A SYSTEM FOR CHANGE DOCUMENTATION BASED ON A SPATIOTEMPORAL DATABASE

M. A. Peerbocus[1], C. Bauzer Medeiros[2], G. Jomier[1], A. Voisard[3]

1 LAMSADE
University Paris Dauphine
Pl du Mal de Lattre de Tassigny
75775 Paris Cedex 16
France

2 IC – UNICAMP
CP 6176
13081-970 Campinas SP
Brazil

3 Computer Science Institute
Freie Universitat Berlin
14195 Berlin
Germany

Corresponding author
Claudia Bauzer Medeiros
IC – UNICAMP – CP 6176
13081-970
Campinas SP Brazil
cmbm@ic.unicamp.br

Abstract

The evolution of geographic phenomena has been one of the concerns of spatiotemporal database research. However, in a large spectrum of geographical applications, users need more than a mere representation of data evolution. For instance, in urban management applications - e.g. cadastral evolution - users often need to know *why*, *how*, and *by whom* certain changes have been performed as well as their possible impact on the environment. Answers to such queries are not possible unless supplementary information concerning real world events is associated with the corresponding changes in the database and is managed efficiently. This paper proposes a solution to this problem, which is based on extending a spatiotemporal database with a mechanism for managing documentation on the evolution of geographic information. This solution has been implemented in a GIS-based prototype, which is also discussed in the paper.

Keywords – Spatiotemporal evolution; database versions; dynamic documentation; urban management documentation

A SYSTEM FOR CHANGE DOCUMENTATION BASED ON A SPATIOTEMPORAL DATABASE

M. A. Peerbocus, C. Bauzer Medeiros, G. Jomier, A. Voisard

Abstract

The evolution of geographic phenomena has been one of the concerns of spatiotemporal database research.. However, in a large spectrum of geographical applications, users need more than a mere representation of data evolution. For instance, in urban management applications - e.g. cadastral evolution - users often need to know *why*, *how*, and *by whom* certain changes have been performed as well as their possible impact on the environment. Answers to such queries are not possible unless supplementary information concerning real world events is associated with the corresponding changes in the database and is managed efficiently. This paper proposes a solution to this problem, which is based on extending a spatiotemporal database with a mechanism for managing documentation on the evolution of geographic information. This solution has been implemented in a GIS-based prototype, which is also discussed in the paper.

Keywords – Spatiotemporal evolution; database versions; dynamic documentation; urban management documentation

1 Introduction

The representation of the spatiotemporal evolution of objects has been the focus of intensive research for over a decade and many spatiotemporal models and prototypes have been developed in the context of geographic applications. As stated by Langran [LAN92], representing spatiotemporal evolution should help to trace and analyze changes in spatial information by responding to *what*, *when* and *where* queries such as:

- *Where and when did such a change occur?*
- *What types of change occurred?*

However, in a large spectrum of geographical applications, especially those involving many participants, a mere representation of the spatiotemporal evolution is not enough. It is also important to know the reasons underlying the changes performed in the spatiotemporal database and their impact on the environment. This paper is concerned with urban development applications, where this kind of concern is found at every administration/decision instance. It is also of practical concern for all organizations that deal with map production and delivery [Bad98]. The solution proposed in this paper is based on associating different kinds of documentation with the changes occurring in a spatiotemporal database in order to support a better understanding of the spatiotemporal evolution.

In geographic applications, the interpretation of what is actually meant by "spatiotemporal change" varies according to the angle from which it is considered. The context in which we base our work is the following. Digital cartographic data about a city are initially stored in the database, and users interact with this database via an information system (e.g., a GIS). As the city changes, modifications have to be reflected in the database (e.g., a cadastral application). Major changes require replacing large amounts of data in the geographic database. Sometimes the solution is to replace the entire database. Smaller changes, on the other hand, occur much more often and are entered by users via the information system interface and reflected on the database. Regardless of the volume of changes, users must be able to keep track of the evolution of the city landscape, as illustrated in Figure 1. There are at least three perspectives that must be considered in this context, namely the *real world*, the *cartographic*, and the *database* perspectives. Let us explain them briefly.

Real-world perspective. From the real world point of view, a change refers to a real world event, which is a consequence of natural phenomena or human action, such as the creation of a new street in an urban development project that cuts across several parcels.

Cartographic perspective (interface). From an information system's view point, we consider that the cartographic level is the user's interface to the spatiotemporal database. Here, a map is how the user sees data stored in the database. Maps are therefore transient graphical displays. Since the cartographic perspective corresponds to the system's interface, it is through this level that real world changes are reflected into the stored database. For instance, following the street creation, the limits of the parcels concerned are changed. The user applies these changes via the system's interface – i.e., the user applies changes on a cadastral map, producing a new map *version*. Thus, from this interface perspective, there are now two map versions – the old and the updated cadastral map. From now on, we will use the term "map version" to denote the distinct graphical renderings of a given area, reflecting real life modifications.

Database level. At the database level, a real world event is represented by updates – i.e., creation, deletion and modification - on the database objects that store data corresponding to the real world. Again, in the street example, this affects the information stored about the parcels and, furthermore, introduces an instance of a "street" object in the database. At this level, we use the term *stored map*, to indicate the set of stored objects concerning an area, and which are combined to be shown at the cartographic level as a map of that area.

Distinct kinds of users may be interested in different levels of changes. For instance:

- A surveyor is concerned with real world events.
- The user of a cadastral map (e.g., city planner) may want to compare city regions where changes occurred within a certain period – and thus ask to see the corresponding versions.
- A database administrator may want to check the last update of a database object.

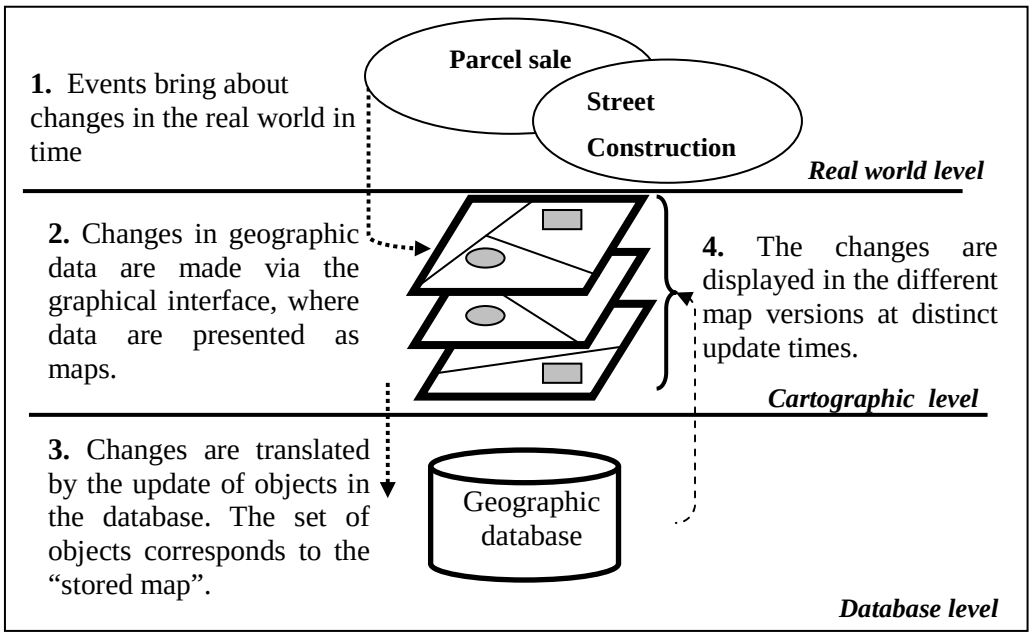


Figure 1. Different levels of changes

The granularity of changes may vary from one level to another. At the database level, each update corresponds to a change, whereas at the cartographic level – which is what the user sees and interacts with - one change may correspond to several data update operations (in database terms, a long transaction). In this paper, we use the term "update" to refer to the database level changes and the term "modification" for the cartographic level changes, whereas "change" refers to both kinds of change. Our goal is to document all these changes within the database itself.

