Modelling spatial object behaviours in object-relational geodatabase

Bright Osei Twumasi
March, 2002

Modelling spatial object behaviours in object-relational geodatabase

by

Bright Osei Twumasi

Thesis submitted to the International Institute for Geo-information Science and Earth Observation (ITC) in partial fulfilment of the requirements for the Degree of Master of Science in Geo-informatics course.

Degree Assessment Board

Chairman:	Prof Dr W. Kainz
External Examiner:	Prof Dr Ir. Van Oosterom
Supervisor:	Drs. R. A. Knippers
Co-supervisor:	M. C. Ellis (MSc)

INTERNATIONAL INSTITUTE FOR GEO-INFORMATION SCIENCE AND EARTH OBSERVATION, ENSCHEDE, THE NETHERLANDS

Disclaimer

This document describes work undertaken as part of a programme of study at the International Institute for Geo-information Science and Earth Observation. All views and opinions expressed therein remain the sole responsibility of the author, and do not necessarily represent those of the institute.

Abstract

Modelling the behaviour of spatial objects constitutes a crucial aspect in the development of any GIS. Object behaviour, together with attributes and position constitute a minimal set of concepts sufficient to capture the special peculiarities of spatial data. Behaviour describes the dynamic aspects of objects. In the real world, we can observe several aspects of object behaviour. Objects in the real world follow their logical spatial arrangements; for example, a river network must always flow down hill, not uphill. Spatial objects have *semantics*, which describe the *meaning* or *role* of the objects in a particular application domain, and these impose conditions on the objects. Spatial objects also have relationships with each other and these relationships impose conditions or constraints on them. In addition, spatial objects change over time. In the context of spatial data handling, behaviour therefore describes the dynamics, interactions, relationships, constraints and semantics of objects in space and time. And any data model that represents (a part of) the real world should be able to capture these dynamic aspects of spatial objects. This will bring a geographic information system more close to the real world which it represents; and enable a more reliable use of the data in spatial planning and decision-making.

Given the need to capture object behaviour, this research aims at utilising the object-oriented concepts to model spatial object behaviours and to realise these behaviours in object-relational geodatabase of ESRI. Spatial object behaviours have been investigated under three aspects, namely: consistency constraints, multiple representations and temporal behaviour. With respect to the consistency constraints, the research seeks to develop a methodology for on-line, automatic monitoring of constraint violations during updating of the database. With the multiple representations, the main issues dealt with are how to maintain topologic and attribute consistency between road areas and their centrelines. The temporal behaviour aims at maintaining the history of objects in the database. This research has developed an integrated model for the automatic enforcement of consistency constraints in the database, maintaining consistency between the multiple representations and keeping the history of objects in the geodatabase. A prototype implementation of the model has been carried on a topographic database. Some of the behaviours have been implemented, and fully realised by extending ESRI geodatabase model.

Keywords: Object behaviour, object-oriented, object-relational, geodatabase, consistency constraints, multiple representations, temporal behaviour, spatial relationships, unified modelling language, component object model

Acknowledgements

This research would not have reached a successful completion without the help, support and encouragement of many individuals, to whom I am forever indebted. First, I thank the Almighty God, my Creator and Guide, who gave me the strength and knowledge to execute this work.

I would like to express my deep and sincere gratitude to my supervisor, Drs. Richard A. Knippers, for his continuing support, suggestions and contributions throughout the thesis period. From the very beginning of the research to the end he has always stood with me. I could go to him without any appointment to discuss the problems. I greatly admire his patience and friendliness I enjoyed during my work with him. Many thanks go to Mr Martin Ellis, my second supervisor for his constructive criticisms, suggestions and comments to this thesis. His support and guidance were extremely useful to the successful completion of this research.

I am also indebted to Prof Dr Wolfgang Kainz, Division of Spatial Information Theory and Applied Computer Science, ITC. I had the honour to discuss many topics with him. He dissolved most of my problems in many aspects of the research, particularly the area of spatial relationships and consistency constraints. My special thanks go to Mr Willem Nieuwenhuis, ILWIS Department, for his immense assistance in learning the Visual C++ language. As a beginner, he helped me many times to find my way through the complex world of C++ programming. I am also indebted to the Topographic Service of the Netherlands. Their support and interest in my research empowered me to come this far. I am thankful to Mrs Ellen-Wien Augustijn with whom I had very nice discussions about my thesis and research directions.

This acknowledgement would not be complete without mentioning Dr Yola Georgiadou, Division of Spatial Data Acquisition, ITC. She encouraged me and helped me realize my research potential from the very beginning of my life in ITC. I will also like to express my thanks to Mr G. C. Huurneman, student advisor, for his fatherly support during my thesis period. To my GFM.2 colleagues who provided the peaceful academic and social environment during my stay in ITC and the Netherlands, I am grateful. To the governments of The Netherlands and Ghana, my country, I remain forever indebted.

Finally, my deep gratitude go to my mother, Hannah Appiah, and my brothers, James, Sammy and Frank for their unwavering support, patience and love, throughout my life. To them I owe all this achievement.

B. Osei Twumasi
March, 2002

Table of contents

List of figures

List of tables

List of abbreviations

ATL	Active Template Library
CAD	Computer Assisted Design
CASE	Computer Assisted System Engineering
CLSID	Class Identifier
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DBMS	Database Management System
DCOM	Distributed COM
DDL	Data Definition Language
DEM	Dimension Extended Model
DLL	Dynamic Link Library
ESRI	Environmental System Research Institute
EXE	Executable
Geodatabase	Geographic Database
GIS	Geographic Information System
GML	Geography Markup Language
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IID	Interface Identifier
LAN	Local Area Network
OLE	Object Linking and Embedding
OMG	Object Management Group
OODBMS	Object-oriented Database Management System
ORDBMS	Object-relational Database Management System
RDMS	Relational Database Management System
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
Top10Vector	Topographic Vector data of the Netherlands at scale 1:10000
UML	Unified Modelling Language
UUID	Universally Unique Identifier
WAN	Wide Area Network

1. Introduction

1.1. Background and motivation

The real world can be seen as consisting of spatially interacting entities or objects that exhibit a wide range of behaviour. By means of the modelling process, each entity relevant to an application domain is represented by a corresponding object in the data model. The object in the model should have properties and behaviour that describe the characteristics of the corresponding entity in the universe of discourse. In general, we can identify three components of spatial objects that need to be modelled, namely: (i) *geometry*, which describe the shape and geographic positions of objects with respect to the earth's surface; (ii) *attributes*, which provide description of the state of an object at an instance in time; and (iii) *behaviour*, which describes objects' dynamics, interactions and the operations.

The last component, object behaviour, has been essentially neglected in traditional GIS systems, although it holds a critical role in spatial databases. Object behaviour, together with attributes and position constitute a minimal set of concepts sufficient to capture the special peculiarities of geographic data in terms of *object-oriented* rationale (Tryfona et al, 1997). Since spatial objects are not static in space and time, there is the need to analyse and model the effects of object dynamics, interactions, and semantics and store them in the database. This will bring a geographic information system more close to the real world, which it represents, and enable a more reliable use of the data in spatial planning and decision-making. Capturing object behaviour means representing both *static* and *dynamic* properties of objects. An object in the real world is characterized by static properties (attributes) and dynamic properties (operations). Static properties describe the *state* of an object at a particular instance of time, while the dynamic properties constitute its *behaviour* in space and time (Abrantes & Pinheiro, 1997). Modelling a real world object therefore involves representing these two aspects of the object.

The data modelling process essentially consists of three levels, namely, conceptual, logical and physical levels. The conceptual level defines the entities and their properties that are important for a particular application domain, their relationships and identify the behaviours that apply to these entities. At the logical level, the conceptual model is translated into a set of constructs supported by the target database management system (DBMS). The physical level deals with the process of mapping the data onto the physical structure of the DBMS. Information is usually lost in mapping from one level to another.

One of the well-established models at the logical level is the *relational model*, which organizes data into tables. These tables are then manipulated by a relational DBMS. Although, the relational model is based on well-established mathematical foundation, it is limited in terms of modelling object behaviour (i.e. object dynamics, interactions, and semantics) as the real world situation becomes more complex (Cooper, 1995; Egenhofer & Frank, 1992). To model object behaviour, object-oriented approaches have been applied for data modelling and implementation. Object-oriented modelling techniques allow the representation of both static and dynamic properties of objects (Abrantes & Pinheiro, 1997; Worboys et al, 1990). GIS systems have traditionally used relational databases for the storage of

their data. In spite of its limited ability for capturing object behaviour, the relational technology is mathematically well established in the database field than the object-oriented model. To take advantage of both the relational and object-oriented concepts, recent developments have focused on combining the two concepts into what is called *object-relational* model. This approach extends the relational model and integrates it with object-oriented concepts (Cooper, 1995; Zeiler, 1999). By this way, the behaviour of objects can be captured while still taking advantage of the full power of the relational technology.

In GIS, object orientation may be applied at three levels (Ruland & Rouvé, 1994), namely:

- The way spatial objects are modelled can be object-oriented.
- Object-oriented programming languages can be integrated into GIS applications to allow great extent customisations.
- Object-oriented database management system (OODBMS) may be used for the storage of the physical data.

The *first* level refers to the conceptual modelling of geographic objects. Modelling geographic objects based on object-oriented abstraction mechanisms (classification, generalization, aggregation and association) has been shown to be more powerful and better correspond to the way humans perceive the real world, than other models like the Entity-Relationship diagram (Egenhofer & Frank, 1992; Clementini & Di Felice, 1994). The *second* level is the approach adopted by object-relational database and GIS systems. Object-oriented features are integrated into the relational model (or built on top of it) in a manner that utilises the power of object orientation while maintaining the full functionality of the relational model. The *third* option constitutes a new representation paradigm, which combines database functionalities with object-oriented programming (Cooper, 1997; see chapter 2).

1.1.1. What is object behaviour?

The term *behaviour* as used in information systems originated from the object-oriented technology. Formal definitions of the term have been covered in chapter 2 in more detail. In general, when applied to objects, behaviour carries the same or similar meaning as when used in ordinary English Language. The Dictionary defines behaviour as ‘the way somebody/something acts or functions in particular situations’ (Oxford Advanced Learners). In object-oriented systems, behaviour is used to describe several aspects of objects dynamics, interactions and semantics. Behaviour has been used to describe the externally visible dynamics of an object in terms of its interactions with others; services/operations provided by objects; messages to which an object can respond; observable effects of operations on objects; the way objects change over time; and constraints on objects and relationships between objects.

In the real world, we can observe several aspects of object behaviour. Objects in the real world follow their logical spatial arrangements; for example, a river always flows downhill, not uphill. Objects also have *semantics*, which describe the *meaning* or *role* of the objects in a particular application domain. These semantics impose conditions or constraints on the objects. For example, in a cadastral application, two polygons representing parcels must not overlap. In addition, objects have relationships with each other and these relationships impose conditions on them. Finally, objects change over time. In the context of spatial data handling, behaviour therefore describes the dynamics, interactions, relationships, constraints and semantics of objects in *space* and *time*. And any data model that would represent the real world more closely should be able to capture these dynamic aspects of spatial objects.

1.2. Problem formulation and justification

Given the need to model the behaviour of spatial objects, modelling techniques should be applied that offer flexibility for handling these requirements. The object-oriented concepts promise a good solution. This research therefore aims at utilising the object-oriented concepts to capture object behaviour and to realise these behaviours in ESRI object-relational geodatabase. The object-relational model has been chosen because it combines the power of both the relational and object-oriented technologies (see chapter 2). The behaviours that are investigated in this research fall into three categories, namely, consistency constraints, multiple representations and temporal behaviour. These are described below.

1.2.1. Consistency constraints

The enforcement of consistency constraints is a very crucial aspect of any reliable spatial database, because the database loses its reliability and usability once inconsistent data are entered. The database must be up-to-date and consistent if it is to be used for spatial planning and decision-making. And since the updating process may disturb the consistency of the data, constraints should be developed to guide the system and the human operator during the updating process (Kufoniyi, 1995). Consistency constraints in spatial databases may be classified into three categories, namely, *topological*, *semantic* and *user-defined* constraints (see chapter 3). Topological constraints are normally handled by the GIS system. But semantic and other user-defined constraints are left to the user because they are largely application-dependent. At present, operational GIS systems usually don't warn the user when, for example, a building is digitised inside a lake. This research therefore aims at applying object-oriented concepts to model semantic and user constraints in the spatial database. It seeks to develop a methodology for on-line, automatic monitoring of constraint violations during updating of the database.

1.2.2. Multiple representations

Frequently, GIS databases must handle multiple representations of the same real world objects. Multiple representations means that the same real world objects are represented more than once in the database. They are needed for many reasons: for example, cartographic map production at various scales to meet multiple users' requirements, for multiple GIS analysis, etc. When such multiple representations coexist in the database, the issue of maintaining consistency between them becomes imperative (Medeiros et al, 1996). Also propagating updates between the various representations is needed in order to reduce update costs. This research seeks to develop techniques for managing and maintaining consistency between multiple representations involving road areas and their centrelines, and to develop automatic update propagation mechanisms using the object-oriented concepts.

1.2.3. Temporal behaviour

Modelling the temporal behaviour of geographic objects has been essentially neglected by traditional GIS systems although it constitutes a crucial aspect of spatial databases (Raza, 2001). Spatial objects are not static in time, and it is only when their temporal aspect has been appropriately handled before queries involving time can be answered, such as 'What was the state of an object at time t_i?' This research aims at utilizing the object-oriented concepts to model the temporal behaviour of geographic objects in the database.

1.3. Research framework and prior work

This research work is within the framework of the proposed research agenda of the International Institute for Geo-information Science and Earth Observation (ITC), the Netherlands. The research actually falls within an initiative by the Topographic Service of the Netherlands (in collaboration with ITC and others) to restructure their datasets and products, and to take advantage of current technologies in order to better respond to increasing customer demands. This research will therefore constitute a preliminary investigation into the suitability of the object-relational approach to meet the requirements of the Topographic Service. Several research activities have been carried out in connection with the main themes of this research, some of which are briefly reviewed below.

In his thesis, Sivanantharajah (2000) investigated the implementation of constraint rules for the Top10Vector database using external constraint repository to control the updating of the database. However, his investigation did not include online, automatic monitoring of constraint violations as the database is being updated. Constraint violations can only be checked after data has been entered into the database. Moreover, the constraint repository was external to the database. However, storing the constraints as part of the database itself and automatically monitoring constraints violations during the updating will enable the system to ensure more readily the integrity of the database. Sivanantharajah acknowledged this and recommended to use object-oriented concepts to model these requirements.

Kufuniyi (1995) researched on spatial coincidence modelling, automated database updating and consistency in vector GIS. In his thesis, he stressed the necessity for the automatic enforcement of constraints during database editing, and modelled these requirements by introducing the container data type. He also dealt with aspects of automatic update propagation among multiple objects in the database.

Raza (2001) applied object-oriented concepts to model the temporal behaviour of geographic objects in the context of urban application. His work was extensive and focused mainly on the core problem of modelling time, namely the spatio-temporal data model. He implemented his data model using the object-relational approach and noted some advantages of it.

1.4. Research objective

The objective of this research is to investigate the behaviour of geographic objects in three aspects, namely, consistency constraints, multiple representations and temporal behaviour; to use object-oriented concepts to model these behaviours and to realise them in the object-relational geodatabase model.

1.5. Research questions

To achieve the above objective, the following research issues will be addressed:

- How to enforce consistency constraints in spatial databases and check for constraint violations during online updating of the database?
- How to maintain consistency between multiple representations of the same real world objects? How to develop automatic update propagation between the multiple representations?
- How to model the temporal behaviour of spatial objects?

- How does ESRI object-relational geodatabase extend the relational model? What support does it offer for implementing object behaviours in the database?
- How to extend ESRI geodatabase model to realize the behaviours?

1.6. Limitations of the research

- With respect to the temporal behaviour, this research is not intended to develop a fully functional temporal GIS system capable of answering any generic spatio-temporal query. Modelling time in spatial databases is an on-going research in itself (Langran, 1992). The main issue that is dealt with in this research is how to maintain the history of objects in the database.
- The issue of managing multiple representations constitutes an active research on its own as can be witnessed from the literature (Spaccapietra et al, 1999). This research is limited to maintaining consistency between multiple representations involving road areas and centrelines. Also, operations for deriving one representation from the other are not considered. That belongs more appropriately to the problem of automatic generalization of features, which is not the theme of this research.

1.7. Methodology

1.7.1. Fundamental concepts

To lay the foundation for the modelling and implementation phases, background theoretical concepts were reviewed. These include:

- A review of the fundamental concepts of object orientation, database models and component-based technology with particular reference to the Microsoft Component Object Model (COM), the technology framework upon which ESRI geodatabase is based.
- A review of the elements of the unified modelling language (UML) relevant to the modelling phase in this research. The UML has been used for the conceptual and logical modelling.
- A review of spatial data models and the mathematical background of modelling spatial relationships and consistency constraints in spatial databases.

1.7.2. System design phase

The design phase began with the requirements analysis for the application domain. We chose a topographic vector dataset of the Netherlands (1:10,000 scale) called the Top10Vector. The requirements of the Top10Vector investigated in this research have been grouped into three categories: (i) consistency constraints, (ii) multiple representations, and (iii) temporal requirements. These requirements have been presented in chapter 4. The requirements were analysed and methodologies were developed to model these requirements. The basic approach is to use object-oriented concepts to systematically identify objects, classes, relationships, constraints, and operations. An integrated conceptual model has been developed for the enforcement of the constraints, the multiple representations and the temporal behaviours. The conceptual model was developed in UML.

1.7.3. Implementation phase

The implementation phase began with a study of ESRI geodatabase model and how to extend that model to handle the specific requirements of the Top10Vector. Supporting technologies were also studied, namely the Microsoft Component Object Model (COM) and Visual C++, in order to gain the necessary mastery for the actual implementation. Logical models were developed in UML using ESRI UML model. The behaviours were implemented using Visual C++ and the geodatabase model, which is based on COM technology. The geodatabase schema was generated from the logical model using ArcInfo tools. The Top10Vector dataset was restructured to conform to the model, and then was loaded into the geodatabase. The implementation phase ended with testing and display of the behaviours in ArcMap, a product of ESRI.

1.8. Research tools

In this research, the following software tools were used:

- ArcInfo 8.1—The integrated development environment for implementing the whole system.
- Microsoft Visio 2000—For designing the models in the unified modelling language (UML).
- Microsoft Visual C^{++} 6.0 —For the actual implementation of the behaviours in the geodatabase.
- Microsoft Visual Basic 6.0—For developing client applications for testing, assessing, and visualizing the behaviours implemented.

1.9. Structure of the thesis

The thesis is organised into six chapters and these are described as follows:

Chapter-1 introduces the research background and motivation, problems, objective, questions, methodology and limitations.

Chapter-2 describes the fundamental concepts of object orientation, database models and Microsoft Component Object Model in order to lay the necessary background for the following chapters. The UML has been used for the conceptual and logical modelling, and is introduced in this chapter.

Chapter-3 describes spatial data models and reviews the mathematical framework for modelling spatial relationships among objects. It also deals with consistency constraints and how the spatial relationships are used to enforce consistency of the database.

Chapter-4 deals with the development of the conceptual model. The three categories of behaviours have been modelled in this chapter, namely, semantic constraints, multiple representations and temporal behaviours.

Chapter-5 deals with the implementation phase and cover the logical design, the actual implementation, and the testing and display of the results.

Chapter-6 summarizes the main conclusions drawn from the research and gives recommendations for further work.

2. Object orientation, database models and component-based technology

2.1. Introduction

This chapter reviews the fundamental concepts of object orientation, database systems and component-based technology. The first part of the chapter covers the concepts of object orientation: i.e., objects, abstraction, inheritance, encapsulation, polymorphism, information hiding, and message passing. The unified modeling language (UML) has been used for designing the conceptual and logical models; therefore the relevant elements have been introduced in this chapter. The sections that follow describe database models, namely, relational, object-oriented and object-relational models, with specific focus on ESRI geodatabase model. The last section describes the Microsoft Component Object Model (COM), upon which ESRI geodatabase is based.

2.2. Object orientation

Object orientation can be described as a strategy for organising a system as a collection of interacting objects that can combine data and behaviour (Blaha & Premerlani, 1998). Object-oriented technology is based on the assumption that the real world can be modelled as distinguishable entities or objects that can be grouped together into classes. These objects have identity, state and behaviour. The object-orientated paradigm has been applied to many technology areas including the following: object-oriented programming, object-oriented analysis and design, object-oriented databases and object-oriented graphical user interface design. All of these bring considerable advantages to the successful implementation of a GIS, especially object-oriented programming languages, which provide the framework for implementing the others. The benefits that come from the object-oriented technology are many, including: extensibility, reusability, reduced complexity, ease of use, etc (Henderson-seller & Edwards, 1994; Blaha & Premerlani, 1998).

The following sections discuss the basic concepts of object orientation. Due to the application of object orientation to different areas, there are often alternative interpretations of the underlying concepts. There are some degrees of dependencies and relationships as well as differences between object-oriented concepts as interpreted in different environments. The concepts explored in this chapter are from the point of view of object-oriented systems in the general sense.

2.2.1. What is an object?

Formally, an object can be defined as an identifiable entity, real or abstract, that has a precise role for an application domain (Roy & Clement, 1994; Blaha & Premerlani, 1998). To constitute an entity, something must be identifiable (have identity), relevant (be of interest to the application domain) and describable i.e. have characteristics (Chen, 1976; cited in Mattos at el, 1993). By means of the modelling process, each entity relevant to an application domain is represented by a corresponding object in

the data model. The object in the model should have properties that describe the characteristics of the corresponding entity in the universe of discourse. In object-oriented systems, each object is defined by three things, namely its *identity* (which distinguishes it from other objects), its *state* (described by its attributes) and its *behaviour* (described by the operations performed by the object). These are further discussed below.

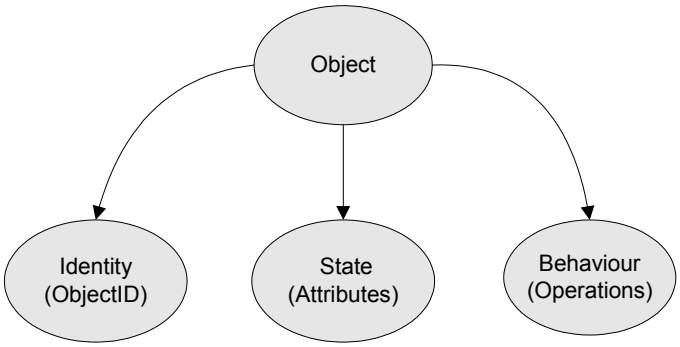


Figure 2.1 Three components of an object

2.2.1.1. Object identity

In an object-oriented system, each object is unique. This uniqueness of an object is achieved by means of the object identity. Object identity is that property of an object that uniquely distinguishes it from all other objects (Khoshafian & Abnous, 1995). By introducing a unique identity for each object, different objects can be distinguished from each other without the need to compare their attributes and behaviour (Ellmer, 1993). The object identity is usually system generated, unique to that object, and invariant for the object lifetime (Cooper, 1997).

2.2.1.2. Object state

The state of an object is described by the values of its *attributes* (or properties) at one moment in time. An object attribute is actually a named property of the object *class* that describes a value held by each object of that class. Objects can have a single state throughout their lifetime, or they can go through many state transitions. When an object moves from one state to another, we talk of a new version of the object (see section 4.7.3).

2.2.1.3. Object behaviour

Behaviour describes object dynamics and interactions with other objects. In object-oriented systems, the term *behaviour* has been used to describe several aspects of object dynamics as the following show:

- The externally visible dynamics of an object (or class) in terms of its interactions with other objects (Firesmith, 1993; cited in Firesmith & Eykholt, 1995).
- How an object acts or reacts, in terms of its state transitions and message passing, the externally visible and testable actions of an object (ordinary English meaning) (Booch, 1994).
- The set of services provided by an object or class via the messages in its interface (Firesmith, 1993; cited in Firesmith & Eykholt, 1995).

- The set of messages to which an object or class can respond (Wirfs-Brock et. al., 1990).
- The observable effect of performing the requested operation including its results (Henderson-Sellers, 1994).
- The operations performed by objects, when requested by other objects in the system (Lee & Tepfenhart, 2001).
- The collection of operations (or methods) the object provides, the responses these operations generate and the changes the operations cause on the object, or other objects in the system (Fussell, 1997).
- A metaphor referring to the way objects change over time within a defined structure (Martin & Odell, 1992).
- Any set of synchronization constraints that defines how objects execute their methods in relation to one another (Atkinson, 1991; cited in Firesmith & Eykholt, 1995).

From these definitions, it is obvious that the term behaviour as applied to objects in a system has similar meaning as the ordinary usage of the word in the English Language. In object-oriented systems, the behaviour of an object is encapsulated in its *methods* or *operations* (also called *services or responsibilities*) performed by the object or performed on the object by other objects. In this thesis, these terms have been used *interchangeably* to describe objects' dynamics, interactions and semantics in a system.

2.2.2. Foundational concepts of object orientation

Object orientation goes beyond the basic concepts of object identity, attributes, and behaviour. It considers other aspects of objects, their interactions, relationships, and organization, and these provide the foundation for object-oriented system modelling and design. In the following sections, the foundational concepts of object orientation are discussed.

2.2.2.1. Abstraction

In the object-oriented environment, abstraction has been defined as both a process and as an entity. As a process, abstraction denotes extracting the essential parts of an object or group of objects while ignoring the unessential details for the time being (Henderson-sellers & Edwards, 1994). As an entity, abstraction denotes a model that includes the most important, essential or distinguishing aspects of something while suppressing or ignoring less important, immaterial or diversionary details (Firesmith & Eykholt, 1995; Booch, 1994). For example, the process of forming a conceptual model of a part of the real world during database design is an abstraction process, since only those entities and their properties that are relevant to the application domain are collected. The resulting model is also an abstraction of the real world. Abstraction is one of the fundamental ways humans cope with complexity. The main purpose of abstraction is to manage complexity of the system being modelled. Abstraction focuses on the essential parts of an application while ignoring the details. Thus abstraction allows one to postpone design details until a later stage when one can make more informed decisions (Blaha, & Premerlani, 1998). Abstraction thus plays important role in modelling and is a central element in object-oriented system analysis and development. Abstraction forms the basis of many concepts of object orientation. There are four abstraction mechanisms that are commonly employed, namely, *classification*, *generalization*, *association* and *aggregation*. These abstractions are in general expressed as relationships between objects or object classes, having as their purpose the organization of the objects in some desired form (Mattos et al, 1993). Each of these concepts is discussed in its proper place in the chapter.

2.2.2.2. Classes

Classification is one of the most important and well-understood forms of abstraction. It is a mechanism for organizing similar objects into groups or *classes*. An object is an instance or occurrence of a class. A class is a description or specification of a group of objects with similar attributes, common behaviour (operations), similar relationships with other objects, and common semantics (Rambaugh et al, 1991). Mattos et al (1993) state three concepts that are associated with the use of classes. First, a class provides the *structural* definition for the instances of that class, that is, the names and types of their attributes and methods. Objects being instances of the class inherit the attributes and methods specified for the class. Second, a class serves as a template for creating new objects. New objects are created as instances of the class; hence they have access to all the attributes and methods of the class to which they belong. This is called *inheritance*, and is discussed in the next section. Third, each class is associated with a group of objects, which comprise its instances, and the class has to administer these.

Some classes are never directly instantiated, that is, they cannot have instances themselves. These classes are called *abstract* classes. The descendants of abstract classes, however, can have instances. Abstract classes are useful for reducing redundancy of class specification: general specifications can be defined in an abstract class that can be shared by other inheriting classes.

2.2.2.3. Inheritance

As mentioned in the previous section, a class is a group of objects with similar attributes and behaviour, and serves as a template for creating new objects. As an instance of a class, an object has all the attributes and operations of its class. Not only can an object inherit from a class; more importantly, a class can inherit from another class. This leads to a key concept of object orientation known as *inheritance*. Inheritance is a concept of object orientation which provides the means to define a new class in terms of existing classes. The new class inherits the characteristics and behaviour of its parent class or classes. The new class is called a *subclass* of its parent class. The parent class in turn is a *superclass* of the child class. The superclass is more general than the subclass. The inheritance hierarchy does not have to stop at two levels. The subclass can be a parent for still another class, and so on. By this way we can construct an inheritance *hierarchy* or *tree* as shown in Figure 2.2.

When constructing an inheritance hierarchy, there are two ways to proceed, either top-down or bottom-up which correspond respectively to two concepts, namely *generalization* and *specialization* (Cooper, 1997). Specialization is the creation of a new class as a subclass of an already existing parent class. The subclass will then inherit the characteristics of the existing parent class. Generalization is the opposite way—taking a set of existing classes and creating a more general class. For example, designing a transport application might start by constructing classes for *cars*, *boats*, *planes*, etc, and out of them extract common features and create a common class called *vehicle*.

Inheritance can be a *single* or *multiple*. With single inheritance, a subclass may inherit attributes and methods from a single parent class, usually adding its own attributes and methods. Multiple inheritance means a subclass inherits from multiple superclasses. For example, consider a *Canal* being a subclass of both *Transportation* and *Hydrography* superclasses (Figure 2.2). Inheritance maximizes code reuse, and offers more flexibility for customizing and developing systems for specific applications (Newell, 1992).

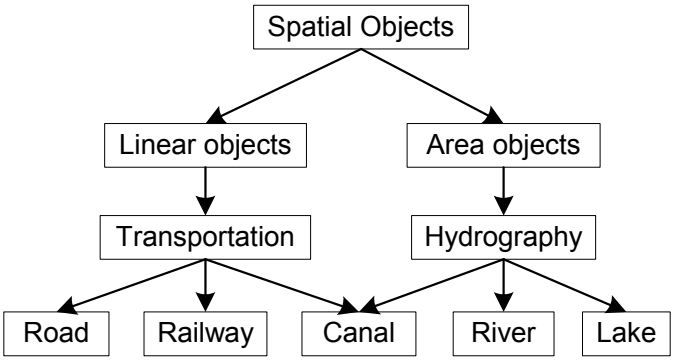


Figure 2.2 Single and multiple inheritance of objects classes

2.2.2.4. Encapsulation

Encapsulation means that each object contains the data (attributes) and the methods (code) that will manipulate the object, and that the object's data are accessed only through its methods (Lee & Tepfenhart, 2001). Thus the data and code are tied together in an integral unit, working as one. An object class encapsulates both the structure and the behaviour in that the characteristics include both attributes (which hold the data of the object), and the methods (which are code fragments that can be executed against members of the class) (Cooper, 1997). The organization of that data is known only to the object itself, and is not accessible from outside the object except through a specific and carefully designed *interface*. Encapsulation thus separates the internal implementation of an object from its public interface. All other details concerning the object are hidden behind its interface. Users of the object access the object's data only through the methods specified in the object's interface. Users don't know the internal implementation of the object, i.e., how its methods are executed. An important benefit of encapsulation is that since the internal implementation is totally hidden within the class, the implementation of a class can be changed without affecting other objects in the system. In other words, the implementation can be extended and modified without affecting the users of the class. This leads to easy software extensibility and reusability (Khoshafian & Abnous, 1995).

