29

Principles of spatial database analysis and design

Y BÉDARD

This chapter covers the fundamentals of spatial database analysis and design. It begins by defining the most important concepts: 'spatial database', 'analysis', 'design', and 'model'; and continues with a presentation of the rationale supporting the use of formal methods for analysis and design. The basic elements and approaches of such methods are described, in addition to the processes used. Emphasis is placed on the particularities of spatial databases and the improvements needed for non-spatial methods and tools in order to enhance their efficiency. Finally, the chapter presents a set of tools, called CASE (computer-assisted software engineering), which are built to support the formal analysis and design methods.

1 INTRODUCTION

This chapter examines the principles of spatial database analysis and design. These two critical phases of system development remain informal in most small GIS projects. However, when one needs to build a large system, to facilitate its maintenance or to work with a team of information technology specialists, formal engineering-like methods must be adopted.

Formal methods for database analysis and design have been developed to master complex problems. They provide fundamental principles and well-defined steps aimed at improving the efficiency of the database development process and the quality of the result. These methods have existed since the 1970s and rely heavily on models and dictionaries. They have been supported for more than ten years by computer assisted software engineering (CASE) tools which facilitate the building of these models and dictionaries. Although a growing number of GIS specialists master formal methods, they rarely use CASE tools to create their spatial database schema and dictionary or to generate GIS code. This is partly because existing methods and tools must be extended to become truly effective with spatial databases.

The following sections cover the fundamentals of spatial database analysis and design. After defining basic concepts, the rationale of formal analysis and design is presented. This is followed by a discussion of the methods used, and by a description of the process to follow for spatial databases. Characteristics of spatial databases are discussed along with examples. Finally, CASE tools are presented and discussed in a spatial database context.

2 BASIC DEFINITIONS AND CONCEPTS

For the purpose of this chapter, 'spatial database' refers to any set of data describing the semantic and spatial properties of real world phenomena (temporal properties are also possible). Such spatial databases can be implemented in a GIS, in a computer-assisted design (CAD) system coupled with a database management system (DBMS), in a spatial engine accessed through an application programming interface (API), and sitting on top of a DBMS, in a universal (object-relational) server with spatial extension, in a web server with spatial viewer, etc. These spatial databases can use flat file, hierarchical, network, relational, object-oriented, multidimensional or hybrid structures. In addition, they can be organised in very diverse architectures such as stand-alone GIS, client-server solutions, intranets or spatial data warehouses. Gone are the days of a spatial database implemented solely on a stand-alone GIS.

Many definitions exist with regards to 'analysis' and 'design', sometimes in contradiction with each other, sometimes very specific to a full life-cycle system development method (see Carmichael 1995 and Olle 1991 for a comparison of methods). This chapter uses the most common and intuitive definitions of analysis and design, but also recognises the fuzziness of the distinction between them (Jacobson and Christerson 1995). In a spatial database context, analysis is the action of understanding and describing what the users need for their spatial database. Thus, it results in a formal and detailed database requirements specification. Similarly, design is the action of defining and describing how the analysis result will be implemented in the selected technology. It is where we consider practical issues such as the limitations of the technology used to manage the spatial database, the desired performance and flexibility, the implementation of security requirements, etc. Design, therefore, results in a formal and detailed programming specification.

Another fundamental definition is that of 'models' since formal models are the thinking tools as well as the final results of database analysis and design. In this context, models are formal representations of something that needs to be understood, remembered, communicated, and tested; they are purposeful surrogates built at a given level of abstraction to include only what is relevant to the system being developed. When a model represents how users' reality is organised in terms of objects, properties, relationships, and processes, then it is described as an 'analysis model', and represents what users want to be implemented in their spatial databases. The analysis model is also called the business model, conceptual model, user's model, and sometimes specification model. When a model represents the database internal structure and related processes as implemented, then it is described as a 'design model'. This latter is also called the implementation model, internal model, or physical model, although the level of detail may vary.

3 THE RATIONALE FOR FORMAL ANALYSIS AND DESIGN METHODS

For any database, analysis and design models determine what can be done easily, with difficulty, or not at all, once the system has been implemented. However, the impact of bad models appears to be higher for spatial databases than it is for non-spatial databases. Consequently, and especially when considering the high cost of spatial data and the long delays in acquiring them, an organisation's return on investment is very sensitive to good analysis and design.

Using a formal method to complete such tasks provides guidance, supports the thinking process and encourages consistent communication and documentation. There are several such methods. The most recent are based on the object-oriented paradigm (Worboys, Chapter 26; Booch 1994; Coad and Yourdon 1991a, 1991b; Cook and Daniels 1994a; Jacobson et al 1993; Martin and Odell 1993; Rumbaugh et al 1991; Shlaer and Mellor 1991). Although these methods have been created to support any type of software development and built to support object-oriented (OO) programming, they can be used efficiently for database development. They all rely on solid theoretical concepts and have all been tested over and over again so they have acquired formal rigour and proved their utility. Once a formal method is mastered, it is faster and better results are delivered, especially when supported by CASE tools (see section 5). Mastering a method also helps developers to solve the most important problems before computerisation; the sooner the problems are solved, the less expensive they are to solve. Finally, good documentation facilitates maintenance (e.g. adding new data types and new processes, migrating to new equipment) as well as software reuse; it also frees the system from its dependency upon individuals. It has been recognised for several years that the higher cost of such a formal approach is lower than the continuous hidden costs of chaotic data and processes. Figure 1 illustrates the impact of good analysis and design methods on the efforts to build and maintain a spatial database.