We are interested in three major types of change documentation in the context of spatiotemporal data management, more precisely: (i) documents about events occurring in the real world, constituting the semantic aspect of the evolution; (ii) documents concerning the cartographic evolution describing the different modifications the user performed at the interface level; and (iii) documentation about the database evolution, containing information about the different objects that have been updated.

However, storing change documentation in a database is not straightforward, especially when the database contains elements that are interconnected. This often arises in geographic applications where updating an object may imply changes on associated objects. Furthermore, whereas some objects are updated for a given (real world) reason, others suffer updates as a consequence – e.g., to maintain topological integrity or semantic consistency. For instance, if the width of a street is doubled, it is not only the parcels along that street that may be affected, but all other kinds of objects, e.g., lamp poles have to be moved, pedestrian crossings have to be extended, and so on. Thus, the problem is first to find

out the kind of documents that should be stored and then to define how they should be stored in order to allow access to both direct and indirect reasons for an update.

This paper focuses on the management of spatiotemporal change documentation in the context of urban development applications. It proposes a solution to this problem based on two principles: (a) the use of a geographic database extended with a version management mechanism; and (b) the selective storing of change documentation for the three levels. Each level requires distinct documentation storage and management criteria in order to minimize redundancy. In this framework, besides executing *what-*, *when-*, and *where-queries* typical in the evolution of geographic phenomena, users may also execute corresponding *why-*, *how-*, and *who-queries*. Each of these queries may be posed on distinct stored versions of the world. This allows urban planners to keep track of the evolution of a city's landscape and therefore helps them in their decision processes.

This paper is organized as follows. Section 2 describes related work. Section 3 presents a simple example of spatiotemporal evolution for an urban development application, which is used throughout the paper. Section 4 presents our proposal for managing change documentation in a spatiotemporal database. Implementation structures are described in Section 5. Section 6 focuses on the querying mechanism used to query change documentation. Section 7 briefly describes a prototype that has been implemented using the concepts presented in the paper. Finally, Section 8 presents our conclusions.

2 Related work

The need for documentation has prompted the appearance on experts on this field, who now have to concern themselves with the requirements of digital data [Grice02]. Issues related to the documentation of spatial data evolution have seldom been considered. It is only recently that database researchers have started analyzing how to better follow up evolution and changes, and even then outside the scope of spatiotemporal data. The workshop on data evolution reported in [RAB+00] enumerates several open problems concerning keeping track of data changes. Relevant queries pointed out concern *what-*, *why-*, *when-*, *where-*, *who-*, and *how-issues*, in the same sense as those discussed in the present paper. In general, related work is mainly concerned with version management and documentation management. It is described below by separating standard applications
from spatiotemporal applications.

2.1 Standard applications

Studies on documentation maintenance are more frequent in the context of software engineering, in cooperative work, or in information science in general. Document engineering is now receiving attention, with a specific conference that discusses standards and tools, for standard applications [FMM02]. Software engineering research is concerned in particular with keeping track of

software evolution and its document semantics (e.g., [CDF+00]), often by using versioning mechanisms and associating documents with versions. Interestingly enough, recent results in code evolution and decay point out to the need for *why-*, *what-*, *who-*, *when-*, *how long- queries* [EG+01]. This kind of research in software engineering is also being motivated nowadays by the need to deal with legacy systems.

In cooperative work, documents serve as one of the communication means among the several persons involved, usually to describe how systems have been designed and constructed. Here, the emphasis is not so much on changes, but on how to retrieve relevant information from stored documents in order to maximize the cooperation among human or software actors (e.g. [SS98, CD00]). In this context, there is an increasing demand for why-queries (e.g., in design rationale) and for the follow up of product development (organizational memory).

Mechanisms for the retrieval of relevant information are left to information science researchers. The most common approaches rely on document marking, hierarchical organization of documents using hyperlinks, or metadata annotations. More recently, the emphasis has turned to dynamic document management facilities, which try to cater to distinct user levels and support context sensitive retrieval, for instance by taking advantage of active facilities (e.g., [DELS99, DEL+00]).

We base our classification of documents on the work described in [VBJ00] This work distinguishes between three types of documents (in the context of engineering artifacts):

1. *What-documents* – e.g., manuals - describe the properties of an artifact, and ensure its maintainability and usability. They are stored as metadata and textual data. Textual data can be organized in hypertext/hypermedia graphs [DL96].
2. *How-documents* describe the process used to build artifacts. They ensure maintainability and repeatability in reproducing the artifact, and are stored as scientific workflows [AIL98].
3. *Why-documents* are used to understand the reasoning behind the construction of an artifact, and are conceptually close to the notion of design rationale of artificial intelligence [MC96].

In general, however, the management of change documentation has not received much attention from database researchers. This is clearly influenced by the fact that standard databases do not consider multiple states for an entity. In the database field, documentation-related research is conducted in the context of information retrieval and the main problem is related to indexing and organizing documents, as opposed to the semantics of changes.

2.2 Spatiotemporal applications

The evolution of spatiotemporal objects has motivated several kinds of research (see for instance [WOR94, PEU94, PT98, TRY98, VCP99]). These approaches address distinct user needs. They vary in

terms of the data models, storage structures, and query constructs adopted. The objects considered may be static or dynamic, e.g., moving object databases [SWCD97, FGN00, PT00,BJKS01].

In geographic applications, more specifically, documentation can be of fundamental support to users - for example, to follow up the evolution of cadastral land parcels [SCL99] or to help in the exchange of geographic data between map producers and customers [BAD98]. The latter issue is an example of a situation that is becoming widespread and affects all kinds of companies that produce digital cartographic data (the "map producers"). These digital files are sold to companies that often embed them in their products. However, in some cases, these companies (the "consumers") may want to update the maps according to their own needs –– e.g., adding features that interest that company's business. At the same time, the producers will be updating their own maps that will be subsequently delivered to the consumers. These consumers will then be faced with two updated versions of a given area. The consolidation of these updates into a consistent digital map is a real problem with many economic consequences [PJB02]. Another related issue where documentation is needed is the integration of heterogeneous geographic data. In this case, the integration concentrates on metadata-based methodologies or, more recently, in the use of ontologies as a basis for semantic comparison – see, for instance, [Fons01, UOM+98,FLP+03].