2.2.2.5. Information hiding

Information hiding is the object-oriented concept that states that users of an object need to know only the essential details of how to utilize and access the object but not all the details of its implementation (Budd, 1991). Each object decides what methods are available to the other objects in the system. Other objects may neither have access nor knowledge of the data (attributes) or how the operations are provided. This principle is called *information hiding*. Information hiding is used to restrict access to a subset of the characteristics of a class (Cooper, 1997). Some authors equate information hiding with encapsulation or use them interchangeably (Booch, 1994; Rambaugh, 1991); others make a distinction between them (Cooper, 1997; Henderson-Sellers et al, 1994). Henderson-Sellers et al (1994) argue that at the implementation level, the data and code can be encapsulated together in a class and yet remain visible to other classes. On the other hand, information hiding requires the visibility of data to be restricted within the encapsulated object. Hence, encapsulation does not guarantee information hiding. Information hiding is used to promote abstraction, support incremental top-down development and to prevent against accidental corruption of the data (Firesmith & Eykholt, 1995).

2.2.2.6. Message passing

Object orientation introduced the concepts of *messages* that are sent to objects. Objects communicate with each other by *sending messages* and responding to those messages. In the object-oriented paradigm, the message passing mechanism is the only way objects communicate with each other (Lee & Tepfenhart, 2001). An action is initiated by a message sent to a specific object and the object responds to that message. Messages that are sent to objects may ask the object to execute one or more of its methods or modify the object's content, changing its state (Khoshafian & Abnous, 1995).

Graphical interfaces represent the most natural way of using the message passing mechanism. These interfaces usually present to the user screens populated by icons for the user to click. Users activate an action by clicking on an icon. For an example, to print a page in a Word Document, the user clicks on the 'Print' icon. The 'Document' object will have a method called 'PrintPage' which has the responsibility of printing a page. To print the page, the 'PrintPage' method sends the 'Print' message to each object on the page, which then prints itself. There may be several objects on the page: Text, Rectangle, Triangle, Circle, etc. The 'Document' object does not need to know what types of objects are on the page, only that each object supports the behaviour of printing. The message passing mechanism allows the 'Document' object to communicate with all other objects on the page, and request them to execute their methods in response to that message.

In general, messages that are sent to objects consist of three things, namely: *a receptor* i.e. the target object, *a selector* or the name of the method and *arguments* (Khoshafian & Abnous, 1995, Mattos et al, 1993). The receptor specifies the object to which the message is sent. The selector tells the target object what the message sender wants, and chooses the appropriate method from all other methods associated with the target object. The arguments are the operation parameters or operands, and may include data passed to the corresponding method or return types. When messages are sent to objects, it is up to each recipient object to figure out how to respond to that message and take the appropriate action. Each object can respond to a collection of messages defined in its *interface*. The interface of an object is the set of messages to which the object can respond, which in turn is determined by the class to which the object belongs. The interpretation of a message can vary with objects from different classes. Objects from different classes may respond to the same message differently. This is referred to as *polymorphism* and is discussed in the next section.

2.2.2.7. Polymorphism

Polymorphism means the ability of a single name to refer to different things having different forms (Firesmith & Eykholt, 1995). In object-oriented systems, this refers *mainly* to the fact that an operation with the same name can be applied to different objects, which may implement that operation differently. Polymorphism allows two or more objects to respond to the same message differently. In our 'Document' object example (referred to in the previous section), when the user clicks on the print icon, the message that is sent to the different objects is the same: 'Print', but the actual implementation of what the objects must do to print themselves varies. Every printable object must know how to print itself. For example, the 'Rectangle' object knows how to respond to the 'Print' message and print a rectangle, not a circle; the 'Triangle' object knows how to react to the same message and print a triangle. This is polymorphism: the same operation is applied to different objects, which implement it differently. Polymorphism thus enhances software reuse by making it possible to implement generic software that will work for not only existing objects, but also for objects to be added later.

2.2.2.8. Relationships among objects

Objects and classes do not exist in isolation. There are three general relationships between objects or classes, namely, *association*, *generalization* and *aggregation*. All these are abstraction mechanisms and aim at simplifying the modelling process. These relationships are described in this section.

Association

Association is a physical or conceptual link between objects, and denotes some semantic dependency between the objects (Blaha & Premerlani, 1998; Booch, 1994). An example is the marriage relationship between a man and a woman. Objects in an association usually play a *role*. For example, the marriage association captures the fact that the man plays a 'husband' role to the woman, while the woman provides a 'wife' role to the man. Association has another property called *cardinality*, which describes the number of objects from one class that relate to a single object in the associated class. There are three general kinds of cardinality across an association, namely: one-to-one, one-to-many, and many-to-many. The marriage association, for example, is one-to-one relationship.

Generalization

Generalization is a relationship between a class (superclass) and one or more variation of the class (subclasses). The superclass holds common attributes and methods and the subclasses inherit them, adding their own attributes and methods. Specialization has the same meaning as generalization except that it takes the top-down perspective, starting with the superclass and splitting out variation. Generalization is known as 'is-a' relationship. For example, a *horse* 'is a' kind (subclass) of *animal*.

Aggregation

Aggregation is an abstraction mechanism, which allows a more complex object to be composed of one or more basic objects. Aggregation is a kind of association between the whole (complex object) and its parts (basic objects). Thus the relationship formed by aggregation is called a 'part-of' relationship (Hornsby & Egenhofer, 1998). This means that the aggregated instances are parts of the aggregate object. For example, an engine is a part of a car. *Composition* is a strong form of aggregation in which the composite object controls the lifetime of the parts. Thus if a composite object is destroyed, it must destroy all of its parts. It is possible to have several levels of aggregation hierarchy among objects or classes. For example, a country is made of provinces, which are composed of municipalities.

2.3. The unified modelling language

The conceptual and logical models have been designed in the unified modeling language (UML). Therefore we briefly introduce the UML in this section and the conventions used in this thesis. The UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems (OMG, 1999). The UML has been chosen because of its solid semantics, notation definitions and its wide acceptance, and especially because ESRI software directly supports it. The UML comprises a number of graphical elements that are combined to form models that provide multiple views of a system. It is important to know that UML models describe what a system is supposed to do and *does not show how to implement the system*. In this section, only the relevant elements of the UML class diagram used in this thesis are briefly described. Complete formal descriptions of all the UML modelling elements and semantics can be found in the Object Management Group (OMG) Unified Modeling Language Specification.

2.3.1. Objects

In the UML notation, an object is shown as a rectangle with two compartments. The top compartment shows the name of the object and its class, all underlined. The second compartment shows the attributes for the object and their values as a list (see Figure 2.3).

2.3.2. Classes

Classes are represented in the UML by a solid-outline rectangle with three compartments separated by horizontal lines. The top compartment holds the class name and other general properties of the class (e.g. stereotypes); the middle compartment holds a list of attributes; the bottom compartment holds a list of operations. The convention used in this thesis is to list the class name in bold face, capitalize the first letter, and use uppercase letters for the first letter of attributes and operation names. Abstract classes have their names in *italics*. In cases the class name is formed from two or more words, we capitalize the first letter of each word irrespective of its position. Optional listing of attribute data types and initial values my also be provided, as shown in Figure 2.3. Operations and methods may also be listed (optionally) with their return types and parameters.

2.3.3. Interface

A class interface is a *specifier* for the externally visible operations of a class without specification of internal structure. The interface of a class specifies the set of operations that the class presents to other classes. An interface is represented in the UML as a rectangle with two compartments, the top showing the interface name with the keyword *«interface»*, and the bottom part showing the list of operations. The relationship between a class and its interface is shown by means of a *refinement* or *realization* symbol (see Figure 2.3).

2.3.4. Association

Association is represented in the UML as a line connecting two classes with the association name just above the line. This kind of association is called *binary association* in the UML. Association *role* is shown at both ends of the line next to the class. Association cardinality or multiplicity is shown just above the association line near the appropriate class. An association may have attributes and operations just like a class. In such case, it is called an *association class*. This is represented in the UML like an ordinary class with a dotted line connecting it to the association line (Figure 2.3).

2.3.5. Generalization

Generalization is a relationship between a superclass and its subclasses. In the UML, generalization is represented by a line that connects the subclass to the superclass, with an open triangle on the end of the line that point to the parent class, as shown in Figure 2.3.

2.3.6. Composition

Composition is a relationship between a composite object and its basic objects. Composition may be shown by a solid filled diamond as an association role adornment. The multiplicity of the composite end may not exceed one (i.e., it is unshared).

In addition, the following conventions are used to define the data members and operations:

«Stereotype» : Tag for interfaces, attributes or operations
Visibility : + Public visibility
: - Private visibility

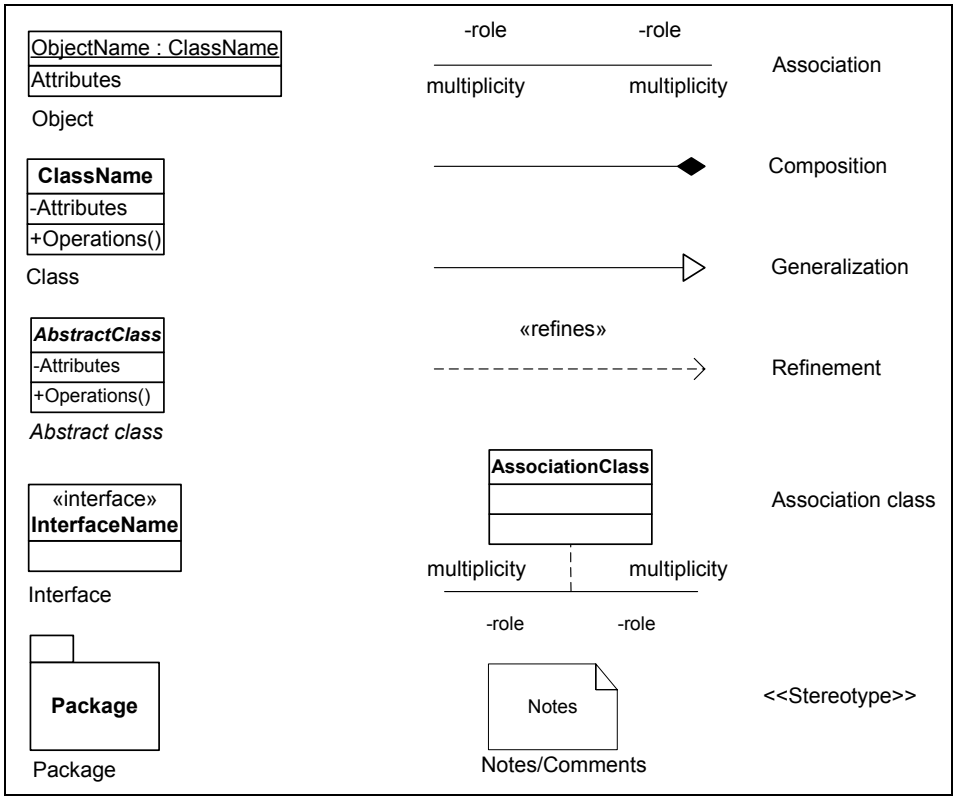


Figure 2.3 UML static structure elements used in this thesis

2.4. Database models

The database is the foundation for a geographic information system. Database systems are usually classified according to the data model they use. In the past, there were the *hierarchical* and *network* data models, which have now being replaced by three models, namely, *relational*, *object-oriented* and *object-relational*. The following sections briefly describe these three models.

2.4.1. The relational model

Relational databases are based on the relational model, which organises the data into relations or tables (Elmasri & Navathe, 2000). A relational schema is a set of attribute names and mapping from each attribute name to a domain (Worboys, 1995). The relational schema gives the structure of the relation, and does not include the data. A *relation* is a finite set of tuples associated with a relational schema according to the number of attribute names in the schema. Some properties of a relation include:

- The ordering of the tuples in the relation is not significant.
- Tuples in a relation are distinct from each other.
- Columns are ordered so that data items correspond to attribute in the relational schema with which they are labelled.

Inherent to relational databases are the notions of *primary key* and *foreign key*, which are used to provide links between tuples in different relations.

2.4.2. The object-oriented model

Object-oriented database management systems (OODBMS) integrate object orientation and database functionality (Khoshafian, 1993). An object-oriented database is characterised by having an object-oriented logical data model and by using an object-oriented programming language as its principal interface (Cooper, 1997). In object-oriented databases, the basic unit of data is an indivisible object. These objects have identity, attributes and behaviour defined through methods, all encapsulated within the object itself. Object-oriented databases were introduced as a means of overcoming the limitations of the relational model and to handle more complex application requirements, which the relational model cannot capture. However, OODBMS have their own limitations. Both the relational and object-oriented databases have their advantages and disadvantages. These are extensively discussed elsewhere and will not be repeated here (see Elmasri & Navathe, 2000; Blaha & Premerlani, 1998; Khoshafian, 1993; Cooper, 1997).

2.4.3. The object-relational model

The object-relational approach, sometimes called the *extended relational model*, is a compromise between the concepts of the object-oriented and relational models. In this approach, object-oriented features are incorporated into relational databases or (built on top of it) in a manner that utilises the power of object orientation while maintaining the full functionality of the relational model (Elmasri & Navathe, 2000). Nowadays, many database management systems and GIS systems are beginning to incorporate object-oriented features into the existing relational database functionality. For example, Oracle 8 introduced this concept to overcome the limitations of the relational model. ESRI's recent model called the geodatabase fully combines object-oriented concepts with the relational model. This model is the one used for implementation in this research. The object-relational model has been adopted in this research because it utilises the power and semantics of object orientation and the full functionalities of the underlying relational database system.

2.5. Geodatabases

Traditionally, most GIS systems have stored the spatial and attribute components of geographic data in separate files linked together by unique identifiers. The data model used often isolates the 'graphics' data from the attribute data. For example, in the old ArcInfo coverage (georelational) model, the spatial data is stored in indexed binary files, and the attributes are stored in separate files (INFO tables). Recent advances in hardware and software engineering have now made it possible to integrate GIS and database systems to create geodatabases. Geodatabases (short for geographic databases) store both the spatial and attribute data together in a single database management system. The spatial data is stored just like an attribute in the database. This possibility has further been enhanced by the integration of the relational model and object-oriented concepts. This section gives a general introduction to ESRI geodatabase concept in the context of the foregoing discussion. The spatial component of the geodatabase model has been covered in chapter 3.

2.5.1. ESRI geodatabase model

With the recent release of their integrated ArcGIS software, ESRI introduced a new object-relational model called the *geodatabase***.** This new model is implemented as extension to the standard relational model by integrating it with object-oriented concepts in a manner that allows geographic objects to be modelled with their behaviours attached to them. The geodatabase model serves as a generic model for geographic information, and provides a framework for creating an object-relational database schema and implementing behaviours on objects in the database (ESRI, 1999). The geodatabase supports many object-oriented concepts such as: inheritance, encapsulation, polymorphism, etc (Zeiler, 1999). Entities in the geodatabase are represented as objects with attributes, relationships and behaviour. ArcInfo also provides the ability for objects in the geodatabase to *message* each other. This provides database developers the ability to build more complex behaviours into objects in the database. The ArcInfo software manages both the attributes and behaviour of objects, and database developers work in an object-component environment that is abstracted from the underlying physical database model and augmented by a programming framework of interfaces and methods. ArcInfo automatically provides the object-relational mapping and manages the integrity of the data in the underlying database. The basic objects in the geodatabase model are shown in Figure 2.4. All the objects in the model are implemented as COM components (see section 2.6) with attributes and behaviour attached to them.

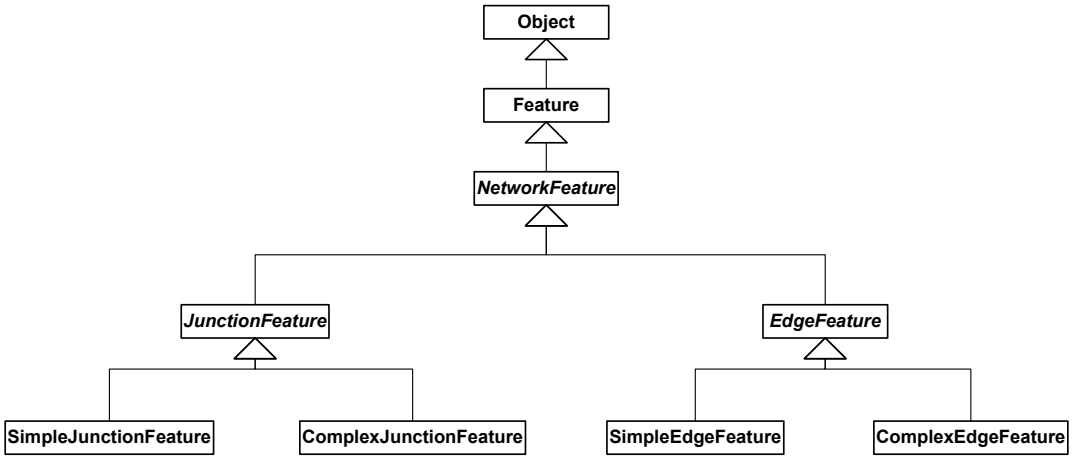


Figure 2.4 Basic objects in the geodatabase model

2.5.2. Personal and multi-user geodatabases

The geodatabase comes in two variants, namely, personal and multi-user geodatabases. The personal geodatabase is implemented on the Microsoft Jet Engine, which stores data in Microsoft Access database, and is built into the ArcInfo software. The personal geodatabase is limited to maximum storage capacity of 2 gigabytes and is therefore suitable for project-oriented GIS. Personal geodatabase supports single editing and several simultaneous viewers. For large enterprise databases, ArcInfo provides a multi-user data access extension called the ArcSDE. ArcSDE provides the gateway between the ArcInfo software and the DBMS to share and manage the spatial data as tables. It allows remote access to spatial data and allows many concurrent editing of the same database. ArcSDE supports many relational and object-relational DBMSs, including—Oracle, Microsoft SQL Server, IBM DB2, and Informix. ArcSDE can also serve data to other ESRI applications such as MapObjects, ArcIMS, ArcView,

and CAD client applications (West, 2001). The behaviours developed in this research are implemented in the personal geodatabase, the only reason being time consideration. However, the same behaviours can also be implemented in the multi-user ArcSDE geodatabase.

2.5.3. How the geodatabase extends the relational model

ArcGIS (hence the geodatabase) is built on a technology framework known as ArcObjects. ArcObjects comprises an integrated library of software components with GIS functionality and programmable interfaces. ArcObjects framework is based on the Microsoft Component Object Model (COM) protocol and is compliant with the OpenGIS simple feature specifications (ESRI, 1999; OpenGIS, 1999). All the objects and classes in the ArcGIS and geodatabase model are COM components that implement programmable interfaces. By means of this technology integration, developers can develop geodatabase applications by either customizing or extending the standard ArcGIS application framework or by using ArcObjects, the extensive COM-based development library for ArcInfo. ArcObjects allows one to extend the geodatabase data model by creating custom (application-specific) models and behaviours that meet the specific needs of the application domain. These custom classes are also COM-based components that can have programmable interfaces and implement custom behaviours, in addition to the behaviours attached to ESRI base classes. This approach has been illustrated in the implementation phase in chapter 5.

2.6. Microsoft Component Object Model

This section introduces the Microsoft Component Object Model (COM), the technology upon which ArcObjects and the geodatabase are based.

2.6.1. What is COM and why is it needed?

The main benefit of object-oriented technology from the computing point of view is the ability to put applications together quickly and easily from *reusable* and *maintainable* code. Object-oriented programming languages (such as C++) offer a great flexibility for software developers to write this kind of code. However, object-oriented programming languages are not sufficient to offer widespread reuse. Their main limitation is that code written in different languages or on different platforms cannot be easily reused (Grimes et al, 1999; De Vet, 1997). Here is where object-component standards like COM, which enable the reuse of binary components, are needed.

COM is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft OLE and ActiveX components (MSDN Library, 2001). COM is a protocol that connects one software component or module with another at the binary level. By making use of this protocol, it is possible to build software components that can be reused at the binary level. This means that the focus of reusability is on the *compiled* code, not the *source* code. Thus COM enables the reuse of software components across different languages and platforms. COM is not a particular object-oriented language, but a binary specification that describes standards for developing interoperable software components (Grime, 1999; Williams et al, 2000). There are other such standards, like *JavaBeans* and *CORBA*, but COM appears to have emerged as the most viable technology for developing high-performance, interactive desktop and client/server applications. COM is language-neutral, and any language that can cope with binary standards can create or use COM components, e.g. C++, Java, Visual Basic, Delphi, etc.

2.6.2. COM objects

COM is built around the notions of *components* (often called *coclass*), *objects*, and *interfaces* (Grimes et al, 1999). These three different entities are defined and related as follows:

- A coclass (named from component object class) is a piece of binary code that implements some kind of functionality.
- A COM object is an instance of a coclass that gets loaded into memory when an instance of the coclass is created.
- COM interfaces are the only means by which other components and programs get access to the functionality of a COM component. An interface is a set of definitions of logically related methods that will control one aspect of the component operation. A coclass can have one or more interfaces. The definition of an interface does not include the implementation of its methods. This enables the actual implementation of the methods to be changed at any time without affecting other components that make use of that interface (see section 2.2.2.4). No matter what functionality they provide, all COM objects must implement a common interface called the *IUnknown*. This interface is the main interface for all others and is the base class from which all other COM interfaces are derived.

2.6.3. COM interfaces as contracts

COM enforces complete *encapsulation* of the data and implementation of a component. A client has no direct access to the data of a COM component except through one of its interfaces. The interface of a COM component therefore serves as contract between the component and its users (clients). This leads to two important issues:

- An interface once published must never change. This means that one must not add methods or remove methods; one must not change the signature of the methods or the calling conventions (Grimes et al, 1999). Any change in the published interface of COM component result in a completely new interface. However, coclasses can be upgraded by exposing new interfaces as long as they continue to support the original interfaces.
- Once a component has said it exposes an interface, any future version of the component should also support the original interface. This will avoid malfunctioning of existing clients.

2.6.4. COM identifiers

Because COM is language independent, the names of coclasses and interfaces must also be unique and language neutral. To achieve this, COM associates with every component a unique string of numbers known as *globally unique identifier* (GUID) or *universally unique identifier* (UUID). A GUID for a coclass is called *class identifier* (CLSID) and that of an interface is called *interface identifier* (IID). A GUID is a large, statistically unique, 128-bit number generated from the address of the Ethernet card in the computer. A typical GUID (CLSID or IID) in the registry looks like this:

{04FC747F-0C53-4B1C-B9AA-734505AE11A0}.

GUIDs ensure that there is no name clash and that every COM component is uniquely identified. The GUIDs of COM components are registered in the System Registry, the Microsoft Transaction Server (or the COM+ catalog) depending on the operating system and how the component is configured. The

system registry contains information that is used to map CLSIDs to the actual file locations. Because of this, a COM client never needs to know the actual location of the server it is using, not the directory or the drive, not even the machine it's on (Grimes et al, 1999). All that the client needs to know is that the component exists and that it supports the particular interface it is using. In the implementation phase (chapter 5), we will explain how the geodatabase uses the COM technology to attach behaviour to feature classes in the geodatabase.

2.6.5. Types of COM servers

COM components can be implemented in two types of code modules, namely, as *in-process* server or as *out-of-process* server. In-process servers are implemented in a dynamic-link library (DLL) and are run in the process space of the client, making access to the component very efficient. However, the client's memory is not protected and any error on the server side could result in a crash in the client. An out-of-process component is an executable (EXE) file that runs in its own process outside that of the client. Communication between a client and an out-of-process component is therefore called *cross-process,* and this tradeoffs performance for robustness because cross-process calls are slower, but does not affect the client code (Figure 2.5).

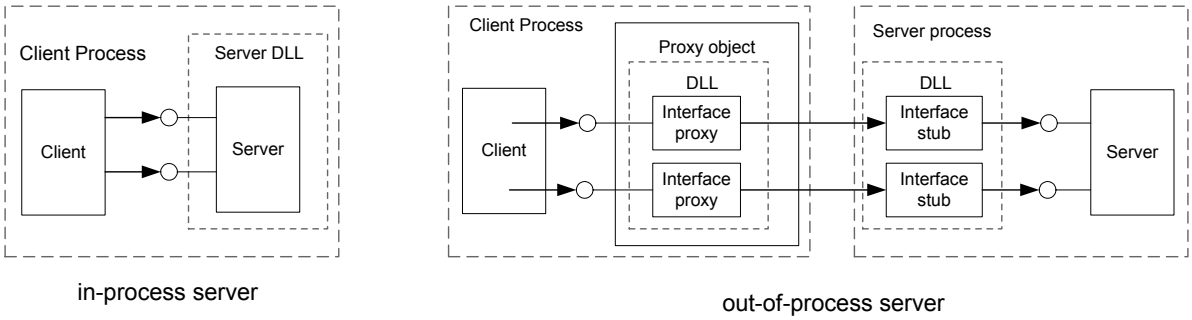


Figure 2.5 In-process and out-of process COM servers (Grimes et al, 1999)

2.6.6. Containment and aggregation

COM technology greatly enhances code reuse by packaging it as programmable components. In COM, there are two primary mechanisms for reusing one component from within another: *containment* and *aggregation*. Containment is similar to having one component as a member variable of another. This allows a component to reuse some or all of the interface implementations of one or more components. The outer object (controlling, containing) acts as a client to the other object, delegating implementation when it wishes to use the services of the contained object. The outer component manages the lifetime of the inner component. With aggregation, the outer component exposes directly the interfaces of the inner one as if they were implemented on the outer object itself (MSDN Library, 2001). To the client, aggregation (like containment) looks like one seamless object; it has no knowledge about this architecture. These techniques allows higher level of reusability within COM programming environment as existing components can be reused and easily extended without having to write components from scratch. This will be illustrated in the implementation phase.

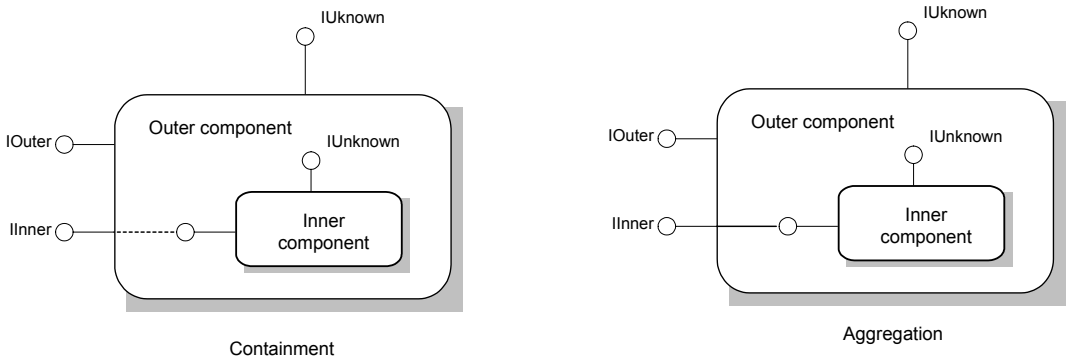


Figure 2.6 Containment and aggregation (Grimes et al, 1999)

2.6.7. Distributed COM

COM components can be created and accessed both on local and remote computers. In the remote case, we talk of distributed COM components, also known as DCOM. DCOM is the basic extension of binary COM objects across LANs (Local Area Network), WANs (Wide Area Network), or the Internet. DCOM provides a standard protocol that can sit on top of any standard networking protocol, such as TCP/IP, HTTP, etc. If connectivity is established between computers at the network layer, DCOM-based communications can occur. DCOM communications also work between dissimilar computer hardware platforms and operating systems. For example, the following platforms support DCOM: Windows 9x, Windows NT, Unix, Linux, Sun Solaris, Macintosh (Apple), etc. Other extensions of the COM technology include COM+, which is not discussed here (see the MSDN Library).

2.7. Concluding remarks

In this chapter, we explored three quite distinct technologies: object orientation, database and component-based technologies. All of these contribute considerably to the successful implementation of a geographic information system. Object orientation enhances both the design and implementation of a GIS; the database is the foundation of the geographic information system; and component-based technology facilitates the development of reusable, extensible GIS systems. We have introduced ESRI geodatabase and described how it integrates these three technologies to develop a system that can be easily extended to meet application-specific requirements. This will be illustrated in the implementation phase. In the next chapter, we will discuss spatial data models, spatial relationships and consistency constraints in spatial databases, to lay the necessary foundation for the design phase.

3. Spatial data models, spatial relationships and consistency constraints

3.1. Introduction

This chapter describes spatial data models, spatial relationships and consistency constraints. First, we introduce spatial data models, and describe the geodatabase model and show how it differs from the coverage model in terms of how topology is handled. The next section covers the mathematical background for modelling spatial relationships among objects (points, lines and polygons). These relationships, apart from their use in formulating spatial queries, are also used to enforce consistency of the database. The last section describes consistency in spatial databases and how the spatial relationships are used to define consistency constraints in spatial databases.

3.2. Spatial data models

Traditionally, the spatial component of real world features is structured according to either the raster (field-based) or the vector (object-based) approaches. The raster representation uses tessellations for geometric modelling. A tessellation is a partition of space into union of disjoint areas. This is further categorised as a regular (grid) and irregular tessellation (partitions with varying sizes and shapes). In the vector mode of representation, an individual object is the basic unit of representation. These object-based data models have been classified into unstructured and structured models. The former is known as a spaghetti data model and the later as a topological model. Topological models are the most popular models and can store rich topologic information although they are not simple in structure like the spaghetti structures (Raza, 2001; Molenaar, 1995; Worboys, 1995).

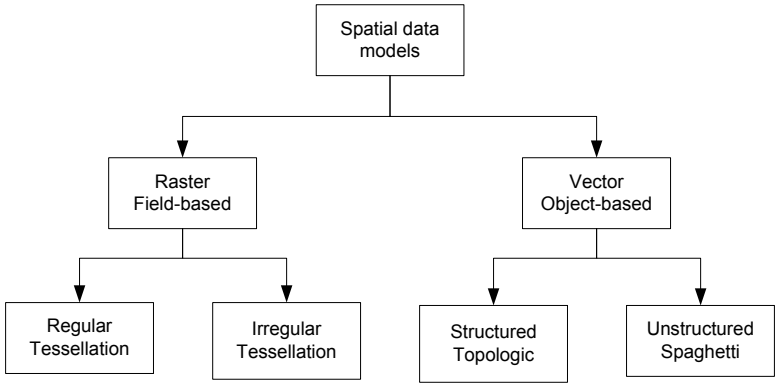


Figure 3.1 Categories of spatial data models

3.2.1. ArcInfo geodatabase model

Previous versions of ArcInfo (before version 8) were based on the coverage or the georelational data model in which objects have explicit topology. Topology deals with the spatial relationships between objects. In the georelational model, objects are structured into topological elements of nodes, arcs and polygons. The practical result of topology is that points know which arcs are connected to them, arcs know which points constitute their origin and destination, and which polygons they form the perimeters of (AGI, 1996). Topology is based on shared boundaries between areas. Any time the feature geometry is edited, the topological relationships must be recaptured and refreshed by the software. These topological associations between features are stored explicitly in (INFO) tables.

The geodatabase model of ArcInfo 8+ constitutes a major difference compared to the coverage model. Apart from the fact that both spatial and attribute data are stored in a relational database, topology in the geodatabase differs from topology in the georelational model. Topology in the geodatabase is based on coincident geometry rather than shared geometry. In other words, separate coordinates exist at the same location. In the geodatabase model, the following two classes of features are distinguished (Zeiler, 1999): *simple features* and *network features*. These are explained in the following sections.