Use of formal analysis and design methods is also being pushed by the increasing complexity of the spatial database development process. The recent evolution of the software industry indicates that spatial databases are becoming mainstream solutions seamlessly integrated with non-spatial corporate data. This is happening more and more, with new categories of tools outside the traditional GIS packages. In comparison to non-spatial databases, these solutions offer a higher level of diversity both within and across categories. The complexity of spatial objects is also inherently higher; issues such as geometry, spatial reference systems, movement, spatial precision, spatial

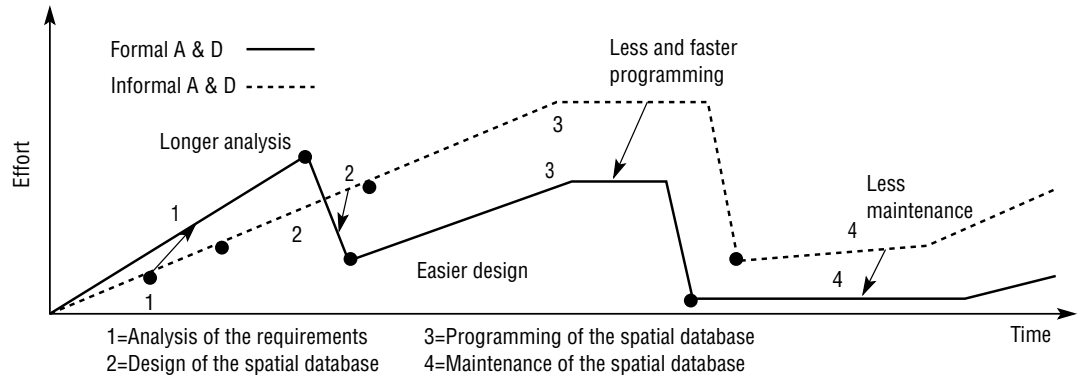

Fig 1. Impact of formal analysis and design methods on efforts to build and maintain a spatial database.

integration, metadata management, database versioning, data quality analysis, and so on may have a tremendous impact when designing a database for spatial querying, spatial analysis, spatial data exchange, and system interoperability (Oosterom, Chapter 27). A look at the ISO SQL3/MM spatial standard (ISO 1996a), the ISO TC211 document *Geographic Information – Rules for Application Schema* (ISO 1996b), and the *Open GIS Virtual Geodata Model* (OGC 1996) rapidly convinces the reader of this inherent complexity of spatial data (see Sondheim et al, Chapter 24). Such complexity adds to the omnipresent need for very high performance, a need which encourages database structures which are denormalised and difficult to understand. Thus, spatial database analysis and design must rely more than ever on formal methods to cope efficiently with such levels of complexity, especially in large projects. Consequently, spatial database developers have a higher need to split the problem between analysis and design and to deliver separate models. According to previous definitions, analysis focuses on real-world issues independent of the technology, while design focuses and depends completely on the technology selected. Such a separation helps to better understand users' needs, to structure the database, to facilitate maintenance, and to encourage metadata management and software reuse. This split is essential for multi-platform environments and systems interoperability – one of today's most challenging issues for spatial databases (Sondheim et al, Chapter 24). Such a 'divide and conquer' strategy has been used for over 20 years in database design (ANSI/SPARC 1975). It is clearly defined in most formal methods, including OO methods, in spite of a different claim in their first years. (The initial claim stated that there is a perfect one-to-one mapping between the objects of the analysis model and the objects implemented with OO programming tools. However, with real and large projects, it became obvious that the 'analysis-to-design' translation was not straightforward, even with the best OO systems. In addition, most of today's commercial database technologies still rely on the relational approach, with or without OO extensions, thus offering only limited and indirect capability to support OO concepts; this is especially true for spatial databases.) As clearly stated by Cook and Daniels (1994b), 'an important question is the extent to which the activities of analysis and design can be merged. The simplistic approach is to say that object-oriented development is a process requiring no transformations, beginning with the construction of an object model and progressing seamlessly into object-oriented code . . . While superficially appealing, this approach is seriously flawed. It should be clear to anyone that models of the world are completely different from models of software.'

Today's practice is to use at least two levels of models, separating the 'what' from the 'how' and leading to more robust and reusable results. Depending on the formal method being used, these levels are based either on different modelling techniques (e.g. entity/relationship schema for the conceptual level, relational schema for the logical level, structured query language, SQL, code for the physical level) or on the same technique (e.g. the additive approach where more details are introduced while going from the analysis level to the implementation level). Batini et al (1992) offer an excellent description of the common three-levels approach used with the entity/relationship

paradigm. Cook and Daniels (1994b) explain their three levels in an OO paradigm. Rumbaugh (1996) clearly presents the layered additive approach common in the OO community (Worboys, Chapter 26). When applied correctly, the multi-level approach is quite powerful since it allows developers to work at different levels of abstraction for different purposes. Integrating all the above mentioned issues into only one model makes the work much less efficient and the result less reusable. Consequently the GIS community is also embracing the multi-level approach in major efforts such as the OGIS Object Model (OGC 1996) and the ISO-TC211 rules for application schema (ISO 1996b).

While doing analysis and design, spatial database developers benefit from a method with a higher power of expression than traditional methods. Research in recent years has made high-level modelling more efficient and semantically complete, including for spatial databases (e.g. Bédard et al 1996; Caron and Bédard 1993; Golay 1992; ISO 1996b; Pantazis and Donnay 1996). However, spatial database analysts still need to improve their modelling techniques to depict better the spatial and temporal properties of geographic features (e.g. Figure 2) and to include geometric considerations better. This is needed to help the developer to convince the user that he or she understands how every piece of information is semantically and geometrically defined and related to the others, how it is used, what are the allowed values, where it comes from, how reliable it is, etc. Similarly, spatial database designers must convince users that they have created a solution offering the best way to map every element of the analysis model into a piece of code supported by the client's technology. To do so

