. This includes the selection, classification and annotation of terrain objects of interest (to form the topographic layer) on the photographs using mirror stereoscope and the manual overlay of the two layers on a transparency. The m-containers were then identified on the transparency and used as a guide during the stereo-compilation.

- Formulation of a coding system to facilitate collection of explicit topologic information during stereo-compilation. The data structure of the Kork system (see Kork, 1992) was designed mainly for computer-assisted mapping without adequate support for topologic data acquisition. Map details are collected as (aggregates of) strings (a string is a chain of straight-line segments used for the representation of polygon boundaries and line objects), symbols (for point objects and map symbols) and annotation (for place names, etc). Each string represents a single object in Kork and is assigned the code of the object. Since we are working in 2D topologic space, each string does have a polygon on the left and a polygon on the right and may represent a linear object.

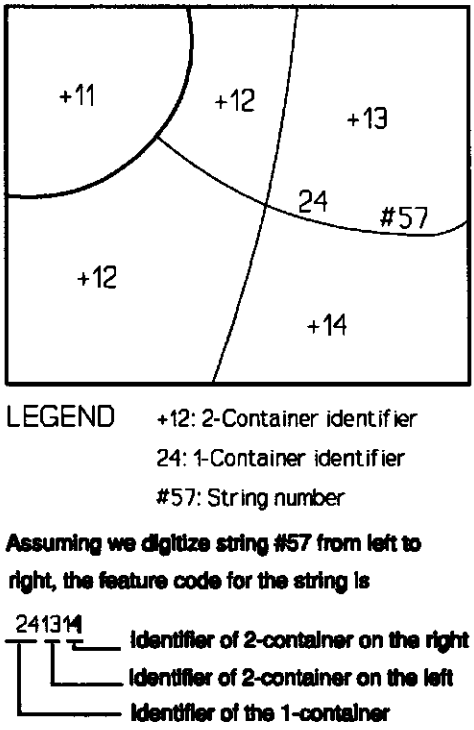


Figure 8.4 Example of feature coding during digitizing, for automatic topology building.

Thus a different coding scheme was devised for the strings in order to digitize the strings in an approximate topologic format. This coding format (see Figure 8.4) was derived from the combination of Kork's colour code and feature code for strings (see Table 8.2 for a sample output in Kork's format) as follows. A Kork manuscript contains 12 fields per record (Kork, 1992) where field 1 represents the nature of the point (whether part of string, STR, or single point, SYM, or annotation, ANN), field 2 represents the string number, field 3 represents the sequential number of the points in a string, field 4 the total number of points in the string, field 5 the X coordinate, field 6 the Y coordinate, field 7 the Z coordinate, field 8 the rotation angle of symbols and annotations, field 9 the pen status (0 for pen up and 1 for pen down), field 10 the colour code, field 11 the feature code of the terrain object represented by the string (negative value means permanent feature and positive means temporary feature), and field 12 represents the line code. Kork allocates one digit (U $\in$ {1,2,..,9}) for the colour code and a 2-byte integer (with five digits VXXYY) for the

feature code. By concatenating the two codes, a six-digit code (UVXXYY) can be derived in which the first (left justified) two digits (UV) are used as an integral colour code and 1-container identifier, i.e., the colour code is synchronised with the 1-container identifier. By facing the direction in which the string is being digitized, the next two digits (XX) contain the identifier of the 2-container on the left side of the string while the last two digits (YY) contain the identifier of the 2-container on the right side of the string. Each string will therefore be digitized only once. For example, string number 510 (having nine points) in Table 8.2 has colour code 6 and (permanent) feature code -10735 which, in our coding method, translates to code 610735, meaning that the string is part of the 1-container with identifier 61, has 2-container with identifier 7 on the left and 2-container id 35 on the right.

This coding scheme, due to the limitation of the Kork system (version 8.0), constrains the maximum number of 2-containers to 99 and the maximum number of 1-containers to 27. This was more than adequate in our implementation but another coding device must be designed for a very large project area with high density of different terrain objects. This constraint can be overcome by using one of the following feature coding methods:

(a) By using digitizing software that supports 4-byte integer for feature codes, thereby increasing substantially the number of 1- and 2-container identifiers that can be coded. In the Kork system, this can be done by changing the computer length (from 2-byte to 4-byte integer) of the feature code in the source program (in collaboration with Kork).

(b) After preparing the compilation guide, all the m-containers, $m \in \{0,1,2\}$ are assigned unique identifiers. Then, a look-up table (LUT) of feature codes is defined. The LUT will consist of four fields per record, each field being a 4-byte integer long. Each record will then contain the topologic information of each string as follows (the order can be rearranged):

. field #1: the string number (this should tally with the feature code assigned to the string during digitizing and serves to relate the LUT strings with the coordinates of the strings in the system) and must be non-null if other fields have values,
. field #2: the identifier of the 2-container on the left side of the string (non-null),
. field #3: the identifier of the 2-container on the right side of the string (non-null),
. field #4: the identifier of the 1-container which the string is part of. If the string does not represent a 1-container, then a value 0 is recorded.

This LUT can be filled up in one of two ways:

(i) During digitizing, the operator enters information about each string as a record of the LUT. As he digitizes the string, the same feature code entered in the system for the string is also recorded as the first field of the string's record in the LUT, and the 1- and 2-container identifiers are also recorded in their respective fields as indicated above. If the system permits intervention and temporary transfer of control, this recording can be done on the system, otherwise it has to be done on paper. The actual assignment of the 1- and 2-containers to the strings will then be done during the data conversion phase using a computer program. This method requires less effort during the preparation phase (strings do not have to be pre-coded) but has the disadvantage of slowing down the operator.

(ii) The strings are also coded and annotated on the compilation guide like the m-containers. The LUT can then be filled up for all strings before the digitizing process. During digitizing, the operator will then enter the corresponding string number on the LUT as the feature code

of the string being digitized. As in the first method, the LUT will be related to the strings' coordinate information during data conversion and formatting. This method requires less effort by the operator during digitizing but more work is involved in the preparation phase. Coding errors and string omission can also be more easily checked in this approach.

8.2.2.2 Data Collection

The major data collection operations consist of the tasks shown in the upper part (above the dashed line) of Figure 8.5. and are outlined in the following:

- The soil map of the chosen test area was initially digitized in "spaghetti" format on a Calcomp drawing board using the Track module of the Kork system. Boundary lines of soil units (as area objects) and sample points (as point objects) were digitized with approximate height (average elevation of the area) to reduce shift between the digitized map layer and the stereo model of the topographic layer during superimposition. Collection of all line objects appearing on the soil map was deferred till the stereo compilation of the topographic layer since they occur on the photographs which are more recent than the soil map.

- The second map layer, the topographic map showing major land uses and land cover types, was derived from two stereomodels of the same area covered by the soil map layer. The first model was set up and oriented for digitizing in multi-valued mode on the Planicomp C120 using the 3D Track module of the KDMS. Through the videomap, the digitized soil map was superimposed in another colour on the stereomodel, thus facilitating a combined (multi-valued) stereocompilation of the two map layers. Every closed polygon on the stereomodel becomes (part of) a 2-container representing a certain soil unit in the soil map layer and a certain area object in the topographic map layer. Likewise, line and point objects are mapped into component 1-containers and 0-containers, respectively. Using the compilation guide prepared during the preparation phase and the devised feature coding system, the 1-containers and boundaries of the 2-containers were digitized as strings while the 0-containers were digitized as symbols. The identifier (code) of the 2-container topologically containing the 0-container being digitized was assigned as a symbol code (user defined), while the symbol number (system generated) served as the identifier of the 0-container. Snapping functions are provided by the software to snap onto an already digitized node.

A check plot of the manuscript was then made followed by editing before changing the model. The editing phase took care of some consistency checks which are within the capabilities of Kork. These include undershoot and overshoot errors. The second model was digitized using the same procedure. The two models were then combined with the Merge command of Kork. The hard copies of the two map layers and the multi-valued map are shown in Figures 8.6 (soil map layer), 8.7 (topo map layer) and 8.8 (the multi-valued map). The maps were plotted by the Plothp program of Kork. The multi-valued digital map was then converted to an ASCII file in the Kork output format (.LST) using the PTLIST command. Table 8.2 shows a sample of the output.

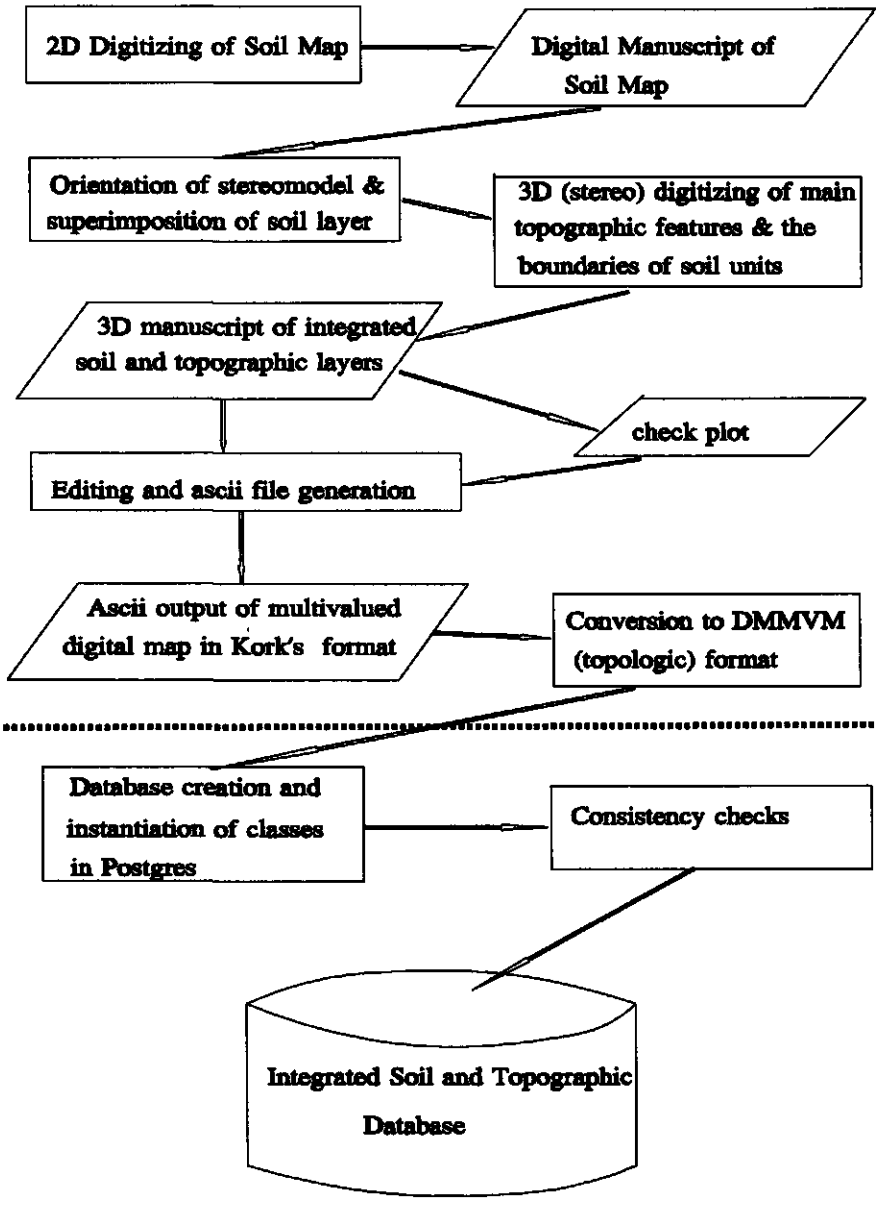


Figure 8.5 Procedure for creating the integrated topographic and soil database

Table 8.2 Sample Ascii output of the digital manuscript in Kork's format

```
! Coordinate output for file: MVVMGL
!
!
!          Kork Digital Mapping System Data File
!
!          File Name: MVVMGL
!    Manuscript Name:
!           Operator: KORKTRAN
!               Date: 29-AUG-94
!     Project Number:
!     Project   Name: Goult
!     Model   Number: 62/64/66
!
! Number of centroids is       16     Next unused STR record is   1266
! Number of strings is        389
!
!     X coordinate range is from     832025.6 to     837662.9
!     Y coordinate range is from     173725.7 to     179669.8
!
STR  504   2   11   836468.778   177764.000   160.89          1 6-10707   600
STR  504   3   11   836185.333   177715.333   166.89          1 6-10707   600
STR  504   4   11   836112.889   177702.333   167.44          1 6-10707   600
STR  504   5   11   836066.444   177689.222   168.00          1 6-10707   600
STR  504   6   11   835952.333   177649.667   168.00          1 6-10707   600
STR  504   7   11   835849.667   177612.889   162.89          1 6-10707   600
STR  504   8   11   835664.111   177547.333   161.00          1 6-10707   600
STR  504   9   11   835566.778   177508.778   163.11          1 6-10707   600
STR  504  10   11   835511.444   177474.889   162.00          1 6-10707   600
STR  504  11   11   835463.444   177438.333   162.56          1 6-10707   600
STR  508   1    2   835463.444   177438.333   162.56          0 6-10707   600
STR  508   2    2   835403.889   177386.444   161.89          1 6-10707   600
STR  510   1    9   835403.889   177386.444   161.89          0 6-10735   600
STR  510   2    9   835365.667   177359.556   161.67          1 6-10735   600
STR  510   3    9   835304.667   177329.889   160.56          1 6-10735   600
STR  510   4    9   835164.111   177272.333   156.56          1 6-10735   600
STR  510   5    9   835025.667   177217.556   154.56          1 6-10735   600
STR  510   6    9   834945.778   177184.333   152.89          1 6-10735   600
STR  510   7    9   834908.444   177164.444   152.67          1 6-10735   600
STR  510   8    9   834886.333   177152.778   150.67          1 6-10735   600
STR  510   9    9   834877.111   177142.889   150.67          1 6-10735   600
STR  513   1    3   835388.333   177529.111   167.11          0 5-10735   500
STR  513   2    3   835471.444   177474.778   162.11          1 5-10735   500
STR  513   3    3   835463.444   177438.333   162.56          1 5-10735   500
STR  515   1    3   835463.444   177438.333   162.56          0 7-13535   700
STR  515   2    3   835408.222   177462.111   162.78          1 7-13535   700
STR  515   3    3   835344.444   177491.778   173.44          1 7-13535   700
  ↓    ↓    ↓    ↓       ↓            ↓           ↓       ↓    ↓ ↓  ↓      ↓
  1    2    3    4       5            6           7       8    9 10 11     12  ← fields
```

8.2.3 Creation of the Integrated Database

This phase involved the instantiation of the database and consistency checks. Before instantiating the classes, the ASCII output of the Kork manuscript was converted to the format of the DMMVM.

8.2.3.1 Conversion of the Kork output to DMMVM format

With the aid of the devised feature coding method, the output of the data acquisition phase contains the necessary topologic information, with each string carrying information about the 2-containers on each side and the 1-container which it is part of (this value will be 0 if the string is not part of any 1-container). Although the snapping function was used during digitizing, the software still recorded the coordinates of the common point for each string in which the point occurs. On the other hand, the DMMVM format treats each line segment as an arc (thus all points as nodes), and coordinates of a node, irrespective of the degree of the node, must be recorded once. A conversion program was therefore developed to map the Kork output into a format that is compatible with the DMMVM such that each point became a node, duplicate nodes were eliminated and each line segment became an arc, each arc having