3.2.1.1. Geodatabase simple features

Simple features have no connections or topological associations explicitly stored in the geodatabase. Topological information is inferred from the geometry of the features. Two polygons are adjacent if they have coincident segments, and each polygon stores all of the coordinates for itself. In Figure 3.2, eight coordinate pairs are stored in the geodatabase: four coordinates for each polygon (A and B). The two polygons are adjacent because they both have a segment that is made up of common coordinates (100.50, 200.50) and (100.50, 220.00). Each polygon also has coordinates that are not coincident with the other polygon: points 1, 2, 7, and 8. The same applies to simple point and line objects. The geodatabase simple features are similar to points, lines and polygons of shapefiles in ArcView. The topological elements of simple features, however, can be derived through on-the-fly editing in ArcMap (Zeiler, 1999).

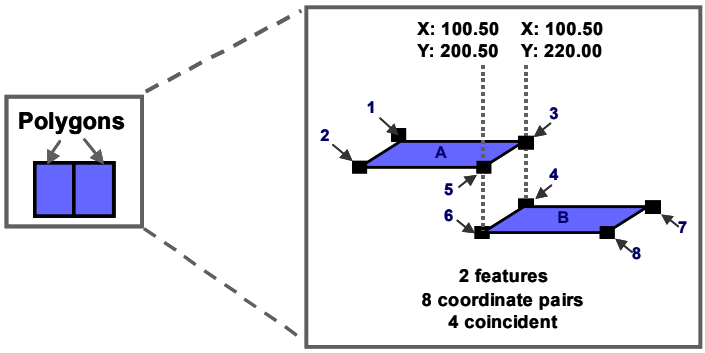


Figure 3.2 Geodatabase simple features (ESRI User Conference, 2000)

3.2.1.2. Geodatabase network features

In the geodatabase, features can be connected to form a geometric network, which consists of a matrix of connected points and lines that have coincident geometry. A geometric network is made up of lines that are connected through points. These are special lines and points that have a role in topology, and they are called *edges* and *junctions*, respectively. Edges are connected through junctions. The network features junction and edge correspond to *node* and *arc* in the georelational model, although the network features in ArcInfo 8 incorporate meaningful advances in design over nodes and arcs. In a geometric network, each feature stores all of the coordinates for itself. In Figure 3.3, there are seven coordinate pairs stored in the geodatabase. Each edge stores two coordinate pairs and each junction stores one coordinate pair. The coordinate (122.50, 200.50) is stored three times in the geodatabase, once for each edge and once for the junction connecting the edges. The connectivity between the edges and junctions is maintained when editing a geometric network. For example, if the junction between two edges is moved, the coordinates for the edges are modified with the junction so that their coincidence is maintained. Thus the network features are more similar to the node and arc concepts of the coverage model.

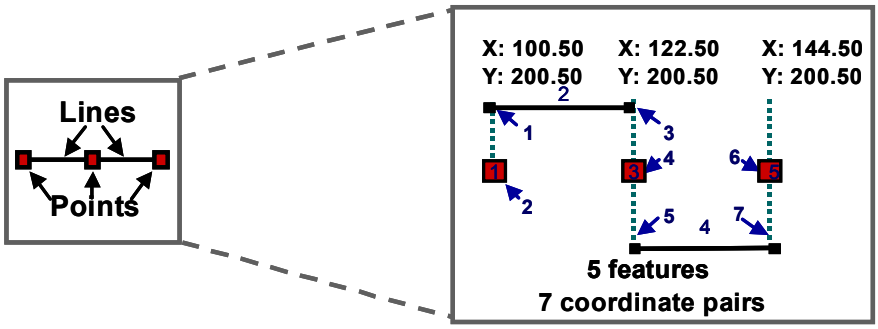


Figure 3.3 Geodatabase network features (ESRI User Conference, 2000)

3.3. Spatial relationships

This section describes the mathematical background for modelling spatial relationships. Spatial relationships describe the relationships between spatial objects and their geometric elements (Raza, 2001). In spatial databases, spatial relationships are needed for two main purposes:

- *For performing spatial queries.* Queries in spatial databases or GIS are often based on the relationships among spatial objects. For example, 'Retrieve all parcels that are adjacent to parcel A'. Such queries involve spatial conditions which standard query definition languages like SQL do not adequately support. Spatial relationships are needed at both the query formulation and processing levels (Egenhofer & Herring, 1991; Clementini et al, 1993).

- *For enforcing consistency of the database.* Spatial relationships are also used to formulate consistency constraints in spatial databases. For example, a violation of the constraint that 'two parcels should not overlap' in a cadastral database can be detected by checking the spatial relationship that exists between the two parcels (Kufoniyi et al, 1994). The spatial relationships provide the means for defining and monitoring these constraints in the database. Hence the formalization of the basic spatial relationships is an essential component in GIS development.

3.3.1. Categorization of spatial relationships

Relationships between spatial objects are usually classified according to their embedding space. In general, spatial relationships can be categorized into three classes, namely, *topological*, *metric* and *order* relationships (Egenhofer, 1989; see Figure 3.4).

- Topological relationships describe the connectivity, orientation, containment, and adjacency relationships among spatial objects (Raza, 2001). These relationships are invariant under topological transformations such as translation, scaling and rotation. Examples of topological relationships are terms like *neighbour* and *disjoint*.
- Metric relationships are those relationships that are described in terms of distances and directions. Examples are such descriptions as, 'within a distance of 2000m', '6m away from'. These descriptions depend on the absolute positions of objects relative to a given reference system (Worboys, 1995). For example, 'within 1000 metres from the first class road' depends on the absolute geometric position of the first class road.
- Order relationships: Spatial relationships can be formally described based on the mathematical theories of order and set (Kainz et al, 1993). In general, order relationships have converse relationships; for example, the order relation, 'behind' has the converse relation 'in front of'.

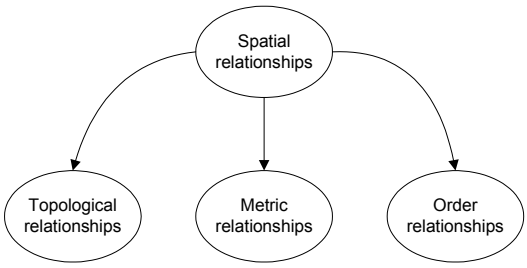


Figure 3.4 Categories of spatial relationships

The focus of this section is on the topological relationships, which are more detailed than order relationships (Kufuniyi, 1995). For the metric relationships, they are normally computed from the database using the coordinates of the objects. In the following sections, we will discuss the mathematical background for the formalization of the topological relationships between the basic objects, i.e., points, lines and areas.

3.3.2. Topological relationships

Topology is that branch of mathematics that studies the characteristics of geometry that remain invariant under certain transformations (*topological mapping* or *homoeomorphism*)(Kainz, 1995). A topological property is one that is preserved under topological transformations such as scaling, translation and rotation. Examples of topological properties are connectivity, adjacency, etc. There are two general branches of topology, both of which are applied in spatial data handling (Kainz, 1995; Worboys, 1995):

- Point-set (or analytic) topology—focuses on sets of points and is based on real analyses, using concepts such as open sets, neighbourhood and convergence.
- Algebraic (or combinatorial) topology—uses algebraic means to describe the spatial relationships and is based on such concepts as simplified and cell complex and graph theory.

The point set approach is the most general model for topological relationships (Raza, 2001). Using the point-set approach, topological relationships are defined in terms of three fundamental primitives of object parts, which are *interior* (denoted as $^{\circ}$), *boundary* (denoted as ∂), and *exterior* or *closure* (denoted as $^{-}$), which themselves are defined based on *neighbourhood* concepts. Formal definitions of these concepts can be found elsewhere (see Egenhofer, 1989; Egenhofer & Herring, 1991). Figure 3.5 shows illustrations of these primitives using a simple 2-dimensional objects with connected boundaries.

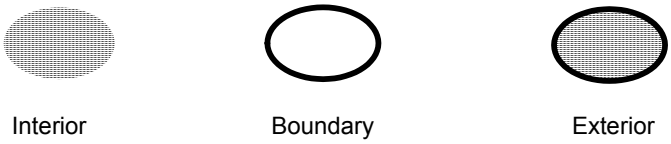


Figure 3.5 Interior, boundary and exterior of a region

3.3.3. The 4-Intersection model

The binary topological relationships between any two spatial objects have been derived based on the well-known 4-intersection model. In this model, all the possible relationships between two objects A and B have been derived by investigating the four intersections of the boundary and interior of A and the boundary and interior of B (Egenhofer & Franzosa, 1991). These are:

- The boundary-boundary intersection, denoted by $\partial A \cap \partial B$
- The boundary-interior intersection, denoted by $\partial A \cap B^o$
- The interior-interior intersection, denoted by $A^o \cap B^o$
- The interior-boundary intersection, denoted by $A^o \cap \partial B$

The 4-intersection model can be represented concisely as:

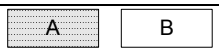
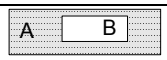
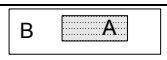
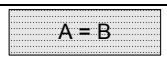
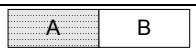
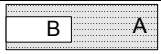
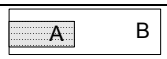
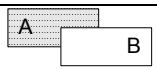
$$I_4(A,B) = \begin{pmatrix} A^o \cap B^o & A^o \cap \partial B \\ \partial A \cap B^o & \partial A \cap \partial B \end{pmatrix} \qquad \text{Equation 3.1}$$

Each of these intersections can have only two possible results, either empty (Ø) or non-empty (¬Ø). Therefore, a total number of 16 binary topological relationships can be distinguished considering the fact that they are topological invariant. However, in the 16 relationships, only 8 of them are meaningful in 2-dimensional space as proved in Egenhofer & Franzosa (1991). These are: *disjoint*, *contain*, *inside*, *equal*, *meet*, *covers*, *overlap* and *coveredBy* (Table 3.1). Among these 8 relationships, there are two pairs of converse relationships, namely:

- A *inside* B implies B *contain* A
- A *coveredBy* B implies B *covers* A

Therefore, the number of relationships can be further reduced to six, which are: *disjoint*, *inside*, *touch*, *equal*, *covers* and *overlaps*.

Table 3.1 Topological relationships between two regions in 2D (the 4-intersection)

Spatial Relationship	4-intersection				Illustration
	$A^o \cap B^o$	$A^o \cap \partial B$	$\partial A \cap B^o$	$\partial A \cap \partial B$	
Disjoint	Ø	Ø	Ø	Ø	
Contains	¬Ø	¬Ø	Ø	Ø	
Inside	¬Ø	Ø	¬Ø	Ø	
Equal	¬Ø	Ø	Ø	¬Ø	
Meet	Ø	Ø	Ø	¬Ø	
Covers	¬Ø	¬Ø	Ø	¬Ø	
CoveredBy	¬Ø	Ø	¬Ø	¬Ø	
Overlaps	¬Ø	¬Ø	¬Ø	¬Ø	

3.3.4. The 9-Intersection model

The 4-intersection considers only the boundaries and interiors of the objects in 2-dimensional space and investigates the intersections of the four parts to derive the topological relationships. However the 4-intersection model cannot distinguish all the detailed relationships between any two arbitrary objects with different dimensions, such as *line-region* and *line-line*, etc, in a 2-dimensional space (Egenhofer & Sharma, 1993). To derive all possible topological relationships between two arbitrary spatial objects, Egenhofer et al (1991) extended the 4-intersection model by considering the exterior ($^-$) of the objects. For any two objects A and B, the topological relationships between them are derived by comparing A's boundary (∂A), interior (A^o), and exterior (A^-) with B's boundary (∂B), interior (B^o) and exterior (B^-). The resulting model is called the 9-intersection and leads to 9 possible intersections that represent all possible topological relationships between any two objects. The 9-intersections can be represented in a 3x3 matrix as shown below:

$$I_9(A,B) = \begin{pmatrix} A^o \cap B^o & A^o \cap \partial B & A^o \cap B^- \\ \partial A \cap B^o & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^o & A^- \cap \partial B & A^- \cap B^- \end{pmatrix} \qquad \text{Equation 3.2}$$

With each intersection being empty or non-empty, this model delivers a total of 2^9=512 topological relationships between any two objects. All the 512 relationships are mutually exclusive and provide a complete coverage. However, not all of the 512 relationships are topologically valid in a 2-dimensional space (Egenhofer & Herring, 1991, Egenhofer & Sharma, 1993).

3.3.5. Group of relationships

The 9-intersection model can be applied to all kinds of geographic objects (Ubeda & Egenhofer, 1997). Considering points, lines and areas, this leads to six groups of relationships, namely: point/point, point/line, point/area, line/line, line/area, and area/area. Egenhofer & Franzosa (1991)

gave the list of relationships that can be realised in each group for objects embedded in 2D space (see Table 3.2).

Table 3.2 Number of relationships per group according to the 4-intersection and 9-intersection

Group of relationships	9-intersection	4-intersection
Point/Point	2	2
Point/Line	3	3
Point/Area	3	3
Line/Line	23	16
Line/Area	19	11
Area /Area	8	8

It can be observed that the 4-intersection is a subset of the 9-intersection, and so the 9-intersection will be able to distinguish greater number of relationships than the 4-intersection. In addition, the actual number of relationships that can be realized between any two objects depends on the dimension of space with respect to the dimension of the objects, and on the topological properties of the objects embedded in that space. Generally, there are more relationships between objects embedded in a higher dimensional space. For region-region configuration in a 2-dimensional space (R^2), only 8 of the relationships make sense and the 9-intersection delivers the same results as the 4-intersection (Egenhofer & Herring, 1991).

3.3.6. The dimension extended model

The 4-intersection and the 9-intersection models have been extended by Clementini et al (1993) by taking into account the dimension of the intersections. Thus instead of distinguishing only empty and non-empty intersections, the dimension of the intersection is also taken into account. The resulting model is known as the *dimension extended model* (DEM). In 2-dimensional space, the dimension of the four intersections can be one of the following: Ø (empty), 0D(point), 1D(line) or 2D(area). Theoretically, these four possibilities result in 4^4=256 different cases, but they proved that many of these are invalid leading to a reduced number of real relationships. To illustrate this extension, a dimensionally extended 4-intersection for the line/area group will result in only the following possible cases:

$$DEI_4(A,B)=\begin{pmatrix}\dim(\partial A\cap\partial L) & \dim(\partial A\cap L^o)\\ \dim(A^o\cap\partial B) & \dim(A^o\cap\partial L^o)\end{pmatrix}=\begin{pmatrix}\{\text{Ø or 0D}\} & \{\text{Ø},0\text{D},1\text{D}\}\\ \{\text{Ø},0\text{D}\} & \{\text{Ø},1\text{D}\}\end{pmatrix} \qquad \text{Equation 3.3}$$

This is due to the fact that the dimension of the intersection cannot be higher than the lowest dimension of the two operands of the intersection: $\dim(\partial A)=1$, $\dim(A^o)=2$, $\dim(\partial L)=0$, $\dim(L^o)=1$. Furthermore, the definitions of line and area features exclude the option that $\dim(A^o\cap\partial L^o)=0$. This reduces the number of cases from 256 to 24. Further geometric considerations bring the number of possible relationships to 17, which have been proved in (Clementini et al, 1993). Similar analysis for the other groups result in reduced number of relationships (Table 3.3).

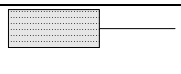
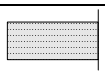
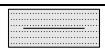
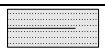
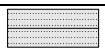
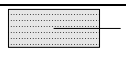
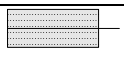
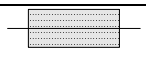
Table 3.3 Number of relationships according to the DE-4IM

Group of relationships	DE-4IM
Point/Point	2
Point/Line	3
Point/Area	3
Line/Line	18
Line/Area	17
Area/Area	9
Total	**52**

3.3.7. The calculus-based model

From the perspective of end-users, all the above models result in too many spatial relationships that may be too difficult for humans to remember and to use in performing spatial queries. Therefore, in Clementini et al (1993, 1995), the authors introduced the *calculus-based* method for describing topological relationships with the aim of reducing the number of spatial relationships to a smaller number as possible. The calculus-based method also takes into account the dimension of the result of the intersection, but uses *object-calculus* concepts to model the spatial relationships. In this method, relationships that are somehow similar are grouped into *five* general relationships that are suitable for the end-user interaction. These general relationships are: *touch*, *in*, *cross*, *overlap* and *disjoint*. The authors have proved that these five relationships are mutually exclusive and that there are no possible relationships outside these five relationship groups. The five relationships may be applied to any groups of points, lines and areas. Table 3.4 gives example illustrations of the topological relationships between line/area group according to the calculus-based model.

Table 3.4 Line/area relationships group according to the calculus-based model

Disjoint			
Touch			
In			
Cross			

3.4. Consistency constraints

Consistency or integrity constraints are the conditions that must hold for the data in the database to be consistent, valid and useful. Consistency is not the same as correctness. Consistency refers to the lack of any logical contradiction within a model of reality (Egenhofer et al, 1994). This must not be confused with correctness, which excludes any contradiction with the reality. And as Egenhofer (1991) observed, database systems can only check for internal contradiction (consistency), not correctness. An inconsistent data violates a defined set of constraints and this can lead to erroneous query results. Consistency constraints are used to enforce the logical consistency of the data in the database. These

constraints are necessary as the updating process may disturb the consistency of the data. The following sections describe the issue of enforcing consistency or integrity constraints in the database.

3.4.1. Classification of consistency constraints

In traditional database management systems, the specification of consistency or integrity constraints is often viewed at three levels of the database system, namely, *inherent*, *implicit* and *explicit* constraints.

- Implicit constraints are inherent in the data model itself and do not need to be specified explicitly in the schema.
- Implicit constraints are specified using the data definition language (DDL). An example is the constraint that every key field must have unique values in a relational database.
- Explicit constraints are also called *business rules* or *adhoc* constraints. They are application dependent and cannot therefore be captured in the data structures. They are defined and enforced by the application program that uses the database.

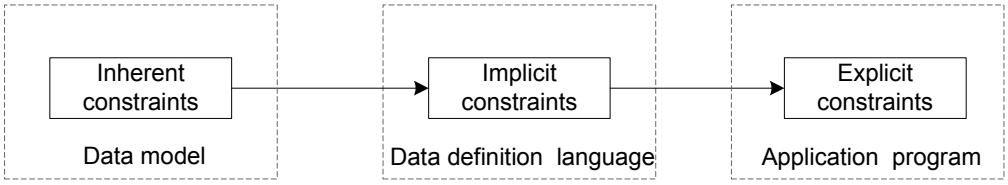


Figure 3.6 Classes of integrity constraints in terms of where they are applied

Integrity constraints have also been categorized (in terms of database states) into *static*, *transition*, and *dynamic* constraints (Elmasri et al, 2000). These are briefly described as follows:

- Static constraints express state-independent properties that must hold for all states of the database. It depends only on the current state, independent on any previous state of the database. Examples of static constraints are *attribute domains*, *primary keys*, and *referential integrity* constraints, which must hold at all times.
- Transition constraints impose restrictions on pairs of states of the database, the *before* and *after* states of a transaction. For example, a constraint can be specified that when a salary is being updated, the salaries should not decrease.
- Dynamic constraints allow expressing conditions over sequence of two or more database states (Pacheco, 1997). For example, 'a renewed book should be returned within a week'. This constraint depends on the previous state of the database, i.e. whether or not the book has been renewed beforehand.

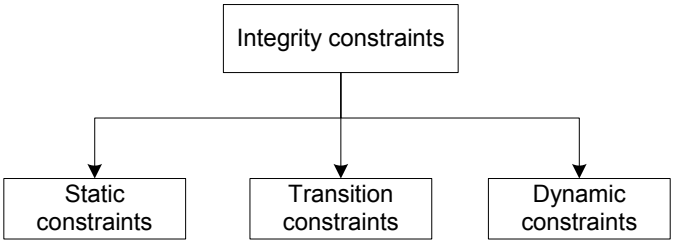


Figure 3.7 Classes of integrity constraints in terms of database states

Dynamic constraints are outside the scope of this research, and are hereafter not discussed. Only static constraints are considered here and in the context of spatial databases. Readers are referred to Pacheco (1997) for detail survey of the various approaches proposed to deal with dynamic constraints.

3.4.2. Consistency constraints in spatial databases

In spatial databases, the static consistency constraints of interest may be categorized into *topological*, *semantic* and *user-defined* constraints Cockcroft (1996, 1997). These are discussed below.

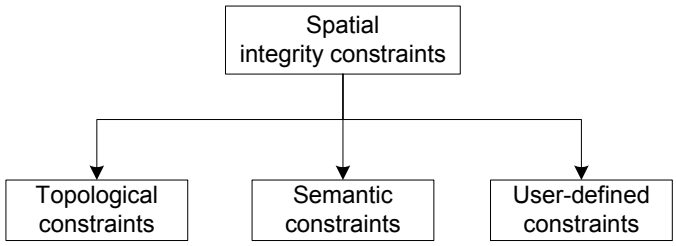


Figure 3.8 A classification of integrity constraints in spatial databases

3.4.3. Topological consistency constraints

Topological consistency constraints are constraints related to the topological/geometric elements of objects in the vector data model. In GIS, two types of geometric objects are often used to represent the geographic space, namely, *simplices* and *cells* (and their complexes). The cell complex is the most general model for topological representations of spatially referenced data (Kainz, 1995). The conditions that are used for the definition of cell complexes are used to define consistency constraints in the topological data model, resulting in the following five topological constraints between *0-cell* (nodes), *1-cell* (arcs) and *2-cell* (polygons):

- Every 1-cell is bounded by two 0-cells
- For every 1-cell, there are two 2-cells (left and right polygon)
- Every 2-cell is bounded by a closed cycle of 0- and 1-cells
- 1-cells intersect only in 0-cells
- Every 0-cell is surrounded by a closed cycles of 1- and 2-cells

These constraints relate to the topological elements of objects and may be used to test the consistency of a topological data model. Kainz (1995) states that all five conditions must hold for a partition of space into area features, and for a network analysis, only the first and second rules are necessary. Topological constraints are usually explicitly coded in the topological data model and well handled by most current GIS systems. For example, in ArcInfo georelational model, the arc-node structure supports these constraints (arc-node topology, polygon-arc topology and left-right polygon). The topological information is stored in a separate table. With respect to the geodatabase model, the network features support the first and last constraints, although the network features in the geodatabase incorporate advanced designs than nodes and arcs. Through connectivity rules, users can define which edges (arcs) connect to which junction (nodes), and other custom constraints. ArcInfo automatically manages these rules. Geodatabase simple features, however, have no topological associations with other features; hence these topological constraints do not apply to them. Each object is independently

stored as a separate row in the geodatabase and can be edited independently from other objects. Therefore constraints on geodatabase simple features are constraints between individual objects, and are mainly semantic or user-defined constraints.

3.4.4. Semantic consistency constraints

Semantic consistency constraints differ from topological constraints in the sense that they are concerned with the *meaning* of geographic objects in reality (Cockcroft, 1996). These constraints are mostly forbidden relationships between objects. A dataset may be topologically consistent (satisfying all the topological constraints) but semantically inconsistent (Kufoniyi, 1994). For example, it is topologically consistent for a line object to cross an area object in so far as topological constraints are fulfilled. But this situation may or may not be semantically consistent depending on the meaning of the two objects in a particular application domain. For example, in a cadastral application, an important semantic constraint is that 'a river (modelled as a line) may not flow into or through a cadastral parcel (an area object)'. Such a situation is considered to be semantically inconsistent since it does not normally occur in reality. Semantic constraints are always application dependent (Molenaar, 1995). For this reason, it is not possible to provide a generic algorithm for resolving their violation; hence current GIS systems leave the enforcement of semantic constraints to the application developer.

Semantic constraints can be spatial (e.g. 'a building should not be inside a road') or non-spatial (e.g. 'the height of a mountain should not be negative'). Non-spatial constraints that restrict attribute values of objects to certain types can be enforced by using domains; for other constraints, code has to be written to enforce them. Binary semantic constraints that have topologic structure can be defined and monitored by translating them into spatial relationships between the objects involved. This approach works under the assumption that the database is topologically consistent. For example, the constraint that 'buildings should not intersect roads' can be enforced by monitoring and checking for *overlap*, *cover* and *equal* spatial relationships between building objects and road objects. Other constraints may apply to single objects, for example, constraints on the geometric size/shape of objects in the database.

3.4.5. User-defined constraints

These differ from semantic constraints in the sense that they are more esoteric in nature and not necessary based on semantics of geographic objects (Cockcroft, 1996, 1998). User-defined constraints are also application-dependent and allow the database consistency to be maintained according to the user-defined constraints analogous to business rules in non-spatial systems. For example, for health or other reasons, it may be desired to locate a waste disposal plant at a certain distance away from residential areas. Such constraints are not necessary based on object semantics but on general requirements such as legal, economic or other requirements.

3.4.6. Constraints enforcement in GIS

Topological consistency constraints are usually handled by the GIS system. The main issue with semantic and user-defined constraints is that because they are application-dependent, it is difficult to develop generic algorithms for them. Thus their enforcement is left to the application modeller. The constraints that have topologic structure can be enforced by using the spatial relationships between the objects. For other constraints, application programs have to be written to enforce the. For example, Cockcroft (1996, 1997) discussed the use of external repository to enforce these constraints. In the object-oriented environment, methods provide the natural means of enforcing consistency constraints in the database. These constraints can be defined as methods on the objects in the class, and each class will enforce its own constraints. This approach has been used in this thesis (see chapters 4 and 5).

3.5. Concluding remarks

This chapter described spatial data models, spatial relationships and consistency constraints in spatial databases. We described the geodatabase model and showed how topology is handled. We described four methods for modelling the spatial relationships among objects, namely: the 4-intersection, 9-intersection, dimension extended model and calculus-based model. The 4-intersection considers only the interior and boundaries of the objects; the 9-intersection includes the exterior; and the dimension extended method includes the dimension of the intersection. The calculus-based method groups similar relationships into five general relationships and aims at reducing the number of relationships to suit the end user. The last section of the chapter discussed consistency constraints in spatial databases. The constraints are classified as topological, semantic and user-defined, according to their nature. In the next chapter, the concepts described in chapters 2 and 3 are employed to design an integrated conceptual model for the Top10Vector topographic dataset chosen for the prototype implementation in this research.

4. Object-oriented design of a topographic database

4.1. Introduction

This chapter deals with the development of the conceptual model. The conceptual model consists of three main components: FeatureClass, ConstraintClass and TemporalClass models. The FeatureClass model describes the feature classes with their attributes structure, operations and relationships. The ConstraintClass model deals with the constraint rules. The TemporalClass model deals with the modelling of time. These components are systematically developed and then unified into one integrated model. The chapter begins with the requirements analysis for the application domain. Then we present the conceptual model at the highest level for modelling these requirements. The FeatureClass model is developed next. The next section describes how we model the consistency constraints. Then we show in detail how we handled multiple representations in the Top10Vector involving road areas and their centrelines. The last section covers how time is modelled in this research.

4.2. Application domain and requirements

Topographic data play an important role in planning and decision-making. In many countries, national mapping agencies have the mandate to produce topographic data for their nation. The Topographic Service of the Netherlands is responsible for producing topographic data for the country. They produce topographic data at different scales; the most important ones are the TopVector series (at scales 1:10,000; 1:50,000; 1:100,000; 1:250,000; 1:500,000), known as the Top10Vector, Top50Vector, etc. The Top10Vector is the most detailed dataset produced by the Topographic Service. It is a vector dataset with a closed structure and built from coded and connected line elements. Traditionally, the Top10Vector has not been structured into a database, but stored as files. The content and data structure are determined for cartographic map production and for this reason, the data cannot always be used directly in applications. Users have to compose an object-oriented dataset for specific applications. There are several problems that arise from the current approach:

- To build an object-oriented dataset from the existing data structures means users have to process their own data. The interpretation of the data gives problems since the data are structured on the basis of a coding system for the production of maps.
- It is difficult to link the existing data with other datasets.
- Checking for inconsistency in the data is done manually. This is a time-consuming and interactive process and mostly depends on visual checking, hence obviously not effective.
- The Top10Vector involves multiple representations such as road sections represented as areas and as centrelines. Traditionally, different representations are stored in separate files with no interrelationships. This leads to lack of data consistency and maintenance problems.

- No temporal information is recorded for spatial objects. Hence no temporal queries or analysis can be performed, except by manually comparing different time slices. It is difficult to ensure that two users are referring to the same version of an object when updates are delivered to them. It is also difficult for users to identify immediate successors and predecessors of objects.

4.2.1. A new data model

Due to above and other problems, the Topographic Service began a project with the aim to restructure their datasets and products, and to take advantage of the current object-relational technology in order to best respond to increasing customer demands. The aim of this project is to replace the existing data model of the Top10Vector and the other TopVector data structure by an object-oriented model. For this purpose, the Topographic Survey started (in 2000) a project with the aim to design a new object-oriented data model that could be eventually implemented in an object-relational DBMS. The project began with the definition of the user requirements. Then the conceptual model was developed using object-oriented approaches. A prototype implementation is currently being carried out in GML (Geography Markup Language). This research will therefore constitute a preliminary investigation into the suitability of the object-relational approach to meet the requirements of the Topographic Service.

4.2.2. Requirements analysis

Any good design is usually preceded by a good analysis of the application requirements and problems. The requirement analysis for the new Top10Vector data model has been carried out by the Centre for Geo-Information (CGI) of the Wageningen University in consultation with the Topographic Service of the Netherlands. The requirements have been grouped into eleven categories (de Zeeuw et al, 2000). The analysis indicated the demand for the following, which constitute the problem of this research as stated in the problem definition phase in chapter 1:

- Enforcement of consistency constraints in the database.
- Management of multiple representations involving road areas ad centrelines.
- Modelling the temporal behaviour of objects in the database.

4.2.2.1. Consistency constraints

The enforcement of consistency constraints in topographic databases is a very important aspect of maintaining the consistency of the data. There are so many constraints that can found in the Feature Collections Rules for the Top10Vector dataset. Given the limited timeframe for this research, only some of the important constraints are considered here. Examples of the important constraints are described below.

Semantic constraints

- Buildings should not intersect roads (apart from a few exceptions e.g. where buildings overlap roads which case is distinguished by height level attributes).
- Road centrelines must meet each other.
- Built-up areas should not be inside water bodies (apart from a few exceptions).
- Road, water, built-up and terrain area features should be adjoining to each other and form a complete coverage. This is a complex constraint and can be decomposed into a set of *touch* relationships between instances of the feature classes involved.

Attribute domains

Most of the attributes of the feature classes have their own domains, which specify the required attribute values that can be entered for an object. The objects cannot have attribute values apart from those specified in the domains. For example, the attribute 'Road_Type' can only be one of the following values: *connection*, *crossing*, or *area*. The complete list of attribute domains for each feature class is given in appendix A.

Minimum size constraints

These are constraints on the minimum size of objects that should be represented in the database. The following are examples of these constraints:

- The minimum area size of a built-up area should be 9 square metres.
- The minimum length for double track in a single-track railway is 500 metres.
- If a road width is more than 2 metres, it is represented as an area object; otherwise it must be represented as a line object.
- The minimum depth of ditches is 50 centimetres.

Height level attributes

Road, railway, water, built-up, terrain and construction objects in the Top10Vector dataset are given a height level attribute. Objects that are allowed to cross or overlap each other must have different height level attribute values, both of which must be greater than 0. For example, two road objects can cross each other if one is underpass (height level is 1) and the other is overpass (height level is 2). Non-overlapping objects are assigned a height level value of 0.