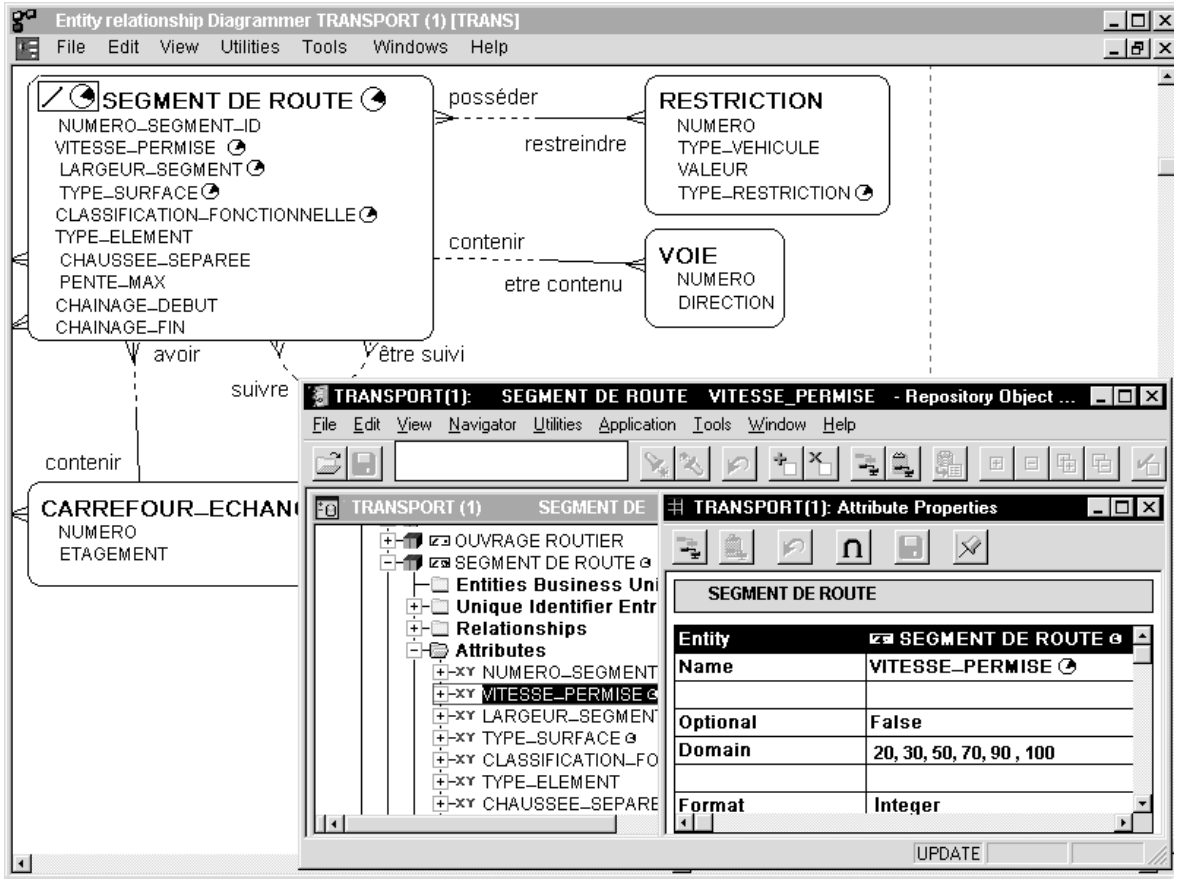

Fig 2. Extract of an analysis model made with Oracle Designer 2000 (Oracle Corporation 1996) where spatial and temporal pictograms (see Figures 4 and 5) are added to indicate the geometry and temporality of geographical features (model in development for Québec Ministry of Transportation at the time of writing the present chapter).

efficiently and to facilitate communication, especially in the multipurpose context of many spatial databases, one must extend the traditional methods, as explained in the next section.

Finally, database developers rely more and more on CASE packages. The arrival of such tools has been a big incentive for databases developers to embrace formal methods, especially when the tools automatically generate code from the models and vice versa (cf. reverse engineering). In the GIS arena, a similar movement has started. For example, there already exist graphical schema builders with an integrated data dictionary for the design of the spatial database (e.g. Intergraph MGE). However, we need to go further and to automate this process right from the analysis model. Although no complete tool exists for spatial databases outside university laboratories (e.g. Orion, developed in 1992 by the author), we can expect that the present international standardisation efforts and the strong demand for GIS interoperability will push the development of such extended methods and tools.

4 FORMAL METHODS FOR SPATIAL DATABASE ANALYSIS AND DESIGN

A formal analysis and design method is a set of guidelines and rules to capture the semantics of users' reality and to build a spatial database supporting it. It is used for thinking, documenting, and communicating in a consistent and coherent manner via models. Thus, modelling is the foundation of analysis and design. Any model is built out of a deliberately limited but sufficiently powerful and crisply defined set of constructs. These constructs, along with a simple notation and a small set of rules, constitute a 'formal language' (also called a formalism). Such a formalism can have a textual notation, a graphical notation, or a mix of the two. Human cognitive research and psychology have shown that graphical languages are more efficient for synthetic views and textual languages for detailed descriptions (see Figure 2 which shows both graphical and textual notations). Cognitive sciences have also shown that combining both graphical and textual languages is necessary to achieve clear understanding. This fact is recognised by formal methods, since they offer graphical notations to create, present, validate, and manipulate models and use textual details in dictionaries and programming code. However, the graphical notations are the most visible part of a method and may mistakenly be thought of as *the* method. This is misleading since graphical notations only reflect a part of the underlying constructs proposed by a method.