2.3 Our approach

The approach taken here is similar to software engineering in the sense that we adopt a particular versioning mechanism to help manage change documentation. At the same time, we consider the context-sensitive dimension of information retrieval since we allow distinct levels of documentation to consider varying user profiles and needs. Unlike all previous research, however, we deal with the specificities of spatiotemporal change documentation and manage this documentation in an integrated way within a spatiotemporal database context. Finally, we support *why-queries*, which, in spite of their applicability, have so far been ignored by researchers on spatiotemporal querying profiles. We point out that for change documentation *how*- and *why*- queries are feasible only when distinct versions are available. Such queries only make sense when it is possible to compare different states of the world.

3 Reference example

This section first presents our reference example of spatiotemporal evolution in a geographic database for an urban development application. This example will be used throughout the paper. We assume that data are stored in an object-oriented database as it is an appropriate paradigm to model geographic applications. Moreover, it makes our explanation easier. Notice, however, that our proposal is general and applies to other types of database management systems such as relational systems extended with abstract data types (ADT). In this paper, we follow the common database terminology and use the expression *complex* (or *composite*) object to refer to any object that has at least one reference to another object as opposed to a *simple (*or *atomic)* objects that does not have such a reference.

3.1 Data model

We consider a cadastral database representing land parcels in a given region (a 2-dimensional space). As it is common in geographic applications, each parcel is considered as a *geographic object* [RSV01], which is an entity consisting of two components: (i) a *description*, implemented as standard (thematic) attributes such as the owner of a parcel and (ii) a *spatial component* describing the object's geometry. The geometry is usually 0-, 1-, or 2-dimensional. A *map* is a graphical representation of a collection of geographic objects. A *stored map* corresponds to the set of objects that completely describe a region, and that are visualized as the map. The geometry of a parcel is a 2-dimensional spatial object. Usually, such objects are encapsulated and correspond to spatial abstract data types. However, in the context of an update-oriented application, we need to access the details of the geometry of the object (the denoted spatial component above), which follows a *spatial data model* (see below). For the sake of simplicity, we consider that the geometry of a parcel is one polygon. A polygon is defined as a list of line segments describing its boundaries. Each line segment refers to two nodes representing its extremities.

Two remarks are noteworthy. First, most geospatial applications consider a richer type system and would associate with such a geographic object a *region*, defined as a set of polygons as opposed to a single polygon. This allows non-connected spatial components to be associated with one geographic entity, for instance, a country composed of a mainland and of islands or other territories. Second, regarding the terminology, one should not confuse the notions of complex geographic objects and complex objects in the database sense. The former denotes a geographic entity that may encompass other entities. For instance a state composed of counties is a complex geographic object. If the spatial component of a county is modeled as an object in an object-oriented database, then a geographic object is also a complex object in the (object-oriented) database sense. This is the reason why we prefer, to the notion of object, the one of *type*.

In geospatial applications, the major spatial data models usually considered are the *spaghetti*, the *network*, and the *topological* model. We chose to use a simplified version of the topological model in which arcs are not associated with nodes. Here nodes only consist of *(x, y)* coordinates. The coordinates indirectly provide the geographic location by means of association of *(x, y)* pairs to some geographic coordinate system. This association is out of the scope of this paper. However, it is important to keep its existence in mind, since we are dealing with urban applications and not mere polygon management. Figure 2 describes the schema of the geographic object ***Parcel*** in an object oriented like language.

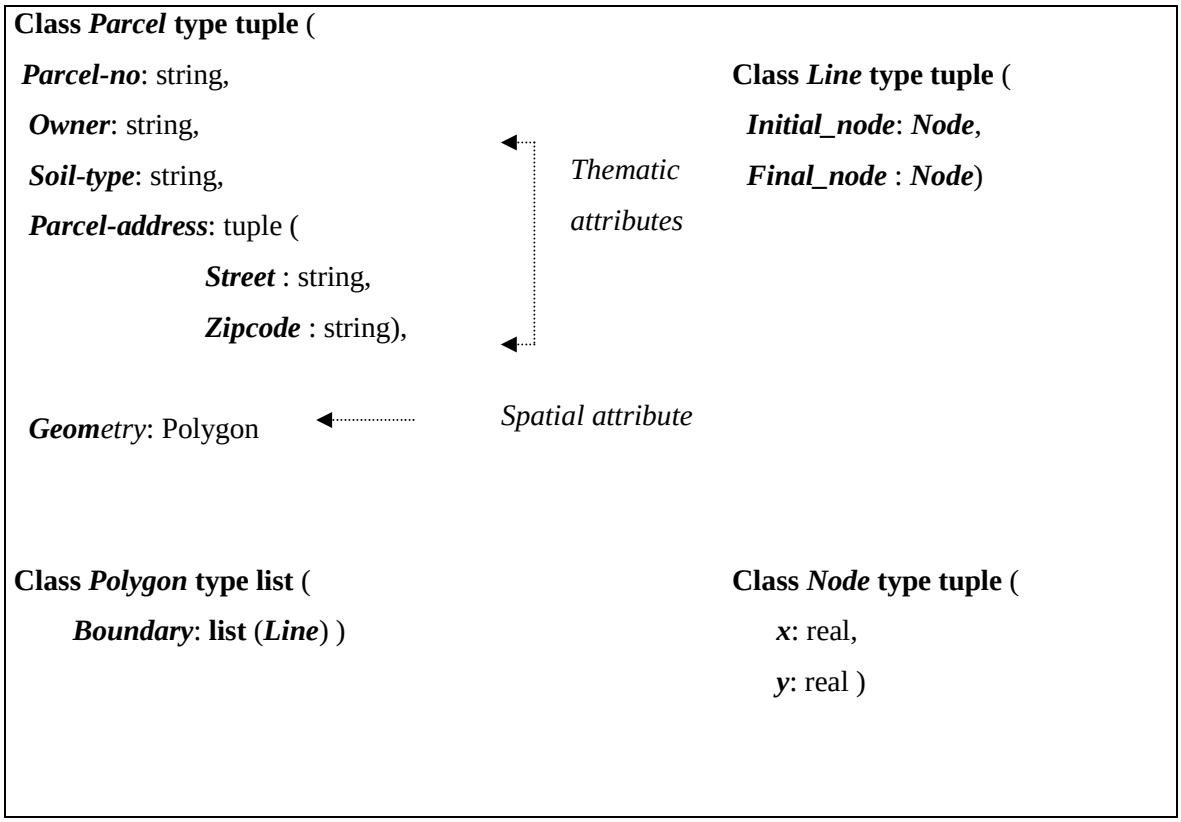


Figure 2. Database schema for parcels and spatial types

3.2 Update Example

Our running example is described graphically in Figure 3. Figure 3a depicts the state of three parcels of the modeled region, P_1, P_2, and P_3, at time t_1. The database objects forming the geometry of the parcels are represented in italics (with an arrow pointing to the specific object). For instance, the geometry of parcel P_1 is represented by Polygon F_1, which is formed by line segments L_1, L_2, and L_3. Line segment L_1 is composed of nodes N_1 and N_2; L_2 is composed of nodes N_2 and N_3; L_3 is made up of nodes N_1 and N_3, and so on. N_i, L_i, F_i, and P_i are internal database identifiers. They are managed by the system itself and are not visible to the real world or map-level users. The latter can only manage the external identifier of the parcel, which is described in the schema by attribute ***Parcel-no*** of type string.