4.2.2.2. Multiple representations

The Top10Vector multiple representations considered in this research involves road areas and their centrelines. The purpose of this research is to develop techniques for managing and maintaining consistency between two representations. This involves maintaining the consistency of the topological relationships between the road areas and the centrelines as well as attribute consistency, by developing update propagation mechanisms.

4.2.2.3. Temporal requirements

The temporal requirements describe the history of objects. As was stated in the problem definition in chapter 1, it is not the objective of this research to develop a fully functional temporal GIS. The main concern here is to develop a database that would keep track of the history of objects in the database. This will enable simple temporal or spatio-temporal queries to be answered as may be needed in a topographical mapping. It is also the aim to illustrate how the object-oriented concepts offer flexibility for capturing the temporal behaviour of spatial objects into the database.

4.3. The conceptual model

The conceptual model consists of three main components which are called the FeatureClass model, the ConstraintClass model, and the TemporalClass model. The FeatureClass model describes the feature classes with their attribute structure, operations, and relationships. The ConstraintClass model shows the constraints classes with their attributes and operations. The ConstraintClass is responsible for the storage and management of the constraints that apply to the objects in feature classes. These constraints are invoked during editing of the data to determine inconsistencies. The TemporalClass model deals with the modelling of time. All modified and deleted objects are removed to the TemporalClass with time stamps. This will enable temporal queries to be performed. The link between the FeatureClass and the TemporalClass is given by the object identifier (ObjectID). Through the ObjectID, it will be possible to retrieve historical objects from the TemporalClass. These three components are systematically developed and then integrated. The approach here is to use object-oriented concepts to systematically identify classes, attributes, relationships and operations or methods for each class.

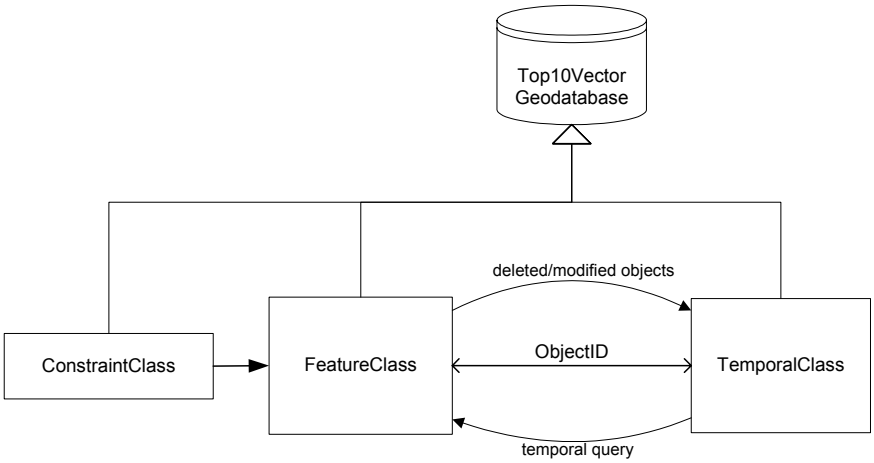


Figure 4.1 The integrated conceptual model

The conceptual model has been designed to be general, and although it has been implemented as extension to the geodatabase model, it can be adapted in other object-relational or object-oriented GIS system. In the sections that follow, we describe how each of the three components has been developed.

4.4. The FeatureClass model

The FeatureClass model consists of the features classes created for each geographic feature type, their attribute structure, operations and relationships. The FeatureClass model is shown in Figure 4.2. It consists of feature classes that hold spatial data as well as abstract classes that provide specifications for a set feature classes. The base class, Top10Object, is an abstract class that defines common specifications for other classes. All feature classes have been defined as specializations of this class. Linear features can participate in a linear network (the NetworkFeatures class). However, network modelling is not part of this research, and thus the details of this component will not be considered further. The model shows composite relationships between RoadArea and the RoadCentreline classes, as well as between RailwayArea and RailwayCentreline classes. These relationships as well as the operations defined for the classes are used for modelling the multiple representations in section 4.6.

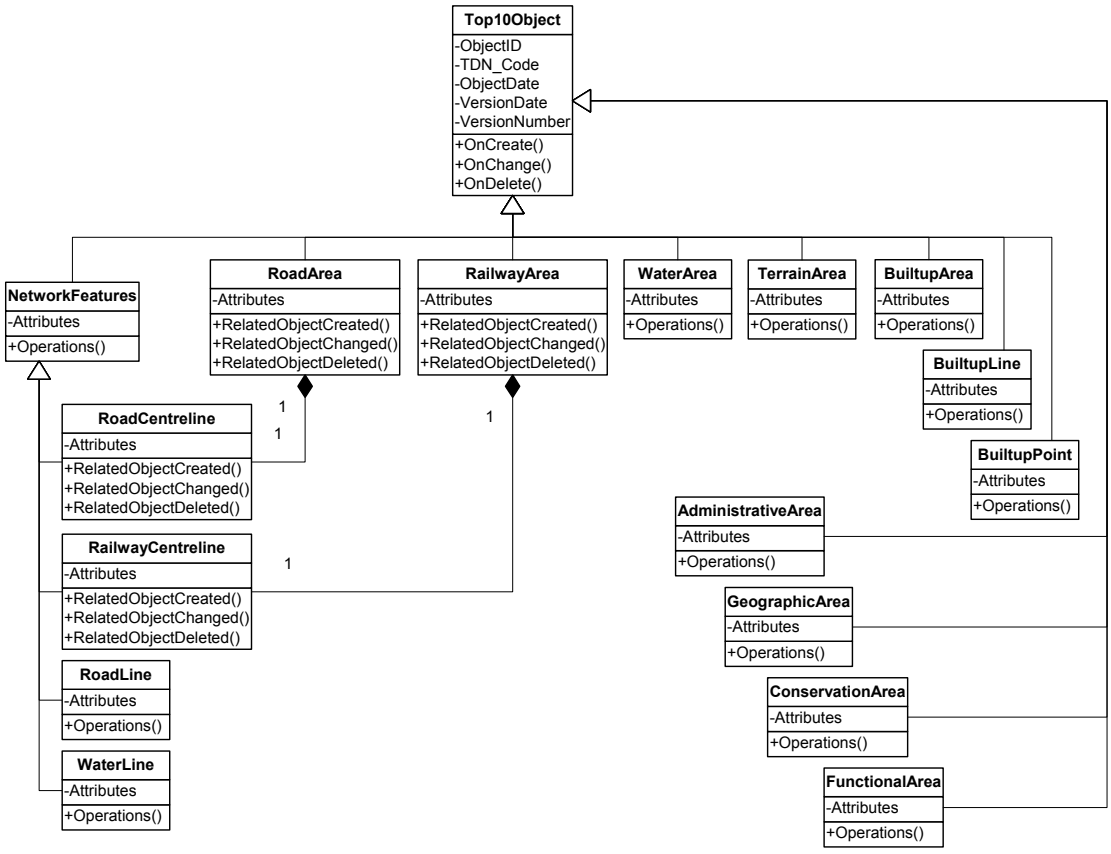


Figure 4.2 The FeatureClass model

4.4.1. The Top10Object class

There are some attributes and operations that are common to all feature classes. These have been generalized to form a more general class represented by the Top10Object class. This class is an abstract class that defines the minimum attribute structure and operations that every feature class in the database should have. Thus all feature classes are defined as specializations of the Top10Object class. The attributes of the Top10Object class are as follows:

- ObjectID—Every object is given a unique identifier called the ObjectID (called TOP10_ID in the original conceptual model).
- TDNCode—Refers to the old object code in the previous Top10Vector dataset.
- ObjectDate—Used for modelling temporal behaviour.
- VersionDate—Used for modelling temporal behaviour.
- VersionNumber—Used for modelling temporal behaviour (see section 4.7).

There are at least three basic operations that are defined for each feature class. These operations are: *OnCreate*, *OnChange*, and *OnDelete*. All feature class must implement at least these three operations. The purpose of these will be used to monitor the three types of events that characterize object lifecycle (i.e., creation, mutation and destruction of objects in the database), and send messages to corresponding feature classes to take the appropriate actions. Since all feature classes must implement these three operations, they are defined in the Top10Object class so that they are inherited by all classes that are defined as specializations of this class. How each feature class will implement these operations may vary depending on the behaviour, constraints, and services provided by that class.

4.5. Modelling consistency constraints

It is not possible to handle all of these constraints within the limited timeframe for the MSc research; therefore only some of the important constraints will be modelled here. However, the principles that are developed here can be applied to other similar constraints. Figure 4.3 shows the ConstraintClass model. The constraints have been grouped into spatial and non-spatial. Three types of constraints have been modelled here, namely, binary constraints, minimum size constraints and attribute domains.

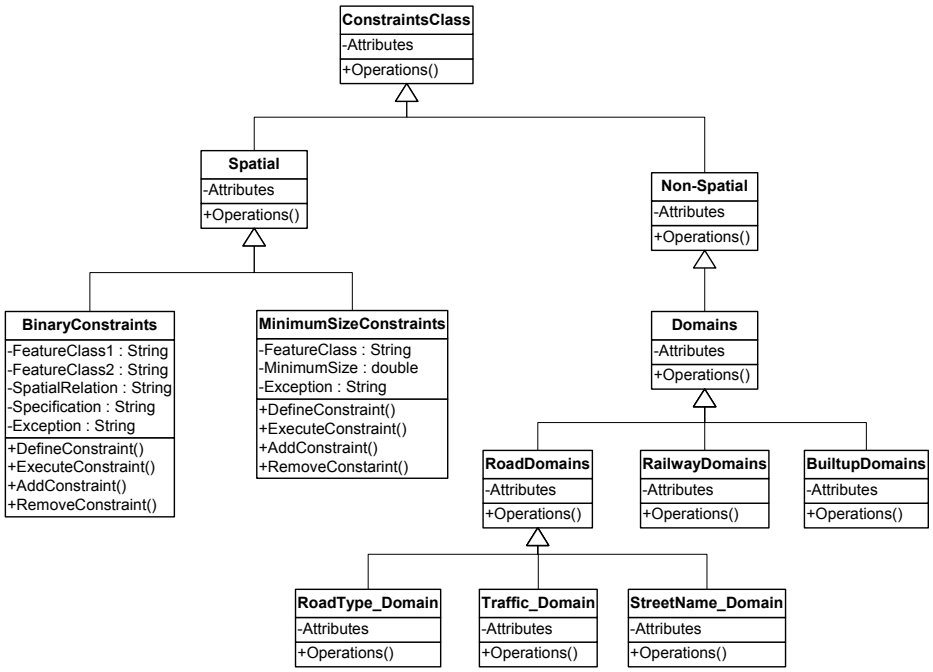


Figure 4.3 The ConstraintClass model

4.5.1. Binary constraints

Most of the constraints are binary semantic or user-defined constraints between two spatial objects. These are the constraints that can be translated into the topological relationships between the objects involved. These spatial relationships will be monitored and used as 'alerters' when they occur to trigger the particular method associated with the constraint class. The general form of a binary constraints definition will be as follows:

Constraint = (FeatureClass1, Spatial Relationship, FeatureClass2, Specification, Exception)

This binary constraint rule is defined between instances of *FeatureClass1* and instances of *FeatureClass2* with a *Spatial Relationship* (operator) between them, followed by a *Specification*, and an *Exception* to the constraint rule.

The *Specification* can be one of the following:

- Allowed: This situation is allowed, but must not necessary be true.
- Not allowed: This situation is not allowed, excluding any exceptions defined.
- At least once: This situation must hold at least once.
- At most one: This situation must hold not more than once.

The *Exception* allows the user to define an exception to the constraint rule. For example, two road objects cannot overlap each other except they have different height level attributes greater than 0. This means that the two roads can overlap if one road is an *underpass* (with height level 1) and the other is an *overpass* (height level 2). In such situations, the two roads can overlap each other in a 2-dimensional layer and the *exception* part will allow the user to specify this exception to the constraint rule.

To provide a simple and useful mechanism for users to easily define the constraint rules, the *calculus-based* method will be adopted in which all the possible spatial relationships between points, lines and areas have been grouped into only five relationships. These five relationships are: *touch*, *in*, *cross*, *overlap* and *disjoint* defined for the six groups of points, lines and areas. Users can select from these spatial relationships, and the system will test for all the relationships in that group to determine the inconsistency. For example, for the constraint that a building should not be *in* a road, the system has to test for *Inside*, *CoveredBy* and *Equal* spatial relationships in the original 9-intersection model (Figures 4.4).

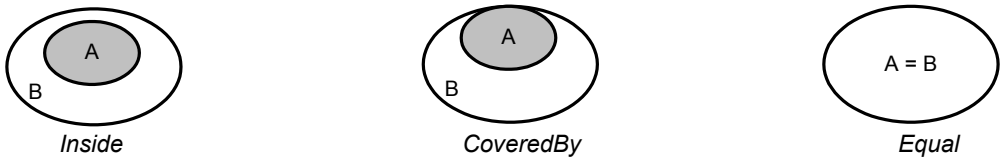


Figure 4.4 'In' relationship group according to the Calculus-based model

4.5.1.1. The BinaryConstraints class

The BinaryConstraints class will store and manage the binary constraints. The attributes of this class will be according to the constraint definition, i.e. FeatureClass1, Spatial Relationship, FeatureClass2, Specification and Exception. The following operations are defined for the user's interaction.

- DefineConstraint: Used to define the constraint rules.
- AddConstraint: Used to add constraints to the constraint class.
- RemoveConstraint: Used to remove existing constraints.
- ExecuteConstraint: Used to execute the constraint rules.

4.5.2. The MinimumSizeConstraints class

These are constraints on the size of objects in a feature class. For example, the minimum size of a built-up area should be 9 square metres. These constraints do not have topologic structure and cannot be enforced by using the spatial relationships. Some other code must be written to enforce them. These constraints are stored in the MinimumSizeConstraints class. The attributes of this class are as follows: FeatureClass, MinimumSize, and Exception; these have similar meanings as explained for the binary constraints. The same operations are also defined for the end-user interaction. These constraints will then be applied to new and modified objects during the updating of the database.

4.5.3. Attribute domains

Constraints that restrict the possible value for attributes will be implemented as domains on the attribute fields. These domains will be assigned to corresponding fields during the creation of the database schema. User domains are usually directly supported in any current system so no special design or code has to be written for them. Interfaces however can be designed for the user to update these domains.

4.5.4. Mechanism for monitoring constraint violations

Within the OnCreate and OnChange operations for each feature class, algorithms will be developed to monitor constraint violations during the creation of new objects and the modification of existing objects. In general, monitoring constraint violations will involve the following routines:

- Searching the corresponding constraints classes for any existing constraints that apply to objects of the feature class being updated.
- Executing these constraints against the new or modified objects.
- Giving warnings to the user in case of constraint violations.

4.6. Multiple representations

4.6.1. What are multiple representations and why are they needed?

GIS databases are intended to keep an integrated and consistent representation of the real world to support a wide range of application requirements. While the real world may be unique, its representation depends on the intended purpose. In other words, any spatial database is built with certain applications in mind; therefore different applications that have different and overlapping views of the real world normally require different representation of the same real world phenomena. The concept of multiple representations is intended to meet these requirements. The same objects are represented more than once at different abstraction levels tailored towards the need of different users or analysis (Egenhofer et al, 1994; Kilpelainen, 2000).

A typical example of applications that require multiple representations is cartographic map production at different scales. Map producers need to make maps of the same geographic features at different levels of abstractions (i.e. at different scales or resolutions) to satisfy multiple users. Traditionally, they maintain one database per resolution, with no interrelationships, hence with no update propagation and no inter-database consistency (Spaccapietra et al, 1999, 2000a). This traditional approach has evident drawbacks and normally leads to maintenance problems and lack of database consistency. Modern data management requires that a new representation paradigm be used, such that multiple representations of the same real world phenomena coexist in one database with mechanisms for maintaining consistency between them. A GIS system that supports multiple representations will allow one to store all representations in a single database, and enforce the consistency of these representations and allow update propagation between them. A geographic information system will be called consistent if it can ensure that whenever an object changes, all other objects which are related to the changed object in one way the other are automatically updated.

4.6.2. Managing multiple representations

When multiple representations of the same real world objects coexist in the database, the main issue that arises is how to manage and maintain the consistency between the different representations. For example, how to maintain topological consistency between the representations at different levels. Topological information is generally considered as first-class information; therefore maintaining topology across multiple representations of the same objects is of prime importance. The mathematical framework for assessing topological consistency between multiple representations has been researched by Egenhofer et al (1994), and will not be discussed here.

In general, when multiple representations coexist in the database, the system should be made aware so that it may manage the situation accordingly. In other words, supporting multiple representations implies that both users and the system are aware that those representations are describing the same real world phenomena (Spaccapietra et al, 1999, 2000b). This can be achieved if existing data models and systems are extended with new concepts to provide links between the different representations, and associated constraints and operators. Spaccapietra et al (1999) noted some benefits of such a model, such as: better real-world modelling, improved consistency and automatic update propagation. In the absence of such existing systems, as is the case for most current systems, it is up to the application designer to cope with the problem of multiple representations whenever it arises, relying on primitive system functionalities to provide links between the different representations.

4.6.3. Multiple representations in Top10Vector database

In the Top10Vector dataset, some objects have multiple geometric representations. For example, road sections are represented primarily as area objects and then as centrelines (Figure 4.5). According to the data collection rules, when the width of a road section is more 2 metres it is represented as an area object; otherwise it is represented as a line object. For roads greater than 2 metres in width, their edges are collected in the field, and their centrelines are derived later. Therefore, the areas are the main representation and the centrelines are the derived or dependent representation. The main purpose of keeping road centrelines in the Top10Vector database in addition to the area representation is for performing network analysis.

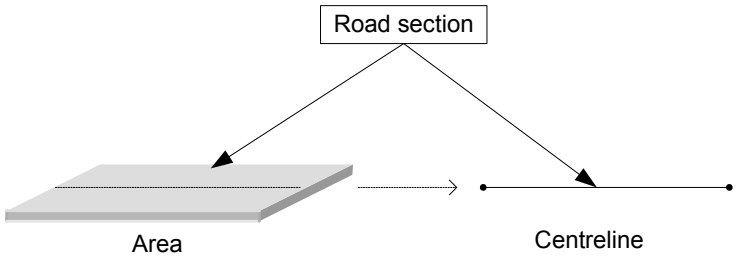


Figure 4.5 Multiple representations of Road sections

The requirements that arise from such multiple representations of objects include the following:

- How to maintain topologic consistency between the two representations?
- How to maintain the consistency of the attribute information?

Topologic consistency can be maintained by identifying all the possible spatial relationships between the two representations, and translating them into constraint rules in the database. Attribute consistency can be maintained by ensuring that whenever the attribute of the main representation (area) changes, corresponding attributes of the centrelines are automatically updated. This can be realised by providing a link between the two representations, and by developing the appropriate update propagation mechanisms. This will also reduce the updating cost. The following sections describe how we model these requirements.

4.6.4. Modelling the multiple representations in the Top10Vector

There are various approaches proposed for implementing multiple representations of objects; these will not be discussed here (see De Carvalho Paiva, 1998; Medeiros et al, 1996; Kilpelainen, 2000). The approach used here is to create separate classes for the primary and derived representations and to use object-oriented concepts to build links between them. In the Top10Vector dataset, the primary representation is road areas and the dependent representation is the road centrelines. Thus we create separate classes for the main representation (RoadArea class) and the dependent representation (RoadCentreline class).

One requirement of the Top10Vector database is that the objects of the primary representation control the lifecycle of objects in the dependent representation. This means that a RoadCentreline object cannot exist apart from its corresponding RoadArea object. To model this, the two classes will be linked by means of a composite association. A composite link allows the parent object to control the lifecycle of the derived object. The multiplicity of this composite association is 1-to-1, i.e., every instance of the RoadArea class is related to one instance of the RoadCentreline class. Objects in the RoadCentreline class will be linked physically to their controlling objects in the RoadArea class through the ObjectID. Through this link, every RoadCentreline object 'knows' the corresponding object in the RoadArea class, which controls its lifecycle. In relational terminology, the ObjectID will be the *primary key* in the RoadArea class and the *foreign key* in the RoadCentreline class. The additional operations defined for the two classes in Figure 4.6 will be discussed in section 4.6.6.

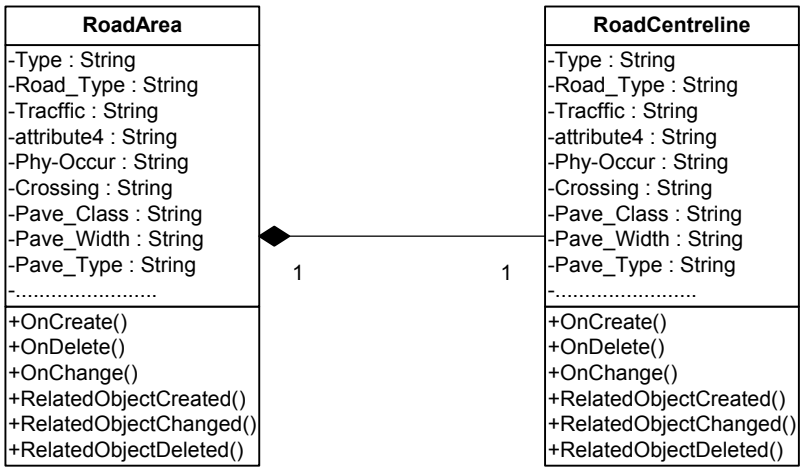


Figure 4.6 Composite relationship between RoadArea and RoadCentreline classes

4.6.5. Topologic consistency between the multiple representations

Topological relationships between instances of the RoadArea and the RoadCentreline should be preserved if consistency of the multiple representations is to be maintained. Topological consistency between the two representations can be maintained by defining constraints rules using the spatial relationships between corresponding objects of the two feature classes. Figures 4.7 and 4.8 show the possible spatial relationships between the RoadArea objects and RoadCentreline objects. It is important to remember that two RoadArea objects are allowed to overlap each other (in 2-dimensional space) if different height level attributes are specified to distinguish them (underpass and overpass). In such cases, their corresponding centrelines also cross each other as shown in Figure 4.7.

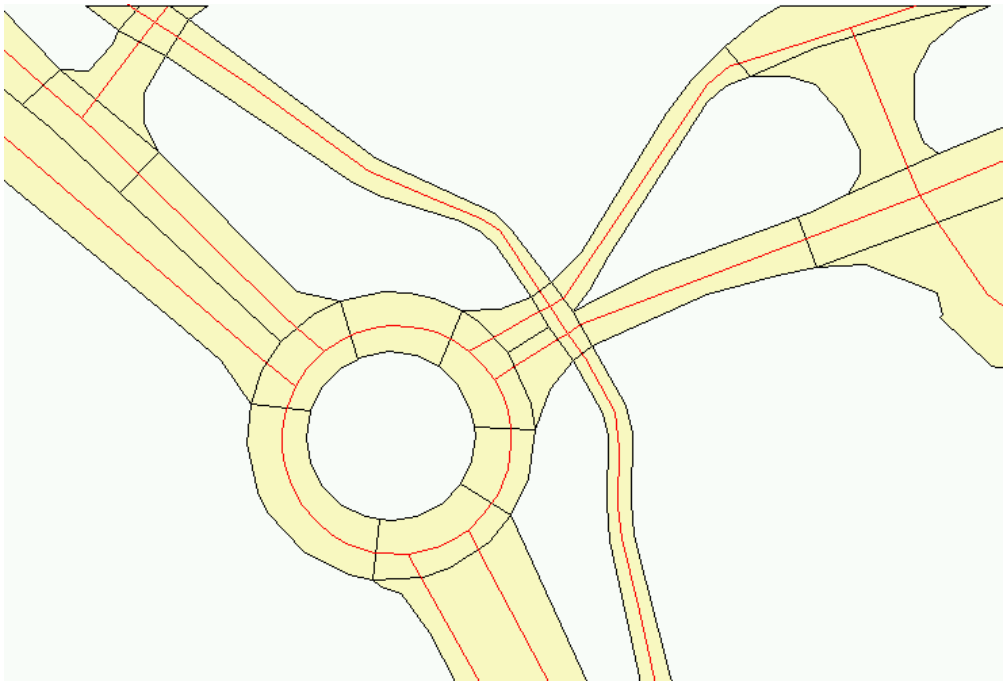
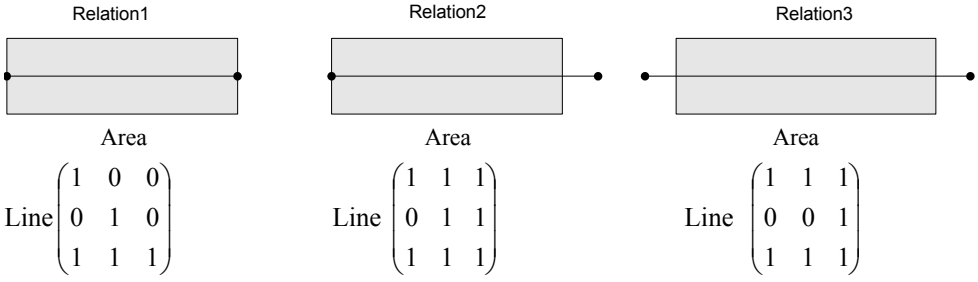


Figure 4.7 Possible topological relationships between road areas and their centrelines

From Figure 4.7, we can identify the possible spatial relationships between a road area object and its centreline. The spatial relationships between the RoadArea objects and the RoadCentreline objects depend on whether the centrelines are modelled as *simple features* or *network features* (see section 3.2.1). In this research, we model the centrelines as simple objects. Thus the spatial relationships should be one of the relationships shown in Figure 4.8. These relationships can be defined by testing for the intersection of the interior, boundary and exterior of area and line objects (see chapter 3).



$$\text{Relation1}: \begin{matrix} \text{Area} \\ \text{Line} \end{matrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \qquad \text{Relation2}: \begin{matrix} \text{Area} \\ \text{Line} \end{matrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \text{Relation3}: \begin{matrix} \text{Area} \\ \text{Line} \end{matrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Figure 4.8 Three valid spatial relationships between road areas and centrelines

Based on these spatial relationships, consistency constraints can be formulated. The following are some of the constraints that can be applied to preserve the topological consistency between road area objects and their centreline objects. Some of these constraints will be implemented in the implementation phase.

- If the ends of two road area objects (A and B) adjoin or meet each other at the same height level, their centrelines should also meet.
- If the end of a road area object (A) connects to the side/edge of another road area object (B) at the same height level, the centreline of A should be extended to meet the interior of the centreline of object B as shown in 4.7. This ensures the connectivity of the linear network.
- The centrelines of overlapping road areas can cross each other provided they have different height levels indicating *overpass* and *underpass*.
- At the same height level centreline objects must not cross each other without a point.
- Every road area object should be related to only one centreline object.

4.6.6. Automatic update propagation mechanism

Automatic update propagation is a mechanism whereby whenever an object in the system changes, all other objects that are related to, or dependent on, the modified object are informed and updated automatically. Update propagation is based on the common principle that derived properties should not be updated explicitly, but only the fundamental properties (Egenhofer & Frank, 1992). This is a powerful feature, as it allows multiple representations to be conveniently and consistently managed. Automatic update propagation is realized through the communication between objects in a system. Whenever an object changes, it sends messages to all other objects that are related to it, leaving them to take their own appropriate actions. The related objects may respond by updating their own attributes or by executing a constraint against the changed object, etc. In our case, the main concern here is how to propagate updates from the (primary) RoadArea class to the (dependent) RoadCentreline class so that every time a road area attribute changes (for example), the corresponding centreline object is automatically updated by the system.

To realise automatic update propagation between the RoadArea and the RoadCentreline classes, methods are defined in both classes that will be used to notify the class that a particular event has occurred in the related class. There are three basic events that can happen to objects in a particular feature class, namely: creation of new objects, modification of existing objects and deletion of existing objects. For each of these events, a method will be defined that will inform the class that the particular event has occurred in the related class. These methods will be defined in both the RoadArea and RoadCentreline classes. The methods are:

- RelatedObjectCreated: Notifies this class that a related object has been created
- RelatedObjectChanged: Notifies this class that a related object has changed
- RelatedObjectDeleted: Notifies this class that a related object has been deleted

The RoadArea and RoadCentreline classes will implement these three methods in addition to the usual methods for all feature classes (i.e. OnCreate, OnChange, OnDelete). These methods will be used to notify both classes when the particular event occurs. For example, when a road area attribute is updated in the RoadArea class, the method RelatedObjectChanged in the RoadCentreline class will be called to notify this class that a related road area object has changed. This will enable the RoadCentreline class to take the appropriate action, such as automatically updating the corresponding attribute of

the related object. The following sections summarize the behaviour for automatic update propagation between the two classes.

4.6.6.1. Updating the RoadArea class

Creation of new objects

When a new object is created in the RoadArea class, the method RelatedObjectCreated will be called to notify the RoadCentreline class. The ideal setting will be when new objects are created in the RoadArea class; their corresponding centreline objects are automatically computed through cartographic generalization methods and stored in the RoadCentreline class. However, because this derivation cannot be fully automated, and given the fact that this research is *not* concerned with operations of how to derive one representation from the other, the system will be modelled to flag the user to insert the corresponding centreline object when new RoadArea objects are created.

Modification to existing objects

An object can be modified in several ways:

- *Attribute updates*: All attribute updates must be made in the RoadArea class, which is the primary representation. These updates will then be automatically propagated to the RoadCentreline class by update propagation mechanisms.
- *Object resized*: If the geometric shape of the area object changes, the system will notify the user to update the corresponding centreline object related to the modified area object, if it detects no valid spatial relationship between the two.
- *Object moved*: If a RoadArea object is moved to a new position, the related RoadCentreline object will be moved together with the area object to maintain consistency.
- *Objects merged or split*: This will result in the creation of new object(s), and will therefore be treated as outlined above.

Deletion of objects

Since objects in the RoadArea class control the lifetime of objects in the RoadCentreline class, a road centreline cannot exist without its corresponding area object. Thus deleting a RoadArea object will cause the system to automatically delete the related object in the RoadCentreline class.

4.6.6.2. Updating the RoadCentreline class

Creation of new centreline objects

If a new centreline object is digitised, the system will test for the valid spatial relationships between the new centreline and the objects of the RoadArea class. If a RoadArea object is found which has a valid relationship with the new centreline, and if that object is not related to any centreline; then the system will confirm from the user to propagate the common attributes of the existing area object to the new centreline object. If user's answer is yes, propagate the update. If no valid relationship is found, the system will inform the user.

Geometric modifications

If the geometry of existing centreline objects changes (e.g. reshaped or moved), check the spatial relationship between the centreline and its controlling object in the RoadArea class. If the relationship is no longer one of the valid relationships, inform the user and break the original relationship between

the two objects. Search for any valid relationships with other objects, if found confirm from the user, create a relationship between the centreline and the new object and propagate the attributes of that area object to the centreline. If no valid relationship is found, inform the user.

Updating attributes of centreline objects
All attribute updates should be made in the RoadArea class and they will be propagated automatically to the RoadCentreline class. This will preserve the consistency of the attribute information. However, if a user attempts to update the attribute of a road centreline directly, the system will give a warning that those updates would not be propagated to the RoadArea class.