The basic constructs are very similar in nature across formal methods of the same type but differ among types (relational, entity relationship, and object oriented). The relational approach relies on one basic construct called a relation which is a table of columns (attributes) and rows (occurrences of a phenomenon) manipulated with a relational algebra. The elegance of the relational approach lies with its simplicity, while its popularity relies on the fact that most commercial DBMS have implemented the relational structure. However, it is widely known that 'the relational model is limited with respect to semantic content (i.e. expressive power) and there are many design problems which are not naturally expressible in terms of relations. Spatial systems are a case where the limitations become clear.' (Worboys et al 1990)

The entity/relationship approach utilises more constructs, such as 'entity', 'attribute', and 'relationship'. This provides a better expressive power; however, few DBMS support the entity/relationship structure. Also, an entity/relationship schema must be translated into a relational schema to be implemented in a relational DBMS. Worboys et al (1990) mention that 'many systems may be modelled using entities, attributes and relationships, including systems with a dominating spatial component … However, experience has shown that for many systems the initial set of modelling constructs (entity, attribute, and relationship) is inadequate'. In fact, the last few years have witnessed the addition of several extensions to entity/relationship constructs, including aggregation, generalisation, geometry, and temporality (e.g. Modul-R: Bédard et al 1996; Caron and Bédard 1993).

Finally, the OO approach relies on: (1) 'objects' encompassing 'properties' (or attributes) with the 'operations' modifying data (also called methods and procedures); (2) on 'relationships' between objects; (3) on aggregation of objects into more complex objects; and (4) on generalisation or specialisation of the types of objects to more general or more specific types, respectively. The OO approach also uses 'states', 'events', and 'messages' to show the behaviour of objects. Such an integrated description leads to richer database analysis and design (see the Unified Modelling Language for the

most recent and robust constructs: Booch et al 1996; Worboys 1995). It is undoubtedly the most powerful modelling paradigm nowadays. Part of the success of the OO approach lies in the fact that 'the object-oriented paradigm, which originated from programming languages, has been successfully applied to the analysis and design and even earlier phases of system development' (Magrogan et al 1996).

Like entity/relationship, it is possible to extend OO methods with spatial and temporal 'plug-ins' to increase efficiency (Figure 3). The new constructs must include abstraction mechanisms powerful enough to facilitate the analysis phase. At the design phase they must also map to the built-in proprietary components of GIS, CAD, universal servers, and similar tools on the market. It must be remembered that these generic structures and operators deal with geometric primitives, graphic properties, spatial reference systems, topological relationships, spatial operators, and so on which do not exist in traditional DBMS.

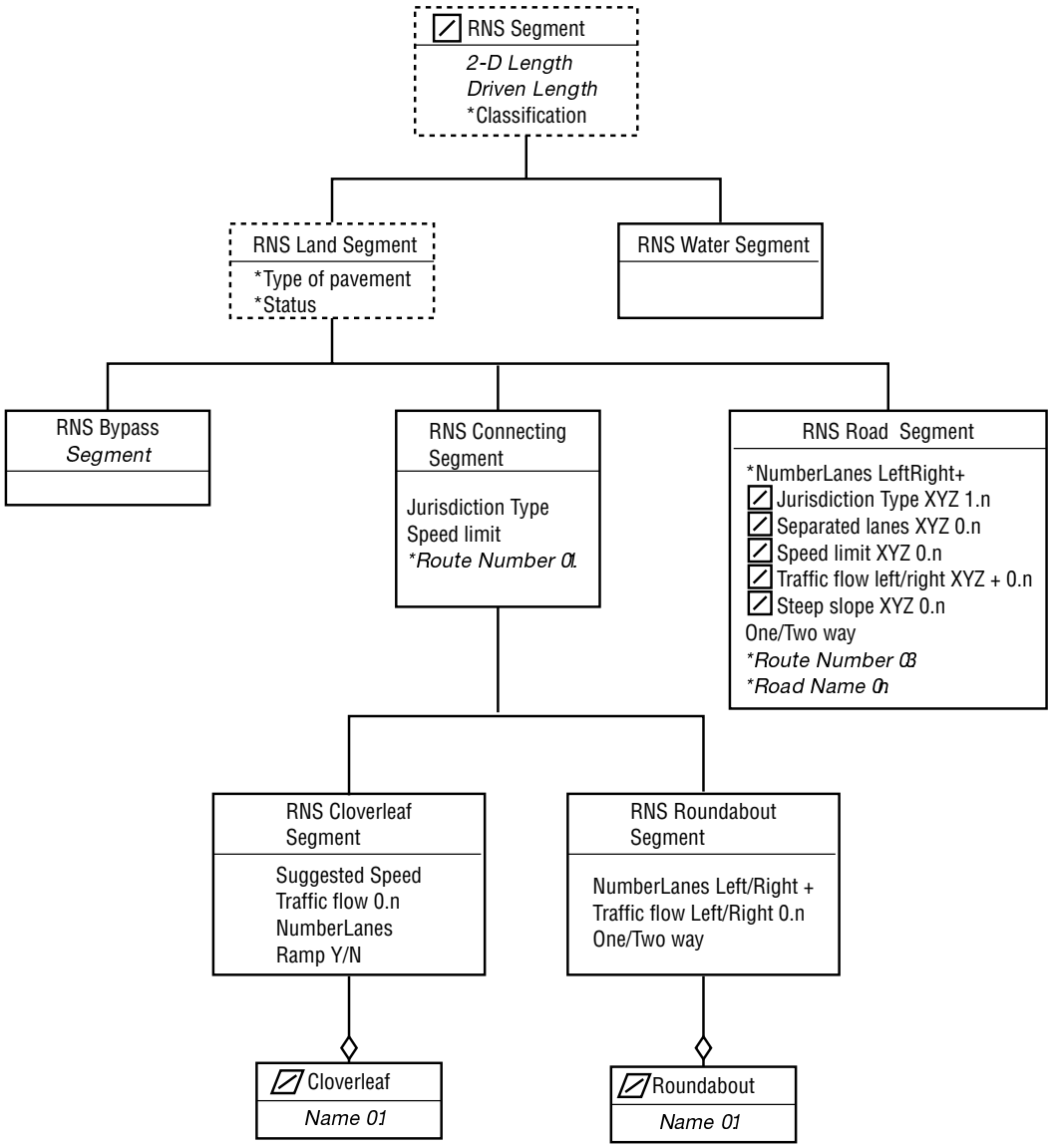

Fig 3. Extract of an object model using spatial 'plug-ins' (Bédard and Sondheim 1996); see Figure 4 for explanation of pictograms.