Now suppose that the owner of parcel P_1 buys the parts of parcels P_2 and P_3 that are adjacent to his or her parcel. Before the sale operation can be performed and the boundary change carried out onto the cadastral map, different legal procedures should be followed in the real world, including:

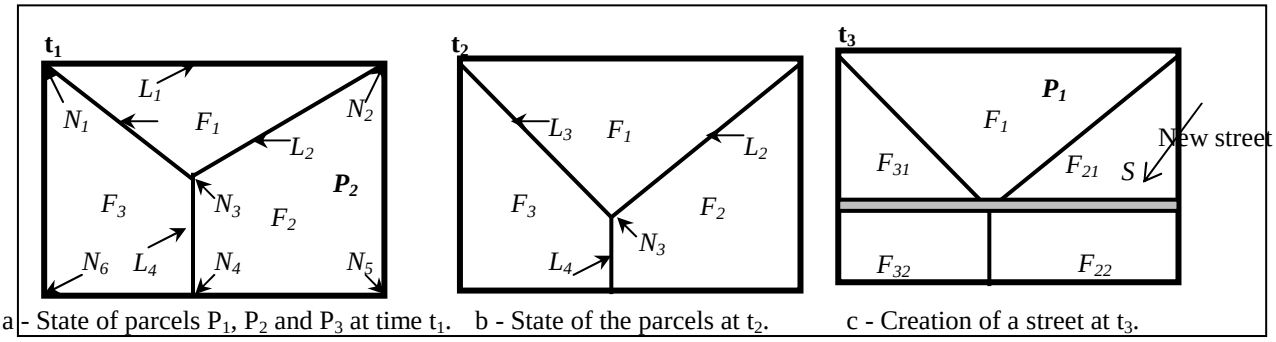
- A land surveying document should be created by a surveyor.
- A deed concerning the sale operation should be prepared by a lawyer.
- The land surveying document and the deed should be sent to the Land Registry.

Once the sale is performed, the limits of the three parcels are modified on the cadastral map and a new map version corresponding to the transaction time t_2 is obtained. Figure 3b shows this evolution. Finally, at the database level, the set of boundary changes is translated into a single update: the change

of the value of node N_3 only. As a consequence, the affected line segments (L_2, L_3, and L_4) and in turn, the polygons (F_1, F_2, and F_3) are implicitly modified as indicated in Figure 3b.

This simple example illustrates that a complex real world operation involving several people and legal procedures results in the update of one node in the database. However, it is clear that recording this kind of update only (i.e., the one database change with associated valid timestamp) is not enough to satisfy the needs of the urban planners who use the spatiotemporal database.

Suppose, next, that the local authority decides to build a new street passing through the three parcels of Figure 3b. After the street creation, the new version of the cadastral map, corresponding to the new state of the modeled world, is presented in Figure 3c, showing the insertion of the street (S) at database time t_3.



a - State of parcels P_1, P_2 and P_3 at time t_1. b - State of the parcels at t_2. c - Creation of a street at t_3.

Figure 3. Evolution of the geometries of parcels

4 Documenting changes

This section is devoted to our proposal for handling documentation. We first introduce various abstraction levels in the documentation, and next consider each level. For each of them we take as an illustration example the running example described previously and we examine the type of documentation needed.

4.1 Levels of change documentation and types of documents

Following our approach of three levels of changes, we propose the following types of change documentation:

- The documentation concerning the real world event.
- The documentation concerning the modifications the user performs on the visualized maps.
- The documentation concerning the update of the database.

Besides, we need to refer to various types of documents. Information systems in general deal with metadata that reflects basic information associated with database objects, in order to find out when, where, and by whom, among other information, some entity was created or updated. Here we need a more complex type of documentation, describing different concerns of end users.

In the following, we introduce the concepts of *what-*, *how-*, and *why-documents*. *What-documents* describe changed objects at the three levels mentioned previously (real world, cartographic,

and database); *how-documents* explain how changes are performed; and finally, *why-documents* provide the reasons for a change. As we next show, not all levels need all kinds of documentation, since some documents do not make sense at some levels. Figure 4 shows the basic structure of *why-* and *how-documents*, which belong respectively to classes ***why-doc*** and ***how-doc***. Note that classical metadata, for instance *who*, is associated with these classes.

Class *why-doc* type tuple (
Reason: text,
Who: string,
When: Date,
Changes: Modification)

Class *how-doc* type tuple(
Description: text,
Who: string,
When: Date,
Changes: Modification)

Figure 4. Schema of why- and how-documents

The ***Changes*** field in the *why-doc* class, of type "Modification", is used for query optimization, in order to allow navigating across documents and map versions. It is only visible at the database level, as explained in Section 5.

4.2 Real-world documentation

In addition to common metadata, at least two kinds of documents should be considered at the real world level: *why-* and *how-documents*. The first one explains the reasons behind a real world change. As previously mentioned, it also includes the actors (the *who*) involved in the change. Figure 5 shows an example of a simplified *why-document* for the parcel sale, which explains the reasons for (spatial) changes, from t_1 to t_2. Notice that the *Who* field names the people involved in this sale and then *When*, its date in the real world. It is the valid time stamp for the sale transaction, but not necessarily the valid time stamp for the boundary (spatial) change, which may be different due to legal reasons. The *What* field enumerates the geographic objects to which this explanation applies.

.

Why₁

Reason : the sale of a portion of parcels P_2
of Mr Jones and P_3 of Mr Paul to
the owner of parcel P_1 (Mr Smith)
(see Deed n° xxx)
Who : Mr Smith, Mr Jones and Mr Paul
When : 01/02/2001
[Changes : …]

Figure 5. *Why-document* concerning a real-world event

The second one, the how-documentation, consists of several documents (***How-doc***) concerning the different legal procedures to be followed for the sale to be really achieved, including:

Hr_1: the creation of the land surveying document.

Hr_2: the formulation of the deed.

Hr_3: the publishing of the legal documents in the Land Registry.

In each ***how-doc***, the *Who* field refers to the different actors involved in a procedure being described by the document - e.g., land surveyor, lawyer- whereas the *When* stores the dates of the different operations such as land surveying or formulation of deed. Notice again that the dates in real-world *why*- and *how-document* concern the validity of operations. This does not necessarily correspond to database updates on geographic objects, and thus may not be automatically managed by temporal mechanisms from a temporal database perspective. Nevertheless, one may consider that these documents are objects with valid and transaction time stamps.

Besides metadata, additional information can be attached to these documents. For instance, the how-documentation concerning the land surveying can also include photos of the land parcels or the surveyor's map.

4.3 Cartographic level

The major cartographic entity to consider is a map and the associated document is a what-document containing map metadata. There are several standards for cartographic metadata, which are outside the scope of this paper (see for instance [GV98] for more information on the topic). Here, it suffices to know that these standards typically include data such as region coordinates *(where)* and cartographic projection, as well as *who* (cartographer) and *when* (valid time). Figure 6 presents a possible schema for a what-document at this level (Class ***what-doc***).