Deletion of existing centreline object
If a centreline object is deleted, check if it is related to an existing area object. If it is, do not allow the deletion of the centreline object without first deleting its controlling area object. If the centreline is not related to any area object, allow delete.

4.7. Temporal behaviour

4.7.1. Temporal behaviour of spatial objects

Spatial objects have both spatial and temporal dimensions. Time is linked with real world objects; their geometry, attributes and relationship change over time. Therefore it is necessary to model the temporal dimension of spatial objects in order to better understand spatial patterns, events and processes that take place in our dynamic environment. Conventional GIS systems represent the real world in only a snapshot that is inadequate for analysing changes and patterns of change over time. Non-temporal GIS assumes the world only exists in the present. No sense of change or dynamics is maintained while updating the database. Every update creates a new map and no attempt is made to establish any link between the snapshots. These systems are therefore not suitable for many applications where facts and data need to be interpreted in the context of time, such as urban applications, environmental monitoring, agricultural applications, forest management, etc (Raza, 2001). The need to model the temporal dimension of spatial objects has given rise to several research activities in this area (Langran, 1992). This research is not intended to develop a fully functional temporal GIS. Our focus is on how to maintain the history of objects in the topographic database. A temporal model has been developed for the Top10Vector database using object-oriented concepts.

4.7.2. What is a change?

The concept of change can be described as follows: ‘if an object O changed, then it cannot be considered as the object O’ (Nunes, 1995; cited in Raza, 2001). This implies that if an object changed, then it is no longer in the same state. Modelling time therefore means modelling change. In spatial databases, variation in temporal behaviour results in changes in one or more of the following three characteristics (Abraham et al, 1999):

- Geometry: for example, relocation of an object.
- Topology: for example, alteration of the spatial relationships among objects.
- Attribute: for example, status updating of an object.

Changes in these three components are not mutually exclusive, i.e., they can occur contemporaneously with each other. For example, merging existing objects to create a new object involves changes in the geometry, attributes and topology of the original objects. There are eight possible changes a spatial object may undergo (Figure 4.9).

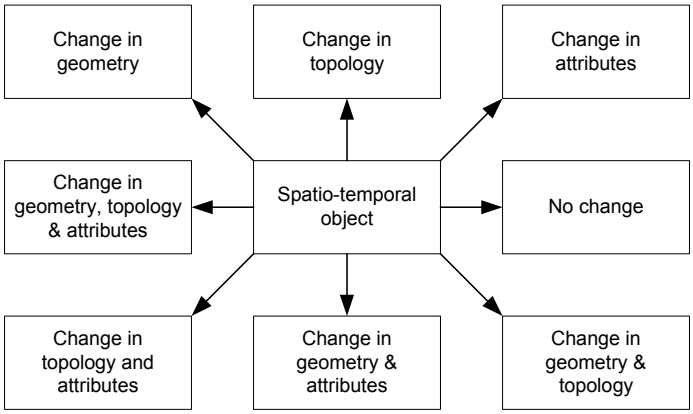


Figure 4.9 Eight possible changes in spatio-temporal object (Abraham et al, 1999)

4.7.3. Objects and versions

Changes in objects' state can be categorized on the basis of their properties as *essential* and *non-essential* changes (Raza, 1996, 2001). An essential change implies the creation of a new object with a new object identifier while non-essential change only means a new *version* of the same object with same identifier (Figure 4.10). The specification of what is essential and non-essential change fundamentally depends on the purpose of the application. Some changes might be relevant to a particular application while they might be irrelevant in other applications domains. For example, in land use applications, land use is the main essence of the application and therefore a change in land use attribute is very essential whereas a change in a boundary may be non-essential.

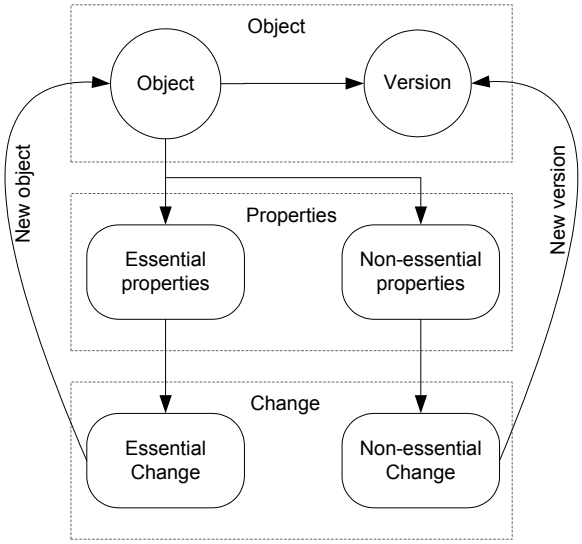


Figure 4.10 Essential and non-essential change model (Raza, 2001)

4.7.4. Fundamental time concepts

Before describing our temporal model, we will, in this section, give brief overview of some fundamental definitions and concepts in the domain of time modelling.

4.7.4.1. Measurement of time

Time can be measured as either *discrete* (e.g. years, months, weeks) or *continuous* variable (Worboys, 1995). Discrete time considers time as consisting of a sequence of event-punctuated intervals (Raza, 2001). Discrete time is useful in applications in which changes are discrete in nature, e.g. changes in cadastral boundaries, administrative boundaries, etc. Continuous time structure considers that there is infinite number of points between any two points. Continuous time is used to measure changes which are occurring continuously, for example, global change management. The smallest, non-decomposable unit of time is known as a *chronon*. This is the time that is supported by a temporal database. *Granularity* refers to the basic unit (precision) of time in a temporal database, and varies from one system to another. The lifespan of an object is measured by a finite number of chronons in the database. An *event* is the transition from one state of an object to another state. An event is a *point* on the timeline, while a state is an interval.

4.7.4.2. Models of time

There are three mathematical models of time, namely: *linear time*, *branching time* and *cyclic time*. Linear time advances from the past to the future in a totally ordered fashion. Branching time advances from past to present where it then splits into several time lines or it may have many pasts and a single future. Cyclic time repeats itself after some time interval.

4.7.4.3. Types of time

Time has been treated as both one-dimensional and multidimensional component. At least three types of time are relevant in a GIS; these are *event time*, *observation t*ime and *database time* (Guptill, 1995). Event time is the time an event actually occurred in the real world being modelled. This is also called *world time* or *valid time* (Langran, 1992). Observation time (also called *evidence time*) is the time the event was observed. The *database time (*or *transaction time*) is the time an event is recorded into the database.

4.7.4.4. Temporal relationships

Temporal relationships are defined in analogy to spatial relationships between events (nodes) and states (arcs). Temporal relationships can also be defined using metric, topological, and order concepts. For example, ‘two hours’ is a metric relation, ‘one hour later’ is a topological temporal relation and ‘four weeks in a month’ is an order relation (Worboys, 1995; cited in Raza, 2001). In this research the order relationships have been used for formulating temporal queries. Examples of ordered temporal relationships are given below. For any two events A and B, we can write the following (Frank, 1994):

- $A < B \rightarrow$ event A is before event B
- $A > B \rightarrow$ event A is after event B
- $A = B \rightarrow$ events A and B are contemporaneous

4.7.4.5. Spatio-temporal models

There are many models that have been proposed for incorporating temporal information into spatial databases. Some of these models treat time as an attribute of spatial objects while others treat time as a separate dimension. Some of the main models are only briefly introduced here. For detailed discussion, the reader is referred to (Langran, 1995; Hazelton et al, 1990; Raza, 2001).

- The *snapshot approach* stores all versions of the map as series snapshots or time-slices. This approach represents a state but does not represent the events that cause changes. Instead of storing the change, its stores the states of the whole map or layer. This quickly leads to large data redundancy, and have serious implications for data management and temporal analysis.

- The *update model (*or *base state with amendment)* stores only the changed objects but not a completely new snapshot of the whole data. Changed objects are represented as versions. This reduces data redundancy and allows object history to be traceable.

- *The Space-time composite model* stores all past and present features by overlapping all time-stamped layers to produce a space-time composite layer (Langran & Chrisman, 1988). This model is not very different from the update model except that past and current features are stored in the same layer.

- The *3D/4D temporal GIS* approach: In this approach, attempts have been made to model time as a dimension of spatial objects. This will lead to a true temporal GIS system, but this cannot be implemented without a major software engineering effort. Many such models have been proposed by researchers, most of them based on the object-oriented paradigm (Raza, 2001; Langran, 1995; Hazelton et al, 1990).

4.7.5. Modelling the temporal behaviour of Top10Vector objects

In this research, time is treated as an attribute of spatial objects. Thus time stamps on objects and attributes would be adequate to represent the historical states of objects in the database. The model developed in his research is similar to the update approach. The modelling process has been described in the sections that follow.

4.7.5.1. Adding time attributes to the FeatureClass model

In order to model the temporal behaviour of the objects, the FeatureClass model has to be extended with additional time attributes. To each feature class, we assign the following time attributes:

ObjectDate:	Records the time the object was created in the database.
VersionDate:	Records the time when the current version of an object was created.
VersionNumber:	Records the version number of the most current version of the object in the feature class.

Our model thus takes into account *only* the database time. We can, of course, include the event time and the observation time. But these are not considered here for the time being. We also allows for multiple versions of the same objects to exist the database. The current objects will be stored in the feature

class while all historical objects will be removed to the temporal class associated with the feature class. The temporal class model is described in the next section.

4.7.5.2. The TemporalClass model

The TemporalClass model deals with the modelling of time. A temporal class will be created for each feature class. The temporal class will store and manage all deleted and modified objects for later retrieval and analysis. The main issue here is how much information should be stored per object to allow temporal queries and analysis. In this research, we have chosen to store objects that are modified or deleted with their changed attributes. Thus, when an object is modified, we store the whole geometry of the object and *only* the attributes that changed. This will be enough for us to reconstruct previous states of the object. Instead of storing the whole geometry of the object, we could store only the geometric changes. However, this will make it more complicated to reconstruct previous situations at any given point in time. Figure 4.11 shows a part of the TemporalClass model.

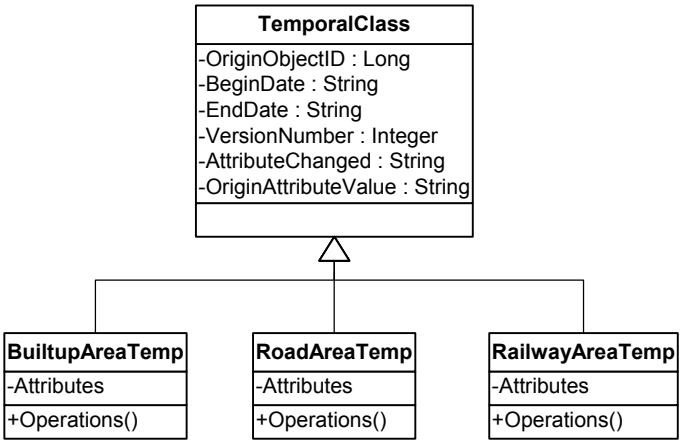


Figure 4.11 The TemporalClass model

Attributes of the TemporalClass

OriginObjectID:	The original ObjectID of the deleted or modified object in its origin feature class. This will provide the link between the feature class and its temporal class.
BeginDate:	The BeginDate stores the creation date for the object in the TemporalClass.
EndDate:	The EndDate stores the date on which the object or version ceased to exist in the feature class.
VersionNumber:	This stores the version number of all modified objects. For deleted objects, the VersionNumber attribute value will be zero, indicating that the object has no current version in the corresponding current feature class.
Event:	This will store the type of event that happened to the object (attribute changed, geometry changed, object moved, deleted, etc).
AttributeChanged:	If an attribute of an object changed, this will record the particular attribute that changed. For deleted objects, this will be NULL.
OriginAttributeValue:	This stores the original attribute value in case the attribute changed. This value together with the name of the attribute that changed will be used to reconstruct previous states of objects.

4.7.6. Objects life cycle rules

Real world objects have lifecycles. An object may be created, go through several mutations, and may be eventually destroyed. For example, a building may be constructed; it may be extended, and finally destroyed. Rules should be established to regulate the lifecycles of objects in the system. These rules should be decided in consultation with the application domain. Since the new Top10vector data model is still in the development stage, these rules are not completely available or not in a form to be directly implemented in any operational system. The rules that are described in this section have been formulated according to the model and based on interviews conducted in consultation with the Topographic Service of the Netherlands.

4.7.6.1. Creation of new objects

According to our model, when new objects are created, they will be given three time attributes, namely, ObjectDate, VersionDate and VersionNumber (1). These attributes will be assigned automatically by the system and would be used to monitor the temporal behaviour of the object.

4.7.6.2. Deletion of objects

When a real world object represented in the database no longer exists, the object is deleted from the database. When a user deletes an object from a feature class, the object is *not* really destroyed, but rather it is removed from the feature class and stored in the temporal class associated with the feature class for later retrieval. Deleted objects will be stored with corresponding time stamps.

4.7.6.3. Modification of existing objects

Objects may go through several changes. These changes either result in the creation of new objects or versions of the same object. New objects are assigned new ObjectID. Versions of the same object maintain the same ObjectID and but are given different VersionDate and VersionNumber attributes. In section 4.7.3, we identified changes to objects' state as essential and non-essential. Essential changes result in new objects while non-essential changes result in new versions. The following rules will guide the system in deciding whether or not change in an object should result in a new object or a new version:

- If the geometric size or length of an object changed, and the change is more than the minimum size of objects stored in the feature class, then the system will consider it as essential change, and will prompt the user for confirmation. If the user confirms that the change should result in a new object, the original object will be deleted, time-stamped and removed to the TemporalClass. A new object will be created with new ObjectID, ObjectDate, VersionDate and VersionNumber. If the user specifies that it is still the same object, the original ObjectID will be retained, the new VersionDate will be recorded and the VersionNumber will be incremented.
- All attribute updates made to objects are, at the moment, considered as non-essential changes; i.e., they result in new versions of the same object and will be treated by the system as outlined above. The basic assumption here is that as long as the object occupies the same space and has the same geometry, it is considered to be the same object.
- When an object is moved to a new position, this will result in new object and will be treated by the system as outlined above.

- When two or more objects are merged to become one object, a new object will be created with a new ObjectID, and the original objects will be marked as deleted. The attributes of the new object will be the combined attributes of the original objects.
- When an object is split into two or more parts, the result is new objects with new ObjectIDs, the original object ceases to exist.

It is important to state that these rules should not be fixed and fast. The computer has no knowledge and cannot make rational decision concerning real world situations. So the system will be designed to assist (not to overrule) the user in deciding whether a change in an object should result in a new object or a new version of the same object. Figure 4.12 summarizes these behaviours.

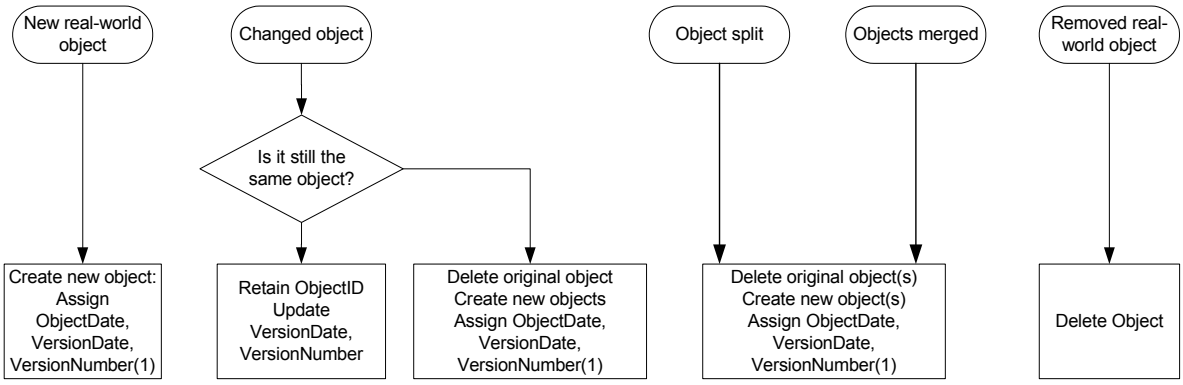


Figure 4.12 Objects life cycle rules

4.7.7. Mechanism for retrieving historical objects

The link between each feature class and its associated temporal class is provided by the ObjectID in the feature class and the OriginObjectID in the temporal class. By storing the original ObjectID (i.e., OriginObjectID) for each object in the temporal class, it is possible to link the current object to all its historical versions in the temporal class. How can we retrieve successive versions of the same objects? This is provided by the VersionNumber attribute. When a new object is created, it is given a Version-Number (1) in the feature class. Now when a new version of the object is created, the VersionNumber in the feature class will be incremented to 2, and the original version (1) of the object will be transferred to the temporal class, and so on. Thus successive versions of the object are identified by the VersionNumber attributes: version 2 is the parent of 1; 3 is the parent of 2, etc. The BeginDate and the EndDate define the lifetime of objects in the database.

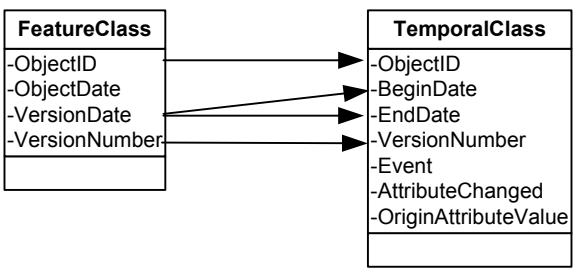


Figure 4.13 Links between FeatureClass and TemporalClass

5. Implementation

5.1. Introduction

This chapter covers the implementation phase. The conceptual model we designed in the previous chapter is implemented by extending ESRI geodatabase model. First the conceptual model is translated into the logical model supported by ArcInfo geodatabase. ESRI also uses the UML for the logical model. Once the logical model has been completed, we can automatically generate initialization code and the geodatabase schema using wizards provided by ESRI. The behaviours are implemented as methods in object-oriented language MS Visual C++. The software components developed are based on the COM technology. These components are developed as extensions to ESRI geodatabase object library, and can themselves be extended to handle more requirements. The chapter concludes with testing and display of the behaviours in ArcMap. It should be noted that not all the behaviours modelled in the design phase could be implemented because of the limited time.

5.2. Methodology for implementation in ArcInfo 8

ArcInfo makes use of Microsoft Visio, which supports the UML, to automate the process of creating the logical model, generating code and creating the geodatabase schema. ESRI provides wizards for generating initialization code and creating the geodatabase schema. The methodology for using Visio and ArcInfo tools to design and create the geodatabase involves the following steps (ESRI, 1999):

- Designing the logical object model in UML.
- Exporting the UML model to the Microsoft Repository.
- Generating start-up code and implementing behaviours of objects by writing code in Visual C++.
- Creating the geodatabase schema from the repository model.

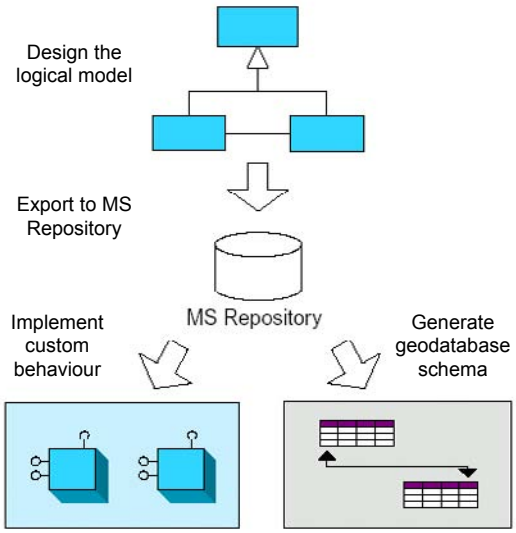


Figure 5.1 Methodology for implementation in ArcInfo

5.3. Designing the logical model in Microsoft Visio

5.3.1. Basic terminology

The following terminologies are used in ArcInfo:

- Object class: A database table with which you can associate behaviour. Rows in a table are object instances that can have behaviour.
- Feature class: A collection of geographic objects or features of the same type.
- Feature attributes: Properties stored as fields in a feature class table.
- Feature dataset: A collection of feature classes with the same spatial reference.

5.3.2. The ArcInfo UML Model

ESRI provides the basic geodatabase model in UML that can be loaded into Visio and extended to create custom models. The ArcInfo UML model contains the geodatabase model and other relevant parts of ArcObjects Library needed for the creation of custom models. The UML model has four packages: Logical View, ESRI Classes, ESRI Interfaces, and Workspace (see Figure 5.2). These UML packages act as directories where different parts of the logical model are maintained.

- Logical View: This package is the root level and contains the other three packages.
- ESRI Classes: Contains existing ESRI COM classes that can be reused by either aggregation or containment.
- ESRI Interfaces: Contains interfaces of the existing ESRI classes.
- Workspace: This holds user-defined models created as extensions to the geodatabase model.

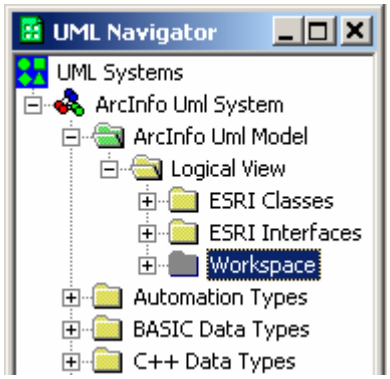


Figure 5.2 ArcInfo UML model

5.3.3. Designing the FeatureClass model

Custom object models should be created as extensions to ESRI geodatabase model. For ArcInfo to read a custom model correctly, certain rules should be followed in creating the model.

- First, all custom models should be created under the 'Workspace' package.
- Classes that will store non-spatial data should be created as specializations of ESRI *object* class.
- Custom feature classes that will store spatial data are created as specializations of ESRI *feature* class. This allows the custom feature classes to inherit the behaviour of ESRI feature class, which in turn inherits from ESRI *object* class, etc.
- Feature datasets are modelled in the UML as packages stereotyped as (ESRI) feature dataset.

Figure 5.3 shows a part of the logical model for the feature classes.

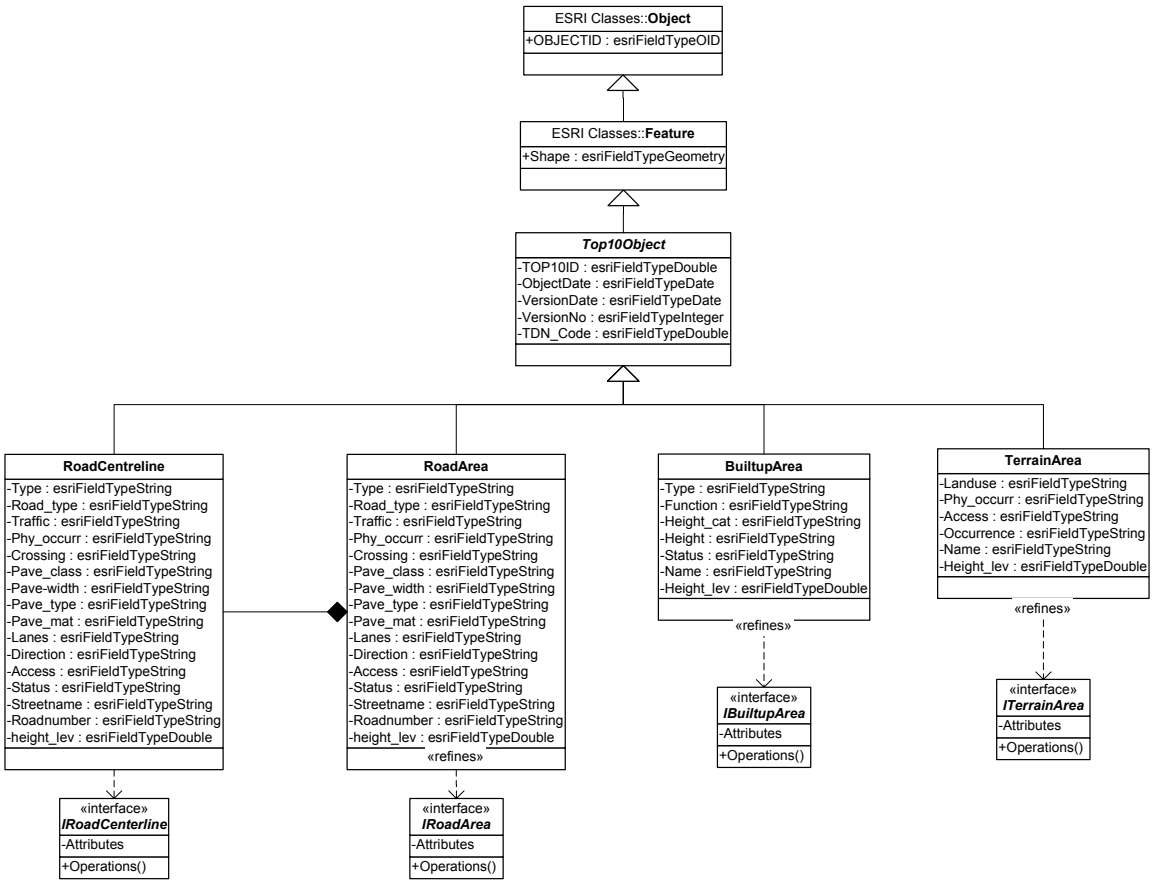


Figure 5.3 The FeatureClass logical model

5.3.4. Attributes

Once each class is created, attribute names and their domains can be specified for the class. Attributes types must be specified as one of ESRI field types or use custom domains. To use custom domains, they must be defined first before they become available for use. Most of the feature classes have their own custom domains, so they are defined for them.

5.3.5. Custom domains

ESRI provides two types of templates for defining custom domains, namely, *range domain* and *coded value domain*. Range domain allows one to define the allowable range of values for an attribute, for example, 20-40. Coded value domain allows one to restrict attribute values to certain types. For example, a landuse attribute may only be ‘Residential’, ‘Industrial’ or ‘Commercial’. We can use coded value domain to accomplish this. Using these templates, custom domains have been created for the attributes of each class where required. Examples of these domains are shown in Figure 5.4. Once the domains are defined, they can be applied to their corresponding attribute.

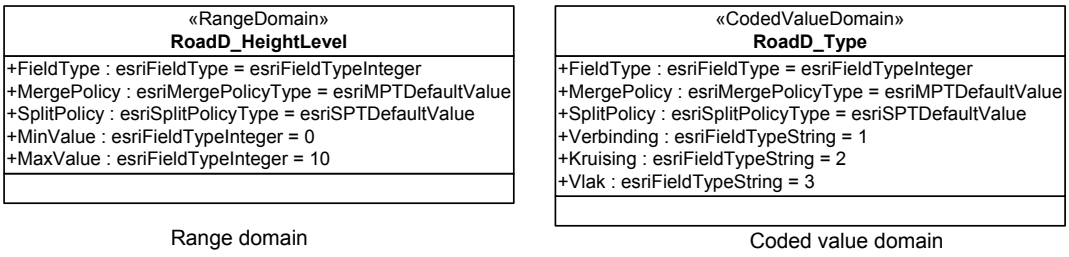


Figure 5.4 Custom range and coded value domains

5.3.6. Tag values

Tag values are used to set additional properties of UML elements and will be used by the ESRI schema generator. For example, we can use tag values to specify that a particular feature class will store polygon features. Table 5.1 lists the tag values used in the model:

Table 5.1 Tag values used in the model

Tag value	Notes
GeometryType	Used to set the geometry type for a feature class
OriginPrimaryKey	The name of the primary key of origin class in an association
OriginForeignKey	The name of the foreign key of origin class in an association
DestinationPrimaryKey	The name of the primary key of destination class in an association
DestinationForeignKey	The name of the foreign key of destination class in an association
Notification	Used to specify direction of messages in an association

5.3.7. Interfaces

At least one interface has been created for each class although not all of them have been programmed. The interface of a class will hold the methods it exposes to the public users of that class. By convention, all interface names are written with 'I' preceding the feature class name, for example, IRoadArea, IBuiltupArea, IRailwayArea, etc. In our model, all the methods have been defined in the class extensions associated with the feature classes. This will be explained in the following sections.

5.3.8. Object class extensions

In the geodatabase, a class extension is a COM class that implements behaviour that pertains to the whole feature class, in contrast to behaviour that apply to single objects. For example, the property inspector of a feature class in ArcMap is implemented by the class extension. The same property inspector is used for all features in the feature class. Now, because the methods of the feature classes (OnCreate, OnChange, OnDelete, etc) will be implemented for all the objects in the feature classes, they will be defined in the class extensions, rather than in the class themselves. ArcInfo tools will find class extensions by naming convention. A valid class extension name is made up of the class name followed by the string 'ClassExtension' (e.g., BuiltupAreaClassExtension, RoadAreaClassExtension).

5.3.9. Reusing existing ESRI interfaces

Class extensions also implement interfaces, which contain the methods. We have at least three methods to implement for all feature classes, i.e., OnCreate, OnChange, and OnDelete. Instead of defining our own interfaces, we can reuse existing ESRI interfaces that have these methods already declared (without implementation). So we use ESRI interface called IObjectClassEvents, which contains definition for the three methods as shown in Figure 5.5. In addition, other existing interfaces have some methods we can reuse. These interfaces are: IRelatedObjectEvents, IRelatedObjectClassEvents, and IRelatedObjectClassEvents2. These are optional interfaces implemented by ESRI COM class called ObjectClassExtension to help users write custom behaviours. And since our class extensions are defined as specializations of this class (through FeatureClassExtension), we can easily *reuse* these interfaces and provide implementation for the methods associated with them. Of course we can define our own interfaces, but reusing existing ones saves the developer lots of effort in writing interfaces from scratch. The developer can then concentrate on the actual implementation of the methods. Figure 5.5 shows a part of the class extension model associated with the FeatureClass model.

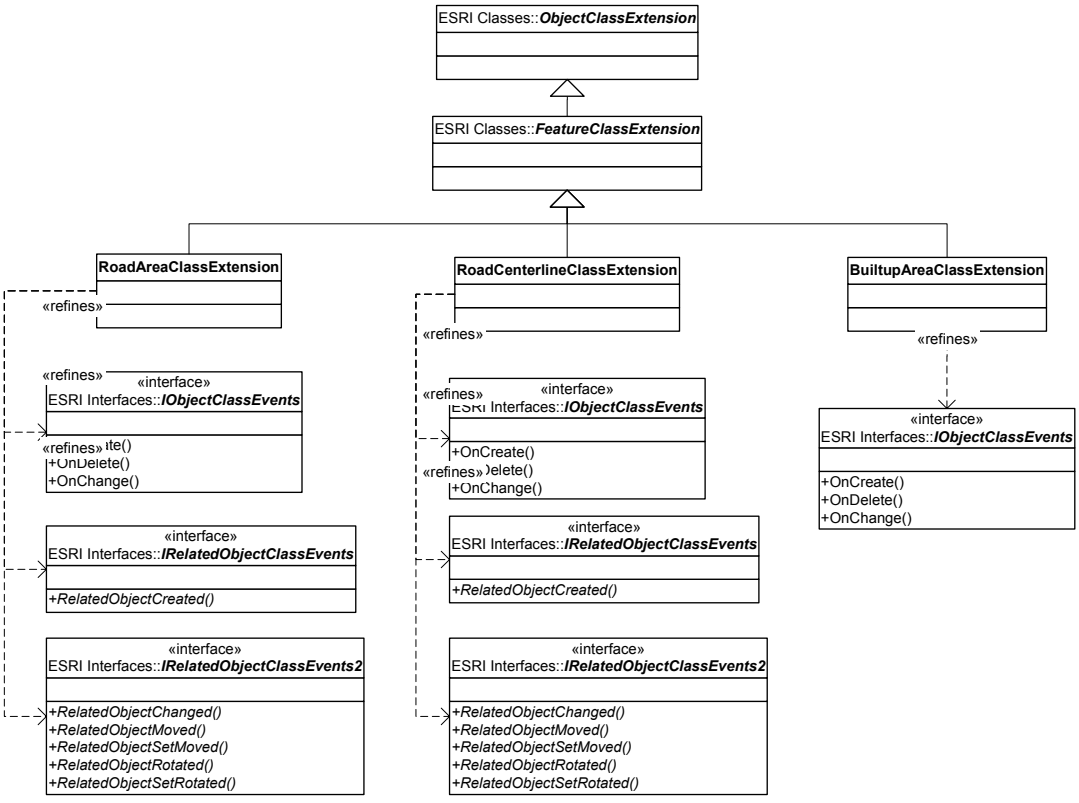


Figure 5.5 The class extension model

5.3.10. Logical models for the TemporalClass and the ConstraintClass

The TemporalClass and the ConstraintClass logical models have also been designed based on the same principles for designing the FeatureClass model. All the temporal feature classes are stored in a feature dataset called 'TemporalClass', so the model is put in a UML package stereotyped as ESRI *feature dataset*. All the constraints classes are tables, so they are designed as specializations of ESRI *object*.