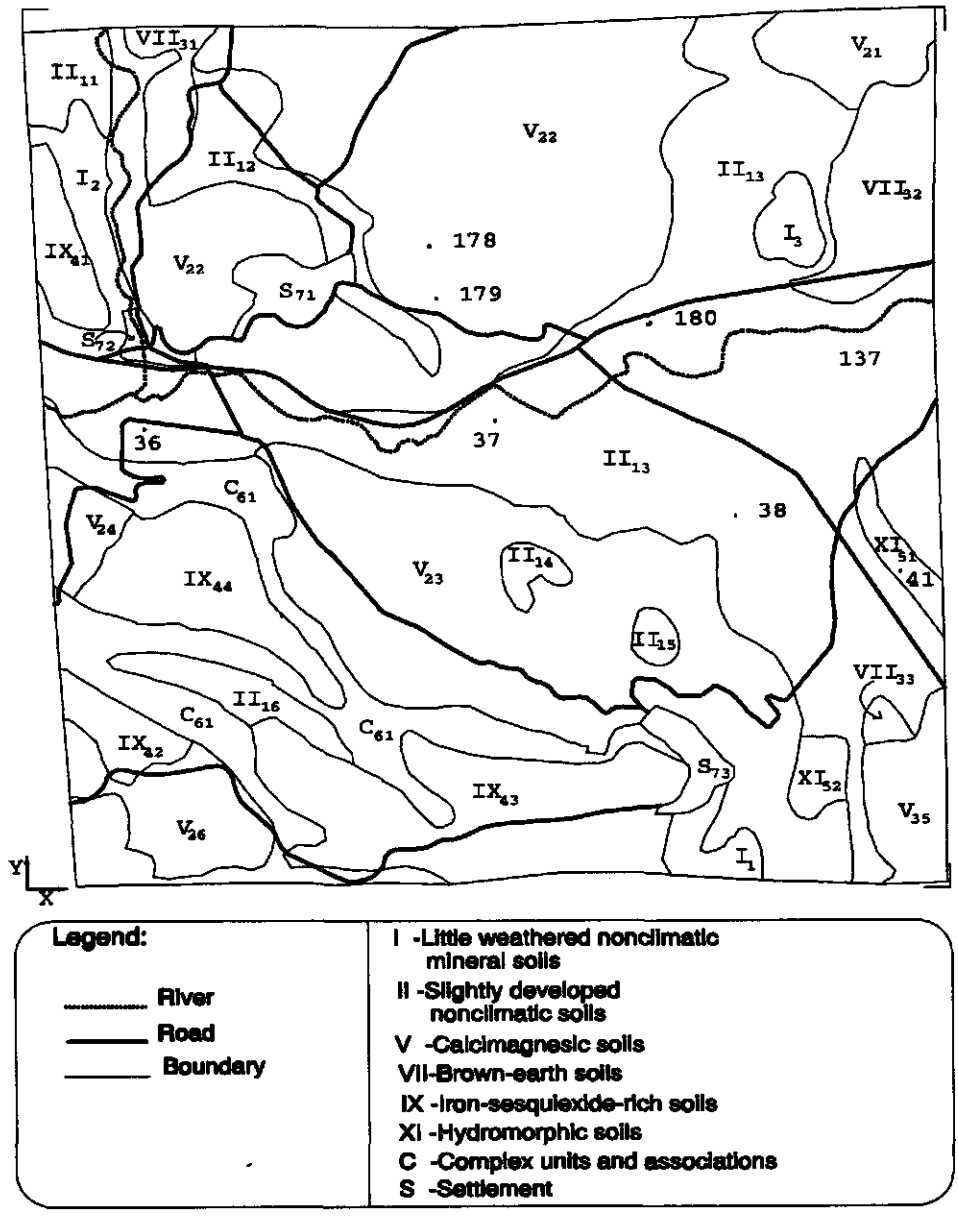


Figure 8.6 The soil map of Goult

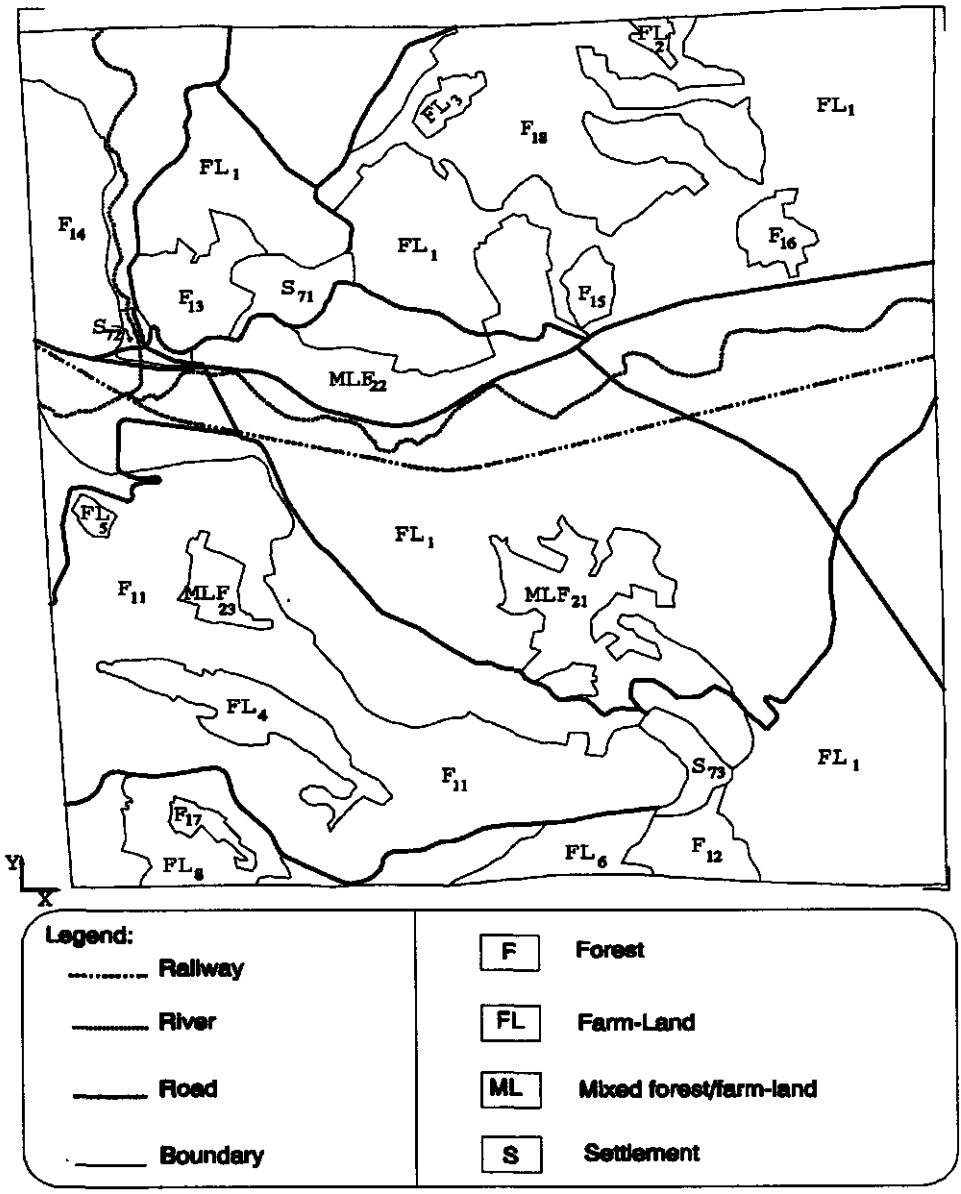


Figure 8.7 Topographic map of Goult showing major features

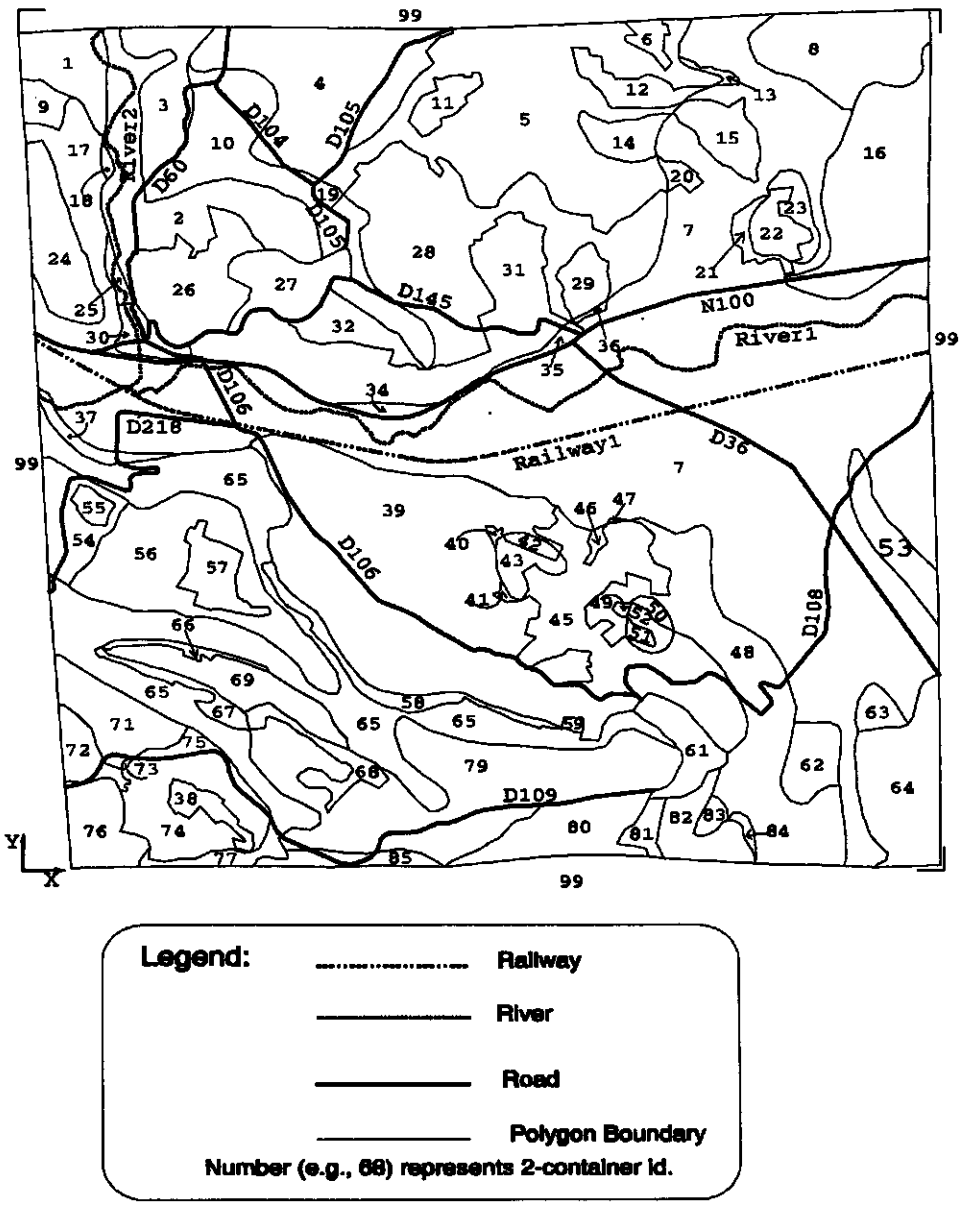


Figure 8.8 Multi-valued vector map of Goult showing soil units and land use/ land cover types

a unique arc identifier, a start and an end node, the identifiers of the 2-containers on its sides, and the identifier of the 1-container which the arc is part of. By ensuring uniqueness of each arc and each node during the conversion process and converting each line segment to an arc, the four geometric primitives constraints (GP_Rule_i, i = 1,4, see §5.1.1) were automatically enforced.

8.2.3.2 Database Creation

After creating a database (called Goult) in Postgres using the **createdb** command, the nine classes (see Table 8.1) representing the multi-valued vector map were created using Postquel's **create** command. The output of the format conversion program described above was then copied into the arc and node classes using the Postquel **copy** command. The other classes were also instantiated. Some instances of the nine classes are shown in Appendix 2.

8.2.4 Consistency Checks

Although a lot of care was taken during the data acquisition phase to ensure geometric consistency and some data consistency rules were enforced during the editing and format conversion phases, it is still essential to check the structural consistency of the database after its creation because of the limitations of the data acquisition software (e.g., lack of provision for node snapping in 3D) and because some inconsistencies may have been introduced during data conversion and, indeed, some may escape detection during the editing phase. This will also verify the effectiveness of the consistency rules. Thus the following consistency rules were monitored and enforced after creating the database:

(a) *Consistency rules for the geometric primitives.* Postquel functions checkArcLoop(), checkRedArc() and checkDupArc() (see Appendix 1.2) were used to monitor the consistency of the geometric primitives. No violation was detected at this level because line segments were automatically treated as arcs, all points as nodes and redundant arcs and nodes were also eliminated during the data conversion.

(b) *Consistency rules for the m-containers.* The consistency rule for 0-container was checked by simple query. The consistency rules for 1-containers (1CC_Rule_1) and 2-containers (2CC_Rule_1) were monitored using the function mCgraph() (see Appendix 1.2). The function checks the degree of each node of a subgraph (the input subgraph can be the geometry of a line object, area object, 1-container or 2-container) and reports the node of the subgraph that has an incorrect degree.

The function was used to check the rule 1CC_Rule_1 for each of the 1-containers in the database. It found a violation of the rule for one 1-container (with oneC_id 91) in which there was a vertical gap (two adjacent interior nodes have the same x and y coordinates but different z values). The violation arose because the height tolerance value set in the program for node snapping in height was too small compared with the measuring precision of the (inexperienced) human operator (the data acquisition software (KDMS)'s snapping function works only in 2D). The mean height of the two nodes was then taken as the most probable value and one of the two nodes was deleted while redefining the node identifiers of the two affected arcs.

The function mCgraph() was then used to check the 2CC_Rule_1 for all the 2-containers in the database. There was no violation of the constraint. However, to ascertain the effectiveness of the function (and thus the algorithm), one arc was manually removed to create a gap in the boundary polygons of two adjacent 2-containers and the function activated again. This time the two 2-containers were returned as violating the structural constraint for 2-containers. Because of the absence of an on-line graphic facility in Postgres as installed, it was possible to retrieve the arc(s) and node(s) in error only in alphanumeric form for examination and editing.

(c) *Consistency rules for the elementary objects.* The same function mCgraph() was used to check the graph consistency of individual line and area objects in the database because an m-container is topologically isomorphic to an n-dimensional object where m = n. Note, however, that the input geometric data into the functions will be different except when, for instance, a 1-container represents only one line object. The topology of each elementary object was consistent most probably because consistency has been enforced at the lower levels (m-containers and geometric primitives).

(d) *Euler constant for planar maps.* The algorithm for checking the planar enforcement of the database (see §5.1.3) was translated to a Postquel function. As indicated in §5.1.3, the return value of the algorithm should equal the number of component graphs if the map is consistent. When the function was administered on the Goult database, we had:

v = 1988, e = 2063, f = 80

thus E = 1988 - 2063 + 80 - 1 = 4

Relating the value to the hardcopy of the database in Figure 8.8, it is true that there are four component graphs in the map because 2-containers 11, 55 and 57 are disconnected from the others (none of the nodes defining each of the three subgraphs can be reached from any other nodes except from those defining the subgraph). Thus they form three additional component graphs (the fourth is the main map minus the three). The result proved the planar enforcement of the map and the effectiveness of the algorithm.

To graphically compare the database after the consistency monitoring operations with the output of the Kork software (before database structuring), a perspective view of the database was produced (see Figure 8.9). The isolated point objects are not shown because the visualisation program was designed only for connected lines. A comparison of this map with Figure 8.8 further confirms the consistency of the database. Figures 8.9, 8.10, 8.11, 8.12 and 8.13 were produced mainly by the PC-based 3D visualization software (Pilouk 1993) and Grasp version 4.0 (Bridges 1991).

To maintain data integrity during subsequent updating of the database, the rules listed in Table 8.3 were defined to guide against invalid insertion to and deletion from the database classes. Two examples are given below.

(1) /* if new node coincides with existing node within the defined tolerance, the system rejects the insertion */

```
define rule node_rule_2 is
on append to node
where float8abs(new.xcoord - node.xcoord) <= 0.02 and
```

```
float8abs(new.ycoord - node.ycoord) <= 0.02 and
float8abs(new.zcoord - node.zcoord) <= 0.03
do instead nothing \g

(2) /* for any arc, arc_id, snode, enode, lftTwoC, rgtTwoC must be nonnull nor zero */

define rule arc_rule_3 is
on append to arc
where new.arc_id ISNULL or new.snode ISNULL or new.enode ISNULL or new.lftTwoC
ISNULL or new.rgtTwoC ISNULL or new.arc_id = 0 or new.snode = 0 or new.enode = 0 or
new.lftTwoC = 0 or new.rgtTwoC = 0
do instead nothing \g
```

Table 8.3 Data integrity rules defined in Postquel

Name of Rule	Target Class	Description of Rule
area_rule_1	area	Disallows values of the properties twoC_id, aobjid, layer and aclass of the Area class from being NULL or zero
area_rule_2	area	Enforces that a new instance of Area class is not assigned the identity of an existing instance of that class
line_rule_1	line	Ensures that values of the properties oneC_id, lobjid, layer and lcass of the Line class are not NULL or zero
line_rule_2	line	Ensures that a new instance of the Line class is not assigned the identity of an existing instance of that class
point_rule_1	point	Ensures that values of the properties zeroC_id, pobjid, layer and pclass of the Pointf class are not NULL or zero, and that the new instance is not assigned the identity of an existing instance of the same class
point_rule_2	point	Ensures that values of the properties zeroC and pnode of the Pointnode class are not NULL or zero and that the new instance is not assigned the identity of an existing instance of the class; in addition, the new instance must exist in the po-intf class
lcross_rule_1	linecross	Enforces the condition that the values of the properties upper1C, lower1C and crosspt of the Linecross class are not NULL or zero; and that the upper1C and lower1C must exist in the Line class

node_rule_1	node	Enforces that the property node_id of a new instance of the Node class is non-null or zero
node_rule_2	node	If new instance of the Node class spatially coincides with an existing instance of the class within defined tolerance, the new node object is not inserted
node_rule_3	node	If a new Node object is mistakenly assigned an existing node_id property value, the insertion should be rejected
node_rule_4	node	Enforces that a shared Node object cannot be deleted until it is no longer required
arc_rule_1	arc	Ensures that if a new Arc object has equal values for the lftTwoC and rgtTwoC properties, then the value of the property aOneC should not be 0
arc_rule_2	arc	Enforces that the values of the properties snode and enode of a new Arc object are not equal (the arc forms a loop if they do)
arc_rule_3	arc	For any Arc object, values of the properties arc_id, snode, enode, lftTwoC, rgtTwoC must be nonnull nor zero
arc_rule_4	arc	If a new Arc object is mistakenly assigned the arc_id value of an existing instance, the new arc should be rejected
arc_rule_5	arc	Ensures that an Arc object is deleted only when the values of the properties lftTwoC = rgtTwoC and aOneC = 0

These rules are used in combination with the consistency rules for geometric primitives (GP_Rule_1 to GP_Rule_12, see Table 5.1) which are implemented in C programming language (see Appendix 1.2).

8.2.5 Query Example

Having ensured data consistency, we can then query the database. An example of a topologic query involving the two map layers which would have necessitated map overlay during the query is given here. The query was:

> *Select all farmlands that have calcimagnesic soil type*

This query requests all area objects belonging to area class "farm_land" (from the topo layer) and have instances of area class "calcimagnesic_soils" (from the soil layer). The geographic names of the farm_land objects and the soil units are required in addition to a graphic output of the objects. The alphanumeric result of the Postquel query is shown in Table 8.4 while Figure 8.10 shows the selected objects.

8.3 Examples of Database Updating Operations

Table 8.4 All farmlands that have calcimagnesic soil type

twoC_id	Farmname	Soilname
2	FL1	V22
4	FL1	V22
6	FL2	V22
8	FL1	V21
11	FL3	V22
12	FL1	V22
14	FL1	V22
28	FL1	V22
32	FL1	V22
39	FL1	V23
46	FL1	V23
55	FL5	V24
64	FL1	V25
73	FL7	V26
74	FL7	V26

To verify the updating algorithms presented in chapter 6, four examples of database updating operations were performed on the created database, as described below. In line with the proposed procedures for automated database updating (see chapter 6) while enforcing geometric consistency (see chapter 5), the algorithms for inserting a new geometric primitive (node or arc) were translated into C functions (see Appendix 1.2). The arc insertion function (Insarc()) calls other functions which computationally detect the existing topologic relationship between a new arc and an existing primitive (arc or node). The return value of the function activates the function that enforces consistency. For example, if Insarc() detects relationship r287(new arc, old arc) (i.e., meet) the function Alternode() will be activated to assign correct values for the properties (left and right 2-containers, 1-container id and the start and end nodes) of the new arc. These functions are then used for inserting a new object (point, line or area). One example of each is given here to illustrate the insertion of point, line and area objects using these functions. Before the insertion examples, an example of a delete operation is given.

8.3.1 Example of a Propagated Delete Operation

To give an example of the automated update propagation algorithms for delete operations, the algorithm for deleting a line object (Delete_Line algorithm, §6.2.4) was translated to a Postquel function. The following query was then executed with the function:

> *Delete line object "railway1" from layer "topo"*

Figure 8.9 shows the graphic representation of the database before the query was run. Figure 8.11 shows the graphic representation of the database after the update propagation while Appendix 3 shows the instances of the Line and Linecross classes after the deletion.

8.3.2 Example of a Propagated Point Insertion Operation

The Insert_Point algorithm (see §6.2.4) was translated to a C function and used to insert a new point object having the following properties:

Object Identifier: 79, Layer: Topo, Geographic name: gps_station1, Thematic class: geodetic_controls, xcoord: 834402.022, ycoord: 176596.472, zcoord: 215.325. (The point has the same lineage and accuracy as the original dataset.)

At the end of the update operation, relation r092(2-container, new node) was detected and the program assigned the identifier of the 2-container (7) as the value of the twoC property of the

object in class Pointnode. A new node number was assigned as the value of the pnode property of the object and a new 0-container identifier for properties zeroC_id in class Pointf and zeroC in class Pointnode. Appendix 4 shows some instances of the Pointf, Pointnode and Node classes after the insertion.

8.3.3 Example of a Line Insertion Operation.

As an example of a line insertion operation, the geometry of the deleted railway1 was modified and reinserted into the database. The line object's data, including its new geometry, now with eight line segments instead of the previous 22, is shown in Appendix 5A. The algorithm Insert_Line (see §6.2.4) was translated into a C program. The program makes use of the Insarc() function. Each line segment of the line object was inserted as an arc. The program generates the Postquel function that appends the new and modified geometric primitives into the database while deleting the obsolete ones. Appendix 5B shows the instances of the Line and Linecross classes and some instances from the Arc and Node classes after inserting the new railway. The perspective view of the database after the insertion is shown in Figure 8.12.

8.3.4 Example of an Area Insertion Operation

To test the algorithm for inserting an area object, one area object with the locational and attribute data as shown in Appendix 6A was inserted into the database. It was assumed that the object will displace the existing area objects of the same layer that it (partially or fully) overlaps (if this is a full overlap, the existing object will cease to exist, and for partial overlap, the existing area object's size will be reduced). Because of the absence of an on-line graphic display facility in Postgres, the insertion was done in two stages. First each line segment of the new object was inserted, one at a time (the program actually checks and enforces all the geometric primitives rules, see Table 5.1, and writes the result into a Postquel command macro file which is then used to update the database in the next stage), using the Insarc() function, and the existing arcs located in the interior of the new area object were detected by running the Ptin2c() function. The Insarc() function displays the identifiers and names of the existing area objects which the new object overlaps for the user's decision on compatibility (semantic consistency) and a prompt to continue or stop the updating operation. Figure 8.13 shows the graphic representation of the database after the insertion with the new area object (now decomposed into two 2-containers in black and red colours at the top right corner of the map). The relevant instances of the new object in the Area, Arc and Node classes are shown in Appendix 6B.

8.4 Summary

This chapter described the test implementation of the object-oriented data structure for multi-valued vector maps presented in §7.2 using the extended relational (evolutionary object-oriented) database management system Postgres version 4.2. Data acquisition was done with a Planicomp C120 photogrammetric stereoplotter equipped with a Zeiss Videomap (for superimposition) and a Calcomp drawing board digitizer. The stereo-compilation was done with the aid of the Kork digital mapping system version 8.0.

The implementation aimed at three objectives: (1) to illustrate the usage of the proposed data structure for multi-valued vector maps (see chapters 4 and 7), (2) to test some of the consistency rules presented in chapter 5 and (3) to test some of the updating algorithms (see chapter 6). Without loss of generality, the data structure was tested with two map layers: a soil map layer and a topographic map layer showing major land use and land cover types. The geo-data from the soil layer were initially digitized on a Calcomp drawing board. The digital manuscript was then superimposed on the stereo-model which contained the geo-data of the second layer. The superimposition resulted in the intersection of the two layers such that closed polygons became 2-containers (representing (part of) a certain area object in the soil layer and (part of) an area object in the topographic layer), line objects were decomposed into 1-containers and point objects became 0-containers. Thus the two layers were compiled in multi-valued mode.