Class *what-doc* type tuple (
Metadata: tuple(fields),
…
Who: string,
When: Date,
Changes: string)
When: date }

Figure 6. Schema of what-documents

We recall that in our framework the cartographic level is the user interface to the geographic database. Updates may concern the thematic as well as the spatial attributes of the objects involved. For instance, the parcel sale requires a map update, which will be propagated to the database by the update of node N_3. As a consequence, future database queries will show the new map version. The what-

document will contain all geographic objects affected by the corresponding map version update – the three parcels. It may further contain metadata fields such as those described previously.

Thus, why-documents at this level are those that concern the real world changes whereas how-documents are those that concern cartographic modification. Hence there are no specific why-documents concerning this level for urban planners. The *why* is the real world *why*.

Moreover, different kinds of users may interact at the cartographic level, such as urban planners or cartographers responsible for maintaining the maps according to predetermined map construction standards. These users may want to assign a new kind of why-document to the maps concerning these standards (e.g., parcel boundaries are drawn with line width *w* because of a given cartographic norm). One may even consider how-documentation to indicate the order in which map changes were performed by the users. To simplify the example, however, we limit ourselves to one type of users only – the ones concerned with urban planning, who are not interested in the niceties of cartographic production.

4.4 Database level

At the database level, the main documents concern *why* and *how* database updates are performed. The *when* at this level can either be directly found from the spatiotemporal transaction timestamp or stored in a how-document. *What*-documents describe an entity's properties. At the database level, this corresponds to schema and extension (the stored data). There is no need for creating specific documents for this kind of information, since the DBMS automatically handles this. The *why* of a database change is in fact that one needs to reflect real world changes. Thus, there are no new *why-documents* created for this level and the *why* at this level is again the real world why.

How-documents at this level describe how an update is actually performed. They contain the whole process of the update of an object from the cartographic level down to the database level. Here the concern is to explain the steps taken to update the database. As we shall see, how-documents at this level are not stored, but derived from the fact that a geographic object has changed. Thus, for instance, the updates of the (database object) parcel P_1 that reflect the cartographic changes from the map version at time t_1 to the map version at time t_2 are associated with the following "how reasoning":

Hd_1. P_1 changed "by changing the Geometry",

Hd_2.The geometry of P_1 has changed "by changing polygon F_1",

Hd_3. Polygon F_1 has changed "by changing L_2 and L_3",

Hd_4. L_2 and L_3 have changed "by updating node N_3",

Hd_5. N_3 has changed "from (x_3, y_3) to (x_{31}, y_{31})"

This reasoning can also be used at the cartographic level to explain how a given map is constructed. However, at the cartographic level, usually only the first kind of procedure is needed. For instance, the query "How was map version at t_2 constructed?" may have an answer such as "By starting from the map at time t_1 and changing the geometry of parcels P_1, P_2, and P_3".

So far, we have exemplified each level of documentation considering only the transition from t_1 to t_2 (and the corresponding spatiotemporal database changes). The same kind of documentation procedure will apply to any such transition, and the corresponding documents are to be stored in the database. Thus, for the real world evolution from t_2 to t_3 (street creation), there will be at least:

- Real world level why- and how-documents associated with the street and parcels affected.
- Cartographic how-procedures and what-metadata associated with the new map version at t_3.
- Database updates that allow describing how the parcels (and their components) were affected, and how the street construction was reflected in the spatiotemporal database.

To sum up, any real world change is documented by distinct kinds of why-, how- and what-documents. These documents differ in terms of the level they are concerned with and the kind of user who needs to access them. Document contents are used to derive the procedures and reasons for evolution (reflected at the database level by spatiotemporal updates). Documentation is dynamic in the sense that it follows this evolution and that it is context-sensitive.

5 Implementation

The whole idea of dynamic change documentation in a database requires, among other things, the use of a version mechanism. We chose to use the multiversion database mechanism of [CJ90] to manage the distinct versions of the stored objects (corresponding to the different states of the world). We recall that the documentation related to the update of a complex object may concern several of its components.

5.1 The implemented version mechanism

The multiversion database mechanism lets end users see the geographic database as composed of a set of independent map versions, representing the same area, which coexist within the same storage space. This means, for instance, that each map version can be managed (read and updated) separately and independently from the other map versions of the same region. At the user level, a new map version is always generated or *derived* as a (logical) copy of an existing map version.

At the database level, one important feature of the multiversion database mechanism is that it automatically allows keeping track of all database objects that "fit together", i.e., that compose a consistent database state. As is typical in any temporal application, several versions of the same real world object coexist in a database (e.g., the different parcel configurations at the distinct time frames). This gives origin to the concept of *multiversion object*, which can be described in a simplified manner as being a "repository" of all versions of a given object. Going back to Figure 3, for instance, multiversion object "parcel P_1" is in fact a complex object, which encapsulates all different states of P_1. The multiversion database mechanism automatically puts each object version in a different database state – e.g., P_1 at time t_1 and P_1 at time t_2 correspond to distinct states of the parcel, and thus different database states.

From a performance point of view, one of the main advantages of the mechanism is that it minimizes storage occupancy and avoids redundancy while storing the multiple versions. Again at the cartographic level, when map features are updated, a new map version is generated. At the database level, however, instead of storing the map data twice, the mechanism just records the updates and unmodified objects are stored only once in the database. The relationship among the distinct (stored map) versions is recorded in a structure called *map version tree*. Furthermore, updates to objects in one map are handled without affecting other map versions, by appropriate management of internal version identifiers.

To obtain the value of an object in a stored map version, the system applies a rule stating that it has the same value as that in the map version from which it is derived, except if another value is explicitly specified. This rule is recursive and called *implicit sharing rule*. When an object is deleted in a map version, its value in the database is set to ⊥, meaning it does not exist. When an object *O* in a map version *v* is involved in a group operation – e.g., splitting or merging operation on land parcels - the link between *O* and the resulting object(s) is stored in a *genealogy tree* [SCL99]. A special value # for *O* in map version *v* is used to denote a group operation. When geographic objects are created from one or several other geographic objects – i.e. the object has one or several ancestors – the resulting geographic objects are initialized with a special value "*" in the map version parent of the map version in which the operation has been performed. Thus, the genealogy graph represents 1-to-N, N-to-1 and N-to-M evolutions of geographic objects.

The system uses the internal identifier of objects to follow the evolution of these objects through time. Internal identifiers are managed only by the system, conversely to external identifiers, which are managed and accessed by users. For further details the reader is reffered to [BJ93, GJ94, CJ00].

Illustration of the multiversion mechanism

Figure 7 shows part of the database without documentation, indicating how different objects evolve, under the perspective of the multiversion mechanism. Versions M_1, M_2, and M_3 correspond respectively to times t_1, t_2, and t_3. The map derivation tree shows that map version M_2 has been derived from map version M_1; and that M_3 is derived from M_2.