5.3.11. Exporting the model to the Microsoft Repository

Once the model has been completed, it has to be exported to the Microsoft Repository from where ArcInfo will read it and use it to generate the database schema. The repository is a Microsoft Access database that stores descriptive information about software components or models and their relationships (MSDN Library, 2001). It provides facilities for cataloging and locating relevant designs, code, and services (interfaces). Before exporting to the Microsoft Repository, it is important that the entire model is checked for semantic errors by using Visio UML semantic checker to verify the model. This will prevent lots of errors during the automatic schema generation.

5.4. Generating initialization code

Once the model has been exported to the Microsoft Repository, we can use ESRI *code generation wizard* to generate start-up and initialization code from the repository model. This saves the developer from having to write some initialization code. The code generation wizard is an add-in to Microsoft Visual C++. The code generated by the wizard is based on the ATL (Active Template Library), a framework designed by Microsoft to facilitate COM programming in C++.
The process of generating start-up code includes the following steps:

- Add the ESRI code generation wizard to Microsoft Visual C++ developing environment.
- Start the code generation wizard and connect to the repository database.
- Select the model and the feature classes to generate code for.
- Set properties for each feature class.
- Define the output Visual C++ project.

5.4.1. Containment and aggregation

In the code generation process, we must specify whether we want to *aggregate* or *contain* the existing ESRI interfaces we are reusing. Aggregation implies that our custom classes will expose those interfaces directly and containment implies forwarding messages to the ESRI classes whenever they occur (see chapter 2). In our case, the only interface we will contain is the IRelatedObjectClassEvents. All the other interfaces are *optional* ESRI interfaces so they will be implemented locally in our custom COM classes.

5.4.2. The output project

The code generator will create a Microsoft Visual C++ workspace with the following files:

- Registration script (*.rgs*), header (*.h*), and implementation or source (*.cpp*) files for each feature class and class extension.
- IDL (Interface Definition Language) file with the definition of the COM classes, interfaces, and the type library.
- Other standard C++/ATL files.

The IDL file contains the definition of all the COM classes and interfaces. The wizard associates a CLSID to each COM class and an IID to each interface (see chapter 2). These are GUIDs and will be registered in the system registry as part of the dynamic link library (DLL) upon compilation of the final code. It is these GUIDs that ArcInfo uses to reference the COM classes that implement the behaviour of the feature classes in the geodatabase (see section 5.6.1).

5.5. Writing custom behaviours

This section describes the process of writing code to implement the custom behaviours. The three categories of behaviours (consistency constraints, multiple representations, and temporal behaviours) are implemented here by writing code for them. The full program code can be found in a separate document associated with this report (see appendix B). This section only summarizes the functionality of the methods and algorithms developed to implement the behaviours.

5.5.1. Private methods

A number of private methods have been developed for the internal implementation of the public methods exposed in the interfaces of the classes. These methods are included in the classes to perform some specific routine tasks. The following are examples of the methods:

- OpenFeatureClass: Used to get a pointer to a feature class
- StoreModifiedObjects: Used to store modified objects
- CheckSpatialRelation: Used to check the spatial relationships between two objects
- SearchConstraintsClass: Used to search for constraints applicable to objects of a particular class
- CheckBinaryConstraints: Used to check for binary constraint violations
- CheckMinimumSizeConstraints: Used to check for minimum size constraint violations
- GetSystemDate: Used to retrieve and format the system date for use in the TemporalClass

5.5.2. Public methods

The following summarize the behaviour of the three methods: OnCreate, OnChange and OnDelete (see the flowchart in Figure 5.6).

OnCreate method

The OnCreate method watches for the creation of new objects in a feature class and takes the specified actions. Any code written in the OnCreate method will be executed when new objects are created. When new objects are created, the following summarize the actions that need to be taken:

- Check for constraints violations (binary and minimum size constraints)
- Assign the ObjectDate
- Assign the VersionDate
- Assign VersionNumber (1)

OnChange method

This method watches for any modification made to existing objects and take the appropriate actions. The following are a summary of the actions to be taken:

- If geometry changed, check for constraint violations.
- Is the object the same object? If same object retain the ObjectID, increment the VersionNumber and update the VersionDate in the feature class, and store the old version in the temporal class (OriginObjectID, Top10ID, Event, BeginDate, EndDate, VersionNumber, AttributeChanged, OriginAttributeValue).
- If new object, delete original object, create new object with new ObjectID, ObjectDate, VersionDate and VersionNumber.

OnDelete method

- Store deleted objects in the corresponding temporal class: OriginObjectID, Top10ID, Event, BeginDate, EndDate, VersionNumber, Attributes.

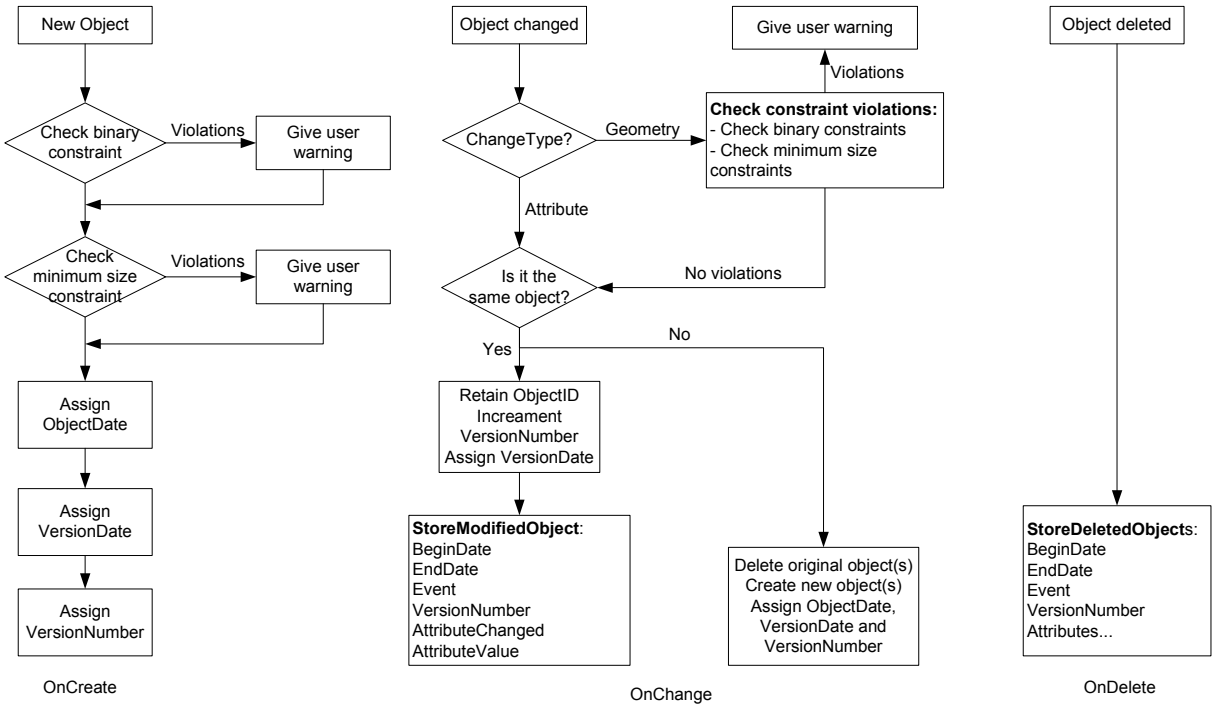


Figure 5.6 OnCreate, OnChange and OnDelete methods

The public methods for the implementation of the multiple representations include: RelatedObjectCreated, RelatedObjectChanged and RelatedObjectDeleted. These methods are defined for the RoadArea and the RoadCentreline feature classes. Their main behaviours were outlined in the design phase and will not be repeated here. Some of these have been implemented; others could not be implemented for time reasons. In section 5.7.2, we display some of the implementation of these methods.

5.5.3. Compiling the output program

Once the custom behaviours have been completed, the program will be compiled as a dynamic link library (DLL) with COM classes (in-process servers, see chapter 2). Custom features in the model for which code were written will become COM classes in the DLL, along with COM classes for the associated class extensions. The DLL will be registered in the system registry from where ArcInfo will locate it and associate it with the feature classes during the geodatabase schema generation process.

5.6. Generating the geodatabase schema

The schema generation wizard of the CASE tools will be used to generate an empty geodatabase schema from the repository UML model, after which data can be loaded into that schema. The model can also be applied to existing geodatabase, which has the same structure as the elements in the model. We will create an empty geodatabase schema and load data into the schema. Table 5.2 shows how ArcInfo maps the elements of the object model into relational database elements.

Table 5.2 Object-relational mapping

Object model elements	Relational database elements
Object	Row
Attribute	Field, Column
Class	Table

In the schema generation process, the wizard will search the system registry for any registered COM classes and class extensions and attach them to the geodatabase feature classes. Therefore it is important that code for custom behaviours are written and compiled before the schema generation process.

5.6.1. How ESRI links COM objects to geodatabase feature classes

The generated geodatabase is in Microsoft Access format, so we can open it with Access. The wizard has generated a table for the feature classes, constraint classes and the temporal classes, together with other tables used by ArcInfo to manage the geodatabase and to reference COM classes that implement the behaviour of objects in the database. Of a particular importance to us is the table named: *GDB_ObjectClasses*. This table is used to store the references to the custom COM classes registered in the system registry that implements the behaviour of the feature classes in the geodatabase. The GUIDs of the registered COM classes and class extensions are stored in this table as shown in Figure 5.7. When the geodatabase is loaded into memory, the DLL associated with the geodatabase is also loaded and ArcInfo uses the GUIDs for the COM classes and extensions to invoke the code associated with each feature class.

GDB_ObjectClasses : Table

ID	Name	CLSID	EXTCLSID
14	AdministrativeArea	{3086DFA8-8AFD-44F9-AA5D-B20972A703E3}	{D814FCA3-B1FD-49DE-88E5-2E9B8BFCECD4}
15	BuiltupArea	{4C093A0F-0B46-4DD8-AC1B-7A8D3FA2DF64}	{E4DF00B6-3A93-4721-AA78-3F653191D943}
18	FunctionalArea	{7203EFC7-9948-4EEE-9A60-7EFF21474200}	{B6303649-1E5A-4043-A818-66341FF75017}
19	GeographicArea	{51EDCE76-B6CC-4202-BCB2-022B79139C6E}	{54320D78-7B09-425F-A3EC-B891CAC3223A}
22	TerrainArea	{D4C647DC-AB3A-4EAB-90F7-85F486440254}	{9F4FE1CC-0189-4dc6-99A3-0D829FF0F878}
23	WaterArea	{BD6E9352-0305-41F1-83A9-EC75B149FF1A}	{2ED6532F-F79D-4246-9EE3-63D4C45DB884}
24	WaterLine	{7BE7C0A6-79D1-4516-AEE9-24DE88B185D8}	{8AA4FEC4-0B38-4332-9EF5-F7801203DBAD}
25	RoadCentreline	{81BB0CAC-1835-4023-B61E-43224E8E381B}	{CFDDC60A-39CF-40E3-9A0B-49F3C2370BE2}
28	RoadArea	{452B1007-834A-4BDE-8CB2-C36D93330CA6}	{4876713C-3BA2-4D2C-A734-6BFDBE124E59}

Figure 5.7 'GDB_ObjectClasses' table stores references to COM classes

5.6.2. Loading data into the geodatabase schema

Once the empty schema has been created, existing dataset can be loaded into that schema. But before data can be loaded into the empty geodatabase schema, the data has to be structured to conform to the model. The existing Top10Vector data are in shapefiles. These data have to be structured according to the FeatureClass model. For example, the attribute names and types of the existing Top10Vector layers should be exactly the same as those in the empty geodatabase schema. Also to each feature class, we have to add additional attribute fields for the ObjectDate, VersionDate and VersionNumber. Once the data has been structured, we can use the *Object Loader* in ArcMap to load the data into the geodatabase feature classes. The *Object Loader* automatically converts the shapefile features into geodatabase objects and generates an error log file if any. If the data have been well structured to march the geodatabase schema, there should not be problems. The only problem we experienced was the custom do-

mains. The *Object Loader* failed to load attribute values for which custom domains have been specified. We did not have time to investigate this problem thoroughly. Custom domains however can be manually assigned to the fields without problems.

5.6.3. User interfaces

User interfaces have to be developed for defining and executing constraints and for performing temporal queries. The user interfaces could also have been developed as COM classes. However, this requires additional knowledge about working with windows and dialogs in ATL COM programming, which could not be mastered within the limited time of the research. Therefore we decided to develop user interfaces in Visual Basic for Application (VBA) inside ArcMap, with which the author is already familiar. Interfaces have been developed for entering, editing and executing the constraints rules. Also interfaces have been developed for performing temporal queries. Visualizing temporal information is outside the scope of this research; therefore temporal query results are displayed in very simplified forms.

5.7. Testing the behaviours in ArcMap

In the following sections, we illustrate some of the implementation of the behaviours in ArcMap, a product of ESRI.

5.7.1. Consistency constraints

Figures 5.8 and 5.9 show the interfaces for defining, modifying and executing the constraints. Users can define the constraints, add them to the ConstraintClass and are they immediately monitored by the system during the updating process. The example given in Figure 5.8 shows a binary constraint defined between BuiltupArea and RoadArea feature classes, and the one in Figure 5.9 shows a minimum size constraint definition for the BuiltupArea class.

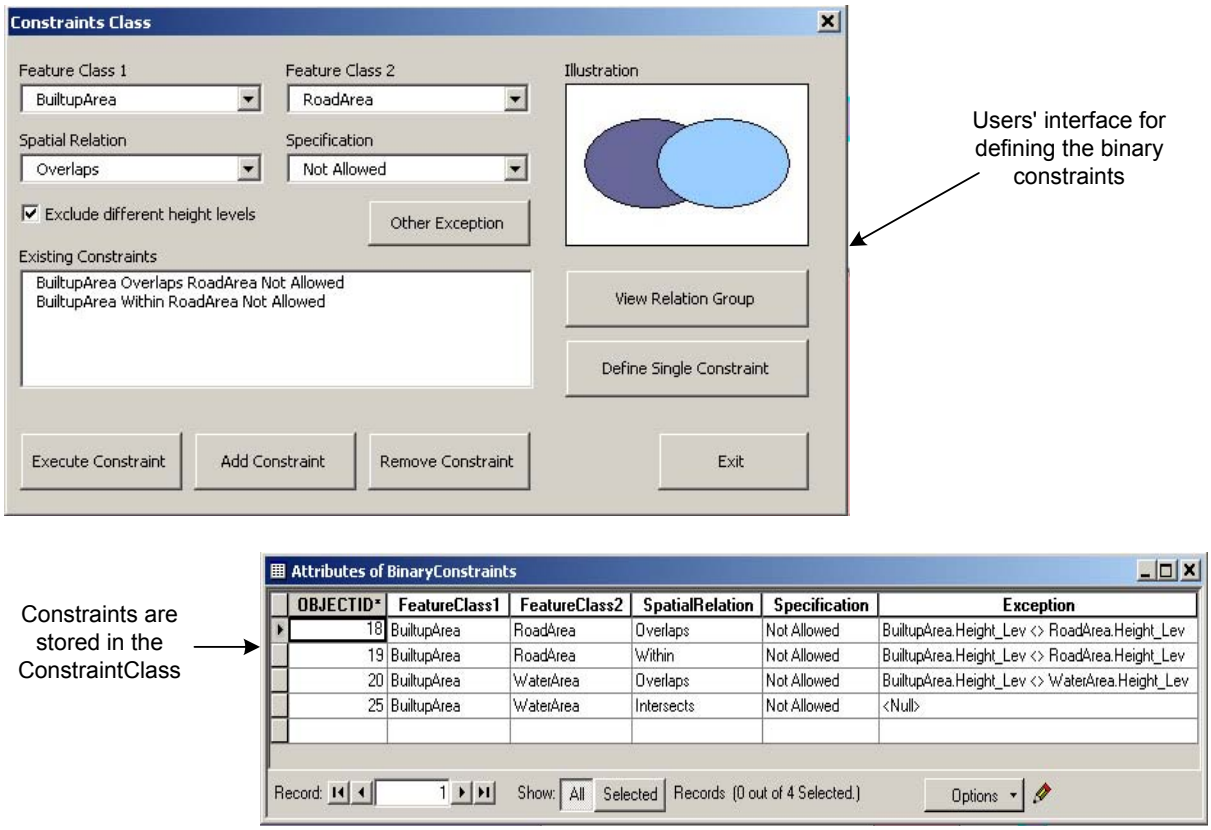


Figure 5.8 Interface for defining binary constraints

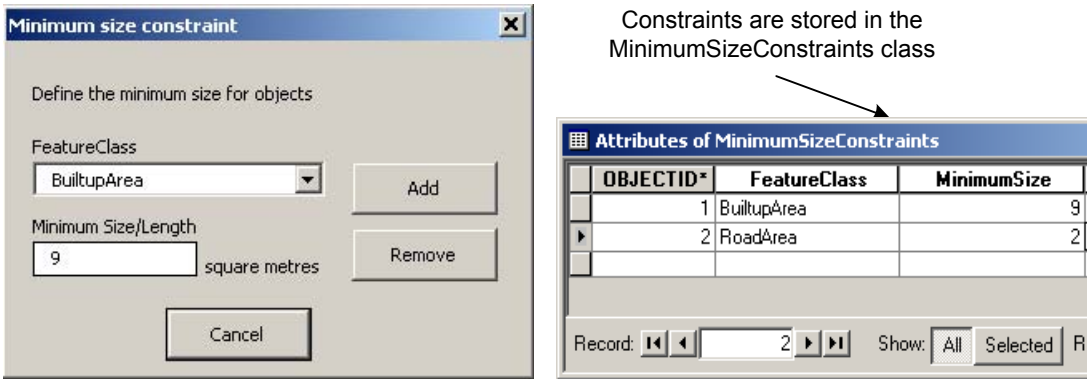


Figure 5.9 Interface for defining minimum size constraints

Figures 5.10 and 5.11 show examples of violations of binary constraints and minimum size constraints respectively. In Figure 5.10, a constraint has already been defined that states that: 'BuiltupArea object should not overlap with RoadArea object'. When the user tries to digitise a BuiltupArea object that overlaps with a RoadArea object, the system immediately issues a violation warning to the user. In Figure 5.11, the area size of the newly digitized BuiltupArea object is less than the minimum size defined for BuiltupArea feature class. Thus a violation message is triggered by the system asking the user whether he/she still wants to add the object. If the user chooses not to add the object, the system will delete the new object.

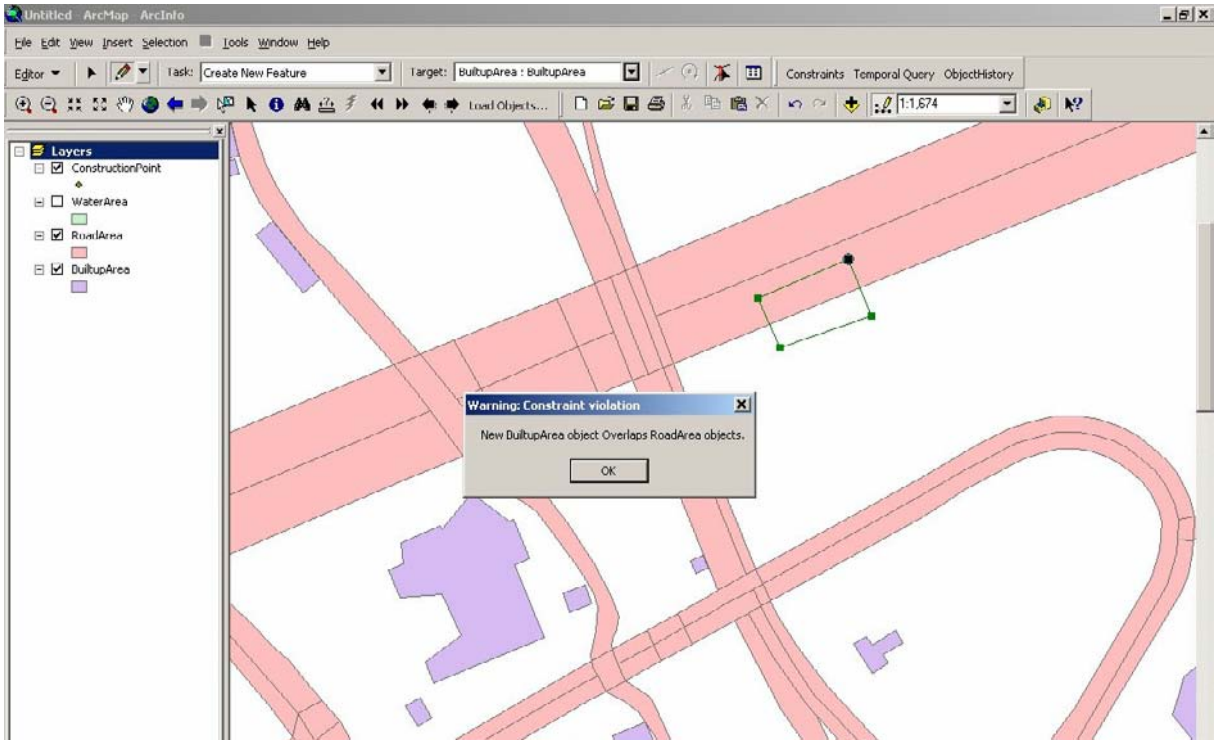


Figure 5.10 Binary constraint violation during digitizing

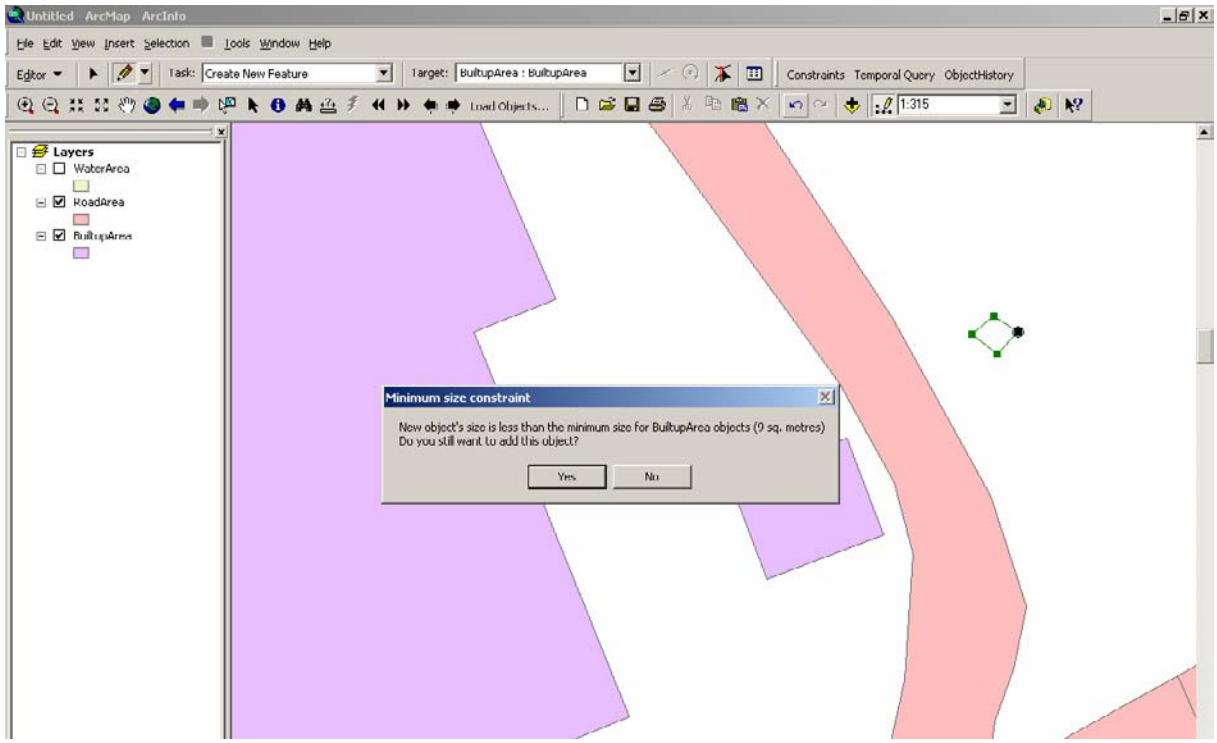


Figure 5.11 Minimum size constraint violation during digitizing

5.7.2. Multiple representations

Two main issues are dealt with concerning the multiple representations, namely: how to maintain the topologic and attribute consistency between the RoadArea and RoadCentreline classes. The following examples illustrate some of the implementation of the behaviours developed in the design phase. In Figure 5.12, when the attribute of the RoadArea object changes, the corresponding attribute of the related centreline object is automatically updated to preserve attribute consistency. In Figure 5.13, when a RoadArea object is moved (or deleted), the related centreline object is automatically moved (or deleted) along with the area object to maintain topologic consistency.

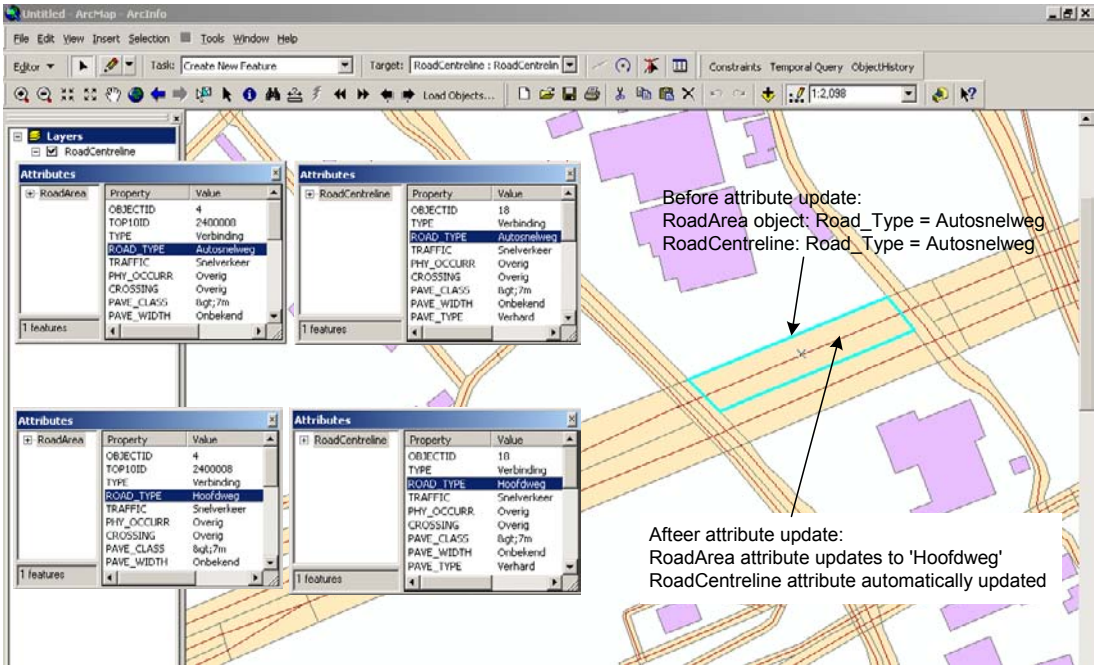


Figure 5.12 Attribute update propagation between RoadArea and RoadCentreline classes

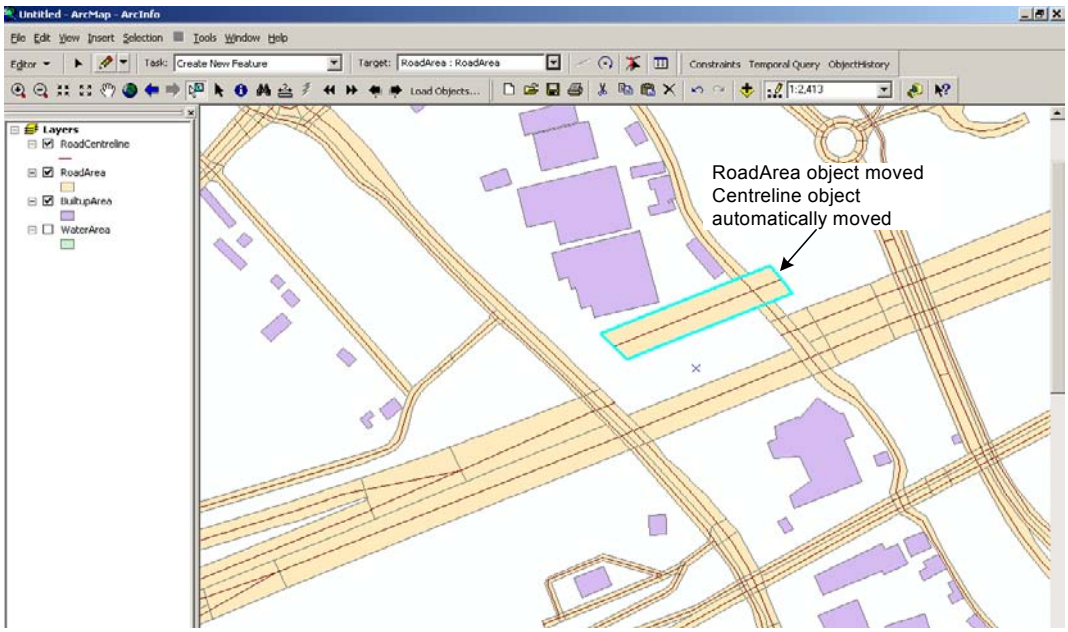


Figure 5.13 Moving a RoadArea object automatically moves the related RoadCentreline object

In Figure 5.14, when a new centreline object is digitized, the system detects a valid relationship with existing RoadArea object and asks the user whether to propagate the attributes of the area object to the new centreline.