A method was designed to assign a feature code to each string (an aggregation of connected line segments representing (part of) a certain line object or boundary of an area object) such that the code contains the identifier of the 1-container represented by the string (or zero if none) and the identifiers of the 2-containers on its sides. The coding method was devised by combining the colour and feature codes of a string in which the colour code, concatenated with the first (left justified) digit of the feature code represents a 1-container identifier, the next two digits of the feature code represent the identifier of the 2-container on the left side of the string and the last two digits of the feature code represent the identifier of the 2-container on the right. A more general coding method was proposed in the chapter whereby a look-up table (LUT) is prepared before digitizing. The LUT, containing four fields per record, will then store the string identifier as one field, the identifier of the 2-container on the left side of the string as the second field, the 2-container on the right as the third field, and the identifier of the 1-container represented by the string (0 if none) as the fourth field. The LUT will then be related to the locational data of the strings as part of the data conversion program before instantiating the database.

The digital manuscript was then converted from Kork format to the format of the DMMVM (shared geometry; line segments as arcs) and used to instantiate the nine classes obtained after the mapping of the object-oriented data structure (§7.2) to the Postgres data model.

The consistency rules proposed in chapter 5 were then verified on the created database using a combination of C functions and Postquel queries. The only geometric inconsistency detected (a vertical gap in one 1-container caused by the constraint in Kork for node snapping only in 2D while digitizing in 3D) was corrected by taking the average height of the two affected nodes. Rules were then defined in Postquel to enforce structural integrity of the database during subsequent updating.

An example of a topologic query involving the two map layers was given to illustrate the capability of the model for multilayer spatial analysis without the need for overlay operation *during* query processing, as is conventionally done in operational systems at present.

Some of the updating algorithms proposed in chapter 6 were also tested on the created database. The algorithms (translated to C functions and Postquel queries) were used to (a) delete an existing line object, (b) insert a new point object, (c) insert a new line object and (d) insert a new area object. The database remained consistent after the update operations,

indicating that the algorithms can be translated into an operational software module in a GIS. However, the experiment confirmed (as indicated in the algorithm) the need for the decision of the human operator during insertion of area objects as to the fate of existing area object(s) in the same layer which are (partly) in the same location as the new object, apart from the semantic consistency situation with the objects from the other layers (which can be handled by rules as proposed in this thesis). The operator should be assisted in this task by a graphic display of the objects involved, a facility which was not available in the DBMS (Postgres) used in this experiment. In the experiment, it was assumed that a new area object takes over the (common) location of the existing area object of the same layer it overlaps with. The semantic consistency was checked by displaying the name(s) of the spatially coinciding area objects from other layers and requesting a prompt from the user to proceed with the updating or to stop.

The Postgres DBMS, though an experimental, public domain software, proved to be effective, especially in rules management, but like most DBMS, a visualization module in the system will improve its capabilities in spatial database management.

This implementation has also proved that the data model for multi-valued vector maps, proposed in this thesis, can indeed be operationalized, this being the third experiment in different systems' environments. In the first experiment, the map layers were manually overlaid and coded into a dBase-IV DBMS to test the information content of the model (see Ayugi 1992). The second experiment used an Arc/Info system for data acquisition and dBase-IV for database management to test the usage of the model in an operational GIS environment (see Chhatkuli 1993). The third experiment reported here (see Essayah 1994 for more details) has been carried out using a photogrammetric workstation for data acquisition and an object-relational system for database management. The data model together with the consistency rules and updating algorithms can therefore be recommended for operational use in GIS and mapping.

Figure 8.9 Graphic representation of the database (perspective view)

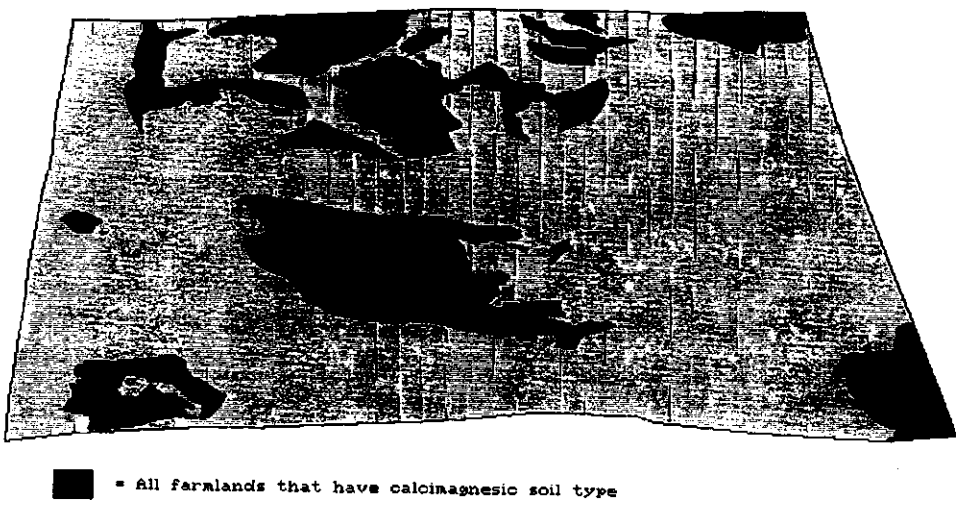


Figure 8.10 All farmlands (from topo layer) having calcimagnesic soil type (from soil layer)

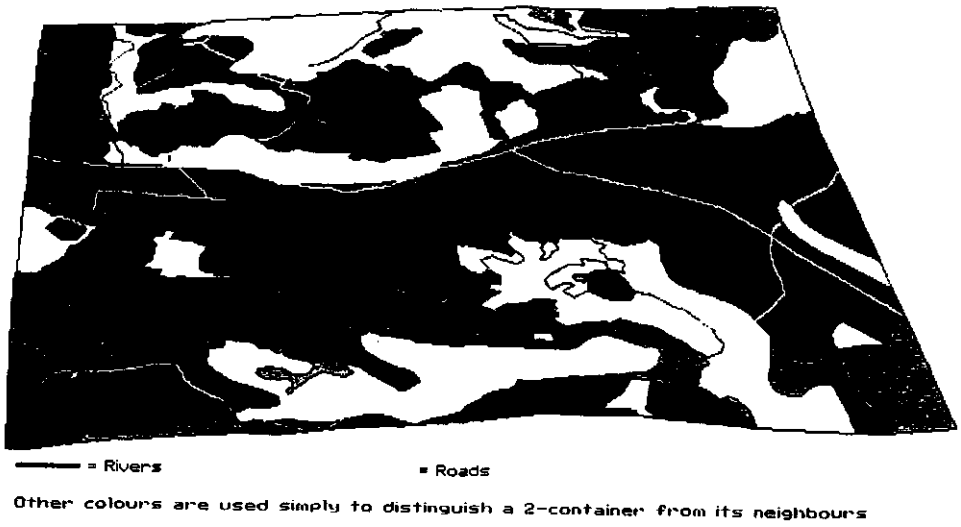


Figure 8.11 Graphic representation of the database after deleting the railway

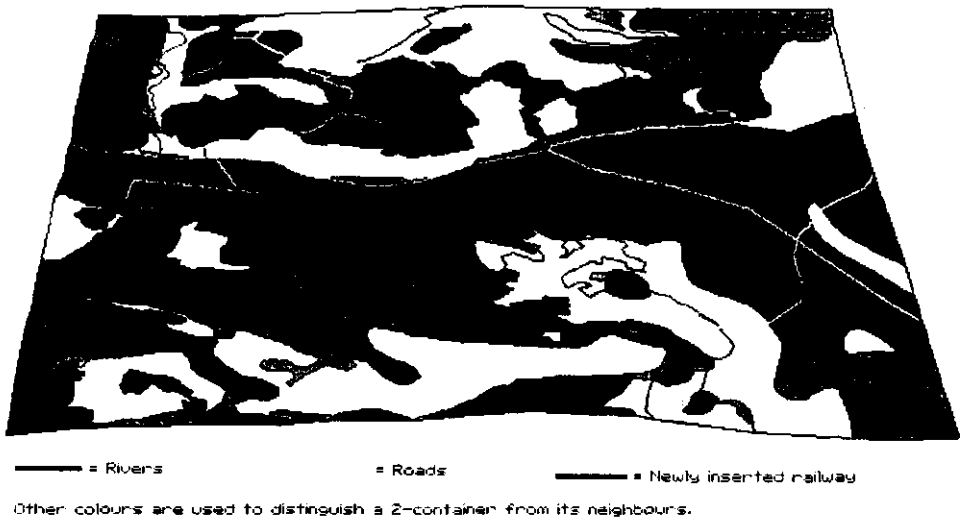


Figure 8.12 Graphic representation of the database after inserting the railway

Figure 8.13 Graphic representation of the database after inserting a new area object (the new area object was decomposed into the two 2-containers shown in red and black colours during the update propagation)

9

CONCLUSIONS

This chapter concludes the research work reported in this thesis and indicates further extensions and improvements. The research aimed at developing a formal approach for automated consistency controls during updating operations in a vector-based GIS. To serve as framework, a generalised conceptual data model was developed for a vector representation of a multi-layer terrain situation. The data model was based on the formal data structure for single-valued vector maps (Molenaar, 1989). In order to test the model and the concepts developed in this thesis and for their future implementation, the extended model was translated to relational and object-oriented data structures (logical data models). Concluding remarks on how these tasks were achieved, including comments on further research and development in this area, are presented in the following sections. A summary of the research work is given in §9.1, followed by an evaluation of the prototype data structure in §9.2. Future research and development are discussed in §9.3.

9.1 Summary of the Research Work

9.1.1 Development of a Data Model for Multi-valued Vector Maps

Geographic information systems are often classified (according to hardware and software aspects) into four main subsystems for handling the four interrelated phases in information processing, namely data collection and input, data storage and retrieval, data manipulation and analysis, and visualization and reporting. When the system is set up for geo-information production, the most vital component on which the four subsystems operate is the spatial database. The database, which represents the real world as seen by the application, must be well-structured and consistent in order to meet the objectives of the system.

To design the database, it is common to view the data at four levels of abstraction, namely reality, conceptual data model, data structure and file structure (Peuquet, 1984). The way the reality (phenomena as they actually exist) is conceptualized in the data model (usually categorised into tessellation and vector) is often tailored to a given application, i.e., different applications normally view a terrain situation differently, thereby extracting only the terrain data that play definite roles within the application. For example, a cadastral surveyor will partition a given region into land parcels with each parcel having unique attribute values. The same region will be partitioned by a soil scientist into different soil units. This implies a layered view of the terrain situation meaning that terrain objects belonging to different applications are spatially coincident in reality. However, spatial analyses and planning often require integration of different views of the world, i.e., to integrate geo-data from different map layers. The term *map layer* is used to denote a geographic dataset describing a certain aspect of the real world (Hoop et al, 1993), i.e., the set of objects belonging to the same mapping context, e.g., cadastral, soil mapping, etc.

At present, the common approach to achieve an integrated analyses in an application involving the use of spatial data from multiple map layers is to structure each layer separately and then perform an overlay operation when joint analysis is desired. The consequences of

this solution include an increase in overhead cost needed for the ad-hoc, repeated overlay computations and the difficulty of predefining the spatial relationships among features from different map layers (vertical topology). For example, to answer a spatial query like " select all cadastral parcels having a soil type V with land use type W, situated within distance X of place Y and having a metric area of not less than Z square units" in the layer approach will involve the overlay of soil and cadastral maps *during* the query processing. Any time such a query (involving multiple map layers) is submitted to the system, an overlay will have to be computed.

In chapter 3, an alternative approach was proposed in which a single model was developed to represent a multi-valued terrain abstraction, especially when frequent spatial analyses across many map layers is envisaged, i.e., in applications involving frequent analysis of multi-layer geo-information. The proposed conceptual data model is an object-based 2.5D (3D position, 2D topology) data model for multi-valued vector maps (DMMVM). Here, a vector map refers to a database representation of the terrain situation as points, lines, surfaces (areas) and bodies in which positional data are given in the form of coordinates of isolated points and the end-points of line segments. A multi-valued vector map then refers to the vector-based representation of terrain objects from multiple map layers whereby two objects of the same geometric type may be spatially coincident. Two objects of the same type are said to be spatially coincident if they (partially) overlap in space. The mathematical framework for the modelling is provided by graph theory, the relevant elements of which were described in §3.1.

In this model, positions of objects are defined in a 3D metric space but embedded in 2D topologic space, i.e., a 2.5D model. This means that only surfaces of objects are represented such that a pair of X and Y coordinates must have a single Z value, thus a single-elevation model. The model was based on the formal data structure (FDS) for single-valued vector maps (Molenaar, 1989). In the 2D FDS (see §3.2), terrain objects play a central role in the terrain description; each object has a thematic component and a geometric component. In the thematic domain, the objects can be grouped into thematic classes in which each class has a specific attribute structure (see Molenaar, 1993), and in the geometric domain, the object types points, lines and areas are distinguished for a 2D or 2.5D terrain description, subject to a constraint that two objects of the same type may not be spatially coincident. The three object types are then completely described by a common set of two types of geometric elements (arc and node), using graph theory as the mathematical framework. In the model, the geometry of a terrain object is clearly distinguished into three independent aspects, namely topology, shape and size, and position (see Figure 2.3). This geometric dataset has been carefully structured in the FDS, leading to a semantically-rich, query-oriented and extendible data model in which information on topology, shape and size, and position can be retrieved. The FDS was extended in this thesis to allow objects of the same type to be spatially coincident, thus facilitating the use of a single structure for the representation of multi-layer geo-data.

A geometric data type, the m-dimensional container, or simply m-container, where $m \in \{0,1,2\}$ was introduced to model spatial coincidence among objects of the same geometric type. Thus a 0-container represents spatially coinciding J point objects from J layers, a 1-container represents (part of) K line objects from K layers and a 2-container represents (part of) L area objects from L layers, where L is the maximum number of layers and J and K may each be less than or equal to L.

By introducing the container data type, overlapping sections across the layers are uniquely identified such that they have their own individual geometric data and non-spatial data, apart from those inherited from the overlapping objects; they can then be maintained and manipulated by the DBMS just like single objects. Thus it is easier to include them in aggregation and association abstractions, thereby improving spatial analyses in GIS.
Using graph theory as a mathematical tool, the three container types are then represented by the topologic primitives arc and node. A node defines one 0-container and/or beginning or end of an arc, while an arc defines (part of) one 1-container and/or (part of) a boundary of a 2-container. The arc is defined by one start node and one end node, and a node is defined by a coordinate triplet X,Y,Z. The geometry of the map is thus represented by a planar graph G(N,A) where A is the set of arcs of the graph and N is the set of nodes. Each m-container C, $m \in \{0,1,2\}$, is then a subgraph of G such that the geometry of C is represented by $G_c(N_c,A_c)$ where $N_c \subset N$ and $A_c \subset A$. For a 0-container, $A_c = \emptyset$.

Thus eight basic geometric data types are defined to represent geo-data from multiple map layers, namely area, line, point, 2-container, 1-container, 0-container, arc, and node. Each data type plays some specific roles in the model. The area, line and point data types abstractly represent terrain objects whereby each terrain object in the application is mapped into one of the three types during implementation. The mapping can be one-to-one or one-to-many, depending on the complexity (shape) of the terrain object, e.g., a two-dimensional object with a connected boundary and interior will be mapped to one elementary area object type, while a two-dimensional object with disconnected boundaries and interiors will be mapped into two or more elementary area objects. These related elementary objects will then be aggregated to reconstruct the parent (original) object during query.

One of the attributes of each of the three object types should be the thematic class of the object. Although the thematic aspects of objects were given less attention in the thesis, the model does not preclude the representation of thematic data. During implementation (when thematic attributes of objects are identified) the thematic data can be arranged in a hierarchic manner as proposed in Molenaar (1993).

The m-container, $m \in \{0,1,2\}$, models spatial coincidence among elementary objects of equal spatial dimension, as explained above. Apart from the attribute values inherited from the spatially coinciding objects, an m-container data type can have additional attributes as required by the user. For example, in a multi-valued vector map that integrates land use map and soil map data, apart from the attribute values propagated to a 2-container by the two spatially coinciding area objects from the two layers, the 2-container can have additional attributes such as metric area, alternative land uses (based on factors such as type of soil, nearness to certain utilities, etc) etc.

Arc and node, as stated above, play the roles of geometric descriptors in the model. The proposed DMMVM can thus be used to organize the result of an initial overlay of all relevant map layers for subsequent single-valued or multi-valued queries.

Spatial relationships (topologic, metric and order) provide the main framework for spatial analysis in GIS. Thus vector data models place great emphasis on the modelling of the topologic spatial relationships among objects. However, little effort is usually made to formalize the consistent set of relationships which a given data model can support in order

to have a priori knowledge of the information content of the model and to provide a spatial query language for retrieving such information. Apart from this acknowledged role of topologic relationships in GIS, they can also serve as useful tools in automated database updating and the maintenance of data consistency in GIS as shown in this thesis.

The possible topologic relationships among the three elementary object types area, line and point, and among the geometric primitives arc and node in the model proposed in this thesis were formalised in chapter 4 using the 9-intersection model (Egenhofer and Herring, 1992) as interpreted for graph-structured vector maps (Molenaar et al, 1994). Algorithms are then defined for detecting the occurrence of any of the elementary relationships for any object combination. The algorithms can be translated to topologic operators and used for topologic queries, as well as providing a tool for detecting violation of and enforcing geometric constraints. When used as a tool for maintaining consistency, the topologic operators will serve as detectors of inconsistencies. The return value of an operator will trigger the relevant rule that will enforce consistency if violation occurs. The rules that enforce the geometric consistency were presented in chapter 5.

An integration of the DMMVM with a DTM was described in §3.5, indicating the extendibility of the model to relief modelling. The integrated model provides a unified representation of multi-layer terrain object data and terrain relief data in a flexible manner, such that DTM specific information can be derived from the same database as any other spatial information under a single database management system, while retaining the ability to perform non-DTM related spatial analysis without involving DTM information. The integration is based on the consideration that most of the skeleton of a DTM is usually contained in terrain objects such as rivers, roads, lakes, etc., and with the importance of DTM in spatial analyses, it is apparently more efficient to integrate geographic and elevation models. The terrain relief is therefore regarded as a mapping layer during data acquisition so that the terrain surface is classified into DTM object types in form of point, line and area on the basis of relief characteristics (slope and height). The DTM class can then become one of the mandatory properties of each terrain object.

The edge-based triangulated irregular network (TIN) was selected for the digital representation of the terrain relief since it can easily be linked with the topologic structure of the DMMVM via the geometric primitives arc and node which are isomorphic, respectively, to the primitives edge and vertex of the edge-based TIN. Thus in addition to the eight data types in the DMMVM, two extra data types, edge and vertex, were added to represent terrain relief in the integrated model. An edge is defined by two adjacent TIN vertices and has one triangle on each side. A vertex is defined by a coordinate triplet X, Y and Z. The geometric connection between the edge-based TIN and the DMMVM is provided by the links among the geometric primitives arc, node, edge and vertex. A TIN edge can be part of zero or one arc while a vertex can be a node.

An algorithm can be provided for deriving relief information in the interior of area objects (see Kufoniyi and Boulloucos, 1994 for an example) because these cannot be resolved through the links among the geometric primitives. The position of an object can therefore be given in 2D or 3D; when defined in 2D, the height value can be interpolated from the DTM subsystem. Thus objects in the object-base can contribute to the generation of a DTM with high fidelity, while the DTM supports the object-base, e.g., when updating via monoplotting

techniques, to provide height information for objects whose Z values could not be determined during the data collection phase, and to provide relief information in general. Additional information on the integration of the DMMVM and DTM can be found in Kufoniyi et al (1994), Kufoniyi and Pilouk (1994), Pilouk and Kufoniyi (1994), and Kufoniyi and Bouloucos (1994).

9.1.2 Translation of the Model to Database Structures for Implementation

A conceptual data model is normally developed without a consideration of the type of system that will implement it. However, for implementation purposes, it is necessary to translate the model into a prototype database structure, based, usually on a database model (e.g., relational, network, etc.). Thus the conceptual data model for multi-valued vector maps (DMMVM) was translated into two families of database structures, namely (1) a relational database structure and (2) an object-oriented database structure (see chapter 7).