In a simplified manner, each multiversion object corresponds to a table, where each line concerns a version. Thus, multiversion object N_3 has three versions recorded, for stored map versions M_1, M_2 and M_3. This means that at time t_1, N_3 has value (x_3, y_3), (first entry of the table), at time t_2, its value is modified to (x_{31}, y_{31}), and at t_3, its value is ⊥, meaning that is does not exist.

Furthermore, the Figure seems to show that multiversion object P_1 has only one version corresponding to map version M_1, and that F_1 and L_3 have two versions each. This is, however, not true, and is in fact due to the management of complex objects [AHV95] and multiversion complex objects [CJ00]. The database stores the shallow value of objects. Thus, for complex objects (having references

to other objects), the identifier of the referred objects are stored. But, on a map version, access is granted to the deep value of complex objects. In the deep value of an object, the identifiers of referred objects are recursively replaced by the value of the referred object. For instance, in Figure 7, N_3 has three (shallow) values stored - (x_3, y_3), (x_{31}, y_{31}) and $\perp$, whereas the shallow value of L_3 is (N_1, N_3) in M_1 and M_2. However, for the user visualizing map version M_1 (see Figure 2b), L_3 is represented by the line with extremities (x_1, y_1) and (x_3, y_3) (deep value), while in map version M_2, it is represented by the line with extremities (x_1, y_1) and (x_{31}, y_{31}).

Suppose that the user wants to know what L_3 looks like in map version M_2. The system "constructs" this version of L_3 recursively as follows:

1) Retrieves the state of L_3 for map version M_2
    1.1 The multiversion object L_3 has no entry for M_2. This means that the line did not change its components at t_2. This means that the shallow value of L_3 is the same as in M_2 and M_1, the map version from which M_2 was derived .
    1.2 The derivation tree shows that M_2 is derived from M_1
    1.3 The components of L_3 in M_1, and thus M_2, are N_1 and N_3 - the same as in M_1.
2) Retrieves the value of N_1 in M_2 following the same procedure (using the derivation tree to obtain the ancestor of the map version M_2 - i.e., M_1 - and reading the value of N_1 in M_1). The value of N_1 in map version M_2 is thus (x_1, y_1).
3) Retrieves the state of N_3 at M_2, which is obtained directly from the second entry of the corresponding multiversion object.
4) Constructs the deep value of L_3 in M_2, using the values of its components (N_1, N_3), which are ((x_1, y_1), (x_{31}, y_{31})).

The same kind of reasoning is applied to all composition levels – e.g., parcel P_1 for map version M_2 has the same components as for map version M_1, and its geometry is described by F_1. Next, polygon F_1 for map version M_1 is retrieved followed by its component lines, finally reaching the node level.

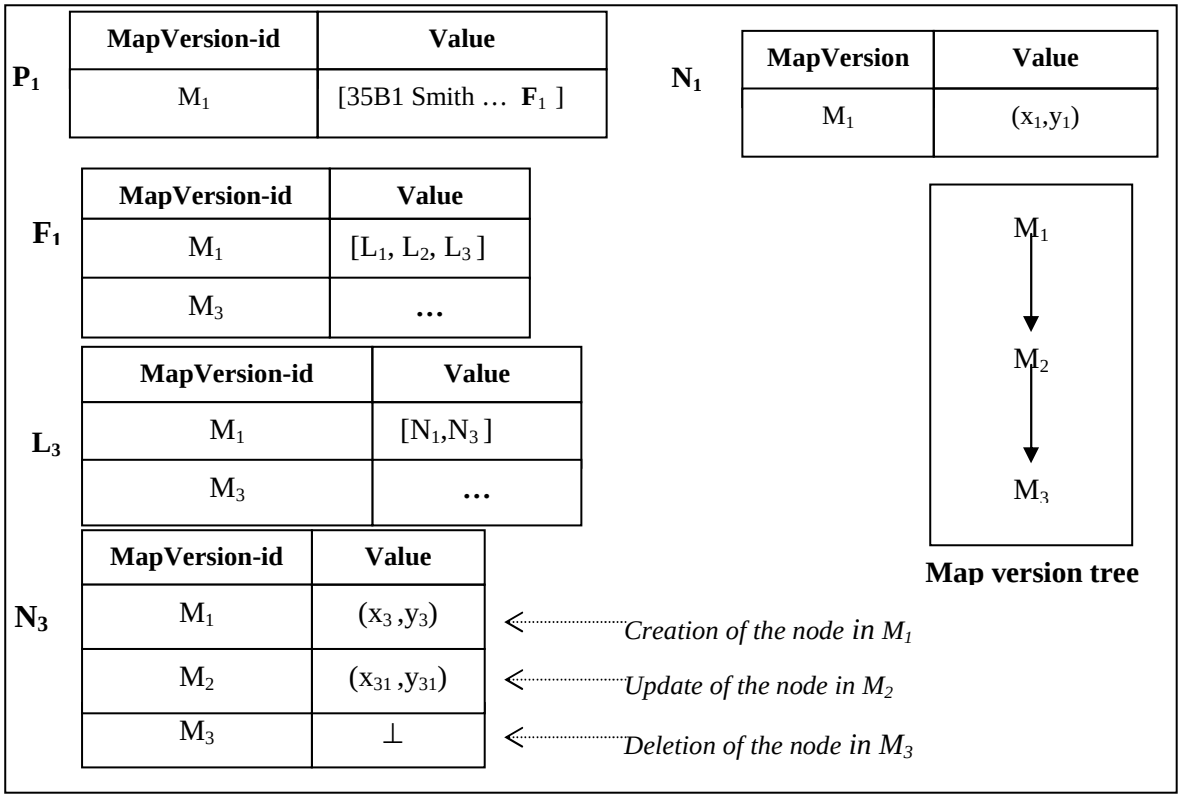


Figure 7. Part of the database state without changes

5.2 Storing documents

In order to avoid duplications and possible inconsistencies, the adopted strategy is to store each document only once. Let us now consider the evolution of an entity over time together with the documentation associated with its successive changes. For instance, we recall that the what-documentation at the cartographic level corresponds to map metadata. From the versioning mechanism point of view, this means that the what-documents must be associated with a *map version* as opposed to *individual object versions*. Why- and how-documents, however, are associated with individual objects.

Each document is also a multiversion object, managed jointly with the associated geographic object. A real world why-document affects many objects (e.g., one why-document for parcels P_1, P_2, and P_3). The difficulty here is that the document should be stored only once to avoid redundancy in the database, while at the same time maintaining links to all objects. Furthermore, to speed up queries, one should be able to navigate from one kind of document to another. The solution adopted was to create a multiversion table that links, to each map version, all the modifications, the corresponding change documentation, and the real world objects affected.

Figure 8 shows a partial view of the database including multiversion document objects. Two new database objects are introduced:

(1) « Doc-Modif-Table » associates each map modification with the corresponding why-, how-, and what-documents. For each modification, the system creates the documents, for which

the user enters the necessary information. Furthermore, to speed up queries, an index can be created to link each map version with the corresponding modification identifiers.

(2) « Modif-Table » details the modifications, by associating a modification identifier with the identifier of updated objects and the map version in which these updates are performed. Each time the (shallow) value of an object is changed, a trigger inserts a new record in this table.