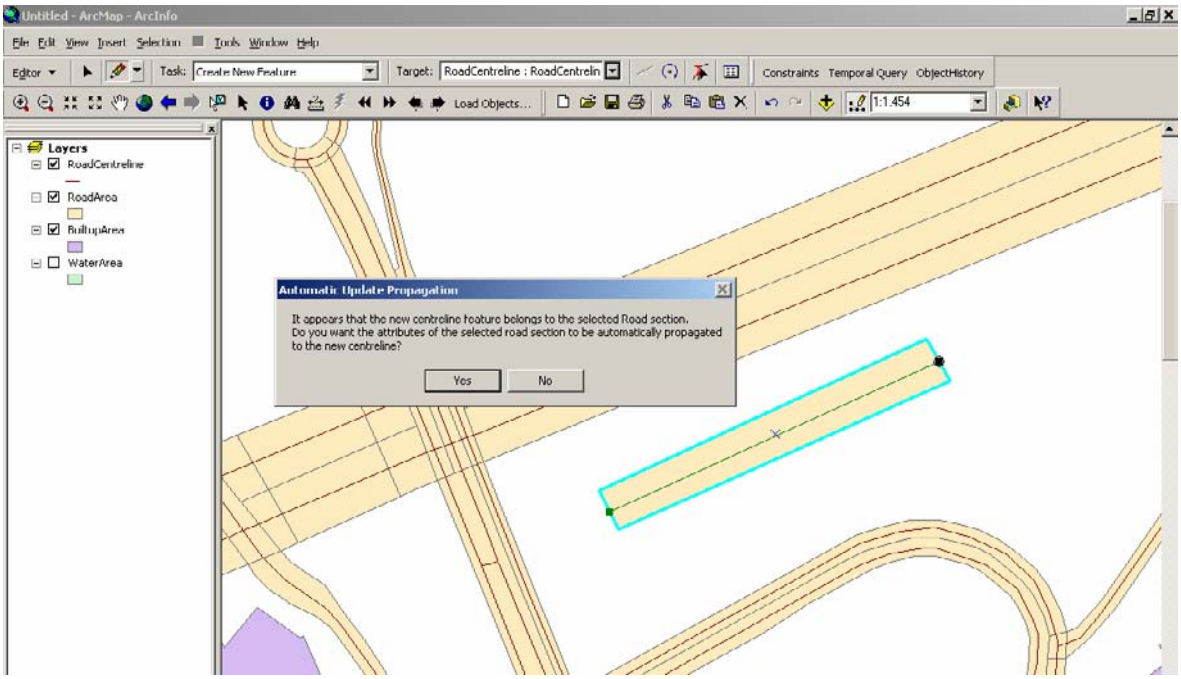


Figure 5.14 Digitizing a new centreline object

5.7.3. Temporal behaviours

5.7.3.1. Time attributes

According to the temporal model developed in the design phase, every object is assigned three time attributes: ObjectDate, VersionDate and VersionNumber (1). When an object attribute is updated, it leads to a new version of the same object with new VersionDate and VersionNumber. In Figure 5.15, when the attribute of the selected RoadArea object is updated, and the system immediately records new VersionDate and VersionNumber for the new version of the object.

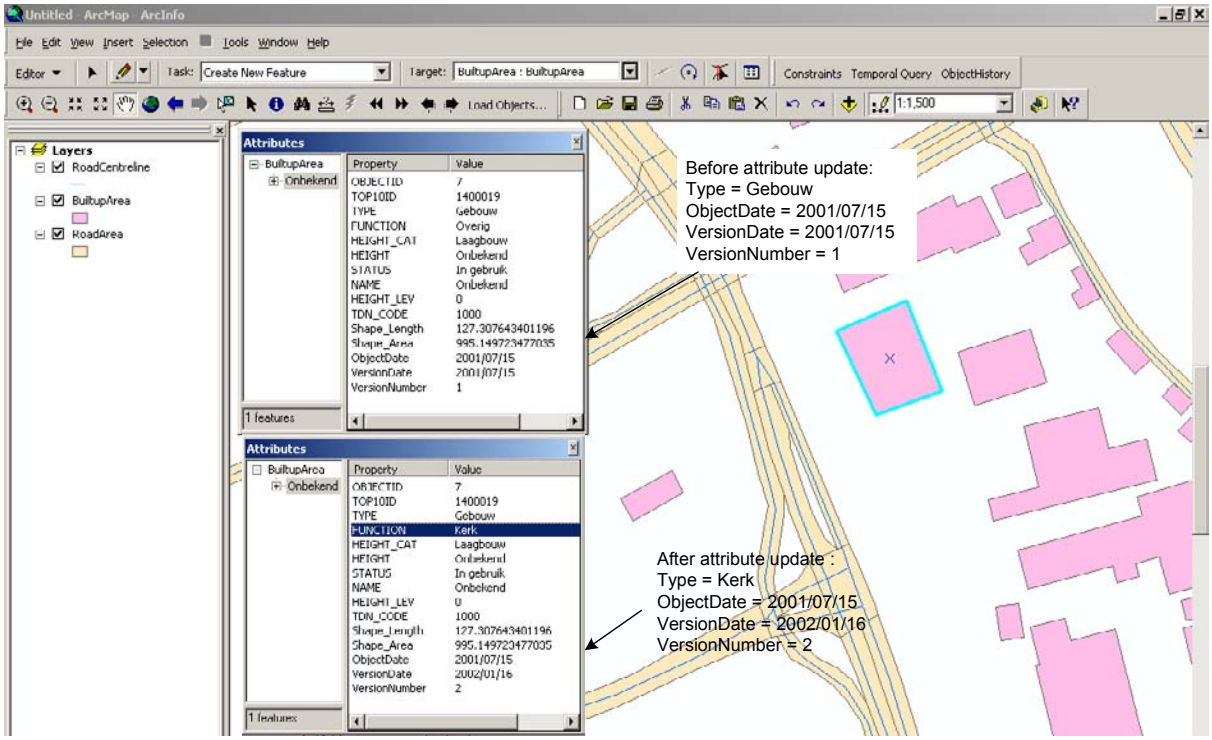


Figure 5.15 Modifying an attribute of an object results in a new version of the object

In Figure 5.16, when the geometry of the object is updated, the system detects that the geometric change is more than the minimum size of objects defined for the BuiltupArea class. The user is prompted whether the change should lead to a new object or a new version. The user's response will cause the system to take the appropriate actions outlined in section 5.5.2.

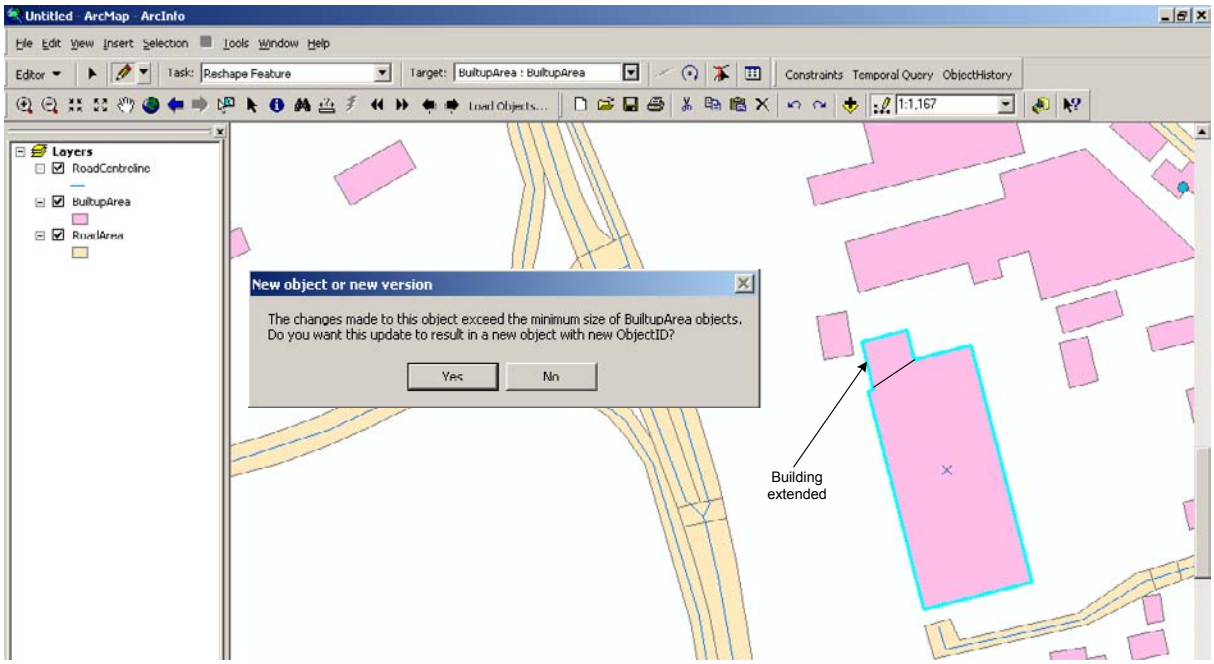


Figure 5.16 Changes made to an object lead to either a new object or version

5.7.3.2. Splitting or merging object(s)

According to the temporal model, when an existing object is split into two or more parts, the result is new objects with new ObjectID, ObjectDate, VersionDate and VersionNumber. The original object is deleted and transferred to the temporal class associated with the feature class. The same consideration applies when objects are merged. In Figure 5.17, a BuiltupArea object (ObjectID = 18) is split into objects A and B. After the split operation, the original object ceases to exist and objects A and B are treated by the system as shown in Figure 5.17.

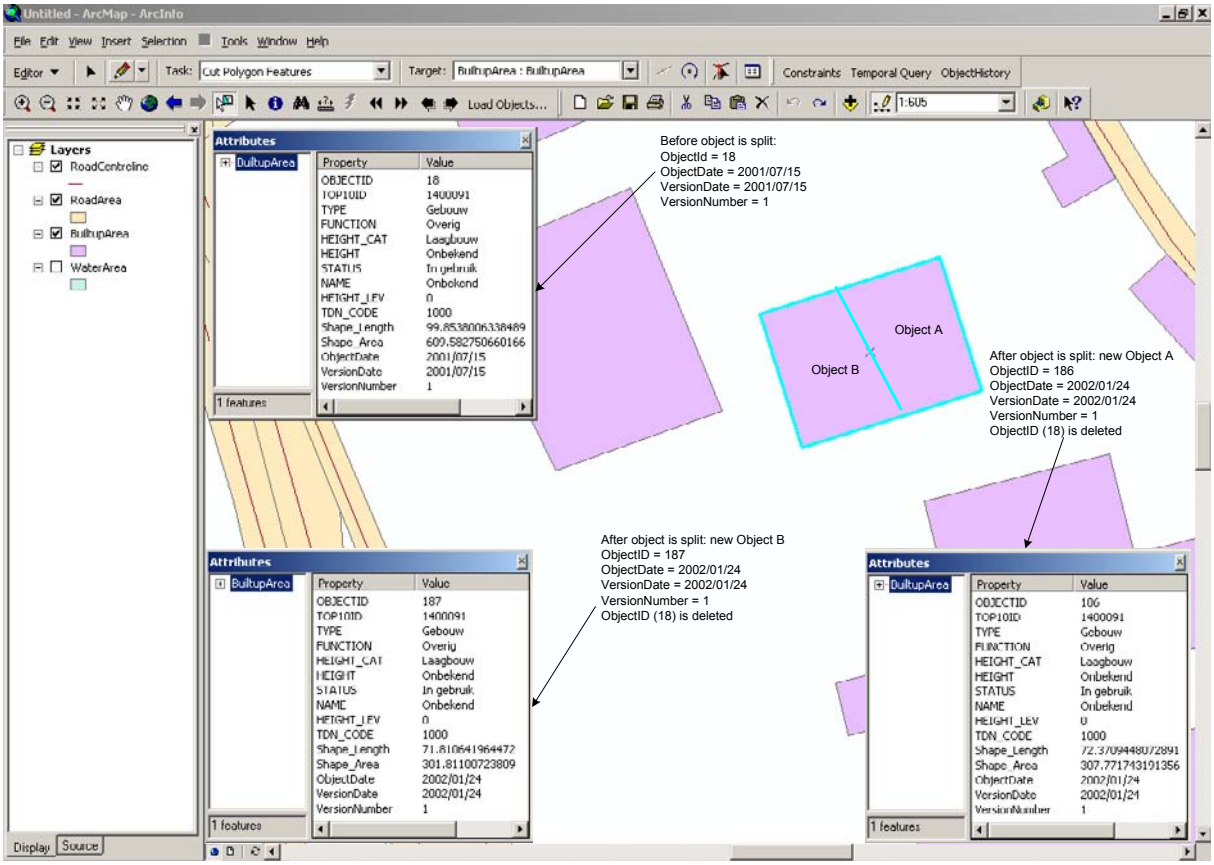


Figure 5.17 Splitting an object into two parts results in new objects

5.7.3.3. Temporal queries

Different types of queries can be answered in a temporal GIS depending on the functionality provided. We have provided simple interfaces for the user to perform simple temporal queries. The following are examples of temporal queries that can be performed.

Query type 1: Scan the history of an object with a time condition?

Figure 5.18 shows the interface for performing this type of query. The query illustrated below says:
Query: 'Show the history of BuiltupArea object (ObjectID = 92), after 21/10/2001'

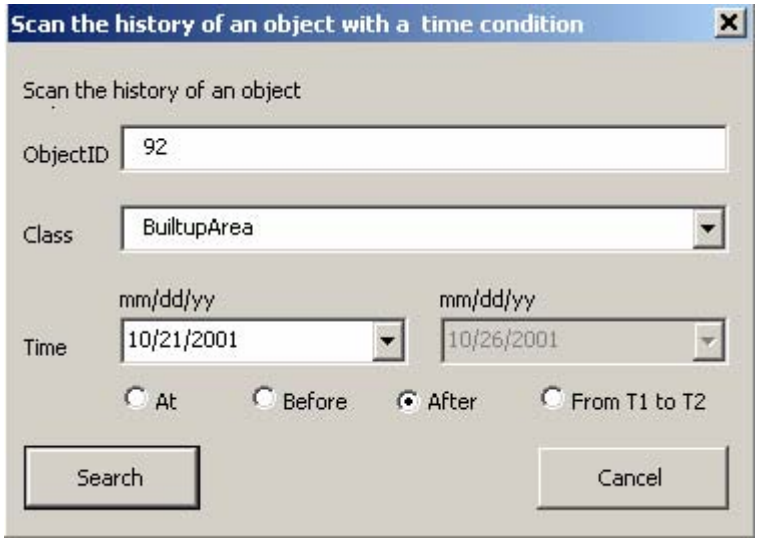


Figure 5.18 Scan the history of object with a time condition

Query Results:

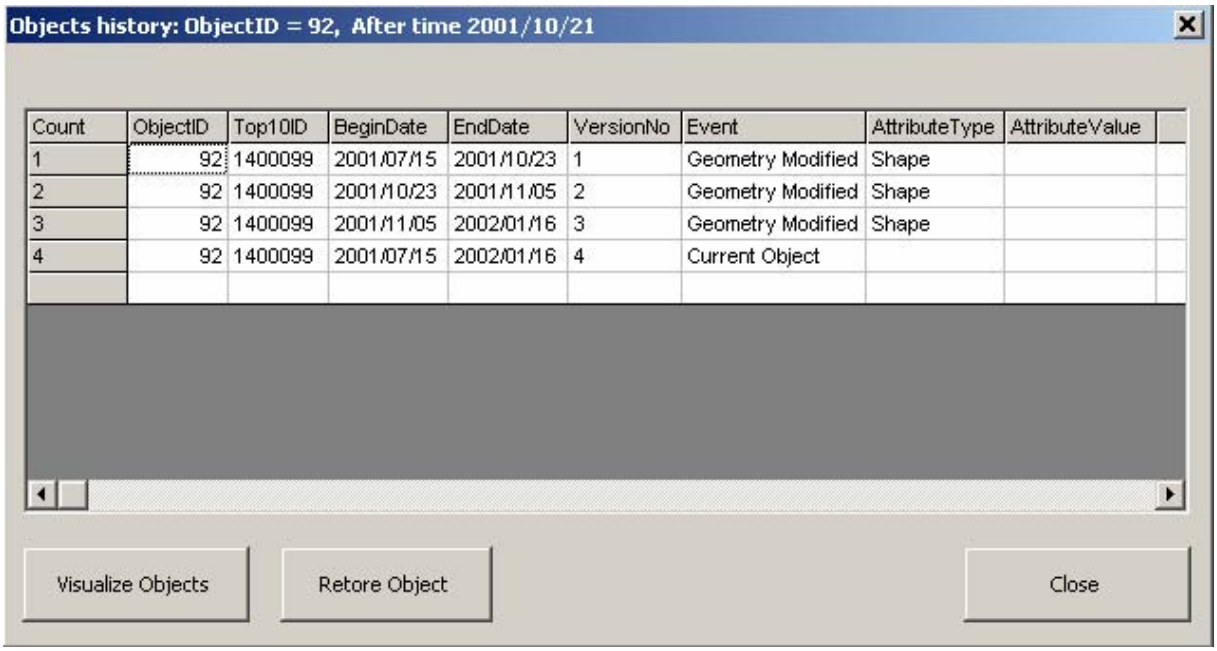


Figure 5.19 The history of ObjectID (92), after 2001/10/21

Query type 2: Which BuiltupArea objects have been modified from 2001/02/26 to 2002/02/26?

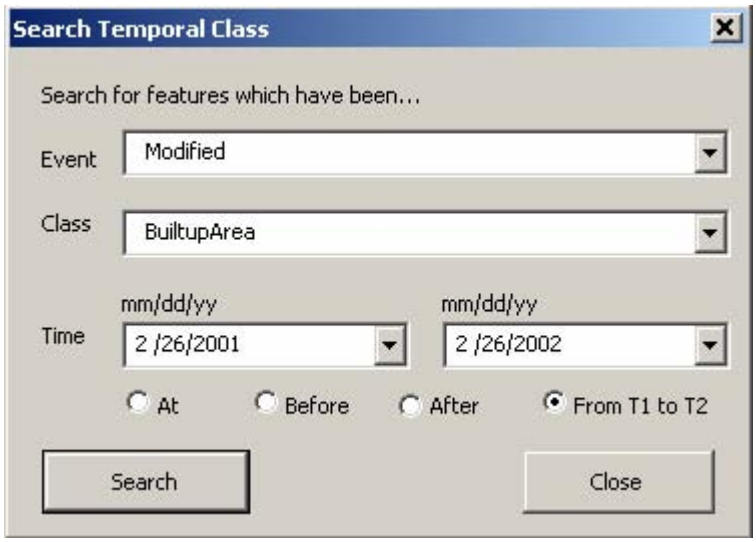
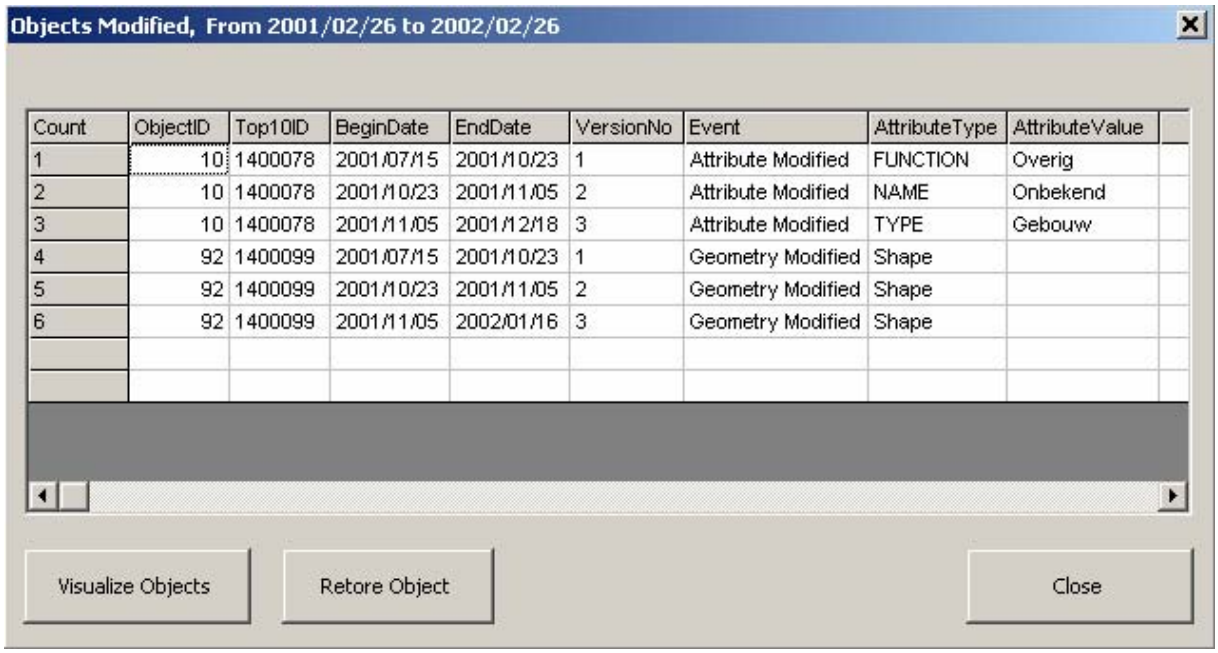


Figure 5.20 Which BuiltupArea objects have been modified from 2001/02/26 to 2002/02/26?



Count	ObjectID	Top10ID	BeginDate	EndDate	VersionNo	Event	AttributeType	AttributeValue
1	10	1400078	2001/07/15	2001/10/23	1	Attribute Modified	FUNCTION	Overig
2	10	1400078	2001/10/23	2001/11/05	2	Attribute Modified	NAME	Onbekend
3	10	1400078	2001/11/05	2001/12/18	3	Attribute Modified	TYPE	Gebouw
4	92	1400099	2001/07/15	2001/10/23	1	Geometry Modified	Shape	
5	92	1400099	2001/10/23	2001/11/05	2	Geometry Modified	Shape	
6	92	1400099	2001/11/05	2002/01/16	3	Geometry Modified	Shape	

Figure 5.21 BuiltupArea objects modified from 2001/02/26 to 2002/02/26

A simple visualization tool

A simple visualization tool has been developed for visualizing the pattern of change over time. The query results obtained in Figures 5.19 and 5.21 can be visualized by clicking the 'Visualize Objects' button, as shown in Figures 5.22 and 5.23. In Figure 5.22, the current objects in the feature class are shown in blue colour. Clicking on an object in the query results shown in the grid on the right displays the temporal object (in yellow) in the map window. The user can click each version in the grid n succession to visualize how the object evolved over time. Figure 5.22 shows version 3 of the object displayed in yellow, and Figure 5.23 shows version 4.

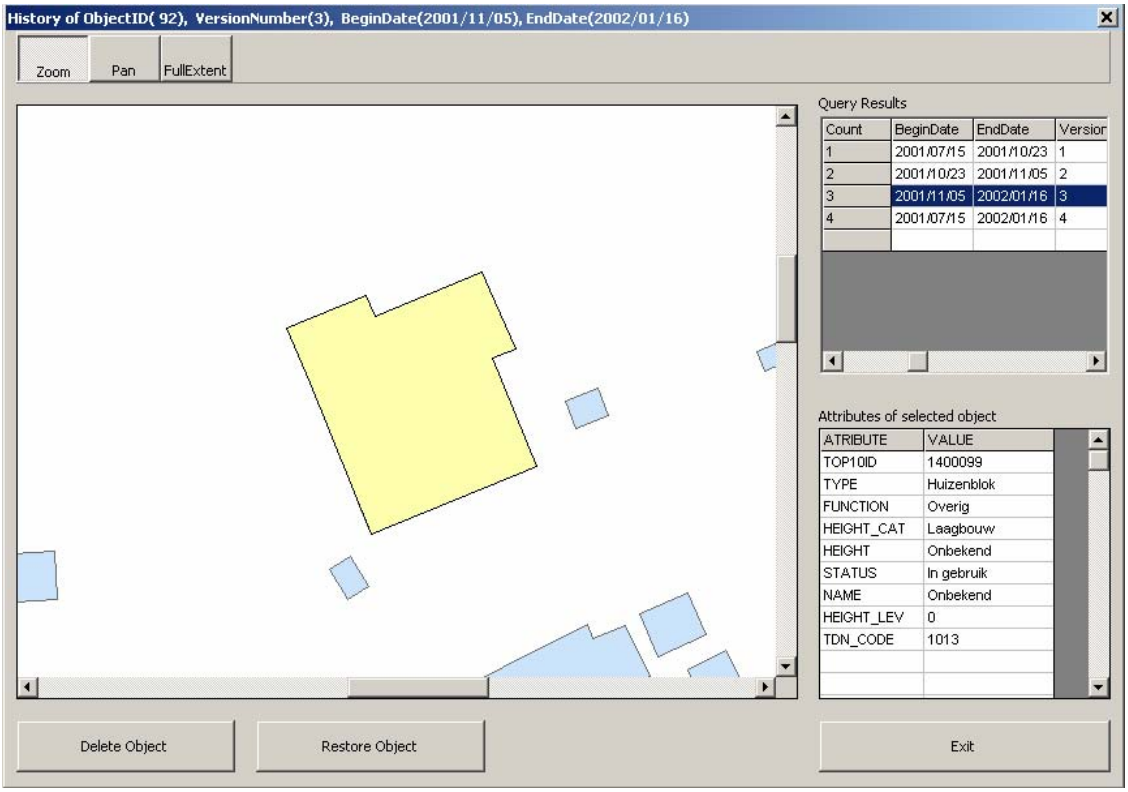


Figure 5.22 Visualizing object history, Version 3

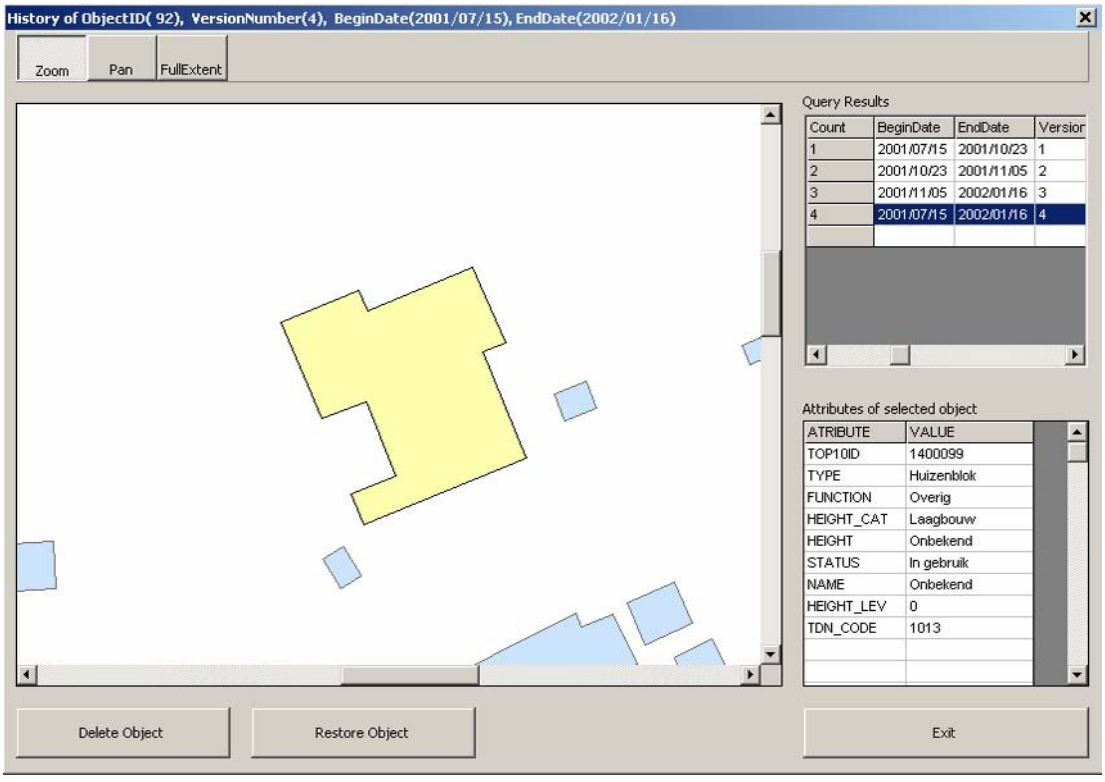


Figure 5.23 Visualizing object history, Version 4

5.8. Concluding remarks

This implementation could not cover all the behaviours designed in chapter 4 due to time reasons. However, the principles illustrated in this implementation can be applied to other similar requirements. With the consistency constraints, the *exception* part was not implemented. In addition, the constraints can only be defined for all objects in the classes, not for a subset, although the *exception* part, if implemented, can be used to specify (partly) this requirement. Automatic implementation of custom domains proved problematic. This may be due to many reasons, including possible software limitation, but there was no time for investigation. For the multiple representations, again not all the behaviours designed have been implemented. The temporal model has been fully implemented, although not everything could be illustrated in this report. The performance of the integrated model has not been tested, neither was it the objective of this research. No statistics have been carried out on performance issues.

6. Conclusions and recommendations

6.1. Introduction

This chapter represents the summary of the research, the main achievements and the conclusions drawn, as well as recommendations for further research.

6.2. Summary of the research

This research investigated spatial object behaviours in three aspects: consistency constraints, multiple representations and temporal behaviour. The objective was to apply the object-oriented concepts to model the behaviour of the spatial objects and to realise these behaviours using the geodatabase of ESRI. In chapter 2, we reviewed the fundamental concepts of object orientation, database models and component-based development. Chapter 3 covered spatial data models, the mathematical background of modelling spatial relationships and consistency constraints. Then in chapter 4 these concepts are employed to develop an integrated conceptual model to handle the specific requirements investigated in this research. The conceptual model was then implemented by extending ESRI geodatabase model in chapter 5. The following summarize the concepts developed:

- The consistency constraints have been modelled as methods defined on the feature classes. The constraints that have topologic structure are defined by using the spatial relationships between the objects. Other constraints are implemented through attribute domains while others are implemented by writing code for them. The constraints are stored and managed by classes inside the database. Simple interfaces have been developed for users to define, modify and execute the constraint rules.
- The multiple representations dealt with in this research concerned road areas and their centrelines. Separate feature classes are used to store the two representations and linked by a composite relationship. Three methods are defined for maintaining consistency between the representations and for realising update propagations (i.e., RelatedObjectCreated, RelatedObjectChanged, and RelatedObjectDeleted). These methods monitor the events that may take place in the feature class and inform the related feature class to take the appropriate actions in order to preserve the consistency between the two representations.
- In this research, time is modelled as an attribute of spatial data. We associate a temporal class to each feature class. Three time attributes are given to objects in the feature class, namely: ObjectDate, VersionDate and VersionNumber. Modification to existing objects either results in new objects or new versions of the same object. In the case of a new object, the original object is deleted and transferred to the temporal class and a new object is created. In the case of a new version, the original object is retained; new VersionDate and VersionNumber are recorded, while the old version is transferred to the temporal class. All deleted and modified objects are stored in the corresponding temporal class with their time stamps.