The prototype relational database structure for multi-valued vector maps was designed using Smith's method for relational database design (Smith, 1985). The method comprises four steps, namely (1) identification of data types and the relationships among them, (2) listing the single-valued and multi-valued dependencies among the data types as dependency statements, (3) translating the dependency statements to a dependency diagram, and (4) composing normalised relations from the diagram. Using this method, seven base tables were developed for creating a relational database for multi-valued vector maps. Additional thematic data of the objects can be introduced by creating extra tables for the thematic classes. Some experimental implementations carried out with the prototype indicate its usability for multi-layer spatial data modelling (see Ayugi, 1992; Bouloucos et al 1993; Chhatkuli, 1993 and Bouloucos et al, 1994). However, if the relational prototype is used for an implementation, then the consistency rules (chapter 5) and the update propagation algorithms (chapter 6) would have to be handled by a high-level programming language (e.g., C or Fortran) and coupled with the RDBMS (see Kufoniyi, 1989 and Kufoniyi et al, 1993 for examples) since most operational RDBMSs are not capable of handling user-defined rules.

The relational structure can serve for immediate implementation given the wide availability of operational relational DBMS as compared with object-oriented systems. Because of the shortcomings of the relational model in handling spatial data (see chapter 2), the model was also translated to an object-oriented data structure (which has been acclaimed to be more suitable for spatial applications than the relational model). The object-oriented modelling constructs classification, generalisation, aggregation and association, together with the concepts of inheritance and propagation, were applied to translate the data model to an object-oriented data structure.

From the basic structure of spatial objects whereby each terrain object has two main characteristics, geometric and thematic, two classification domains were distinguished for the object-oriented modelling, namely (1) thematic domain and (2) geometric domain. Thus each terrain object will be an instance of one of the thematic classes and an instance of one of the geometric classes (i.e., double inheritance). Attention was focused on the latter classification which yielded a total of nine classes (each of the eight data types -- area, line, point, 2-container, 1-container, 0-container, arc and node -- as a class plus the explicit representation of the topologic relationship "cross" between two 1-containers as a class) for the object-

oriented data structure. The consistency rules and updating procedure proposed respectively in chapters 5 and 6 become operations (methods) of the classes. Also, the topologic relationships derived in chapter 4 are expected to be dynamically detected by the system. This can be done by translating the proposed algorithms for detecting a topologic relationship (see chapter 4) into operations of the classes.

9.1.3 Development of Procedure for Spatial Database Updating

Another aspect of this research was to formulate procedures for a consistent automated updating of a vector-structured database, using the DMMVM as a framework. In geo-information production, the cost of data collection has been said to be about seven to ten times more than the cost of the hardware and software needed to establish the database (Peled, 1994). Thus it is very important that the accuracy and currency of the data should be reliable, such that the purpose for setting up the database can be fulfilled with profitable cost recovery. This thesis aimed at contributing towards achieving this by providing algorithms for automated update propagation (see chapter 6) such that topology is automatically updated by the system in a consistent manner. This will improve on the current practice in operational systems, which usually requires a delayed reconstruction of topology whenever the geometry of an existing object changes or when a new one is inserted. In other words, there is usually a time-lag between the time the geometric state of the object changes and when the topology is reconstructed, often by the user having to issue a command for the reconstruction.

Although geoinformation updating includes change detection, data collection and database updating, the focus in the thesis was on automated database updating, under the assumption that the necessary changes have been detected and captured in readiness for input into a DMMVM-structured database. Algorithms were developed for the insertion, deletion or modification of each of the eight data types (area, line, point, 2-container, 1-container, 0-container, arc and node) in the DMMVM. The updating of the two geometric primitives (arc and node) are at the lowest level upon which the updating of other data types are based.

The topology of the database is updated dynamically by the system during the updating by evaluating, using computational geometry, the topologic relationship between the new primitive (arc or node) of an object and the existing primitives in the database. The type of relationship detected will then activate the relevant consistency rule (including update propagation) to validate the topology and consistency of the database. Ideally, the defined algorithms should be translated into computer modules as an integral part of an existing DBMS. However, since most of the operational DBMS are not capable of accepting user-defined rules and data types, the algorithms may have to be programmed in a high-level language and then coupled with the DBMS during implementation.

9.1.4 Handling Data Consistency in Spatial Databases

This research also addressed the problem of data consistency in spatial databases. A large proportion of the cost of setting up a database for spatial information production is attributed to data acquisition. To achieve the aims of setting up the database, the information produced by that system must be reliable, i.e., the quality of the data from which the information is derived must be reliable. This has made the issue of data quality an important aspect in GIS. Data consistency is a component of data quality because consistency is essential for the

database's reliability. Consistency can be categorised into two types: static and dynamic consistencies. The thesis focused on static consistency which can also be analyzed into structural and semantic consistencies (see chapter 5). For a vector map to be structurally consistent, the topology of individual objects represented in the database must be consistent. In addition, the topology of the geometric descriptors of the objects must be preserved. Furthermore, if the objects are embedded in a 2D topologic space as in the model developed in this thesis, the planarity of the map must be enforced by ensuring that the generalised Euler constant (see §5.1.3) holds at any time.

The semantic consistency, on the other hand, deals with topologic consistency between pairs of objects and the application-dependent constraints attached to individual objects. In chapter 5, consistency rules were formulated to ensure structural constraints, while a monitoring strategy was proposed for semantic constraints. In both cases, topology plays the central role as an "alerter" of constraint violations. As shown in chapter 5, the conditions can be translated to topologic constraints (for single objects) and topologic relationships (for object pairs) and stored in the database as *events*. The corresponding responses of the system to enforce consistency can then be defined as *actions*, thus giving a rule-based procedure (using the **if** ***event*** **then** ***action*** convention) for the management of data consistency in spatial databases.

9.1.5 Experimentation of the Prototype Data Structure

An experimental implementation of the proposed model as translated to an object-oriented data structure in §7.2 was carried out using the extended relational (evolutionary object-oriented) database management system Postgres, version 4.2 (see chapter 8). Data acquisition was done with a Planicomp C120 photogrammetric stereoplotter equipped with a Zeiss Videomap (for superimposition) and a Calcomp drawing board digitizer. The stereo-compilation was done with the aid of the Kork digital mapping system, version 8.0.

The implementation aimed at three objectives: (a) to illustrate the usage of the proposed data structure for multi-valued vector maps (see chapters 3 and 7), (b) to test some of the consistency rules presented in chapter 5 and (c) to test some of the updating algorithms (see chapter 6). Without loss of generality, the data structure was tested with two map layers: a soil map layer and a topographic map layer showing major land use and land cover types. The geo-data from the soil layer were initially digitized on a Calcomp drawing board. The digital manuscript was then superimposed on the stereo-model which contained the geo-data of the second layer. The superimposition resulted in the intersection of the two layers such that closed polygons became 2-containers (representing (part of) a certain area object in the soil layer and (part of) an area object in the topographic layer), line objects were decomposed into 1-containers and point objects became 0-containers. Thus the two layers were compiled in multi-valued mode.

A method was designed to assign a feature code to each string (an aggregation of connected line segments representing (part of) a certain line object or boundary of area object) such that the code contains the identifier of the 1-container represented by the string (or zero if none) and the identifiers of the 2-containers on its sides. The coding method was devised by combining the colour and feature codes of a string in which the colour code, concatenated with the first (left justified) digit of the feature code represents a 1-container identifier, the

next two digits of the feature code represent the identifier of the 2-container on the left side of the string and the last two digits of the feature code represent the identifier of the 2-container on the right.

A more general coding method was proposed in chapter 8 whereby a look-up table (LUT) is prepared before digitizing. The LUT, containing four fields per record, will then store the string identifier as one field, the identifier of the 2-container on the left side of the string as the second field, the 2-container on the right as the third field, and the identifier of the 1-container represented by the string (0 if none) as the fourth field. The LUT will then be related to the locational data of the strings as part of the data conversion program before instantiating the database.

The digital manuscript of the compiled multi-valued map was then converted from Kork format to the format of the DMMVM (shared geometry and line segments as arcs) and used to instantiate the nine classes obtained after the mapping of the object-oriented data structure (§7.2) to the Postgres data model.

The consistency rules proposed in chapter 5 were verified during implementation using a combination of C functions and Postquel queries. The geometric inconsistency detected was corrected and rules were defined in Postquel to enforce structural integrity of the database during subsequent updating. In addition, computer programs were developed for detecting the topologic relationships among geometric primitives (i.e., between a new arc and an existing arc, between a new arc and an existing node (or vice versa) and between a new and an existing node) during updating such that the detected relationship will activate the corresponding C function that enforces geometric consistency.

An example of a topologic query involving the two map layers was given to illustrate the capability of the model for multi-layer spatial analysis without the need for an overlay operation during query processing as is conventionally done in operational systems at present. Information relating to a single layer can also be easily retrieved (see Essayah, 1994 for examples).

Some of the updating algorithms proposed in chapter 6 were also tested on the created database. The algorithms (translated to C functions and Postquel queries) were used to (a) delete an existing line object, (b) insert a new point object, (c) insert a new line object and (d) insert a new area object. The database remained consistent after the update operations, indicating that the algorithms can be translated into an operational software module in a GIS. However, the experiment confirmed (as indicated in the algorithm) the need for the decision of the human operator during insertion of area objects as to the fate of existing area object(s) in the same layer which are (partly) in the same location as the new object, as well as the semantic consistency situation with the objects from the other layers.

The human operator should be assisted in this task by a graphic display of the objects involved, a facility which was not available in the DBMS (Postgres) used for the experiment in this thesis. In the experiment, it was assumed that a new area object takes over the (common) location of the existing area object of the same layer it overlaps with. The semantic consistency was checked by displaying the name(s) of the spatially coinciding area objects from other layers and requesting a prompt from the user to proceed with the updating or to

stop. An earlier, related experiment on automated update propagation, using a subset of these algorithms, was also performed on a single-valued vector map by the author. The experiment was done by coupling Microsoft Fortran with an Oracle DBMS in a microcomputer environment at the Department of Land Surveying, Photogrammetry and Remote Sensing, Wageningen Agricultural University. The algorithms were translated into Fortran programs, while Oracle served as the RDBMS retrieving the necessary data from the database into the Fortran program, for updating and consistency operations, and returning the updated data back into the database. Details of this experiment can be found in Kufoniyi (1989) and Kufoniyi et al (1993).

The Postgres DBMS, though an experimental, public domain software, proved to be effective especially in rules management, but like most DBMS, a visualization module in the system will improve its capabilities in spatial database management.

The experimental implementation of the data model for multi-valued vector maps proposed in this thesis has also proved that the model can indeed be implemented, this being the third experiment in different systems' environments. In the first experiment, the map layers were manually overlaid and coded into a dBase-IV DBMS to test the information content of the model (see Ayugi, 1992). The second experiment used an Arc/Info system for data acquisition and a dBase-IV for database management to test the usage of the model in an operational GIS environment (see Chhatkuli, 1993). The third experiment reported here (see Essayah, 1994 for more details) was carried out using a photogrammetric workstation for data acquisition and an object-relational system for database management. The data model, together with the consistency rules and updating algorithms, can therefore be recommended for operational use in GIS and mapping.

Although the DTM aspect of the data model was not addressed in the experiment, a subsequent decision to create a DTM of the same area will not require much extra effort because the locations of objects have been defined in 3D. First, two extra classes, namely Edge and Vertex, should be created in the database with the class Edge having mandatory properties edge-id, beg-vertex, end-vertex, left-triangle, right-triangle and arc-id (identifier of the arc which the edge is part of; this will be zero if none), and class Vertex having properties vertex-id, x-coordinate, y-coordinate and z-coordinate including accuracy and lineage if desired. Then the stereomodel will be set up again using the existing orientation parameters of the model and the digital manuscript superimposed to determine the DTM classes of the already digitized objects and to acquire additional skeleton data as well as filling data. Based on the DTM classes just determined for the objects, the coordinates of those objects would be retrieved from the database and combined with the acquired skeleton and filling data to triangulate the project area (using any triangulation software) and structure the result according to the structure proposed in chapter 3.

9. 2 Evaluation of the Model

As stated earlier in this chapter, the proposed data model for multi-valued vector maps (DMMVM) was based on the 2D formal data structure (FDS) for single-valued vector maps. The extension of the FDS now facilitates the use of a single structure to represent spatially coinciding objects of the same type, i.e., terrain objects from different map layers.

The clear distinction in the FDS of the geometric aspects of terrain objects into topology, shape and size, and position not only facilitates the construction of a semantically-rich, query-oriented spatial database, it also leads to an extendible and flexible data model. For example, having distinguished the semantic characteristics of terrain objects into thematic and geometric, it follows that the geometry of the same terrain situation can be represented either by vector elements (arc and node) or by raster elements (see Molenaar and Fritsch, 1991 and Molenaar and Janssen, 1992), leading to flexibility in the choice of system configuration for its implementation and in data exchange. The flexibility and extendibility of the model have also been demonstrated in this work. The clear articulation of the geometric aspects of an object facilitated key modelling decisions. (1) It became possible to decide on the dimension of the metric space (whether 2D or 3D) independent of the dimension of the topologic space. Here, the extended model was based on 3D coordinate space (position) and 2D topologic space, i.e., a 2.5D data model. (2) It also helped in deciding at which level to integrate geo-data from multiple map layers, whether at geometric level or at thematic level.

At the geometric level (the choice in this thesis), it becomes possible to distinguish four different approaches to the geometric integration by considering how metric (positional) data and topologic data of the different layers are handled. The four possibilities are (i) to structure each layer separately, i.e., combining metric and topology per layer and perform overlay of the layers when necessary; (ii) to structure the geometric data such that all the layers share a metric dataset while topology is kept per layer; vertical topologic query will then be done by overlay computation or by comparison of metric data; (iii) to structure the geometric data such that all layers share a common topology, while the metric information is structured per layer; and (iv) to define a model in which both metric data and topology are shared by all layers as proposed in this thesis. The pros and cons of the four geometric approaches are given in chapter 3 (see also Hoop et al, 1993) but some remarks are in order about the fourth, for which a data model has been proposed in this thesis. The advantages of this approach include the following:
(1) Elimination of redundant data because a single geometric dataset is kept for all layers instead of storing geometric components (position and/or topology) separately for each layer.
(2) Faster multiple-layer queries since it will not be necessary to compute an overlay before answering such queries.
(3) Reduction in overhead cost: overlay is computed once, whereby problems of spurious polygons, sliver lines, etc. are handled once, although there is the disadvantage of performing overlay computation where it is not required (see subsequent paragraphs).
(4) Higher information content, the knowledge of which is also known a-priori; thus a query language can be predefined for retrieval of such information.
(5) Spatial consistency can be maintained at system level since only one data structure is used and only one geometric data set is kept.
(6) Because the overlapping parts among objects of the same type are uniquely identified with their own geometric and non-spatial datasets, they can be maintained and manipulated just like single objects; thus it is easier to include them in aggregation and association abstractions.

Thus the proposed data model is query-oriented, giving high performance efficiency when used in applications that frequently require analysis of multi-layer geo-data with a high density of spatial coincidence among the objects. Even when an implementation starts with separate layers, the proposed database structure can be used later to organize the result of an

overlay computation during query if it is desired to make the overlay result persistent for future queries.

Using the data model to organize a multi-layer terrain situation, however, means that overlay computation would have to be performed in all parts of the geographic space, including areas that may not require it, thereby introducing some unnecessary increase in the storage and computational costs. Note also that query operations concerning a single layer will be slower in the integrated model than in separate structures. Thus it is necessary to have a priori knowledge of the extent of the spatial coincidence among objects in the application and how frequent the "vertical" spatial analysis will be required. If it is possible to model terrain objects in a multi-layer application such that no two objects of the same type overlap in space, then the FDS will be more suitable. And if it is certain that only a limited number of objects will be spatially coincident in the application or more single-layer queries are envisaged with vertical queries required only seldomly, then the layers are best structured separately.

Note that using the FDS instead of the DMMVM, as mentioned in the paragraph above, does not significantly invalidate the consistency rules and update propagation algorithms developed here, since they were formulated at the conceptual level with terrain objects, as defined in the FDS, as the main focus. In addition, an m-container is topologically isomorphic to an n-dimensional elementary object with m = n; it is thus easy to harmonize the operations defined for the m-containers with those of the elementary objects.

The DMMVM is more query-oriented than data acquisition-oriented (except where it is possible to derive all input data from the same source) because data acquisition is usually done within a certain context and by a specialist in that particular discipline. For example, to set up an integrated database incorporating soil and cadastral geo-datasets, the classification and sampling of soil units will be performed by a soil scientist, while the demarcation and survey of the cadastral parcels will be done by a land surveyor. The collection of data for updating will also follow the same trend: the data will be collected per layer and it is easier to update a single layer than a combination of layers.

In a DMMVM-based GIS, the addition of a new object means a further segmentation of the geographic space, leading to higher storage and overhead cost, but this is relevant only if the model is used in a situation where vertical spatial analysis is not often required. When used to organize multi-layer geo-data for continual vertical spatial analysis, it solves the problem of the much higher overhead cost that will occur if overlay is computed for every query. Intuitively, it would also appear that overlay computation will be required each time the database is being updated. This is not necessarily so because the updating algorithm proposed here eliminates the need for overlay computation during updating; the updating is propagated just as it would in a single-layer vector structure. This has been made possible by using the same geometric structure for single- and multi-valued vector maps.

The experience gained during the experimentation with the model indicates that elaborate planning is required to create the database but, as pointed out earlier, the faster multi-layer query and richer information content of the database compensate for this. Data consistency is also ensured because inconsistencies are identified and corrected during the creation or updating of the database.

If this model is compared with other models of its type, e.g., the ATKIS DLM data model (see §2.1.2) and another variant of the FDS extension proposed by Hoop et al (1993) (see §3.3), the DMMVM has a potentially richer information content because of the explicit representation of overlapping sections among objects, which can then be assigned additional attribute values apart from the ones they inherit from the overlapping objects. These are implicitly represented in the two cited examples and will therefore require extra computations to derive them during query.

The provision of conventions for the FDS, with extensions of the conventions as required by the DMMVM, facilitates an unambiguous mapping of the terrain situation to a DMMVM-structured database, making it possible to implement the proposed data structure in any existing GI systems, albeit with additional operations (such as programming) or slight modification of conventions. For example, many existing systems represent an arc differently, usually as an aggregation of straight-line segments; thus additional operations will be required to restructure topology if we keep to the assumption that an arc is a straight-line segment. This, in addition, introduces some data redundancy. Because it does not invalidate the conceptual model (an arc can have any shape subject to the constraints that it is not self-intersecting and it does not close back on itself), this convention can be relaxed during implementation to accommodate a different shape definition for an arc (Molenaar, 1992).

In addition, it is possible to also relax the convention that constrains all objects belonging to the same thematic class to be of the same geometric type, to allow for the construction of complex thematic classes. The requirement for additional operations often arises because the data models of many of the existing systems often lack the topologic richness that is found in the FDS. For example, in order to realise an object-level topology in Arc/Info, such as knowing all the cities (with each city represented as an area object) through which a certain road (with roads represented as line objects) passes, will require the overlay of the line and polygon coverages. The topologic relationships that are explicitly represented in the conceptual data model will therefore be pre-computed in the same manner and stored.

The power of the data structure developed here is in the fact that it can be used in any vector-based application being a generic model. It can be used to set up the primary database and then define aggregation rules (see Richardson, 1993 for an example) to derive databases of lower resolutions from the basic structure.

9.3 Further Research and Development

Much work has been done in this research to meet the stated objectives, but there are still areas for further research and development. As with every prototype, the relational and object-oriented data structures proposed in this thesis still require more experiments before actual implementation. Experiment is needed to formulate the optimum procedure to collect data using any data source (aerial photographs, digital images, etc.) and data acquisition method (photogrammetry, digital image processing of remotely sensed data, land surveying, etc.). Since the main aim is to automate much of the processes involved in the production of geo-information, the data collection procedure should, as much as possible, include dynamic building of topology and consistency enforcement as the data are entered into the system. The consistency rules and update propagation algorithms, together with the topologic coding

method proposed in this thesis, will serve this purpose by translating them into software codes as part of the data collection software for the implementation of the structure.