While at the design level the offered primitives must be considered, at the analysis level these complexities are hidden from the user; only the general geometric information is relevant (e.g. does the user want to have this type of object represented on the map? using which type of shape?). In fact, during the analysis phase, users do not have to deal with the intricacies of points vs nodes vs vertices, lines vs arcs vs polylines, metric vs topology, and so on; they should only deal with houses, lots, streets, and similar concepts of interest. Accordingly, methods should be extended with the proper geometric constructs and coding rules. Their availability in commercial CASE tools will improve the efficiency of analysis and design for spatial databases.

All of the constructs mentioned above result from the fundamental abstraction concepts used by humans to understand the world where they live: classification, association, aggregation, and generalisation. These are used by the system analyst who tries to understand the users' perceptions of their reality (e.g. the types of objects they deal with, their properties, how they relate to each other, how they are geometrically represented, how they behave, how they are used to provide new information, how they are spatially related). These are also the abstraction mechanisms used by the system designer to build efficient programming code on the selected spatial database technology. The process used when applying these abstraction mechanisms is a subtle one. It calls for creativity as well as observation and rigour. When a formal method is used it adds an engineering-like rationality to a work of art. The next paragraphs explain some technical elements of the analysis and design processes for spatial databases.

To start the analysis phase, the different types of features which are of interest to the users (e.g. house, street, owner, contract) must be identified; these features may or may not exist in their present systems. Then the semantic properties and identifiers of these features (e.g. the value, style, and address of the feature 'house') should be selected. In the case of geographical features, the types of geometry must be identified as well as a spatial reference system. Bédard et al (1996) present such geometries. They combine a dimensional pattern (0-dimensional, 1-dimensional, 2-dimensional, 3-dimensional) with a composition pattern (simple, complex, alternate, multiple). More specifically, the 'simple' pattern is used when a geographic feature is geometrically represented by only one occurrence of a given dimension (e.g. a park represented by a single polygon, i.e. a 2-dimensional primitive); this is the most frequent situation. The 'complex' pattern indicates geometric aggregations (e.g. hydrographic networks made of 1-dimensional rivers and 2-dimensional lakes). The 'alternative' pattern indicates mutually exclusive geometries (e.g. parks digitised as points if smaller than 500 square metres or as polygons if larger). Finally, the 'multiple' pattern is the rarest of all. It indicates that more than one shape must be digitised for each occurrence of a geographical feature; this happens when the desired shapes cannot be deduced from each other. Consider, for example, the feature 'city' which is represented by a polygon on certain maps and by a point located downtown on other maps; the point cannot be derived from the polygon and thus requires its own digitising. Figure 4 shows the graphical notation used for these patterns in the Modul-R method while Figure 2 illustrates their use in a model.

For spatio-temporal databases (e.g. temporal GIS), the types of temporality needed for each type of feature must be added. These temporalities follow the same logic as the spatial patterns, i.e. dimensional pattern (instantaneous 0-dimensional, durable 1-dimensional) and composition pattern (simple, complex, alternate, multiple). These patterns apply to the existence of the feature, its presence, its functionality, and its evolution (semantic and geometric). (See Figures 2 and 5.)

Once the desired features are defined with their semantic, geometric and temporal properties, the analyst must identify the relationships of interest between these features. These include semantic relationships (e.g. house 'is owned by' owner) as well as semantically significant spatial and temporal relationships (e.g. house 'is on' lot). Integrity

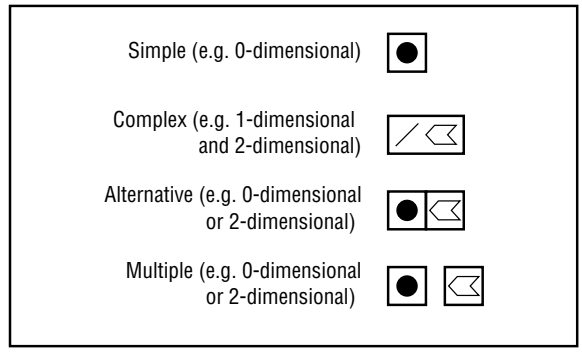

Fig 4. Graphical notation used for spatial patterns.

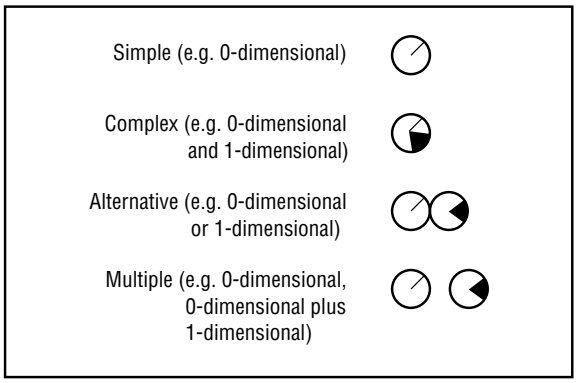

Fig 5. Graphical notation used for temporal patterns; an example of a spatio-temporal pattern is also presented, that is one of a 0-dimensional object (e.g. an emergency vehicle) with a position which sometimes varies continuously and sometimes remains stable.

constraints must also be specified as they restrict the content of the spatial database. There are attribute rules (e.g. lists of values for nominal attributes, ranges for numeric attributes), inter-attribute rules (e.g. building.area smaller than lot.area), and inter-object rules (e.g. lot existed before building). The latter include the cardinalities of relationships (e.g. houses built on only one lot, but a lot may have 0 to N houses).