For instance, when N_3 is updated in M_2, a trigger inserts a new record in the Modif-table corresponding to the first line of the table in Figure 8. Doc-Modif-Table contains the identifiers of the different change documents corresponding to $Modif_1$, which, in turn, points at the relevant real world why-document (Why_1) and how-document (How_1), and database what-documents ($What_1$). In this case $What_1$ refers to only one object, N_3. However, in a major update it would point to all the updated objects. The what-document shown in Figure 8 refers only to the parcel change. However, a what-document for the street construction (Figure 3c) would require a much longer description, since several database and real life objects are affected.

Finally, stored map versions might be created only after several modifications are made. For instance, a new map version M_4 may be created if all parcels neighboring the street are bought by the municipality ($Modif_3$ for map version M_4) and the street is enlarged, occupying part of these parcels ($Modif_4$).

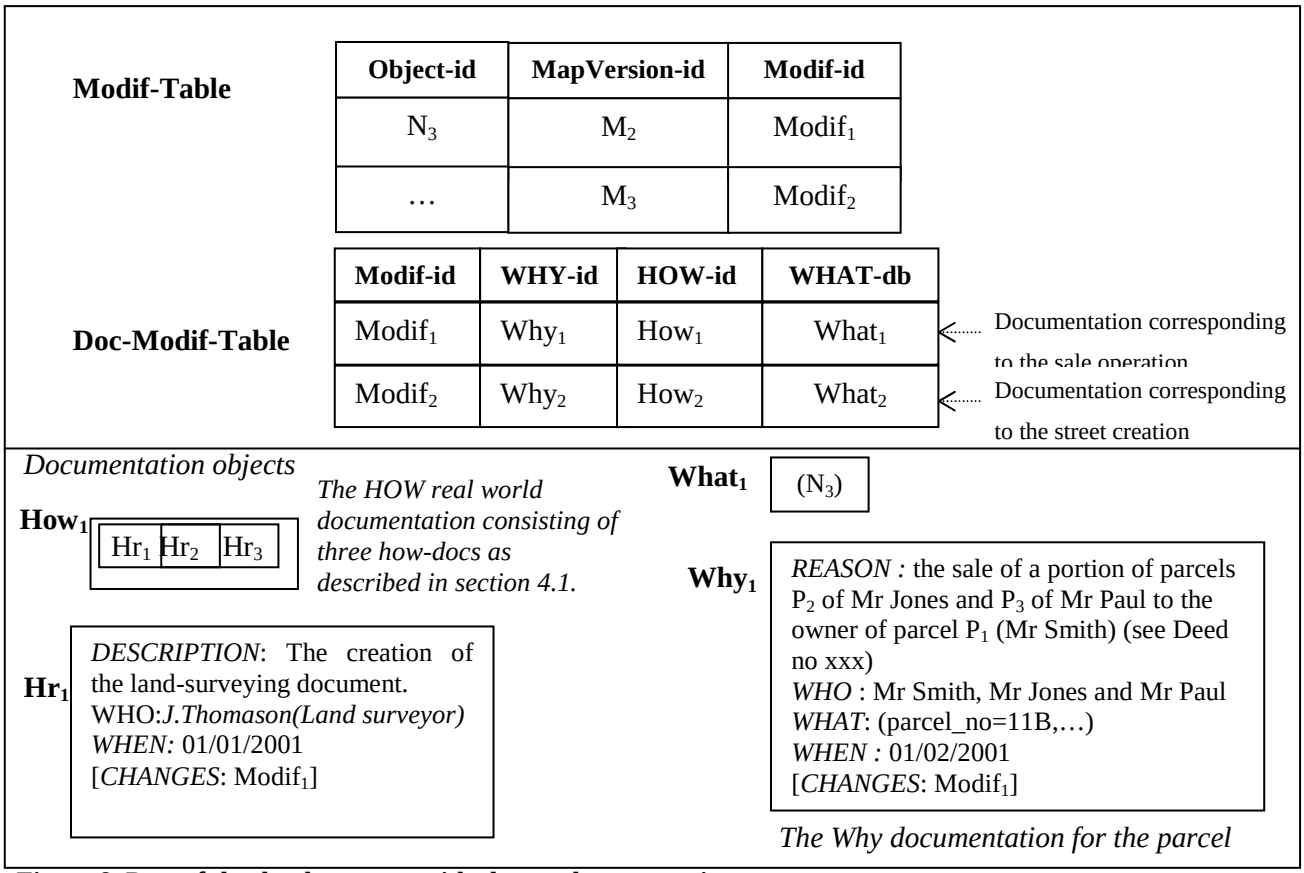


Figure 8. Part of the database state with change documentation

All other structures needed to handle versions are not declared at the database schema level because they are handled internally by the version mechanism [CJ94].

5.3 Querying the update documentation

The previous section explained how update documents are stored. The documentation of changes in a spatiotemporal database extends the range of spatiotemporal queries a user can express. Examples of such queries for our application domain are:

1. *What objects* have changed between time t_1 and t_2?
2. *What is the difference* between map version M_1 and M_3?

Pointing at P_1 in map version M_2:

3. *Why* has this parcel changed?
4. *How* has the change occurred in the real world?
5. *Who* was responsible for performing the corresponding database change?
6. *How* was the change performed in the database?
7. *What are the objects* involved in this parcel change?
8. *What are the names of the owners* of the parcels affected by this change?

Finally, at the database level, the Database Administrator may want to know:

9. *Why* did N_3 change between t_1 and t_2?

Whereas queries 1 and 5 (and possibly 7) may be conceivably implemented in standard spatiotemporal databases, the other queries require dynamic documentation and a versioning mechanism. This section gives a few examples of ways some of these queries can be implemented. For map version M_2, it is assumed here that the user has already selected this map for analysis and wants to find out about its evolution. We show how to deal with queries 1, 3, 4, 5, 6 and 7. To answer queries 2 and 8, the distinct versions must be compared.

5.4 Real world level queries

Looking for documents concerning the *why* and the *how* of the evolution of real world objects is a straightforward operation. It suffices to read the specific why- and how-documents associated with the objects. This solves queries 3 and 4, where the versioning system will recover the appropriate document version.

5.5 Cartographic level queries

At this level, *what* details are obtained from the metadata document associated with a stored map version. The why-document for an evolution of a cartographic object is the same as its real world why-document. The answer to query 5 is obtained from the *who* field of the how-cartographic-document, associated with the map version. Again, why- and how-documents are obtained directly.

5.6 Database level queries

First, let us consider why-queries. Two types of why-queries can be distinguished:

- queries concerning objects that correspond to real world objects (e.g., a parcel)
- queries concerning components of geographic objects, especially their geometry (e.g., node N_3).

In the first case, the why-document is obtained directly from the corresponding multiversion object. In the second case, the system must go "bottom-up" following the database schema of object composition until a real world entity is found, and then retrieve its associated why-document.

For instance, when looking for the why-document concerning the update of the node N_3 in M_2 (query 9), the system:

1. First obtains from Modif-table (see Figure 8) the Modif-id, $Modif_1$ corresponding to M_2;
2. Then finds the Why-id, Why_1, corresponding to $Modif_1$ in Doc-Modif_Table;
3. Finally retrieves the associated why-document corresponding to the parcel, which contains the names of the owners.