Not all the developed rules and algorithms were tested during the experimental phase of the thesis because not all the formalized situations occurred during the experiment. Moreover, the extensiveness of the rules and algorithms implies that more time is required to translate all of them into software codes and test them with real data. More extensive tests are therefore required to verify all the possibilities covered in the consistency rules and the update propagation algorithms. The consistency rules and updating algorithms have been defined for a 2D topologic space. It is necessary to extend them to cover a 3D topologic situation, using, for example, the 3D FDS (Molenaar, 1990) as the framework. This can be done by first extending the topologic relationships to cover 3D topologic space (see for example Hoop et al, 1993), which will then be used as tools for consistency operations using the approach in this thesis as guidelines.

On the basis of the experience articulated in this thesis on the extension of the 2D FDS (for single-valued vector maps) to a 2.5D data model for multi-valued vector maps (DMMVM), it should be possible to also extend the 3D FDS to accommodate multi-valued terrain representation.

A basic assumption made in the modelling process is that time is considered constant such that, during updating, the old data are treated as obsolete and deleted, and only the current data are kept. Although it is possible to archive these obsolete data, because the basic data model does not incorporate time as a variable component of objects, spatio-temporal analysis cannot be performed. This aspect is now very important in GIS; thus it is necessary to extend the proposed model to accommodate the temporal dimension of objects. In the same vein, the boundaries (positions) of objects are assumed to be well-defined (i.e., crisp dataset) in this work; it is also important and relevant to investigate and formalize the aspect of objects with fuzzy boundaries which will then be incorporated in the consistency rules and updating procedure.

9.4 Thesis Recapitulation

9.4.1 Conclusions

- A 2.5D query-oriented spatial data model for multi-valued vector maps was developed by extending the 2D formal data structure (FDS) for single-valued vector maps.

- Three geometric data types, namely 0-container, 1-container and 2-container were added to the five basic data types (area, line, point, arc and node) in the FDS to facilitate representation of geo-data from multiple map layers.

- The addition of the m-container, $m \in \{0,1,2\}$, facilitated an explicit representation of spatial coincidence among objects of the same geometric type, meaning that an m-container can be handled as an individual object with distinct geometric and attribute properties for spatial analysis.

- The proposed model gives faster and richer multi-layer topologic queries in a vector GIS.

- Overlay is computed only once at the time of database creation (if data collection is made per layer) and not during query processing as presently done in most commercial systems; thus production cost is reduced, while increasing the information content of the database.

- By using an edge-based TIN, the model can be integrated with a DTM in a flexible manner as demonstrated in the thesis.

- A method was proposed for the coding, during data collection, of the elementary topologic relationships to be made explicit in the database.

- Consistency constraints in vector-structured spatial databases were identified and analyzed.

- Rules were then defined for monitoring and enforcing the constraints during database creation and updating. Topologic relationships served as a tool in the rule-based scheme; they are used as inconsistency detectors which activate the corresponding operations to be performed by the system to validate consistency.

- The possible topologic relationships in vector maps were formalized for the above-mentioned purpose but they also serve their traditional role in topologic queries.

- The thesis addressed the issue of database updating by providing a procedure for automated updating of the database while maintaining data consistency.

- Algorithms were provided for the updating of the basic data types in a vector-structured spatial database using the proposed data model as a framework. Complex updating operations can then be decomposed into the formalized elementary operations.

- Having translated the proposed model to a relational structure and an object-oriented data structure, implementation in a variety of systems is made possible.

- An experimental implementation of the object-oriented prototype indicated that the model and the consistency rules and the updating algorithms can be used in a production environment. For the experiment, data were acquired by analytical photogrammetry, while an extended relational DBMS (Postgres) was used for database management.

- For frequent spatial analysis across many layers, the proposed model is very suitable.

9.4.2 Recommendations

- Because the proposed model requires a complete segmentation of the combined layers, even in areas where vertical topologic queries will not be performed, it is necessary to have a priori knowledge of the extent of spatial coincidence among objects in the application and how frequent the "vertical" spatial analysis will be required in order to decide on the optimum data model for multi-layer representation.

- If it is possible to model terrain objects in a multi-layer application such that no two objects of the same type overlap in space, then the FDS will be more suitable. And if it is certain that only a limited number of objects will be spatially coincident in the application or more single-layer queries are envisaged with vertical queries required only seldomly, then the layers are best structured separately.

- To facilitate easier implementation of the model (and the FDS on which it is based) in commercial systems, and to further minimize data redundancy, the arc can be implemented as an aggregation of straight-line segments.

- Experiments are needed to formulate the optimum procedure to collect data for multi-valued vector maps using any data source (aerial photographs, digital images, etc.) and data acquisition method (photogrammetry, digital image processing of remotely sensed data, land surveying, etc.); the coding method proposed in the thesis can be used for such an experiment.

- More extensive tests are required to verify all the possibilities covered in the consistency rules and the update propagation algorithms.

- The proposed 2.5D data model for multi-valued vector maps (together with the rules and algorithms) should be extended to full 3D, e.g., by extending the 3D FDS to accommodate multi-valued terrain representation following the same approach used here.

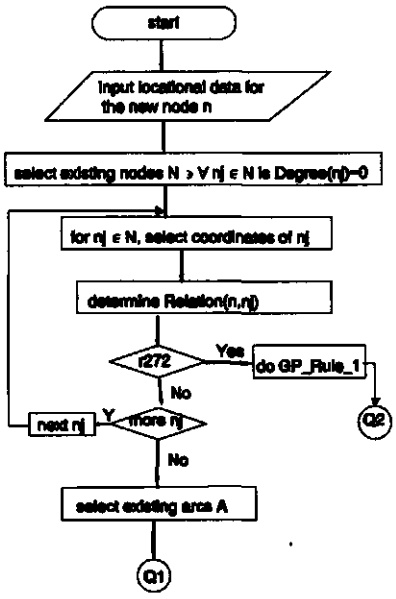
- To be able to handle spatio-temporal analysis, the temporal dimension of objects have to be accommodated in the proposed model.

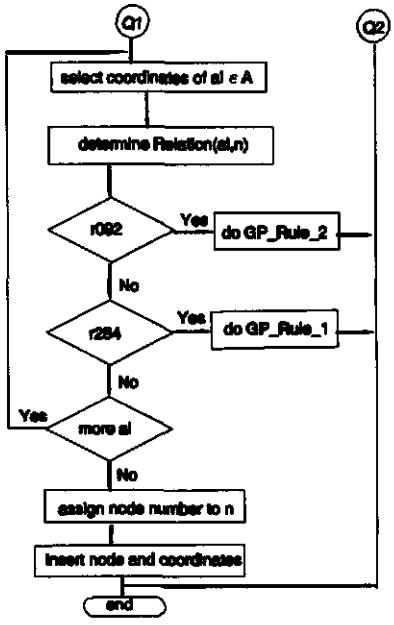
- It is also important and relevant to investigate and formalize the aspect of objects with fuzzy boundaries so as to incorporate this aspect in the consistency rules and updating procedure.

APPENDICES

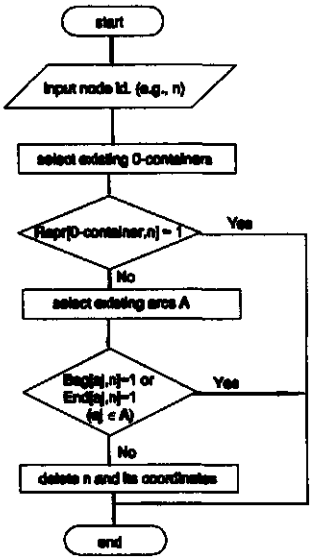
APPENDIX 1.1 Block Diagrams of the Updating Algorithms
(see Chapter 6 for the notations and algorithms)