All along this process the analyst must build a schema and dictionary which are sufficiently complete for the programmers while keeping the result understandable to the users. Such a compromise between two contrasting objectives can be accomplished through the building of views, that is, exact or modified-but-compatible subsets of the global model and dictionary. Views also help to divide the problem into smaller and more manageable parts. The analyst must also decide the level of detail for the model and dictionary:

'it is not always possible or desirable to capture every nuance and restriction in a model: similarly, there is always a danger in producing a convoluted model that captures every small detail at the expense of general understandability. As a basic principle, therefore, we follow the maxim: "If you must choose between undermodelling and overmodelling, choose the undermodel and add textual commentary".' (Booch et al 1996)

The analyst verifies the logic, coherence, and completeness of his or her model, dictionary, and views. At this point the model is normalised if the approach is relational or entity/relational. Normalisation is used to eliminate data redundancies and dependencies, to facilitate the maintenance of the integrity of the database, and to build application-independent structures. If the analyst works with an extended entity/relationship or an OO approach, then objects can be generalised with common properties and relationships to create supertypes. In the case of OO methods, operations may be added and the dynamic models may be built. The last verifications are made with additional interviews and site visits, with formal walk-throughs with the users, or with testing scenarios.

With regard to the design phase, one needs new basic constructs. These constructs vary widely among spatial database management systems, since they depend on their primitives. For example, relational systems require foreign keys to materialise relationships, object-oriented systems require messages to activate operations, and commercial GIS require specific links between geometric primitives and semantic objects. As stated by Günther and Lamberts (1994), 'for the geometric representation there exists a large variety of spatial data structures that each support a certain class of spatial operators'. It may also be required to optimise the database structure with careful denormalisations of relational tables or with proper fusion and separation of objects. In particular, denormalisation is a very popular technique for spatial databases because of the large amount of spatial data handled for every single request, coupled with the need for very high performance (Egenhofer and Frank 1992). This is especially true when one wants to accelerate spatial analysis. However, alternative ways should be investigated to improve the performance of the system, either through better programming of queries and procedures or through better indexing, clustering, and buffer management.

To facilitate the preceding steps of analysis and design, different techniques may be used, like building throw-away and evolved prototypes to validate user requirements. One may also include 'use cases' or 'scenarios' (Jacobson and Christerson 1995) and 'CRC Cards' (Wilkinson 1995) which detail how users interact with a database. Finally, database 'patterns' offer well-documented and tested solutions to common problems (Coplien 1996; Fowler 1995; Gamma et al 1995).

In spite of such aids, certain steps of analysis and design lead to revisiting the models, redefining objects, adding attributes, etc. Such trips back happen naturally because the more we advance into the details, the more we understand the subtleties. Thus, analysis and design are not clear-cut sequential processes, but rather are incremental and iterative processes. In the past, most methods and CASE tools forced the developer to create very definite breaks between phases, and going back was difficult. Nowadays, with CASE tools offering reverse engineering, and OO methods suggesting additive models, it has become natural to iterate. In spite of this iterative nature and the resulting fuzzy boundary between the analysis and design processes, there remain two clearly distinct results: the technology-independent analysis model describing the users' spatial reality as understood once the project is complete, and the design model describing the spatial database as developed on the selected technology.

5 SOFTWARE SUPPORTING SPATIAL DATABASE ANALYSIS AND DESIGN

In the first years of formal methods, all schemas and dictionaries were made by hand. As a result, the drawing and editing process took more time than the thinking process, and manually keeping the coherence among evolving models, views, and dictionary proved to be an almost impossible task. This problem was solved by the new category of software created in the mid-1980s, namely CASE. These packages offered drawing, dictionary, checking, and reporting functions which accelerated the creation and modification of the schemas, views, and reports suggested by formal methods. Using such tools greatly increased productivity, especially when one CASE could import and export results to another CASE used in the preceding or subsequent step (Bédard and Larrivée 1992). According to Roux (1991), 'the objectives of CASE tools are, in order of importance: to reduce maintenance considerably, to provide real quality, to speed up delivery, to develop at less cost' (author's translation). Figure 6 illustrates the impact of CASE tools on formal methods of database development.

The first CASE tools copied what was done by hand and specialised in one type of schema or in one task of the development process (e.g. an entity/relationship tool, a data flow diagram drawer, a screen painter). Most of them were not compatible with other CASE tools automating the preceding and following tasks, resulting in limited productivity gains. Nowadays, most CASE tools support all the schemas, views, and reports suggested by a formal method, and the result of a task situated early in the development process can be used by another task situated later in the process. This is done via a common dictionary (also called encyclopaedia, repository, or information resources dictionary system) which maintains the coherence among models.

As opposed to diagramming, drawing, and CAD packages such as Visio, Corel Draw and AutoCAD respectively, CASE tools store the meaning of each construct of a method and of each piece of a model (e.g. objects, properties, relations). They use this intelligence to enforce the rules of a method and to control the behaviour of the constructs. For example, a CASE tool may refuse an object in the dictionary which is not depicted in a schema, or may automatically fill parts of the dictionary from a schema. When an object is moved in a diagram, the software keeps intact its relationships with the other objects. When one tries to draw a relationship