6.3. Conclusions

On the basis of the issues addressed in this research, the following conclusions can be drawn:

- This research has developed a methodology for the online monitoring of consistency constraints in the geodatabase. The constraints have been defined as methods and invoked each time new objects are created and existing objects are modified. The constraints are stored and managed by classes inside the database. Storing the constraints as part of the database and automatically monitoring constraints violations during database updating will enable the system to ensure more readily the integrity of the database. Constraints can be defined by users and immediately monitored by the system during the creation and modification of objects in the database.
- Two main issues are dealt with in the case of the multiple representations, namely, how to maintain topologic and attribute consistency between the two representations. Topologic consistency has been realised by identifying the possible topological relationships between the two representations and translating them into consistency constraints in the database. Attribute consistency has been realised by developing update propagation mechanisms to automatically propagate updates from the main representation to dependent representation. The techniques developed in this research use three key concepts to maintain consistency: spatial relationships, link between the two classes and the object-oriented concept of message passing between the classes. These offer the means for maintaining consistency between the multiple representations.
- The temporal model keeps the history of objects in the database. The model stores the geometry of changed objects and only the attributes that changed. The model allows identifying the successors and predecessors of different versions of objects in the database. This will ensure users that they are referring to the same object when updates are delivered to them. Some temporal queries can be performed as illustrated in chapter 5.
- ESRI geodatabase model was used for the implementation. This is an object-relational model in which both spatial and attribute data are stored in a single relational database. The geodatabase model extends the relational model by integrating it with object-oriented concepts and developing software components based on Microsoft Component Object Model (COM). Feature classes in the geodatabase can have COM components that implement the behaviour of objects in the feature class. The geodatabase stores references to these components and invoke them during editing of the data.
- Many benefits can be realised with the geodatabase approach. First, the geodatabase constitutes a uniform repository of both spatial and attribute data in a single database system. Objects in the geodatabase can have behaviour associated with them. Integration with object-oriented concepts and COM technology allows great level of customization and reuse of the model to create application-specific models. Furthermore, the ArcObjects Library is compliant with the OpenGIS simple feature specifications, which may (in the future) provide the framework for interoperability. The main problem we encountered was the custom domains, but this was not investigated. In addition, although the ArcObjects Library is extensive, more functionality needs to be added to allow high level of customization.

6.4. Recommendations for further research

The main areas of further research directly relevant to this work are the following:

- The custom behaviours modelled in this research were implemented in the personal geodatabase, which is limited to small amount of data (about 2 gigabytes) and is suitable for project-oriented GIS. The same principles illustrated here also apply to the multi-user geodatabase. It is therefore recommended to implement the behaviours in the multi-user geodatabase such as Oracle.
- The temporal model developed in this research is simple and limited. The multi-user ArcSDE geodatabase allows multiple versions of the database to coexist. If the concepts developed in this research are combined with the versioning mechanism in ArcSDE geodatabase, a more capable temporal GIS may be developed.
- The temporal model can be extended to include (for example) event and observation times.
- This research did not address visualization of temporal information. How to visualize temporal data is an active research on its own. Further research could concentrate on visualization issues.
- The performance of the model should be tested with different datasets.
- Not all the behaviours modelled in this research were actually implemented for time reasons. More implementation is needed to realize the full capabilities and limitations of the geodatabase model.

References

1. Abraham, T. & Roddick, F. J. (1999), Survey of spatio-temporal databases. Geoinformatica, Vol. 3, No. 1, pp. 61-99.
2. Abrantes, G. & Pinheiro, A. (1997), Behaviour modelling in geographic applications. In: Hodgson, S., Rumor, M., Horts, J. J. (eds.), Geographical Information '97: Proceedings of the 3rd Joint European Conference and Exhibition on Geographical Information, Vienna, Austria, Vol. 1, pp.254-257.
3. AGI (1996), The Association for Geographic Information (AGI) GIS dictionary. Website: http://www.agi.org.uk/pag-es/dict-ion/dict-agi.htm, accessed on November 3, 2001.
4. Blaha, M. & Premerlani, W. (1998), Object-oriented modelling and design for database applications. Upper Saddle River: Prentice Hall.
5. Booch, G. (1994), Object-oriented analysis and design with applications. The Benjamin/Cummings Publishing Company.
6. Budd, T. (1991), An introduction to object-oriented programming. New York: Addison-Wesley Publishing Company.
7. Candy, J. T. (1995), Development of a prototype temporal Geographic information system. , Unpublished Master's Thesis, Simon Fraser University, Burnaby, B. C., Canada. Website: http://gisnt15.soe.bcit.ca/geocandy/jcandy/thesis/sabs.html, accessed on September 15, 2001.
8. Clementini, E. & Di Felice, P. (1994), Object-oriented modelling of geographic data. Journal of the American Society for Information Sciences, Vol. 45, No. 9, pp. 694-704.
9. Clementini, E. & Di Felice, P. (1995), A comparison of methods for representing topological relationships. Information Sciences, Springer-Verlag, Singapore, Vol. 3, pp. 149-178.
10. Clementini, E., Di Felice, P. & Van Oosterom, P. (1994), A small set of formal topological relationships suitable for end-user interactions. In: Abel, D. & Ooi, B. C. (eds), Proceedings of the 3rd International Symposium on Large Spatial Databases, SSD'93. Lecture Notes in Computer Science, Springer-Verlag, Singapore, Vol. 692, pp. 277-295.
11. Cockcroft, S. (1996), Towards the automatic enforcement of integrity constraints in spatial database systems. In: Pascoe, R. T., Sutherland, N. C., & Gorman, P. (eds.), Proceedings of the 8th Colloquium of the Spatial Information Research Centre, University of Otago, New Zealand, pp. 33-42.
12. Cockcroft, S. (1997), A taxonomy of spatial integrity constraints. An International Journal on Advances of Computer Science for Geographic Information Systems, Geoinformatica, Vol. 1, No. 4, pp. 327-343.
13. Cockcroft, S. (1998), User defined spatial business rules: storage, management and implementation—a pipe network case study. In: Firns, P. G. (ed.), Proceedings of the 10th Colloquium of the Spatial Information Research Centre, University of Otago, New Zealand.
14. Cooper, R. (1997), Object databases: An ODMG approach. International Thompson Computer Press.
15. Creating custom features and geodatabase schemas: CASE tools tutorials. ESRI, Website: http://arconline.esri.com/arcobjectsonline/, accessed on September 15, 2001.

16. De Vet, R. M. A. (1997), The application of object technology in GIS. In: Hodgson, S., Rumor, M., Horts, J. J. (eds.), Geographical Information '97: Proceedings of the 3rd Joint European Conference and Exhibition on Geographical Information, Vol. 1, pp.197-200.
17. De Zeeuw, C. J., et al (2000), Gebruikersspecificaties TOP10-21ste eeuw, Centre for Geo-information (CGI), Alterra, Wageningen University.
18. Egenhofer, J. M. & Sharma (1993), Topological relations between regions in R^2 and Z^2. In: Abel, D. & Ooi, B. C. (eds.), Proceedings of the of the 3rd International Symposium on Large Spatial databases (SSD' 93), Lecture Notes in Computer Science, Springer-Verlag, Vol. 692, pp. 316-336.
19. Egenhofer, J. M. & Frank, U. A. (1992), Object-oriented modelling for GIS. Journal of the Urban and Regional Information Systems Association, Vol. 4, No. 2, pp. 3-19.
20. Egenhofer, J. M. & Franzosa, R. D. (1991), Point-set topological spatial relations. International Journal of Geographical Information Systems, Vol. 5, No. 2, pp. 161-174.
21. Egenhofer, J. M. & Herring, J. R. (1991), Categorizing binary topological relations between regions, lines and points in geographic databases. Technical report, Department of surveying Engineering, University of Maine, Orono.
22. Egenhofer, J. M. (1989), A formal definitions of binary topological relationships. In: Latwin, W & Schek, H. J. (eds.), Third International Conference of Data Organization and Algorithms (FODO), Lecture Notes in Computer Science, Springer-Verlag, Vol. 367, pp. 457-472.
23. Egenhofer, J. M., Clementini, E., Di Felice, P. (1994), Evaluating inconsistency among multiple representations. Sixth International Symposium on Spatial Data Handling, Edinburgh, Scotland, pp. 901-920.
24. Ellmer, E. (1993), Basic principles and concepts of object orientation. University of Vienna, Institute of Applied Computer Science and Information Systems. Website: http://citeseer.nj.nec.com/ellmer93basic.html, accessed on November 3, 2001.
25. Elmasri, A. R. & Navathe, S. (2000), Fundamentals of database systems. New York: Addison-Wesley.
26. ESRI (1999), Exploring ArcObjects: Vol 1 & 2, Edited by Zeiler M., USA: Environmental System Research Institutes, Inc.
27. ESRI User Conference (2000), Geodatabase concepts for the new user, Twentieth annual ESRI International Conference, June 26-30, 2000.
28. Fernando D. N. et al (1997), Towards a foundation for object-oriented GIS. In: Hodgson, S., Rumor, M., Horts, J.J. (eds.), Geographical Information '97: Proceedings of the 3rd Joint European Conference and Exhibition on Geographical Information, Vol. 1, pp.58-67.
29. Firesmith, D. G. & Eykholt, E. M. (1995), Dictionary of object technology: The definitive desk reference. New York: SIGS Books, Inc.
30. Frank, A. U. (1994), Qualitative Temporal Reasoning in GIS: Ordered Time Scales. In: Waugh, T. C. & Healey, R. C. (eds.), Proceedings of the Sixth International Symposium on Spatial Data Handling, London, Taylor and Francis, pp. 410-430.
31. Fussel, L. M. (1997), Foundations of Object-Relational Mapping. Website: http://www.chimu.com/publications/objectRelational/objectRelational.pdf, accessed on November 3, 2001.
32. Grimes, R., Templeman, J., Stockton, A., Watson, K. & Reilly, G. (1999). Beginning ATL COM programming. Wrox Press.
33. Guptill, C. S. (1995), Temporal information. In: Guptill, C. S. & Morrison, J. L. (eds.), Elements of spatial data quality. Published on behalf of The International Cartographic Conference, Pergamon Press, pp. 153-159.

34. Hardy, P. G. & Woodsford, P. A. Mapping with life features: Object-oriented representation. Website: http://oldsite.laser-scan.com/papers/, accessed on July 15, 2001.
35. Hazelton, N. J., Leahy, F. & Williamson, I. (1990) On the design of temporally referenced, 3-D geographic information system: development of four-dimensional GIS. In the proceedings of GIS/LIS '90, Anaheim, California, Vol. 1, pp. 357-372.
36. Henderson-sellers, B. & Edwards, J. M. (1994), Book two of object-oriented knowledge: The working object. Sydney, Australia: Prentice Hall.
37. Hornsby, K. & Egenhofer, J. M. (1998), Identity-based change operations for composite objects. Eighth international symposium on spatial data handling, Canada. July, 1998, pp.202-213
38. Kainz, W. & Egenhofer, J. M. (1993), Modelling spatial relations and operations with partially ordered sets. International Journal of Geographical Information Systems, Vol. 7, No. 3, pp. 215-229.
39. Kainz, W. (1995), Logical consistency. In: Guptill, C. S. & Morrison, J. L. (eds.), Elements of spatial data quality. Published on behalf of The International Cartographic Conference. Pergamon Press, pp. 108-137.
40. Khoshafian, S. & Abnous, R. (1995), Object Orientation: Concepts, Analysis and Design, Languages, Databases, Graphical User Interfaces and Standards. New York: John Wiley & Sons, Inc.
41. Khoshafian, S. (1993), Object-oriented databases. John Wiley & Sons.
42. Kilpelainen, T. (2000), Maintenance of multiple representations database for topographic data. The Cartographic Journal, Vol. 37, No. 2, pp. 101-107.
43. Kufoniyi, O. (1995), Spatial coincidence modelling, automated database updating and data consistency in vector GIS. PhD Thesis, ITC Publication Number 28.
44. Kufoniyi, O., Molenaar, M. & Bouloucos, T. (1994), Topology as a tool for consistency operations in vector maps. International Archive of Photogrammetry and Remote Sensing, Vol. 30, Pt. 3-1, pp. 455-462.
45. Langran, G. & Chrisman, N. (1988). A framework for temporal geographic information. Cartographica Vol. 25, No. 3, pp. 1-14.
46. Langran, G. (1992), Time in geographic information systems. London: Taylor and Francis Ltd.
47. Lee, R. C. & Tepfenhart, W. M. (2001), UML and C++: A practical guide to object-oriented development. Prentice-Hall, Inc.
48. MacDonald, A. (1999), Building a geodatabase, USA: Environmental System Research Institutes, Inc.
49. Martin, J. & Odell, J. J. (1992), Object-oriented analysis and design. Englewood Cliffs, NJ: Prentice Hall.
50. Mattos, N. M., Meyer-Wegener, K. & Mitschang, B. (1993), Grand tour of concepts of object-orientation from a database point of view. Data & Knowledge Engineering, Vol. 9, pp. 321-352.
51. Medeiros, C. B., Bellosta, M. J. & Jomier, G. (1996), Managing multiple representations of georeferenced elements. In the proceedings of the 7th International Workshop on Database and Expert Systems Applications (DEXA). Website: http://citeseer.nj.nec.com/31254.html, accessed on November 3, 2001.
52. Molenaar, M. (1998), An introduction to the theory of spatial object modelling for GIS. London: Taylor & Francis Ltd.
53. MSDN Library, April 2001, Microsoft Corporation.
54. Newell, R. G. (1992), Practical experiences of using object orientation to implement a GIS. In the proceedings of GIS/LIS '92, Annual Conference and Exposition, San Jose, California, Vol. 2, pp 624-629.

55. OMG Unified Modelling Language Specification, Version 1.4, September 2001. Website: http://www.omg.org/technology/documents/formal/uml.htm, accessed on November 3, 2001.
56. OpenGIS (1999), Simple feature specification for OLE/COM. Revision 1.1, OpenGIS Project Document 99-050, Release Date: May 18, 1999, Open GIS Consortium, Inc.
57. Pacheco, M. A. (1997), Dynamic integrity constraints definition and enforcement in databases: a classification framework. In the Proceedings of the IFIP TC-11 Working Group 11.5, First Working Conference on Integrity and Internal Control in Information Systems, Zürich, Switzerland, pp. 65-87.
58. Raza, A. (1996), Design and implementation of a temporal GIS for monitoring land use change: a case study of Dar-es-Salaam, Tanzania, MSc Thesis, ITC, Enschede, The Netherlands.
59. Raza, A. (2001), Object-oriented temporal GIS for urban applications. PhD Thesis, ITC Publication Number 79.
60. Roy, D. & Clément, G. (1994), Object-oriented database technology for geographic applications: A case study. Proceedings of the Canadian Conference on GIS, Ottawa, Vol. 1, pp. 664-677.
61. Ruland, P. & Rouvé, G. (1994), Advantages of object-oriented GIS for the integration of hydraulic models. In: Verwey, Minns, van Maksimovic, B. (eds.), Hydroinformatics' 94, p.253-259.
62. Shaner, J. & Wrightsell, J. (1999), Editing in ArcMap, USA: Environmental System Research Institutes, Inc.
63. Sivanantharajah, S. (2001), Constraint rules for spatial database updating, MSc Thesis, ITC, The Netherlands.
64. Spaccapietra S., Parent, C., Vangenot, C. (2000a), GIS Databases: From multiscale to multirepresentations. Website: http://lbdwww.epfl.ch/e/publications/publist.html, accessed on November 3, 2001.
65. Spaccapietra S., Parent, C., Zimanyi, E. (1999), MurMur: A research agenda on multiple representations. Website: http://lbdwww.epfl.ch/e/publications/publist.html, accessed on November 3, 2001.
66. Spaccapietra S., Parent, C., Zimanyi, E. (2000b), MurMur: Database management of multiple representations. Website: http://lbdwww.epfl.ch/e/publications/publist.html, accessed on November 3, 2001.
67. Tryfona, N., Pfoser, D. & Hadzilacos, Th. (1997), Modelling the behaviour of geographic objects: An experience with the Object Modelling Technique. Proceedings of the 9th International Conference on Advanced Information Systems and Engineering (CaiSE' 97), Lecture Notes in Computer Science Vol. 1250, Berlin: Springer-Verlag.
68. Ubeda, T. & Egenhofer, J. M. (1997), Correcting topological errors. In: Scholl, M. & Voidsard, A (eds.), Proceedings of the 5th International Symposium on Large Spatial Databases SSD' 97, July 1997, Springer-Verlag, Vol. 1262, pp. 283-297.
69. West, R. (2001), Understanding ArcSDE, USA: Environmental System Research Institutes, Inc.
70. Williams, M., Bennett, D., et al (2000), Visual C++ unleashed. USA: Sams Publishing.
71. Wirfs-Brock, R., Wilkerson, B. & Wiener, L. (1990), Designing object-oriented software. Englewood Cliffs, NJ: Prentice Hall.
72. Worboys, M. F. (1995), GIS: A computer perspective. London: Taylor and Francis Ltd.
73. Zeiler, M. (1999), Modelling our world. USA: Environmental Systems Research Institute, Inc.

Appendices

Appendix A: Top10Vector feature classes

Appendix A lists the feature classes, attributes, and domains for the Top10Vector database.

Appendix B: Implementation code

Because a large number of pages of code were written during the implementation phase, the code could not be put in the report. Instead, they have been bound separately in a different document. Interested readers are referred to the document entitled: 'Implementation code: Modelling spatial object behaviours in object-relation geodatabase'. The document can be found with Drs. Richard Knippers, my supervisor.

Appendix A: Top10Vector feature classes

The following tables list the feature classes, attributes, and domains in the Top10Vector database. The original list is in Dutch but has been partly translated into English for use in this research.

Table A-1 Topographic basic objects

Dutch	English
Wegdeel	Road section
Spoorbaandeel	Railway section
Waterdeel	Water section
Bebouwing	Built-up
Terrein	Terrain
Inrichtingselement	Construction
Administratief gebied	Administrative area
Beheersgebied	Conservation area
Geografisch gebied	Geographic area
Functioneel gebied	Functional area

Table A-2 Road section (wegdeel)

Attribute (Attribuut)	Domain (Domein)	Cardinality (Multipliciteit)	Option (Optionaliteit)
Type (Type)	Connection (Verbinding) Crossing (Kruising) Area (Vlak)	Single	Mandatory
Road type (Wegtype)	Highway (Autosnelweg) Main road (Hoofdweg) Regional road (Regionale weg) Local road (Lokale weg) Street (Straat) Other road (Overige weg)	Single	Mandatory
Main traffic use (Hoofdverkeersgebruik)	Fast traffic (Snelverkeer) Mixed traffic (Gemengd verkeer) Bus traffic (Busverkeer) Slow traffic (Langzaam verkeer) Bikes/Mopeds (Fietsers/bromfietsers) Pedestrian (Voetgangers) Parking (Parkeren) Other (Overig)	Single	Mandatory
Physical occurrence (Fysiek voorkomen)	On fixed part of bridge (Op vast deel van brug) On movable part of bridge (Op beweegbaar deel van brug) On bank (Op talud) On raised area (Op verhoogd vlak) On lowered area (Op verlaagd vlak) Covered (Overkluisd) In tunnel (In tunnel) Ferry (Veer, pont) Other (Overig)	Single	Mandatory

Crossing type (Kruisingstype)	Part of roundabout (Deel van rotonde) Part of traffic junction (Deel van verkeer-sknooppunt) Other (Overig)	Single	Mandatory
Pavement width class (Verhardingsbreedte-klasse)	>7m 4-7m 2-4m <2m	Single	Mandatory
Pavement width (Verhardingsbreedte)	< actual width in dm > (< Werkelijke breedte in dm >)	Single	Mandatory
Pavement type	Hard surface (Verhard) Loose or light surface (Half verhard) Not paved (Onverhard)	Single	Mandatory
Pavement material (Verhardingsmateriaal)	Asphalt (Asfalt) Very open asphalt (Zeer open asfalt beton) Bricks (Klinkers) Gravel (Grint) Sand (Zand) Other (Overig)	Single	Mandatory
Nr of lanes (Aantal rijstroken)	< amount > (< aantal >)	Single	Mandatory
Direction of the traffic (Rijrichting)	One way (Eenrichting) Two ways (Tweerichting)	Single	Mandatory
Accessibility (Toegankelijkheid)	Public (Openbaar) Not Public (Niet-openbaar)	Single	Mandatory
Status (Status)	Planned (In ontwerp) Under construction (In aanleg) In use (In gebruik) No longer in use (Niet meer in gebruik)	Single	Mandatory
Name (Name))	< street name, road number, exit number > (< straatnaam, wegnummer, afritnummer>)	Multiple	Optional
Road Number (Wegnummer)	< number > (< nummer >)	Multiple	Optional
Height level (Hoogteniveau)	0 (no level assigned) 1 (lowest level) 2 (lowest level plus 1) 3 (lowest level plus2) <Etc.>	Single	Optional

Table A-3 Railway section (Spoorbaandeel)

Attribute	Domain	Cardinality	Option
Type (Type)	Connection (Verbinding) Crossing (Kruising) Area (Vlak)	Single	Mandatory
Verkeersgebruik	Train (Trein) Tram (Tram) Subway (Metro) Other (Overig)	Single	Mandatory
Physical occurrence (Fysiek voorkomen)	On fixed part of bridge (Op vast deel van brug) On movable part of bridge (Op beweegbaar deel van brug) On bank (Op talud) On raised area (Op verhoogd vlak) On lowered area (Op verlaagd vlak) On dike (Op dijk) Cutting (Ingegraven) Covered (Overkluisd) In tunnel (In tunnel) On road (Op weg) Other (Overig)	Single	Mandatory
Track width (Spoorbreedte)	Normal gauge (Normaalspoor) Narrow gauge (Smalspoor)	Single	Mandatory
Number of tracks (Aantal sporen)	*< number > (< nummer >)*	Single	Mandatory
Function (Functie)	Mixed use (Gemengd gebruik) Only Passenger traffic (Alleen personen-vervoer) Only Cargo traffic (Alleen goederenvervoer)	Single	Optional
Electrification (Elektrificatie)	Electrified (Geëlektrificeerd) Not electrified (Niet geëlektrificeerd)	Single	Mandatory
Accessibility (Toegankelijkheid)	Public (Openbaar) Not Public (Niet openbaar)	Single	Mandatory
Status (Status)	Planned (In ontwerp) Under construction (In aanleg) In use (In gebruik) No longer in use (Niet meer in gebruik)	Single	Mandatory
Name (Naam)	< name track> (< naam baanvak >)	Multiple	Optional
Height level (Hoogteniveau)	0 (no level assigned) 1 (lowest level) 2 (lowest level plus 1) 3 (lowest level plus2) <Etc.>	Single	Optional

Table A-4 Water part (Waterdeel)

Attribute	Domain	Cardinality	Option
Type (Type)	Connection (Verbinding) Crossing (Kruising) Area (Vlak)	Single	Mandatory
Water type (Watertype)	Watercourse (waterloop) Lake, pool, pond (Meer, plas, ven, vijver) Ditch, trench (Sloot, greppel) Sea (Zee) Wetlands (Droogvallend)	Single	Mandatory
Width category (Breedteklasse)	< 1m 1-3 m 3-6 m 6-12 m 12-20 m 20 - 50 m 50 - 150 m 150 - 300 m 300 - 450 m > 450 m	Single	Mandatory
Width (Breedte)	*< actual width in dm >* *(< werkelijke breedte in dm >)*	Single	Optional
Main drainage (Hoofdafwatering)	Yes (Ja) No (Nee)	Single	Mandatory
Salt level (Zoutgehalte)	Sweet (Zoet) Salt (Zout) Brackish (Brak)	Single	Mandatory
Accessibility (Toegankelijkheid)	Public (Openbaar) Not-Public (Niet-openbaar)	Single	Mandatory
Physical occurrence (Fysiek voorkomen)	In lock (In sluis) On aqueduct (Op aquaduct) In culvert (In duiker) In culvert siphon (In grondduiker) Other (Overig)	Single	Mandatory
Use (Gebruik)	Fish cultivation (Viskwekerij) Water disposal (Waterzuivering) Swimming pool (Zwembad) Other (Overig)	Single	Mandatory
Flow direction (Stroomrichting)	One direction (Eenrichting) Two directions (Twee richtingen (getijde invloed)) Still water (Stilstaand)	Single	Mandatory

Status	Planned (In ontwerp) Under construction (In aanleg) In use (In gebruik) No longer in use (Niet meer in gebruik)	Single	Mandatory
Name (Naam)	<name water > (< naam water >)	Multiple	Optional
Height level (Hoog-teniveau)	0 (no level assigned) 1 (lowest level) 2 (lowest level plus 1) 3 (lowest level plus2) <etc>	Single	Optional

Table A-5 Built Up (Bebouwing)

Attribute	Domain	Cardinality	Option
Type	Building (Gebouw) Building block (Huizenblok) Tower (Toren) Installation (Installatie)	Single	Mandatory
Function (Functie)	Municipality (Gemeentehuis) Police office (Politiebureau) Post office (Postkantoor) Religious building (Religieus gebouw) Hospital (Hospitaal) Railway station (Treinstation) Subway/fast tram station (Metro/sneltramstation) Station (Station) Water tower (Watertoren) Lighthouse (Vuurtoren) Light tower (Lichttoren) Radio tower (Zendtoren) Windmill (Windmolen) Watermill (Watermolen) Pumping station (Gemaal) Dock (Dok) Greenhouse (Kas) Storage tank (Opslagtank) Crematorium (Crematorium) Manege (Manege) Radar post (Radarpost) Other (Overig)	Single	Mandatory
Height category (Hoog-teklasse)	Low-rise buildings (Laagbouw) High-rise buildings (Hoogbouw)	Single	Mandatory

Height (Hoogte)	< height above ground in meters > (< hoogte boven maaiveld in meters >)	Single	Mandatory
Status	Planned (In ontwerp) Under construction (In aanleg) In use (In gebruik) No longer in use (Niet meer in gebruik)	Single	Mandatory
Name (Naam)	< name building > (<naam gebouw >)	Multiple	Optional
Triangulation (Triangulatie)	Yes (Ja) No (Nee)	Enkelvoudig	Verplicht
Height level (Hoogteniveau)	0 (no level assigned) 1 (lowest level) 2 (lowest level plus 1) 3 (lowest level plus2) <etc>	Single	Optional

Table A-6 Terrain (Terrein)

Attribute	Domain	Cardinality	Option
Land use (Landgebruik)	Deciduous forest (Loofbos) Coniferous forest (Naaldbos) Mixed forest (Gemengd bos) Osier (Griend) Poplar (Populieren) Heath (Heide) Sand (Zand) Arable land (Akkerland) Grassland (Grasland) Orchard (Boomgaard) Fruit nursery (Fruitkwekerij) Tree nursery (Boomkwekerij) Abandoned field (Dodenakker) Railway body (Spoorbaanlichaam) Pavement, basalt (Plaveisel, basaltblokken) Other (Overig)	Single	Mandatory
Physical occurrence (Fysiek voorkomen)	On bank (Op talud) On raised area (Op verhoogd vlak) On lowered area (Op verlaagd vlak) On dike (Op dijk) Cutting (Ingegraven) Covered (Overkluisd) In tunnel (In tunnel) On bridge (Op brug) Other (Overig)	Single	Mandatory
Accessibility (Toegankelijkheid)	Public (Openbaar) Not public (Niet-openbaar)	Single	Mandatory
Occurrence (Voorkomen)	With reed (Met riet) Marsh (Dras, moerassig)	Single	Optional

Name (Naam)	< name terrain > (< naam terrein >)	Multiple	Optional
Height level (Hoog-teniveau)	0 (no level assigned) 1 (lowest level) 2 (lowest level plus 1) 3 (lowest level plus2) <etc>	Single	Optional

Table A-7 Construction (Inrichtingselement)

Attribute	Domain	Cardinality	Option
Type	High tension line (Hoogspanningsleiding) Cable railway (Kabelbaan) Pipe line (Leiding) Poles (Paalwerk) Hedge (Heg, haag) Row of trees (Bomenrij) Power line pole (Hoogspanningmast) Pole (Paal) Border sign (Grenspunt) Fire pipe (Vlampijp) Beacon (Baak) Radio post (Zendmast) signal post (Seinmast) Water level gauge (Peilschaal) Oil-pumping unit (Oliepompinstallatie) Wall (Muur) Hedge (Hekwerk) Monument (Gedenkteken, monument) Bollard (Dukdalf) Road sign (Wegwijzer) Tree (Boom) Megalithic grave (Hunebed) Grave hill (Grafheuvel) (Tol) (Stuw) Road bloack (Wegafsluiting) Crucifix, Chapel (Kruis, Kapel) Little Windmill (Windmolentje) Wind turbine (Windturbine) Pillar (Pijler) Ferry (Veer) Floodgate (Sluisdeur)	Single	Mandatory

	Telescope (Telescoop) Load platform (Laadperron) Landing stage (Aanlegsteiger) Light post (Lichtopstand) Shooting track (Schietbaan) Milestone (Kilometerpaal) Mile post (Kilometerraaipaal) Mile sign (Kilometerraaibord) Cable railway pole (Kabelbaanmast) Wall (Wal) Dike (Dijk) Warf (Boezemkade) Steep edge (Steile rand) GPS triangulation point (GPS kernnetpunt) RD stone (RD steen) Remarkable object (Markant object) Cape (Kaap) Chimney (Schoorsteen) Antenna (Antenne)		
Function (Functie)	Noise barrier (Geluidswerend) Separation (Scheidend)	Single	Optional
Height (Hoogte)	< height above ground level > < hoogte boven maaiveld in meters >	Single	Mandatory
Name (Naam)	<name construction > (< naam inrichtingselement >)	Multiple	Optional
Status	Planned (In ontwerp) Under construction (In aanleg) In use (In gebruik) No longer in use (Niet meer in gebruik)	Single	Mandatory
Height level (Hoog-teniveau)	0 (no level assigned) 1 (lowest level) 2 (lowest level plus 1) 3 (lowest level plus2) <etc>	Single	Optional

Table A-8 Administrative area (Administratief gebied)

Attribute	Domain	Cardinality	Option
Type	Country (Land) Province (Provincie) Municipality (Gemeente) Bundesland (D) Regierungsbezirk (D) Kreis (D)	Single	Mandatory
Name (Naam)	< naam >	Multiple	Mandatory

Table A-9 Conservation area (Beheersgebied)

Attribute	Domain	Cardinality	Option
Type	Nature area, Nature reserve area (Natuur-gebied, natuurreservaat) Forest reserve area (Boswachterij) National park (Nationaal park)	Single	Mandatory
Name (Naam)	< naam >	Multiple	Mandatory

Table A-10 Geographic area (Geografisch gebied)

Attribute	Domain	Cardinality	Option
Type	Region, field (Streek, veld) Hill, dune, mountain (Heuvel, duin, berg) Island (Eiland) Polder Forest area (Bosgebied) Heath area (Heidegebied) Cape (Kaap, hoek) Sea, Arm of the sea (Zee, zeegat, zeearm) Lake, pond, pool (Meer, plas, vijver) Trench, fairway (Geul, vaargeul) Ford (Wad) Bank, shallow, shoal (Bank, ondiepte, plaat) City, town, populated place (Plaats, be-woond oord) District, neighboorhoud (Wijk, buurt) Other (Overig)	Single	Mandatory
Name (Naam)	< naam >	Multiple	Mandatory

Table A-11 Functional area (Functioneel gebied)

Attribute	Domain	Cardinality	Option
Type	Industrial area (Industriegebied) Military terrain, firing range (Militair oefengebied, schietterrein) Cemetery (Begraafplaats) Sports field (Sportterrein) Sports field Sportterrein Airport, airfield (Vliegveld, luchthaven) Bungalow park (Bungalowpark) Camping Recreation area (Recreatiegebied) Swimming pool (Zwembad) Golf terrain (Golfterrein) Lock complex (Sluizencomplex) Building complex (Gebouwencomplex) Yacht basin (Jachthaven) Harbour (Haven) Zoo, Safari park (Dierentuin, safaripark) Cross track (Crossbaan)	Single	Mandatory
Name (Naam)	< naam >	Multiple	Mandatory