1.1.1 Insert Node

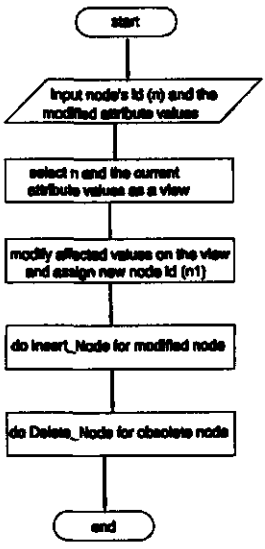




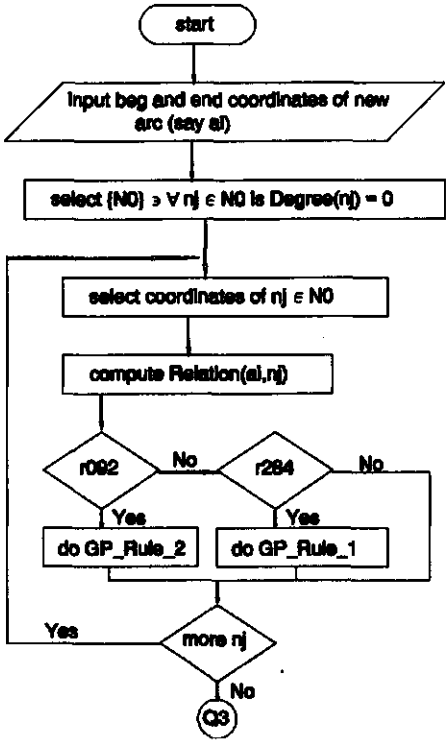
1.1.2 Delete Node

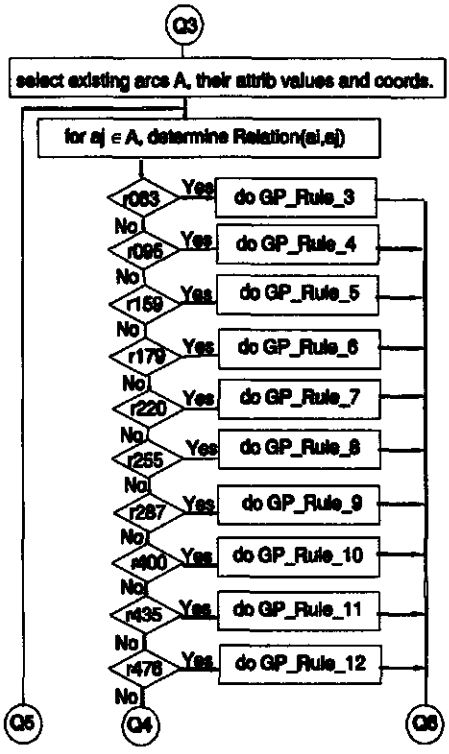


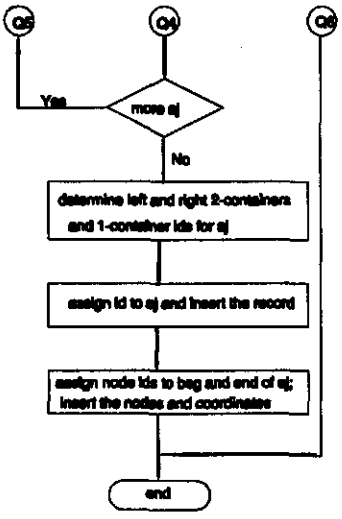
1.1.3 Modify Node



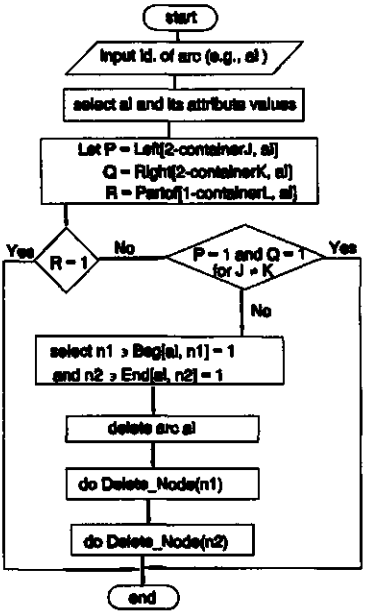
1.1.4 Insert Arc



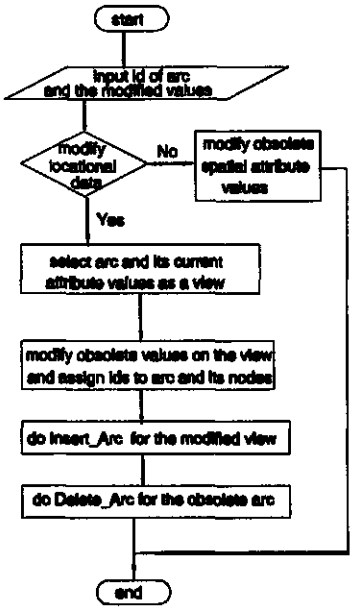




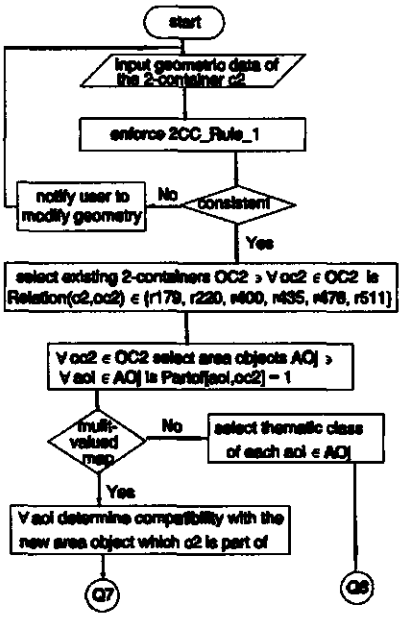
1.1.5 Delete Arc

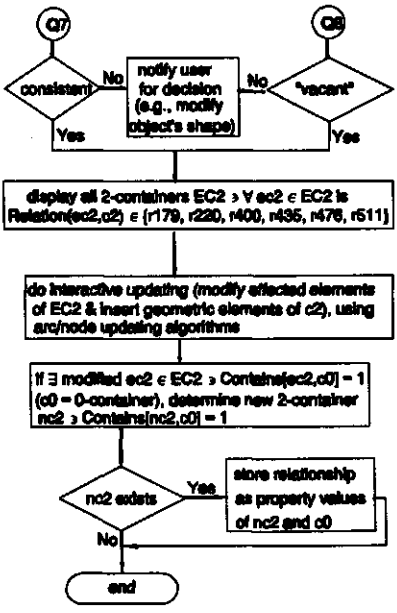


1.1.6 Modify Arc

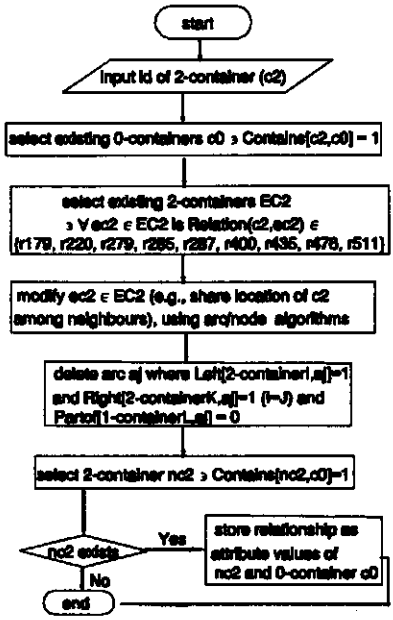


1.1.7 Insert 2-container

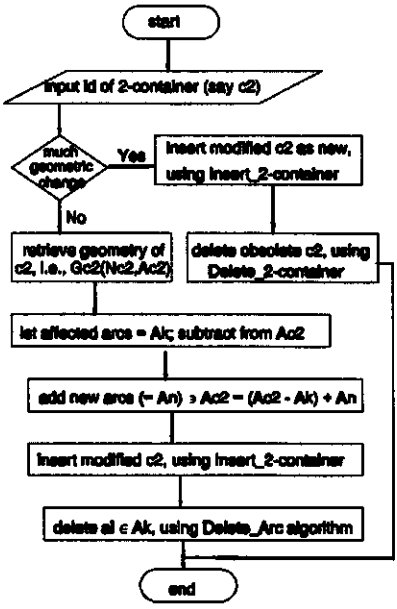




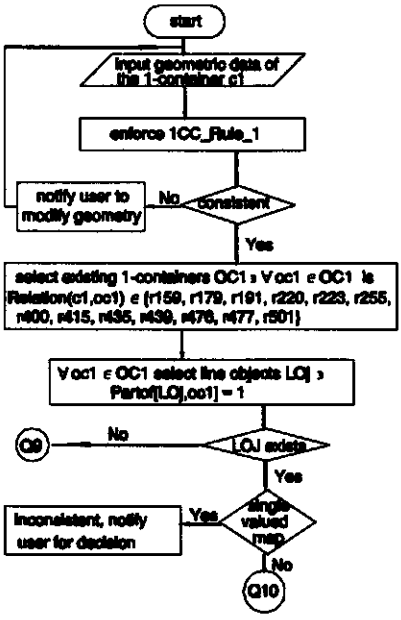
1.1.8 Delete 2-container

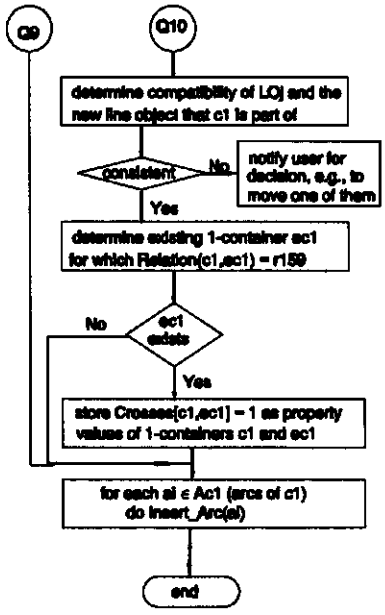


1.1.9 Modify 2-container

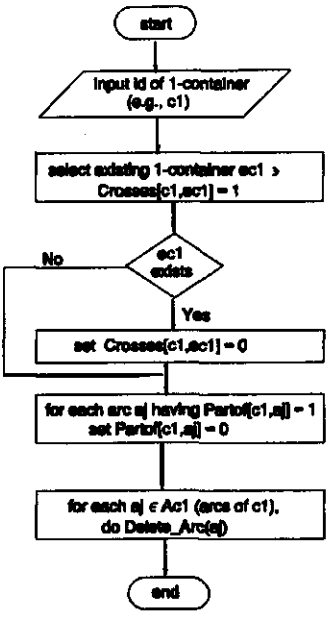


1.1.10 Insert 1-container

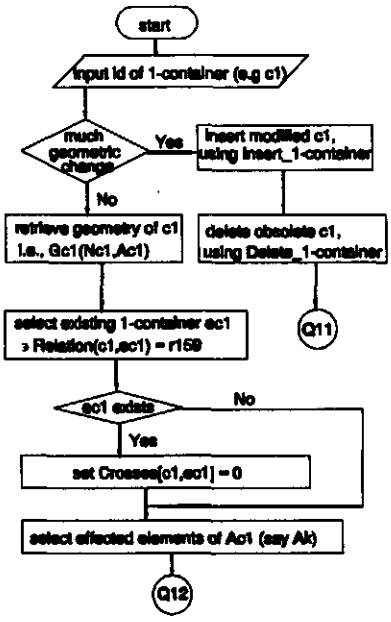


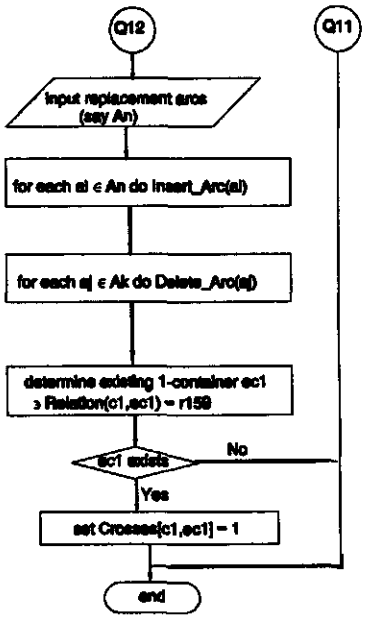


1.1.11 Delete 1-container

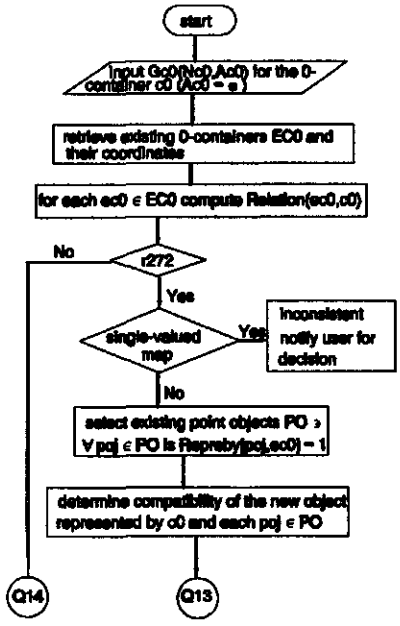


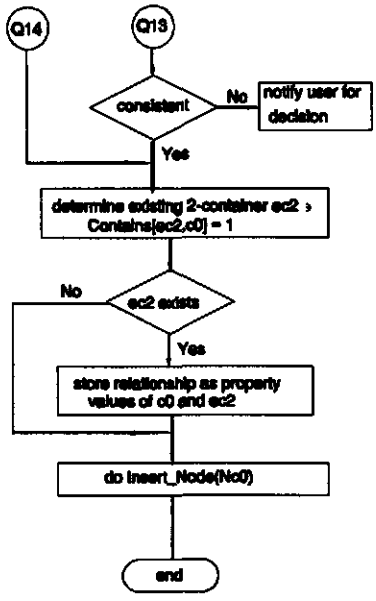
1.1.12 Modify 1-container



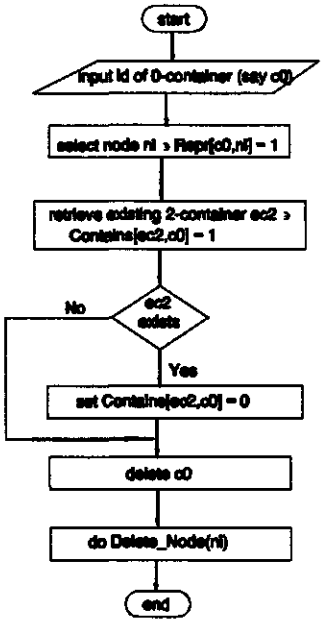


1.1.13 Insert 0-container

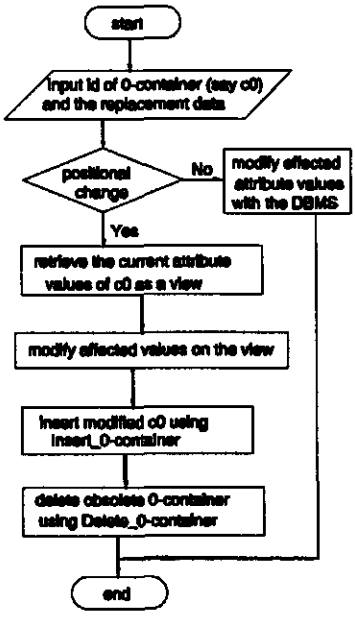




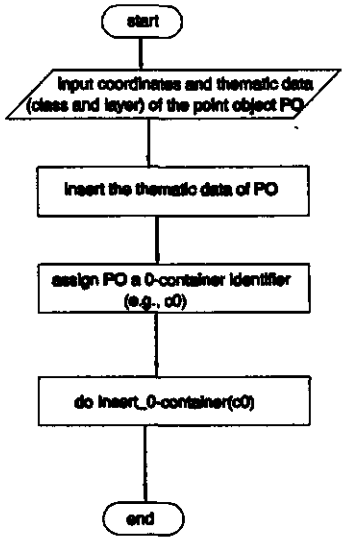
1.1.14 Delete 0-container



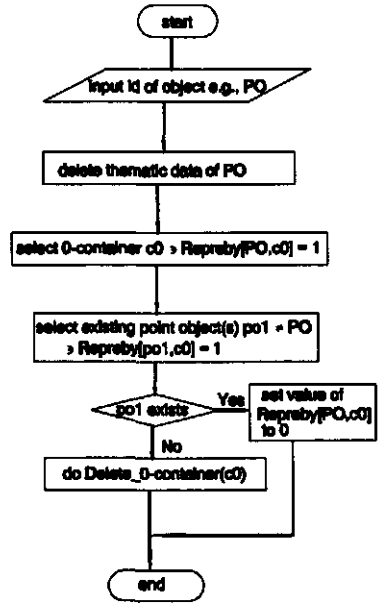
1.1.15 Modify 0-container



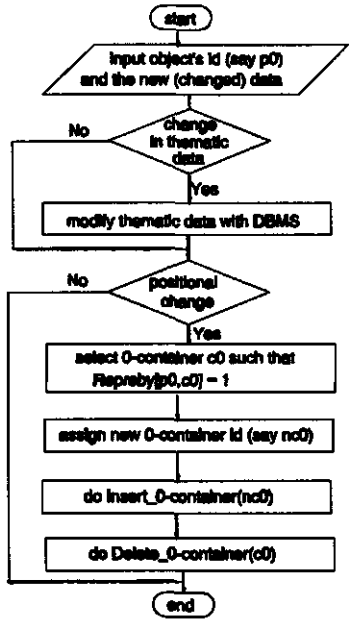
1.1.16 Insert Point



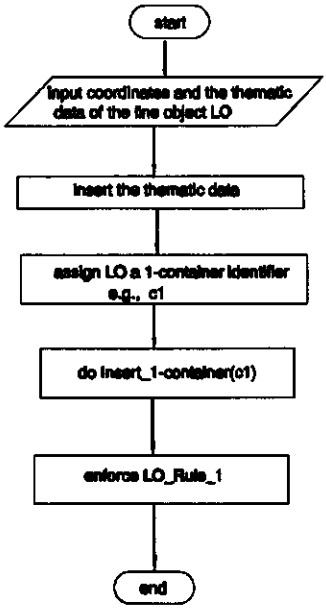
1.1.17 Delete Point



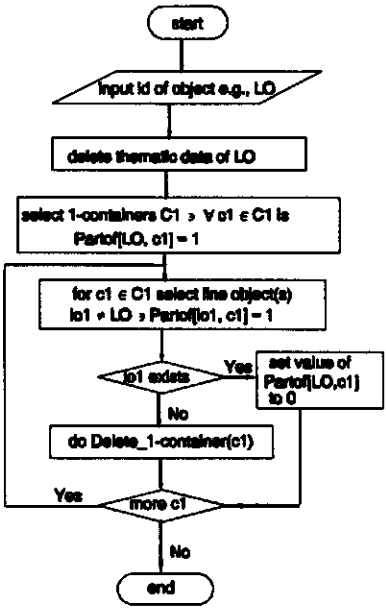
1.1.18 Modify Point



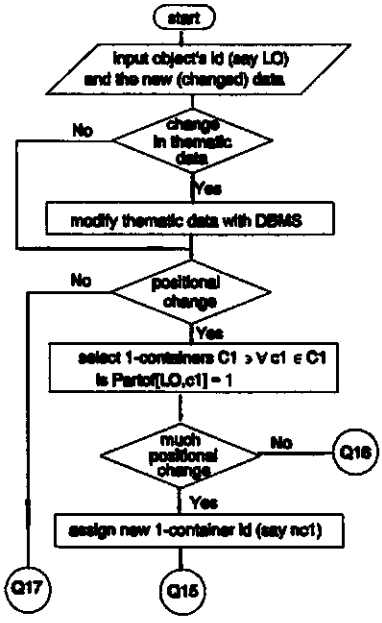
1.1.19 Insert Line

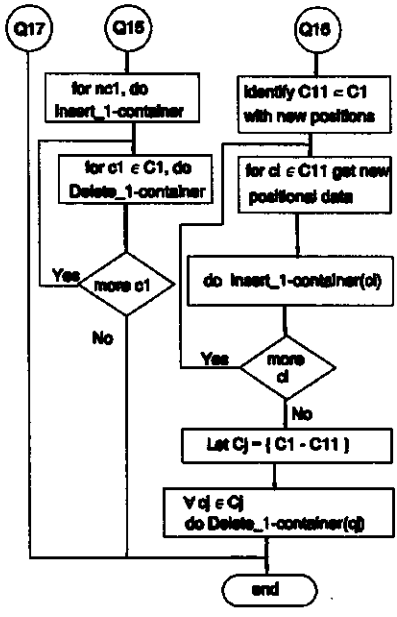


1.1.20 Delete Line

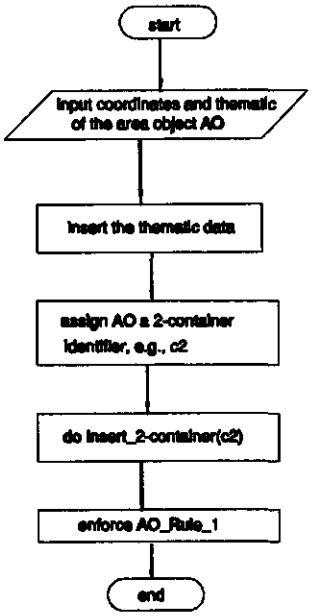


1.1.21 Modify Line

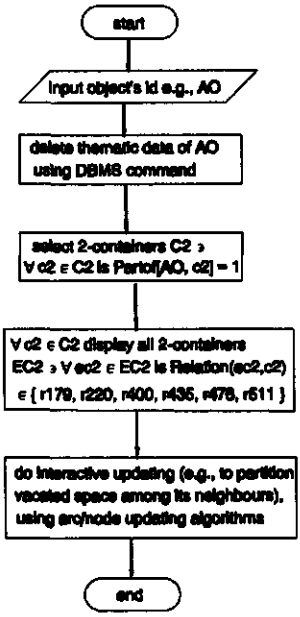




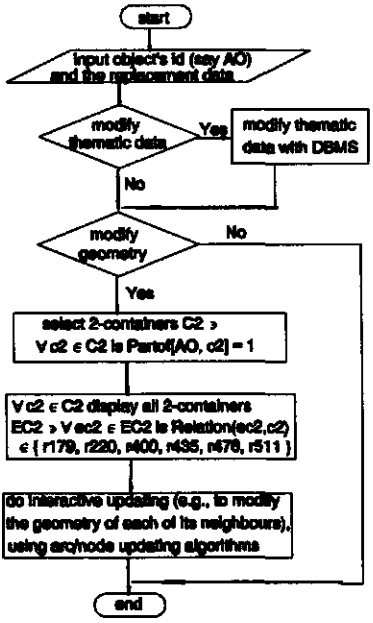
1.1.22 Insert Area



1.1.23 Delete Area



1.1.24 Modify Area



Appendix 1.2 Descriptions of Selected C and Postquel Functions Written for the Implementation (see chapters 6 & 7 for the algorithms)

Function Name	Language	Argument (input)	Output	Description
checkArcLoop	Postquel	arc class	null (if no violation) or arcs with snode = enode (if violation occurs)	consistency check for arc loop
checkDupArc	Postquel	arc class	null or duplicate arcs	consistency check for duplicate arcs
checkRedArc	Postquel	arc class	null or redundant arcs	consistency check for arcs that carry no information (dangling arcs) e.g. lftTwoC = rgtTwoC but aOneC = 0
mCgraph()	C	subgraphs of all 1-containers or 2-containers or line objects or area objects	identifiers and nodes of 1- or 2-containers or line or area object that violate graph structure	monitoring 1CC_-1/1CC_2 and 2CC_1 (see §6.-2.2)
insarc()	C	arc of new line or area object and all existing arcs (including coordinates)	property values (snode, enode, lftTwoC, rgtTwoC and aOneC) of new arc, and data file for updating database geometry	monitor and enforce GP_Rule_1 to GP_Rule-_12 (see Table 6.1)
arctopo()	C	properties (including coord.) of new arc; properties of all existing arcs	topologic relationship between new arc and an existing arc	computation of existing topologic relationship between a new and an existing arc using computational geometry
meet()	C	new arc and existing arcs	existing arc for which r287(new arc, existing arc) is True; 0 if none	computation of relation r287(new arc, existing arc)
arcint()	C	new arc and existing arcs	existing arc for which r063 or r159(new arc, existing arc) is True	computation of relations r063 and r159 between two arcs

ptin2c()	C	new node and existing 2-containers and their locational data	existing 2-container in which the new node lies	point-in-polygon computation
interlr()	C	new arc and the existing arc having relation r159	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_5 (see Table 6.1) and assigns property values to new arc
alternodelr()	C	new arc and the existing arc having relation r287	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_9 and assigns property values to new arc
equallr()	C	new arc and the existing arc having relation r400	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_10 and assigns property values to new arc
insidelr()	C	new arc and the existing arc having relation r179	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_6 and assigns property values to new arc
containslr()	C	new arc and the existing arc having relation r220	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_7 and assigns property values to new arc
indentlr()	C	new arc and the existing arc having relation r255 (indents)	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_8 and assigns property values to new arc
outdentlr()	C	new arc and the existing arc having relation r255 (outdents)	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_8 and assigns property values to new arc
coveredbylr()	C	new arc and the existing arc having relation r435	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_11 and assigns property values to new arc
coverslr()	C	new arc and the existing arc having relation r476	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_12 and assigns property values to new arc
toucheslr()	C	new arc and the existing arc having relation r063	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_3 and assigns property values to new arc

touchedbylr()	C	new arc and the existing arc having relation r095	property values of new arc; modified version(s) of existing arc	enforces GP_Rule_4 and assigns property values to new arc
r272nn()	C	coordinates of new point object and all existing nodes with degree 0	existing node for which r272 (new node, existing node) is True	computation of relation r272 between new node and existing nodes (with degree 0)
r092an()	C	new node and existing arcs	existing arc for which r092(existing arc,new node) is True; returns 0 if none	to compute r092(-arc,node) between a new node and an existing arc
arcnodecoord	Postquel	arc and node classes	G(N,A) of the map (database) including coordinates	to retrieve the arcs, nodes and coordinates in the database
euler	Postquel	area and arc classes	Number of component graphs of the map	to ascertain planar enforcement
delline	Postquel	id and layer of line object		to delete a given line object & propagate the update
inspt()	C	coordinates, object id, name, layer and class of the new point object	file containing property values with enforced consistency rules; the file is run as Postquel query file to propagate the update	insertion of new point object while maintaining consistency
insline()	C	as in inspt but for line object	as above	as above, for line object
insarea()	C	as above, for area object	as above	as above, for area object

Appendix 2 Samples from the Database

Arc Class

arc_id	snode	enode	lftTwoC	rgtTwoC	aOneC
967	1096	1097	7	7	92
968	1097	1098	7	7	92
970	186	1101	53	53	92
971	1101	1102	53	53	92
1070	679	1206	39	39	21
1071	1206	1207	39	39	21
1106	1244	1246	10	27	71
1107	1246	1247	10	27	71
1112	1252	1253	32	27	71
1155	1299	1300	30	30	71
1156	1300	1302	30	30	71
1157	1300	1304	30	30	91
1167	1314	1315	2	26	91
1	1	2	99	5	0
2	2	4	99	6	0
:	:	:	:	:	:
:	:	:	:	:	:

Area Class

twoC_id	aobjid	layer	aname	aclass
1	1	topo	FL1	farm_land
80	6	topo	FL6	farm_land
73	7	topo	FL7	farm_land
81	12	topo	F12	forest
82	12	topo	F12	forest
40	21	topo	MLF21	mixed_ld_for
30	72	topo	S72	built_up_area
82	1	soil	I1	little_weathered_nonclimatic_mineral_soils
84	1	soil	I1	little_weathered_nonclimatic_mineral_soils
9	11	soil	II11	slightly_developed_nonclimatic_soils
21	13	soil	II13	slightly_developed_nonclimatic_soils
31	22	soil	V22	calcimagnesic_soils
32	22	soil	V22	calcimagnesic_soils
16	32	soil	VII32	brown-earth_soils
63	33	soil	VII33	brown-earth_soils
24	41	soil	IX41	iron-sesquioxide-rich_soils
71	42	soil	IX42	iron-sesquioxide-rich_soils
53	51	soil	XI51	hydromorphic_soils
58	61	soil	C61	complex_units_and_associations
65	61	soil	C61	complex_units_and_associations
99	99	soil	O99	out_area
:	:	:	:	:
:	:	:	:	:

Line Class

oneC_id	lobjid	layer	lname	lclass
11	1	topo	railway1	railway
41	2	topo	river1	river
42	3	topo	river2	river
61	4	topo	N100	road
22	10	topo	D105	road
42	3	soil	river2	river
61	4	soil	N100	road
92	13	soil	D108	road
:	:	:	:	:
:	:	:	:	:

Linecross Class

upper1C	lower1C	crosspt	lower_ht
81	41	1033	149.89
81	11	1028	156.11
71	42	1303	135.33
61	42	1886	133.33
21	41	2051	137.33
11	41	2005	133.33
21	11	2048	139.11

Accuracy Class

ac_id	pl_acc	ht_acc
1	0.45	0.68

Pointf Class

zeroC_id	pobjid	layer	pname	pclass
1	1	topo	bridge1	rd_river_bridge
2	2	topo	bridge3	rd_river_bridge
12	12	soil	pt_sample180	point_sample
13	13	soil	pt_sample179	point_sample
14	14	soil	pt_sample178	point_sample
15	15	soil	pt_sample36	point_sample
:	:	:	:	:
:	:	:	:	:

Pointnode Class

zeroC	twoC	pnode
1	7	1033
2	30	1302
3	7	1886
4	7	2051
5	7	1028
:	:	:

Lineage Class

lin_id	pl_lin	ht_lin
1	The soil data were extracted from the 'Pedologique map of atlas No2 , South-Est Sector at scale of 1/50.000. The map was issued by the Ministry of Agriculture of France, 'Direction Departementale de l'Agriculture de Vaucluse' ,in 1978. The features of interest were initially digitized by using a 2D tablet digitizer and the control points used for coordinate transformation were taken from the topographic map of Cavaillon at 1/50.000 ,re--edited by IGN of France in 1988. For map projection, the ellipsoid of Clarke 1880, Lambert conique conform projection was used. The topographic layer informations (major land use/cover types) were extracted from aerial photograghs at scale of 1/30.000, which were taken in 1989, with a focal length of 152.00 mm. The two layers were superimposed and 3D digitizing of the outcome multivalued map was made in stereomode using Planicomp C120, with KORK-KDMS software, in August 1994.	The height information was extracted by using the phogrammetric techniques (during the stereocompilation) and from the same data sources(aerial photos. The flying height of the used photographs is about 4560 meters, and by using Planicomp C120 , the expected height accuracy is : 0.10 to 0.15 per mil of the flying height.