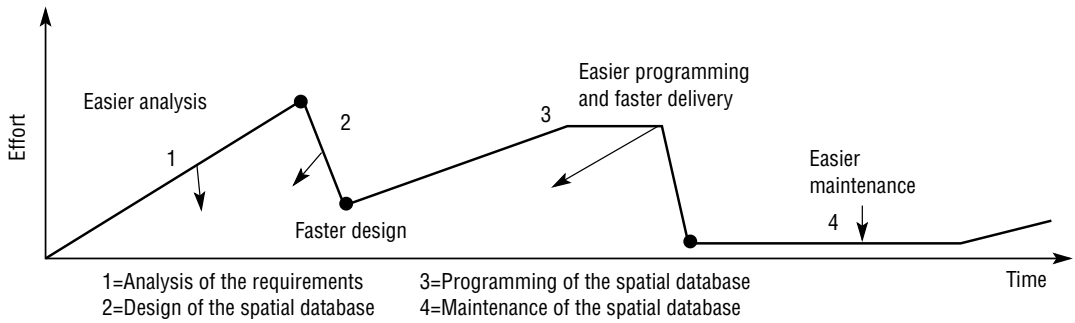

Fig 6. Impact of CASE tools on formal methods of spatial database analysis and design.

between two other relationships, the software refuses this operation if it is not allowed by the method. When one uses the same name for different objects, the software requires a change, and so on. Products such as ERWin, S-Designor, Sylverrun, IEF, Designer2000, OMTool, and Rational Rose are a few examples of intelligent CASE tools covering the entire development and maintenance cycle. Like GIS, some CASE tools cost thousands of dollars while others cost a few hundreds, and the latest trend is to embed limited CASE engines within programming environments (e.g. Delphi, Visual Basic).

In the case of spatial databases, present commercial CASE tools can be used. However, extensions for spatial constructs, rules, and code generation are needed. Therefore, spatial database developers must turn towards a more advanced category of CASE tools: metaCASE tools. These are specialised development packages working at the metamodel level; they can be programmed to accept the new constructs and rules of a new method and afterwards to run like a CASE tool for that method. Several large organisations rely on such a solution to have a CASE tool adapted to their proprietary method. This was used to build our spatial CASE tool called Orion (Figure 7). ObjectMaker and Paradigm Plus are such products. In comparison to traditional CASE tools, they offer a large number of methods. Some recent CASE tools offer similar but limited extension capabilities. MetaCASE and extensible CASE products represent the best choice for spatial databases.

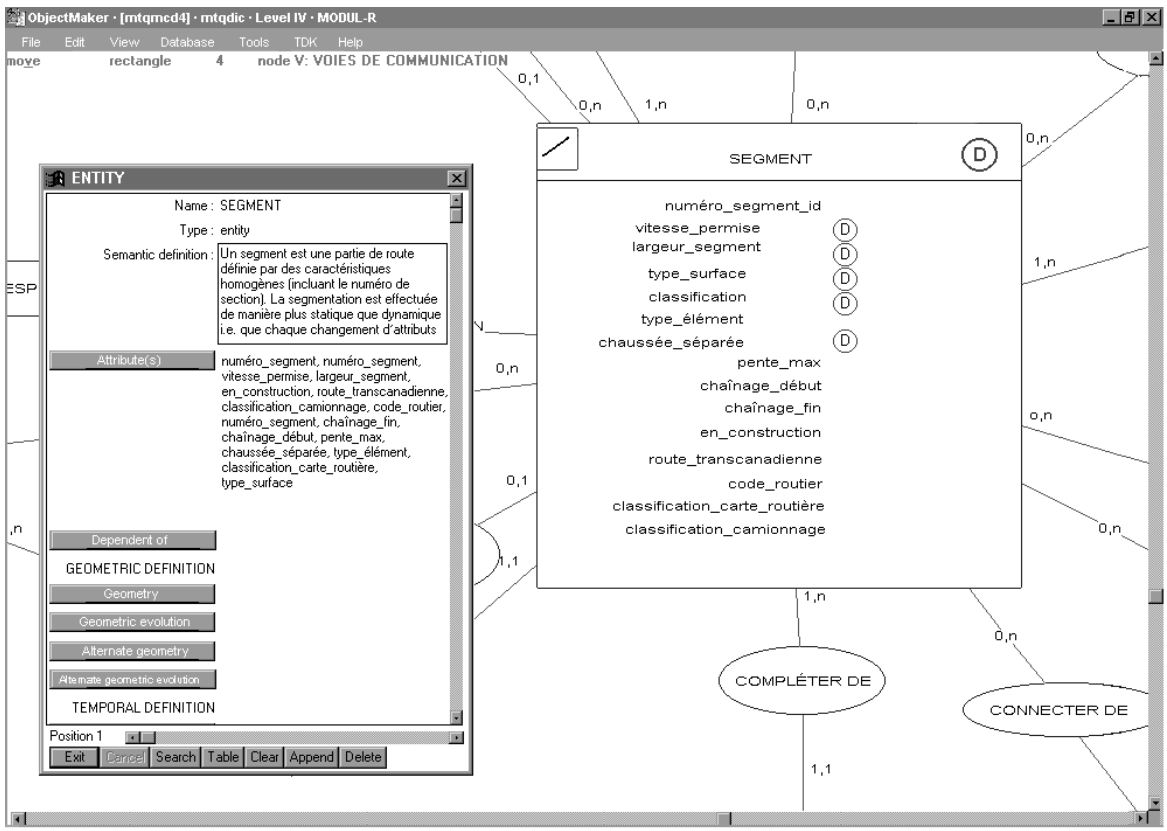

Fig 7. Example of a screen display of the working prototype Orion built with the Object Maker metaCASE; it shows an extended entity/relationship analysis model with spatial and temporal pictograms, as well as an extract of the extended data dictionary which is partly filled automatically from the schema; it is used for automatic code generation for Intergraph GIS.

6 CONCLUSION

This chapter has presented the fundamental elements of spatial database analysis and design. Good analysis and design rely on formal methods. Such methods help us to work in a more rigorous manner directed by principles, techniques, and guidelines. This is particularly important for the analysis phase: 'although it represents only a small proportion of the total development effort, its impact on the final system is probably greater than any other phase . . . Analysts estimate that a change that costs $1 to fix in the requirements stage will cost $10 in design, $100 in construction, and $1000 in implementation!' (Moody 1996).