Node Class

node_id	xcoord	ycoord	zcoord	ac_id	lin_id
1	834800.875	179542.4375	268.890015	1	1
2	835732.6875	179587.671875	243.440002	1	1
4	835964.6875	179604.109375	227.110001	1	1
6	836144.6875	179609.109375	227.110001	1	1
2959	832303.875	175914.109375	199.220001	1	1
2960	832236.125	175768.4375	192.220001	1	1
2965	832232.4375	175682.671875	193.440002	1	1
2966	832226.000	175666.000	193.110001	1	1
2963	832226.125	175725.5625	192.000	1	1
2964	832229.6875	175703.78125	192.559998	1	1
2967	832215.125	175659.890625	193.440002	1	1
2974	832740.6875	178054.000	143.559998	1	1
2975	832740.4375	178080.4375	147.110001	1	1
:	:	:	:	:	:

Appendix 3 Some instances from the Line and Linecross after deleting an existing line object (railway1)

Instances of the Line class after the automatic deletion of the line object:

oneC_id	lobjid	layer	lname	lclass
41	2	topo	river1	river
42	3	topo	river2	river
61	4	topo	N100	road
21	5	topo	D106	road
31	6	topo	D109	road
71	7	topo	D145	road
81	8	topo	D36	road
91	9	topo	D60	road
22	10	topo	D105	road
32	11	topo	D218	road
72	12	topo	D104	road
92	13	topo	D108	road
41	2	soil	river1	river
42	3	soil	river2	river
61	4	soil	N100	road
21	5	soil	D106	road
31	6	soil	D109	road
71	7	soil	D145	road
81	8	soil	D36	road
91	9	soil	D60	road
22	10	soil	D105	road
32	11	soil	D218	road
72	12	soil	D104	road
92	13	soil	D108	road

Instances of the Linecross class showing that the system has deleted the property values of the line object:

upperlc	lowerlC	crosspt	lower_ht
81	41	1033	149.89
71	42	1303	135.33
61	42	1886	133.33
21	41	2051	137.33

Appendix 4 Some instances from the Pointf, Pointnode and Node classes after inserting new point object

Some instances of the Node class after inserting the point object (its node_id = 3181) by propagation:

node_id	xcoord	ycoord	zcoord	ac_id	lin_id
:	:	:	:	:	:
:	:	:	:	:	:
3137	837109.25	177420.890625	161.320007	1	1
3178	834357.4375	178006.5625	220.600006	1	1
3179	834526.75	177746.890625	217.160004	1	1
3180	835836.875	177542.890625	155.639999	1	1
3181	834835.6875	176707.046875	247.774994	1	1

Instances from Pointnode class after propagating the insertion of the point object (its zeroC_id = 16):

zeroC	twoC	pnode
1	7	1033
2	30	1302
3	7	1886
4	7	2051
5	7	1028
6	7	2048
7	7	2005
8	8	3038
9	9	3037
10	53	3041
11	7	3137
12	7	3180
13	28	3179
14	28	3178
15	7	3036
16	7	3181

Instances of the Pointf class after inserting a new point object:

zeroC_id	pobjid	layer	pname	pclass
1	1	topo	bridge1	rd_river_bridge
2	2	topo	bridge3	rd_river_bridge
3	3	topo	bridge4	rd_river_bridge
4	4	topo	bridge5	rd_river_bridge
5	5	topo	bridge2	rd_road_bridge
6	6	topo	bridge7	rd_road_bridge
:	:	:	:	:
:	:	:	:	:
12	12	soil	pt_sample180	point_sample
13	13	soil	pt_sample179	point_sample
14	14	soil	pt_sample178	point_sample
15	15	soil	pt_sample36	point_sample
16	79	topo	gps_station1	geodetic_controls

Appendix 5A Data of the new line object for insertion

Type of Object: Line
Geographic name: railway1
Thematic class: railway
Layer: topo
Geometry:

Segment#	Points	X	Y	Z
1	1	837592.250000	177319.671875	167.889999
	2	836953.750000	177149.781250	161.779999
2	1	836953.750000	177149.781250	161.779999
	2	836071.312500	176917.109375	156.110001
3	1	836071.312500	176917.109375	156.110001
	2	835466.437500	176753.000000	155.559998
4	1	835466.437500	176753.000000	155.559998
	2	834735.562500	176568.437500	149.559998
5	1	834735.562500	176568.437500	149.559998
	2	833341.312500	176821.000000	139.110001
6	1	833341.312500	176821.000000	139.110001
	2	832892.437500	176931.328125	138.779999
7	1	832892.437500	176931.328125	138.779999
	2	832645.750000	177072.328125	138.559998
8	1	832645.750000	177072.328125	138.559998
	2	832122.125000	177401.562500	134.559998

New 1-container Id: 93

Appendix 5B Some instances from the Line, Linecross, Arc and Node classes after inserting new line object

Instances of the Line class after inserting the line object (it has oneC_id = 93):

oneC_id	lobjid	layer	lname	lclass
41	2	topo	river1	river
42	3	topo	river2	river
61	4	topo	N100	road
21	5	topo	D106	road
31	6	topo	D109	road
71	7	topo	D145	road
81	8	topo	D36	road
91	9	topo	D60	road
22	10	topo	D105	road
32	11	topo	D218	road
72	12	topo	D104	road
92	13	topo	D108	road
41	2	soil	river1	river
42	3	soil	river2	river
61	4	soil	N100	road
21	5	soil	D106	road
31	6	soil	D109	road
71	7	soil	D145	road
81	8	soil	D36	road
91	9	soil	D60	road
22	10	soil	D105	road
32	11	soil	D218	road
72	12	soil	D104	road
92	13	soil	D108	road
93	1	topo	railway1	railway

Instances of the Linecross class after inserting "railway1" (it now has oneC_id = 93):

upper1C	lower1C	crosspt	lower_ht
81	41	1033	149.89
71	42	1303	135.33
61	42	1886	133.33
21	41	2051	137.33
81	93	1028	156.110001
21	93	2048	139.110001
93	41	2005	133.33

Some instances of the Arc class showing arcs of the new railway:

arc_id	snode	enode	lftTwoC	rgtTwoC	aOneC
2683	17	3182	7	7	93
2684	3182	1028	7	7	93
2685	1028	3185	7	7	93
2686	3185	3186	7	7	93
2687	3186	2048	7	7	93
2688	2048	3187	7	7	93
2689	3187	2005	7	7	93
2690	2005	3189	7	7	93

Instances of the Node class showing nodes of the new railway:

node_id	xcoord	ycoord	zcoord	ac_id	lin_id
17	837592.25	177319.671875	167.889999	1	1
1028	836071.3125	176917.109375	160.440002	1	1
2005	832645.75	177072.328125	138.559998	1	1
2048	833341.3125	176821.000	143.889999	1	1
3182	836953.75	177149.78125	161.779999	1	1
3185	835466.4375	176753.000	155.559998	1	1
3186	834735.5625	176568.4375	149.559998	1	1
3187	832892.4375	176931.328125	138.779999	1	1
3189	832122.125	177401.5625	134.559998	1	1

Appendix 6A Data of the new area object for insertion

Object Name: Potato_farm
Thematic class: Farm_land
Layer: Topo
Object_Identifier 74
Locational data:

Segment#	Point	X	Y	Z
1	1	837350.5000	179500.050	189.127
	2	837512.0000	179486.225	193.400
2	1	837512.0000	179486.225	193.400
	2	837568.1150	178205.515	184.005
3	1	837568.1150	178205.515	184.005
	2	837357.0080	178425.375	184.113
4	1	837357.0080	178425.375	184.113
	2	837350.5000	179500.050	189.127

Appendix 6B Some instances of the Area, Arc and Node classes after inserting the new area object

Area Class showing addition of 2-containers 86 and 87 after inserting new area object "potato_farm":

twoC id	aobjid	layer	aname	aclass
81	12	topo	F12	forest
82	12	topo	F12	forest
84	1	soil	I1	little_weathered_nonclimatic_mineral_soils
83	23	soil	V23	calcimagnesic_soils
81	43	soil	IX43	iron-sesquioxide-rich_soils
86	32	soil	VII32	brown_earth_soils
87	21	soil	V21	calcimagnesic_soils
85	11	topo	F11	forest
85	61	soil	C61	complex_units_and_associations
86	74	topo	potato_farm	farm_land
87	74	topo	potato_farm	farm_land

Some instances from the Arc class showing arcs related to the inserted area object:

arc id	snode	enode	lftTwoC	rgtTwoC	aOneC
2693	3192	3194	16	86	0
2696	3194	3195	16	86	0
2697	3195	3197	16	86	0
2700	3197	233	87	86	0
191	233	234	87	86	0
192	234	235	87	86	0
193	235	236	87	86	0
2694	236	3192	87	86	0
2691	3190	3191	8	87	0
2692	3191	3192	8	87	0
2698	3197	3190	8	87	0
2700	3197	233	87	86	0
191	233	234	87	86	0
192	234	235	87	86	0
193	235	236	87	86	0
2694	236	3192	87	86	0

Instances from the Node class showing nodes and locational data of the "potato_farm":

node id	xcoord	ycoord	zcoord	ac id	lin id
233	837376.3125	179142.5625	185.669998	1	1
234	837448.5625	179144.328125	183.559998	1	1
235	837485.4375	179161.000	190.440002	1	1
236	837509.6875	179176.78125	192.669998	1	1
3190	837350.5	179500.046875	189.126999	1	1
3191	837512.000	179486.21875	193.399994	1	1
3192	837523.689453	179198.650391	193.786942	1	1
3194	837568.125	178105.515625	184.005005	1	1
3195	837357.000	178225.375	184.113007	1	1
3197	837352.307129	179145.720703	187.733231	1	1

Bibliography

Ahuja, N. (1983): On approaches to polygonal decomposition for hierarchical image decomposition. Computer Vision, Graphics & Image Processing, 24, pp 200-214.

Alagic, S. (1989): Object-oriented database programming. Springer-Verlag, New York, 320pp

Alfred, C. (1994): Maximizing leverage from an object database. IBM Systems Journal, 33, 2, pp 280-299.

Aronoff, S. (1989): Geographic Information Theory: A management perspective. WDL Publications, Ottawa, Canada, 294 pp

Ayugi, S.W.O. (1992): The multivalued vector map. Unpublished M.Sc Thesis, ITC, Enschede, 104 pp.

Beech, D. (1988): A foundation for evolution from relational to object databases. Lecture Notes in Computer Science, 303, Springer-Verlag, New York, pp 252-270.

Benoit, S. (1990): Digital spatial capture and revision - a photogrammetric solution for the 1990's. Proc. 1st European Conference on Geographical Information Systems, Amsterdam.

Boudriault, G. (1987): Topology in the Tiger file. Proc. Autocarto 8, Falls Church, pp. 258-269.

Bouloucos, T., Kufoniyi, O. and M. Molenaar (1990): A relational data structure for single-valued vector maps. Progress in data analysis, ISPRS Com III, Wuhan, pp 64-74.

Bouloucos, T., Kunarak, R. and K. Tempfli (1992): Low-cost feature extraction from aerial photographs for database revision. Int. Arch. of Photogr. and Rem. Sen. 29, Part B4, pp 493-498.

Bouloucos, T., Ayugi, S.W.O. and O. Kufoniyi (1993): A data structure for multivalued vector maps. Proc. EGIS '93, Genoa, Italy pp 237-245

Bouloucos, T., Chhatkuli, R.R. and O. Kufoniyi (1994): A multi-theme vector data structure for modelling data quality in spatial databases. Int. Arch. of Photo. and Rem. Sen., 30 Part 4 pp 649-656.

Boursier, P. and Faiz, S. (1993): A comparative study of relational, extensible and object-oriented approaches for modelling and querying geographic databases. Proc 4th European Conference and Exhibition on Geographical Information Systems, Genoa, Italy, pp 764-772.

Braun, J. (1989): Current status of the Phocus development. Proc. 42nd Photogrammetric Week, Stuggart Univ., 13, Stuggart.

Burrough, P.A. (1986): Principles of geographical information systems for land resources

assessment. Claredon Press, Oxford, 194pp

Chance, A, Newell, R.G. and D.G. Theriault (1990): An object-oriented GIS - issues and solutions. Proc European Conference on Goegraphical Information Systems, Amsterdam, pp 179-188.

Chen, P.S. (1976): The entity relationship model: Towards a unified view of data. ACM Transactions on Database Systems, 1, 1, pp 9-36.

Chhatkuli, R.R. (1993): Modelling data quality parameters in a multiple-theme vector data structure and its implementation in a geographic information system. Unpublished M.Sc Thesis, ITC, Enschede, 129pp.

Chung, K.L., et al. (1988): Process management and assertion enforcement for a semantic data model. Lecture notes in computer science, 303, Springer-Verlag, New York.

Clementini, E., Felice, P.D. and P. van Oosterom (1993): A small set of formal topological relationships suitable for end-user interaction. Lecture Notes in Computer Science, 692, Springer-Verlag, New York, pp 277-295.

Codd, E.F. (1970): A relational model for large shared data banks. Communications of the ACM, 13, 6, pp 377-387.

Cuthbert, A. (1991): GIS need structure not topology. Proc. Second European Conference on Geographical Information Systems, Brussels, Belgium.

Date, C.J. (1990): An introduction to database systems. Vol 1, 5th ed. Addison-Wesley, Reading, 854 pp.

DCDSTF (Digital Cartographic Data Standards Task Force), (1989): The American Cartographer, 15, 1, pp 9-140.

Doyle, F.J. (1978): Digital Terrain Models: An Overview. Photogrammetric Engineering and Remote Sensing 44, 12, pp 1481-1485.

Ebner, H., Hössler, R. and R. Würländer (1990): Integration of an efficient DTM program package into Geographical Information Systems. Int. ArchPhRS, 28 Part 4, pp 116-121.

Ebner, H. and K. Eder (1992): State-of-the-art in digital terrain modelling. Proc. EGIS '92, Munich.

Eck, J.W. van and M. Uffer (1990): A presentation of System 9. P&LIS 90, pp 139-178.

Egenhofer, M.J. (1991): Extending SQL for graphical display. Cartography and Geographic Information Systems, 18, 4.

Egenhofer, M.J. (1992): Why not SQL! Int. Journal of Geographical Information Systems, 6, 2, pp 71-85.

Egenhofer, M.J. and A.U. Frank (1988): Designing object-oriented query languages for GIS: human interface aspects. Proc 3rd Int. Symp. on Spatial Data Handling, Sydney, Australia, pp 79-96.

Egenhofer, M.J. and A.U. Frank (1989): Object-oriented modelling in GIS: inheritance and propagation. AUTOCARTO 9, Baltimore pp 588-598.

Egenhofer, M.J. and J.R. Herring (1990): A mathematical framework for the definition of topological relationships. Fourth International Symposium on Spatial Data Handling, Zurich, Switzerland, pp 803-813.

Egenhofer, M.J. and J. Herring (1991): High-level spatial data structures for GIS. In: D.J. Maguire, M.F. Goodchild and D.W. Rhind (Eds.) Geographical Information Systems Principles and Applications, Longman Scientific & Technical, New York, pp 227-237.

Egenhofer, M.J. and R.D. Franzosa (1991): Point-set topological spatial relations. Int. Journal of Geographical Information System, 5, 2, pp 161-174.

Egenhofer, M.J. and J.R. Herring (1992): Categorizing binary topological relationships between regions, lines, and points in geographic databases. Technical report, Department of Surveying Engineering, University of Maine, Orono 59pp.

Egenhofer, M.J. and K.K. Al-Taha (1992): Reasoning about gradual changes of topological relationships. In: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, A.U.Frank, I. Campari and U. Formentini (Eds.), Springer-Verlag, Berlin pp 196-219.

Ehlers, M. et al (1991): Integration of remote sensing and GIS: data and data Access. PE & RS, 57, 6.

Ehlers, M., Edwards, G. and Y. Bédard (1989): Integration of remote sensing with geographic information systems: a necessary evolution. Photogrammetric Engineering and Remote Sensing, 55, 11, pp 1619-1627.

Essayah, E. H. (1994): Object oriented multi-theme vector data structure and its implementation. Unpublished M.Sc Thesis, ITC, Enschede 99 pp

Faust, N.L. et al. (1991): Geographic information systems and remote sensing future computing environment. PE & RS, 57, 6.

Felgueiras, C.A., Erthal, G.J. and L.A.V. Dias (1988): A digital terrain model system for a microcomputer. Com. III ISPRS, Kyoto.

Frank, A.U. (1984): Requirements for database systems suitable to manage large spatial databases. Proc. Int. Symp. on Spatial Data Handling, Zurich, pp 38-60.

Frederiksen, P. (1992): Updating elevation data bases - merging old and new data. Com IV ISPRS, Washington.

Freiling, M.J. (1982): Understanding database management. Alfred Publishing Co. Inc., Sherman Oaks.

Fritsch, D. (1991): Raumbezogene Informationssysteme und Digitale Gelandemodelle. Series C Nr. 369, Munich

Gatrell, A.C. (1991): Concepts of space and geographical data. In: D.J. Maguire, M.F. Goodchild and D.W. Rhind (Eds.) Geographical Information Systems Principles and Applications, Longman Scientific & Technical, New York, pp 119-134.

Giblin, P.J. (1977): Graphics, surfaces and homology. Chapman and Hall, London.

Goodchild, M.F. (1992): Geographical data modelling. Computers and Geosciences, 18, 4, pp 401-408.

Hadzilacos, T. and N. Tryfona (1992): A model for expressing topological integrity constraints in geographic databases. In: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, A.U. Frank, I. Campari, U. Formentini (Eds.), Springer-Verlag, London, pp 252-268.

Herring, J.R. (1987): TIGRIS: Topologically integrated geographic information system. Proc Autocarto 8, Falls Church, pp 282-291.

Hesse, W. and F.J. Leahy (1990): Authoritative topographic-cartographic information system (ATKIS). The Project of the State Survey Authorities for the Creation of Digital Landscape Models and Digital Cartographic Models, Landesvermessungamt Nordrhein-Westfalen, Bonn, 29 pp.

Höhle, J. (1991): Digital orthophotos and object-oriented height models in GIS. Proc. AM/FM conf., Mariehamn.

Höhle, J. (1992): The object-oriented height model and its application. Int. Archives of Photogrammetry and Rem. Sensing, 29, Part B4, pp 869-873.

Hoop, S. de and P. van Oosterom (1992): The storage and manipulation of topology in Postgres. Proc. EGIS '92, Munich, Germany, pp 1324-1336.

Hoop, S. de, Meij, L. v.d. and M. Molenaar (1993): Topological relationships in 3D vector maps. 4th European Conference and Exhibition on Geographical Information System, Genoa, Italy, pp 448-455.

Hoop, S. de, Oosterom, P. van and M. Molenaar (1993): Topological querying of multiple map layers. Proc COSIT 93, Elba Island, Italy

Hughes, J.G. (1991): Object-oriented databases. Prentice Hall, New York, 268pp.

Illustra Information Technologies Inc., (1994): Illustra object-relational database management system. Oakland, California 6 pp.

Jianya, G. (1990): Object-oriented models for thematic data management in GIS. Proc European Conference on Geographical Information Systems, Amsterdam, pp 494-503.

Kainz, W. (1990): Spatial relationships - topology versus order. Proceedings Fourth International Symposium on Spatial Data Handling, Zurich, Switzerland pp 814-819.

Kemppainen, H.H. (1992): Handling integrity constraints of complex objects in spatial databases. In: International Archives of photogrammetry and Remote Sensing, Com. III, 29, Part B3, pp 456-464.

Kemppainen, H. (1994): Modelling update propagation in spatial databases. Int. Arch. of Photogr. and Rem. Sen.Vol 30 Part 4 pp 625-633.

Kent, W. (1983): A simple guide to five normal forms in relational database theory. Communications of the ACM, 26, 2 pp 120-125

Kidner, D. and C. Jones (1991): Implicit triangulations for large terrain databases. Proc. EGIS '91, Brussels.

Kork System Inc. (1992): Kork digital mapping system manual version 8.0., Bangor, U.S.A.

Kotz, A.M. et al. (1988): Supporting semantic rules by a generalised event/trigger mechanism. Lecture notes in computer science, 303, Springer-Verlag, New York.

Kufoniyi, O. (1989): Editing of topologically structured data. Unpublished M.Sc Thesis, ITC, Enschede 113pp

Kufoniyi, O., M. Molenaar, T. Bouloucos, (1993): Topologic editing of relationally structured single valued vector maps. ITC Journal, 1993-4, pp 332-339.

Kufoniyi, O., and T. Bouloucos (1994): Flexible integration of terrain objects and DTM in vector GIS. In: Proc. Int. Colloquium on Integration, Automation and Intelligence in Photogrammetry, Remote Sensing and GIS, Wuhan, pp 111-122.

Kufoniyi, O., T. Bouloucos and M. Molenaar (1994): An integrated model for multi-layer vector maps and DTM. Int. Arch. of Photo. and Rem. Sen., Vol. 30 Part 4 pp 664-671.

Kufoniyi, O., and M. Pilouk (1994): A vector data model for integrated multitheme and relief geoinformation. In: Advances in GIS Research Proceedings, 6th Int. Symposium on Spatial Data Handling (SDH 94), 1, pp 1061-1071.

Kufoniyi, O., M. Molenaar, T. Bouloucos, (1994): Topology as a tool for consistency operations in vector maps. ISPRS Com. III Symposium, Munich.

Lemmens, M.J.P.M. (1991): GIS: The data problem. Proc. Second European Conference on Geographical Information Systems, Brussels, Belgium.

Lemmens, M.J.P.M. (1990): Strategies for the integration of raster and vector data. Proc. 1st

European Conference on Geographical Information Systems, Amsterdam.

Liu, C.L. (1986): Elements of discrete mathematics. McGraw-Hill Book Co., New York pp 137-173.

Makarovic, B. (1988): Differential Terrain Relief Modelling. Technical Seminar of IPGH, Bogota.

Mantyla, M. (1988): An introduction to solid modeling. Computer Science Press, Maryland.

Mark, D. and M.J. Egenhofer (1994): Modelling Spatial Relations between Lines and Regions: Combining Formal Mathematical Models and Human Subjects Testing. National Center for Geographic Information and Analysis, Technical Report 94-1.

Molenaar, M. (1989): Single valued vector maps - a concept in GIS. Geo-Informations-Systeme 2, 1, pp 18-26.

Molenaar, M. (1990): A formal data structure for three-dimensional vector maps. Proc 4th Int. Symp. on Spatial Data Handling, Zurich, pp 830-843.

Molenaar, M. (1991a): Terrain Objects, data structures and query spaces. In: M. Schilcher (ed), Geo-Informatik, Siemens Nixdorf Informationsysteme AG, Munchen.

Molenaar, M. (1991b): Status and problems of geographical information system. ISPRS Journal of Photogrammetry and Remote Sensing, 46, 2, pp 85-103

Molenaar, M. (1991c): Formal data structures, object dynamics and consistency rules. In: Digital Photogrammetric Systems eds. Ebner/Fritsch/Heipke; Wichmann, Karlsruhe.

Molenaar, M. (1992): A topology for 3D vector maps. ITC Journal, 1992-1, pp 25-33.

Molenaar, M. (1993): Object Hierarchies and Uncertainty in GIS or why is Standardisation so Difficult. Geo-Informations-Systeme, 6, 4, pp 22-28.

Molenaar, M. and D. Fritsch (1991): Combined data structures for vector and raster representations in Geographic Information Systems. Geo-Informations-Systeme, 4, 3, pp 26-32.

Molenaar, M. and L.L.F. Janssen (1992): Integrated processing of remotely sensed and geographic data for land inventory purposes. Proc. ISPRS Conference XVII, Washington.

Molenaar, M., Kufoniyi, O. and T. Bouloucos (1994): Modelling Topological Relationships in Vector Maps. In: Advances in GIS Research Proceedings, 6th Int. Symposium on Spatial Data Handling (SDH 94), 1, pp 112-126.

Mortenson, M.E. (1985): Geometric Modeling. John Wiley & Sons, New York.

Nagy, G. and S. Wagle (1979): Geographic data processing. Comput. Surv., 11, pp 139-181.

Nassif, R., Y. Qiu, and J. Zhu (1991): Extending the object-oriented paradigm to support relationships and constraints. In: Object-Oriented Databases: Analysis, Design & Construction (DS-4), R.A.Meersman, W. Kent and S. Khosla (eds.), North-Holland, Amsterdam pp 305-329.

Newby, P.R.T. (1994): Revision policy and practice for Great Britain's topographic database. Int. Arch. of Photog. and Rem. Sensing, 30, Part 4, pp 260-267.

Oosterom, P. van (1990): Reactive data structures for geographic information systems. Doctoral dissertation, Department of Computer Science, Leiden University, Leiden, The Netherlands, 197 pp

Oxborrow, E. and Z. Kemp (1989): An object-oriented approach to the management of geographical data. Proc. Managing Geographical Information Systems and Databases, Lancaster University, 12 pp.

Peled, A.(1994): Revision of Digital Maps and GIS databases. Int. Arch. of Photog. and Rem. Sensing, 30 Part 4, pp 268-272

Peucker, T. and N. Chrisman (1975): Cartographic data structures. The American Cartographer, 2, pp 55-69.

Peuquet, D.J. (1984): A conceptual framework and comparison of spatial data models. Cartographica 21, 4, pp 66-113.

Pigot, S. (1991): Topological models for 3D spatial information systems. Proc. 1991 ACSM-ASPRS AUTO-CARTO 10, Baltimore, 6

Pilouk, M. and K. Tempfli (1993): An integrated DTM-GIS data structures: a relational approach. AUTOCARTO 11, pp 278-287.

Pilouk, M. and O. Kufoniyi (1994): A relational data structure for integrated DTM and multitheme GIS. Int. Arch. of Photo. and Rem. Sen., Vol. 30 Part 3/2 pp 670-677.

Postgres Group (1994). The Postgres user manual. Computer Science Division, Dept. of EECS, University of California at Berkeley, 81pp.

Proctor, D.W. and P.R.T. Newby (1988): Revision of large scale maps at the Ordnance Survey. Int. Arch. of Photogr. and Rem. Sensing, 27, Part 4, pp 298-307.

Pullar, D.V. and M.J. Egenhofer (1988): Toward formal definitions of topological relations among spatial objects. Proc. 3rd Int. Symp. on Spatial Data Handling, Sydney, Australia pp 225-241.

Radwan, M.M. et al (1991): Guidelines for a digital database for topographic and terrain relief information at Bakosurtanal, Indonesia. ITC Journal, 1991-3.

Richardson, D.E. (1993): Automated spatial and thematic generalization using a context

transformation model. R & B Publications, Canada, 149 pp.

Roessel, J.W.van (1986): Design of a spatial data structure using the relational normal forms. Proc. 2nd Int. Symp. on Spatial Data Handling, Seattle pp 251-272.

Roessel, J. van and D. Pullar (1993): Geographic regions: a new composite GIS feature type. AUTOCARTO 11, Minneapolis, pp 145-156.

Rowe, L. and M. Stonebraker (1987). The Postgres data model. Proc. 1987 VLDB Conference, Brighton, England.

Salge, F. and M.N. Sclafer (1988): The IGN cartographic database, from the users' needs to the relational structure. Proc. Eurocarto seven, Enschede.

Samet, H. (1989): The applications of spatial data structures. Addison-Wesley, Reading, 507 pp.

Sandgaard, J. (1988): Integration of a GIS and a DTM. Proc. XVI Int. Congress of Photogrammetry and Remote Sensing, Com IV, Kyoto.

Servigne, S. and R. Laurini (1994): Detecting changes in geographic database using aerial photos. Int. Arch. of Photogr. and Rem. Sensing, 30, part 4 pp 318-324.

Silver, J. (1978): GBF/DIME system - an overview. Proc. 1st Int. Adv. Study Symp. on Topological Data Structures (ed. G. Dutton), Laboratory for computer graphics and spatial analysis, Harvard.

Smallworld Systems (1991): Smallworld Magik Overview. Smallworld Systems Limited, Cambridge. 13 pp.

Smith, H.C. (1985): Database design: composing fully normalised tables from a rigorous dependency diagram. Communications of the ACM, 28, 8, pp 826-838.

Stonebraker, M. (1994): Object-relational data base systems. Illustra Information Technologies Inc., Oakland, California 6 pp

Tsai, V.J.D. (1991): PC photogrammetric graphic system and its bridge to GIS and CADD. Proc. 1991 ACSM-ASPRS Annual Convention Baltimore, 5

U.S. Dept. of Commerce, Bureau of the Census (1970): The DIME geocoding system. In: Report No 4, Census Use Study.

Vaidyanathaswamy, R. (1960): Set Topology. Chelsea Publishing Company, New York, 2nd Ed.

Vijlbrief, T., and van Oosterom, P. (1992): The GEO++ System: An extensible GIS. Proc 5th Int. Symposium on Spatial Data Handling, Charleston, South Carolina, pp 40-50.

Webster, C. (1990): The object-oriented paradigm in GIS. Progress in Data Analysis, Com. III ISPRS, Wuhan, China, 37 pp

Webster, C.J. and C.N. Omare (1991): A formal approach to object oriented spatial database design. Proc. Second European Conference on Geographical Information Systems, Brussels.

Welch, R., (1989): Desktop mapping with personal computers. Photogr. Eng. & Rem. Sensing, 55, 11, pp 1651-1662

Wilson, R.J. (1990): Introduction to graph theory. 3rd. Edition, Longman Scientific & Technical, England, pp 59-78.

Wintraecken, J.J.V.R. (1985): Informatie-analyze volgens MIAM. Control data Academic Service, Den Haag.

Woelk, D. and W. Kim (1987): Multimedia information management in an object-oriented database system. 13th VLDB conference, Brighton, England.

Worboys, M.F. (1992): A generic model for planar geographical objects. Int Journal of Geographical Information Systems, 6, 5 pp 353-372

Worboys, M.F., H.M. Hearnshaw and D.J. Maguire (1990): Object-oriented data modelling for spatial databases. Int. Journal of Geographical Information Systems, 4, 4, pp 369-383.

Zdonic, S. and D. Maier (1990): Fundamentals of object-oriented databases. In: Readings in Object-Oriented Database Systems, Morgan-Kaufman, San Mateo, California.

De modellering van samenvallende ruimtelijke objecten, de geautomatiseerde bijhouding van databases en data consistentie in een vector GIS.

Samenvatting

In deze dissertatie zijn formele procedures ontwikkeld voor de automatische controle van dataconsistentie en voor het bijhouden van databases in een vectorgestructureerd GIS. Het kader daarvoor wordt gegeven door een conceptueel datamodel voor de representatie van een terreinbeschrijving in meerdere lagen in een vectorstructuur. Om het model en de ontwikkelde concepten te testen en om de implementatie mogelijkheden te onderzoeken, is het model vertaald naar relationele en objectgestructureerde gegevensstructuren.

Geografische informatie systemen (GIS) geven de mogelijkheid voor ruimtelijke gegevensverwerking voor een breed scala van toepassingen. Het hart van zo'n systeem is de ruimtelijke database, daarin wordt de wereld gerepresenteerd zoals die vanuit de toepassingen gezien wordt. Deze database moet goed gestructureerd en consistent zijn. Bij het ontwerp van de database worden de gegevens op vier abstractie niveaus bekeken: de werkelijkheid, het conceptuele datamodel, de datastructuur en de filestructuur. De wijze waarop de werkelijkheid in een conceptueel datamodel wordt afgebeeld hangt meestal sterk van de toepassing af. Ruimtelijke analyse en planning vereisen meestal dat verschillende visies op de wereld geïntegreerd worden, dit betekent meestal de integratie van verschillende "kaartlagen". De term "kaartlaag" wordt hier gebruikt voor een verzameling geo-data, die een bepaald aspect van de wereld beschrijven. Tegenwoordig wordt meestal iedere laag apart gestructureerd, integratie wordt dan via overlays gerealiseerd. Deze oplossing leidt vaak tot een toename van de overheadkosten voor herhaalde ad-hoc overlay berekeningen (tijdens query bewerkingen), bovendien is het moeilijk om in dit geval van tevoren relaties te definiëren tussen objecten uit verschillende lagen (verticale topologie).

In deze dissertatie wordt een alternatief gegeven met een conceptueel model voor een vectorrepresentatie voor meerwaardige terreinabstracties, vooral wanneer veelvuldig ruimtelijke analyses worden uitgevoerd over meerdere lagen. Dit model is een object gestructureerd 2.5D datamodel voor meerwaardige vectorkaarten (DMMVM); dit is een vectorgestructureerde representatie van terrein objecten uit meerdere kaartlagen, waarbij objecten van hetzelfde geometrische type kunnen samenvallen, i.e., elkaar overlappen. In dit model is de positie van objecten in 3D coördinaten gegeven, terwijl hun topologie door een 2D vlakken graaf beschreven wordt; dit geeft een 2.5D model. Het model is gebaseerd op de 2D formele datastructuur (FDS) voor enkelwaardige vectorkaarten (Molenaar, 1989). In de FDS spelen terrein objecten een centrale rol; ieder object heeft een thematische- en een geometrische beschrijvingscomponent. Thematisch gezien worden objecten ingedeeld in klassen, waarbij iedere klasse een eigen specifieke attribuutstructuur heeft. Geometrisch worden de punt-, lijn- en vlakobjecten onderscheiden, de geometrie van deze objecten kan worden uitgedrukt in knooppunten en zijden waarvan de samenhang volgens de regels van de grafentheorie kan worden bestudeerd. De FDS werd in deze dissertatie uitgebreid zodat objecten van hetzelfde type elkaar kunnen overlappen, zodat geodata uit meerdere kaartlagen in een structuur gebracht kunnen worden. Dit gebeurde door de introductie van het begrip m-container (m duidt de dimensie aan). Een 0-container representeert samenvallende puntobjecten uit meerdere lagen, een 1-container samenvallende lijnobjecten en een 2-

container samenvallende vlakobjecten. Voor vlakken worden altijd alle lagen betrokken, bij punt- en lijnobjecten kunnen dat minder lagen zijn. Via de containers worden de samenvallende delen van objecten geïdentificeerd, deze delen kunnen hun eigen geometrie en thematiek hebben naast die van de oorspronkelijke objecten. Ze kunnen door de database als aparte (sub)objecten behandeld worden, zodat ze in aggregatie en associatie operatie gebruikt kunnen worden.

De geometrie van de containers wordt ook beschreven door middel van knooppunten en zijden: een knooppunt beschrijft een 0-container en/of het begin of einde van een zijde, terwijl een zijde een deel kan zijn van een 1-container en tegelijkertijd van de grens van een 2-container. Een zijde wordt gedefinieerd door de begin- en eindknooppunten, een knooppunt heeft een positie uitgedrukt in X,Y,Z-coördinaten. De geometrie van een kaart wordt beschreven door een vlakke graaf G(N,A), daarin is A de verzameling van alle zijden en N van alle knooppunten. Iedere m-container C wordt beschreven door een subgraaf van G_c van G met $G_c(N_c,A_c)$ met N_c is een deelverzameling van N en A_c is een deelverzameling van A. Voor een 0-container is A_c leeg.

De geo-data in multi-kaartlagen worden dus beschreven door acht basistypes van geometrische gegevens: punt-, lijn- en vlakobjecten, 0-, 1- en 2-containers en knooppunten en zijden. Ieder gegevenstype speelt haar eigen rol in het datamodel. Net als in de FDS representeren de punten, lijnen en vlakken de terreinobjecten, ieder object kan op een of meer van deze typen worden afgebeeld, de m-containers worden gebruikt om overlappende delen van objecten te beschrijven, terwijl de feitelijke geometrische beschrijving van de objecten wordt uitgedrukt in de knooppunten en zijden.

In dit model konden DTM's geïntegreerd worden via een op zijden (edges) gebaseerde TIN representatie, de twee geometrische primitieven van TIN, te weten edges en vertices werden aan de acht datatypen van het DMMVM toegevoegd. De topologische relaties tussen de punten, lijnen, vlakken, knooppunten en zijden werden geformaliseerd. Daarna werden algoritmen gedefinieerd voor het opsporen van de mogelijke elementaire relaties tussen objecten. Deze algoritmen kunnen vertaald worden in topologische operatoren voor het uitvoeren van topologische bevragingen, maar ze vormen ook het gereedschap voor het opleggen van topologische voorwaarden aan de gegevens. Als de topologische operatoren gebruikt worden voor het handhaven van consistentie regels, dan kunnen er inconsistentie mee opgespoord worden. Ingeval van inconsistentie activeert de operator de relevante regels waarmee de consistentie hersteld wordt.

De DMMVM werd vertaald naar twee types van database-structuren, het relationele en een objectgestructureerd model. Het prototype in het relationele model werd genormaliseerd met de methode van (Smith, 1985). Zeven basistabellen werden gedefinieerd, waarin experimentele implementaties werden gerealiseerd om de bruikbaarheid van het multi-lagen model te toetsen. De consistentie en bijhoudingsoperaties werden in C geprogrammeerd en aan de RDBMS gekoppeld. Het relationele model kan direct geïmplementeerd worden omdat er vele RDBMSen beschikbaar zijn, dat is minder het geval voor het object-georiënteerde model.
Omdat relationele databases vele tekortkomingen hebben voor het behandelen van ruimtelijke gegevens, werd het model ook geïmplementeerd in een object-gestructureerde database. Hierbij werden negen objectklassen gedefinieerd, voor ieder van de acht genoemde geometrische elementtypen één plus één voor het onderscheid tussen kruisende en snijdende

lijnobjecten. De consistentie en bijhoudingsoperaties werden als aan deze klassen verbonden methoden behandeld.

Dit onderzoek richtte zich ook op gegevensconsistentie in ruimtelijke databases. Er zijn twee typen van consistentie, de dynamische en de statische. Hier werd vooral gekeken naar de statische consistentie, welke uitgesplitst kan worden in structurele en semantische consistentie. Er werden regels geformuleerd om structurele voorwaarden te handhaven, terwijl een strategie werd voorgesteld voor de monitoring van de semantische regels. In beide gevallen speelt de topologie een belangrijke rol voor het opsporen van inconsistenties. Deze dissertatie toonde aan dat de consistentie regels vertaald kunnen worden in topologische voorwaarden voor de afzonderlijke objecten en topologische relaties tussen paren van objecten. Overtredingen kunnen in de database opgeslagen worden als "events" en de overeenkomstige respons als "actie"; hiermee kan een rulebased procedure ontwikkeld worden met de conventie "IF event THEN actie".

In de dissertatie werden procedures ontwikkeld op basis van het DMMVM voor de consistente automatische bijhouding van vectorbestanden. Hierbij werden algoritmen ontwikkeld om bijhoudingsoperaties automatisch zo te laten doorwerken dat topologische relaties consistent blijven. Hierdoor wordt de huidige praktijk verbeterd, waarbij de systemen later een herstelslag moeten uitvoeren als de geometrie in de database is bijgewerkt. Er zijn algoritmen ontwikkeld voor het invoegen, verwijderen en veranderen van alle acht geometrische typen in DMMVM. De interactie tussen mens en machine vindt daarbij op objectniveau plaats, de verandering van objecten wordt dan door de machine doorvertaald naar de andere gegevenstypen. De topologie van de database wordt door het systeem onder bijhoudingsoperaties dynamisch bijgewerkt. Hierbij wordt gebruik gemaakt van "computational geometry", de topologische relatie tussen de nieuwe ingevoerde geometrische primitieve van een object en de al aanwezige. De gevonden topologische relaties activeert dan de relevante consistentie regel, zodat de consistentie van de database hersteld wordt. Het systeem waarschuwt de operateur als het er zelf niet uitkomt.

Een experimentele implementatie van object-georiënteerde datastructuur werd in Postgres versie 4.2 gerealiseerd. De data acquisitie werd gedaan met een Planicomp C120, een fotogrammetrische stereoplotter voorzien van een Zeiss Video Map en een Calcomp digitizer. De stereocompilatie werd uitgevoerd met een Kork digital mapping system, versie 8.0. Het datamodel werd getest voor een situatie met twee lagen: een bodemkaart en een topografische laag (met de belangrijkste landgebruik en landbedekkingstypes). De consistentie regels werden geverifieerd met behulp van C-functies en Postquel queries, een aantal consistentie regels werden ook getest.

De ervaring met deze experimenten leerde dat deze methodiek een grotere inspanning vraagt voor het ontwerp van een database, dat wordt dan gecompenseerd door snellere query afhandeling in het multi-lagen model en door het grotere informatiegehalte van de database. Dataconsistentie is bovendien verzekerd doordat inconsistenties al bij de creatie en bijhouding van de database gevonden worden en doordat topologie dynamisch door het systeem wordt bijgewerkt.

De conventies van de FDS aangevuld met de conventies van de DMMVM ondersteunen een eenduidige afbeelding van terreinsituaties op de databases, de voorgestelde datastructuur kan

in ieder bestaand vector-GIS worden geïmplementeerd, ook al is het soms met toegevoegde operaties of met een lichte aanpassing van de conventies. Veel systemen representeren zijden bijvoorbeeld verschillend, meestal als een aggregaat van rechte lijnstukken; aanvullende operaties zijn dan nodig om de topologie te herstructureren als we de veronderstelling dat een zijde recht is willen handhaven. Hiermee worden dan wel wat dataredundanties ingevoerd. De conventie dat zijden recht moeten zijn kan wel ontspannen worden door ook andere vormen toe te laten.

De experimentele implementatie toonde aan dat het datamodel met de consistentie regels en bijhoudingsalgoritmen geïmplementeerd kunnen worden. Ze kunnen daarom aanbevolen worden voor implementatie in GIS en kaarteringssystemen, daartoe moeten de algoritmen en consistentie regels aan het DBMS toegevoegd worden.

Curriculum vitae

Olajide Kufoniyi, born on 22 September 1954, received his BSc degree in geography in 1980 (with a second class upper division) from the University of Ife (now Obafemi Awolowo University), Ile-Ife, Nigeria. After completing the mandatory one-year national youth service scheme, he proceeded to the University of Lagos in October 1981 for the postgraduate diploma course in land surveying. He completed the course in December 1982. He then worked as a surveyor in the Federal Survey Department, Lagos, until September 1986 when he went to the International Institute for Aerospace Survey and Earth Sciences (ITC), Enschede, The Netherlands, for the postgraduate diploma and the MSc courses in photogrammetry. He completed the diploma and the MSc degree courses in August 1987 and March 1989, respectively. Both degrees were awarded with distinctions. After the MSc course, he returned to the Federal Survey Department, Lagos, from where he was seconded to the Federal School of Surveying, Oyo, Nigeria. From 1989 till January 1992, he taught photogrammetry and computer programming at the school. He was the head of the department of photogrammetry and also the head of the computer services unit. He is a registered (land) surveyor. He went back to ITC in January 1992 for the PhD research project reported in this thesis. The research was carried out in cooperation with the department of surveying and remote sensing, Wageningen Agricultural University, The Netherlands, and funded by ITC. He is the author and co-author of many scientific articles. He is married to a charming wife and they have three great kids.