Formal methods rely on specific constructs, models, and processes like the ones presented in this chapter. The most recent methods are based on the object-oriented paradigm, and they are the most powerful. Like other methods, they are supported by CASE tools to facilitate the building of models, dictionaries, documentation, and maintenance. Present methods and tools may already be used for the analysis and design of spatial databases. However, recent research and standardisation efforts will help to adapt these methods and tools to spatial information technology. This should further encourage the development of spatial databases which are robust and flexible, faster to build, easier to maintain, and closer to interoperability. Finally, this will help GIS developers to enter the mainstream of information technologies which is a natural evolution.

References

ANSI/SPARC 1975 ANSI/X3/SPARC Study Group on data base management systems, interim report. *ACM SIGFIDET* 7: 3–139

Batini C, Ceri S, Navathe S B 1992 *Conceptual database design, an entity–relationship approach*. Redwood City, Benjamin Cummings

Bédard Y, Caron C, Maamar Z, Moulin B, Vallière D 1996 Adapting data models for the design of spatio-temporal databases. *Computers, Environment, and Urban Systems* 20: 19–41

Bédard Y, Larrivée S 1992 Développement des systèmes d'information à référence spatiale: vers l'utilisation d'ateliers de génie logiciel. *Journal of the Canadian Institute of Surveying and Mapping* 46: 423–33

Bédard Y, Sondheim M 1996 *Road Network System data model. Technical report*. Geographic Data. *http://www.env.gov.bc.ca/gdbc/rns*

Booch G 1994 *Object-oriented analysis and design with applications*, 2nd edition. Redwood City, Benjamin Cummings

Booch G, Rumbaugh J, Jacobson I 1996 *The Unified Modelling Language for object-oriented development, documentation set version 0.9 addendum*. Santa-Clara, Rational Software Corporation. *hhtp://www.rational.com/ot./uml.html*

Carmichael A (ed.) 1995 *Object development methods*. New York, SIGS Books

Caron C, Bédard Y 1993 Extending the individual formalism for a more complete modelling of urban spatially referenced data. *Computers, Environment, and Urban Systems* 17: 337–46

Coad P, Yourdon E 1991a *Object-oriented analysis*, 2nd edition. Englewood Cliffs, Prentice-Hall

Coad P, Yourdon E 1991a *Object-oriented design*. Englewood Cliffs, Prentice-Hall

Cook S, Daniels J 1994a *Designing object systems: object-oriented modelling with Syntropy*. Englewood Cliffs, Prentice-Hall

Cook S, Daniels J 1994b Software isn't the real world. *Journal of Object-Oriented Programming* (May): 2–28

Coplien J O 1996 *Software patterns*. New York, SIGS Books

Egenhofer M J, Frank A U 1992 Object-oriented modelling for GIS. *Journal of the Urban and Regional Information Systems Association*: 3–19

Fowler M 1995 *Analysis patterns: reusable object models*. Reading (USA), Addison-Wesley

Gamma E, Helm R, Johnson R, Vlissides J 1995 *Design patterns: elements of reusable object-oriented software*. Reading (USA), Addison-Wesley

Golay F 1992 'Modélisation des systèmes d'information à référence spatiale et de leurs domaines d'utilisation spécialisés, aspects méthodologiques, organisationnels et technologiques'. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland

Günther O, Lamberts J 1994 Object-oriented techniques for the management of geographic and environmental data. *The Computer Journal* 37: 16–25

ISO 1996a *SQL multimedia and application packages (SQL/MM) part 3: spatial*. SQL/MM: MAD-005. International Organisation for Standardisation

ISO 1996b *Geographic information – rules for application schema*. ISO 15046-9. ISO TC211: WG2 N030 Geographic information-Geomatics, Working group 2. International Organisation for Standardisation

Jacobson I, Christerson M 1995 Modelling with use cases: a confused world of OOA and OOD. *Journal of Object-Oriented Programming* (September): 15–20

Jacobson I, Christerson M, Jonsson P, Overgaard G 1993 *Object-oriented software engineering*. Reading (USA), Addison-Wesley

Magrogan P J, Schardt J A, Chronoles M J 1996 Object-oriented conceptualisation. Report on object analysis and design. *Journal of Object-Oriented Programming* (September): 54–63

Martin J, Odell J J 1993 *Principles of object-oriented analysis and design*. Englewood Cliffs, Prentice-Hall

Moody D 1996 The seven habits of highly effective data modellers. *Database Programming and Design* (October): 57–64

OGC (Open GIS Consortium) 1996c *The Open GIS abstract specification. http.//www.openings.org/public/abstract.html*

Olle T W 1991 *Information systems methodologies*, 2nd edition. Reading (USA), Addison-Wesley

Oracle Corporation 1996 *Oracle Designer/2000, Release 1.3*

Pantazis D, Donnay J P 1996 *La conception de SIG, méthode et formalisme*. Paris, Hermès

Roux F G 1991 Comment le génie vient au logiciel. *L'informatique professionnelle* 93(April): 19–26

Rumbaugh J 1996 Layered additive models: design as a process of recording decisions. *Journal of Object-Oriented Programming* (March–April): 21–48

Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W 1991 *Object-oriented modelling and design*. Englewood Cliffs, Prentice-Hall

Shlaer S, Mellor S 1991 *Object lifecycles: modelling the world in states*. Englewood Cliffs, Prentice-Hall

Wilkinson N 1995 *Using CRC cards: an informal approach to object-oriented development*. New York, SIGS Books

Worboys M F 1995 *GIS: a computing perspective*. London, Taylor and Francis

Worboys M F, Hearnshaw H, Maguire D J 1990 Object-oriented data modelling for spatial databases. *International Journal of Geographical Information Systems* 4: 369–8