For building the "how" reasoning, at some composition level of a composite object, the system navigates top-down, using the database schema, until it reaches that object's simple components. Thus, for instance, a how-query at the line level will explain line and point updates; a how-query at the geometry level will return polygon - line - node update how-documents. For instance, in order to know how parcel P_1 has been updated in the database at time t_2 (query 6), the system:

1. First deduces the class composition hierarchy of the class Parcel which is [Parcel, Polygon, Line, Node]
2. Then navigates through the composition hierarchy of the components of parcel P_1, in the specific map version, until it reaches the object that has been physically updated (N_3). For this it selects for map version M_2:
   - polygon F_1, the first level component of P_1
   - the lines L_1, L_2, and L_3, which are components of F_1,
   - the nodes N_1, N_2, and N_3, components of lines L_1, L_2, and L_3. It deduces that N_3 has been updated in M_2, since it has an entry for this map version
3. Finally builds the how-reasoning by deducing the part of the composition hierarchy of P_1 that refers to N_3 - i.e., the polygon F_1, the lines L_2 and L_3, and the node N_3.

This reasoning can be generalized by an algorithm whose basic principle is explained below. There are two possible cases:

- The user may either point to an object whose shallow value has changed (e.g., N_3 in M_2) (Case 1),
- or may choose an object whose deep value has changed (e.g. P_1 in M_2) (Case 2).

In Case 1, the system:

1. First extracts the object identifier and the map version identifier (e.g., N_3 and M_2),
2. Then extracts the corresponding modification identifier from the Modif-Table (e.g. $Modif_1$), and the identifiers of all documents (why-, how-, and what-documents) from the Doc-Modif-Table, (e.g. why_1, how_1, $what_1$)
3. Finally extracts the contents of these documents, and particularly the identifiers of all the objects, whose shallow values have been modified by $Modif_1$ (e.g. here, only N_3)

In Case 2 (when the deep value of the pointed object is changed in the map version) the system needs to find out the elementary components whose shallow values have been modified (e.g. P_1 in M_2):

1. First, the system extracts the object identifier and the map version identifier (e.g., P_1 and M_2),
2. Next, it deduces the class composition hierarchy of the object - here, the class hierarchy is [Parcel, Polygon, Line, Node]
3. Then it navigates through the composition hierarchy of the components of the object in the specific map version (e.g., parcel P_1), until it reaches the objects that have been physically updated in the specific map version. See the previous discussion for query 6, which starts from polygon F_1, passes through the three lines L_1, L_2, and L_3, which form this polygon, and then to their nodes to reach N_3, the only node affected.

From then onwards, the system proceeds as in Case 1 to obtain the modification identifier.

In general, real world modifications imply the update of more that one object in the database, while the example used in this article, portrays a very simple situation aiming at explaining our approach. For instance, in the road construction, many nodes may be created, deleted or updated. The list of all the updated nodes is obtained from the *what* field of the corresponding modification. Then, from this list, all geometric and geographic objects concerned by the modification can be deduced as explained in Cases 1 and 2.

From a performance point of view, these distinct documents are stored as database objects. Querying can be performed using the OQL object query language. For instance, document Why_1 can be retrieved as an instance of class ***why-doc*** using the following query expression

```
Select q -> reason
From q in WHY  //WHY is the persistency root of WHY-DOC objects
Where q.oid = WHY1
```

Alternatively, documents can be interactively examined by clicking on maps. In this case, the corresponding queries are generated when the interface captures the click action.

6 Prototype description

This section presents the prototype implemented at the University of Paris at Dauphine which supports interactive queries. Our proposal was implemented using the MapInfo GIS [MapInfo02], which was extended with the multiversion mechanism and additional database and interface facilities in order to query documentation and trace data evolution. Interactive queries were mapped to SQL.

The prototype supports two kinds of interactive modes, namely *query* and *update*. Within the query mode, users can navigate through the several stored map versions in a database and query different documentation aspects, at the three levels concerned. The update mode concerns any kind of data insertion, deletion or modification, either thematic or spatial. In this case, the user is prompted by the system to provide all relevant documentation to be stored in the database together with the actual data being changed. The language of the prototype interface is French. However, our explanation text will provide translation details, when relevant. We point out that all interactions are performed within an enhanced version of MapInfo. The basic interface is a MAPINFO interface with additional menu options. Thus, the user is not required to switch to a new system in order to query or update maps; all happens within the GIS environment.

6.1 Update mode

From a GIS point of view, users can edit or modify thematic or cartographic data as usual. The difference, however, is that the prototype allows the users to update data within the Documentation mode. If the user chooses to use the GIS within this mode, the prototype prompts the user to provide all necessary documentation by means of forms to be filled. Figure 9 gives a brief indication of some of the kinds of information the user is asked to provide at this mode, for cartographic and real-world levels.

The leftmost part of Figure 10 schematically shows the sequence of actions performed by the system to request from the user the documentation information needed. The right part of the Figure shows the geometric updates performed on the spatial data while the documentation forms are filled.

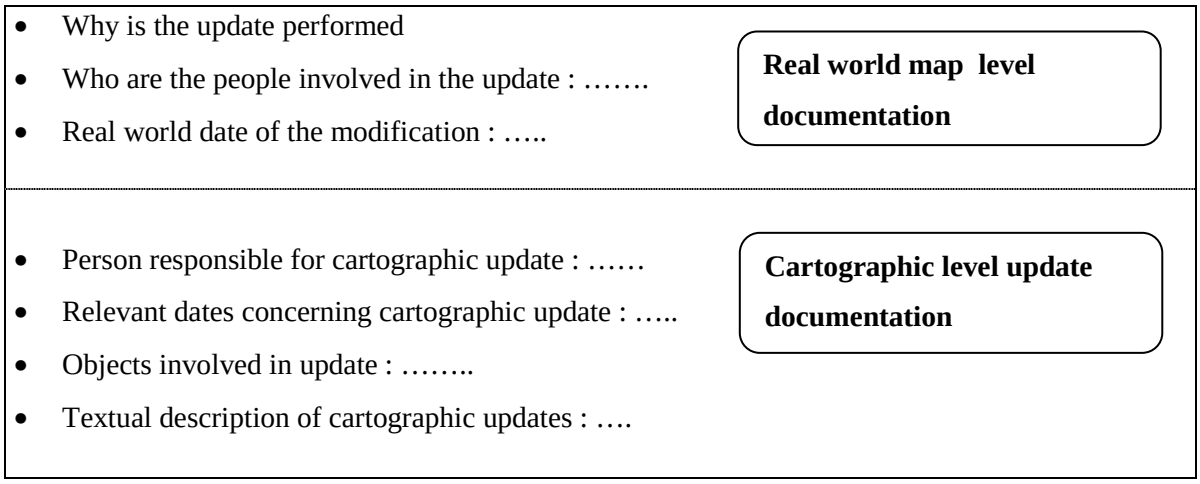


Figure 9 Partial description of documentation data required from user (update mode)

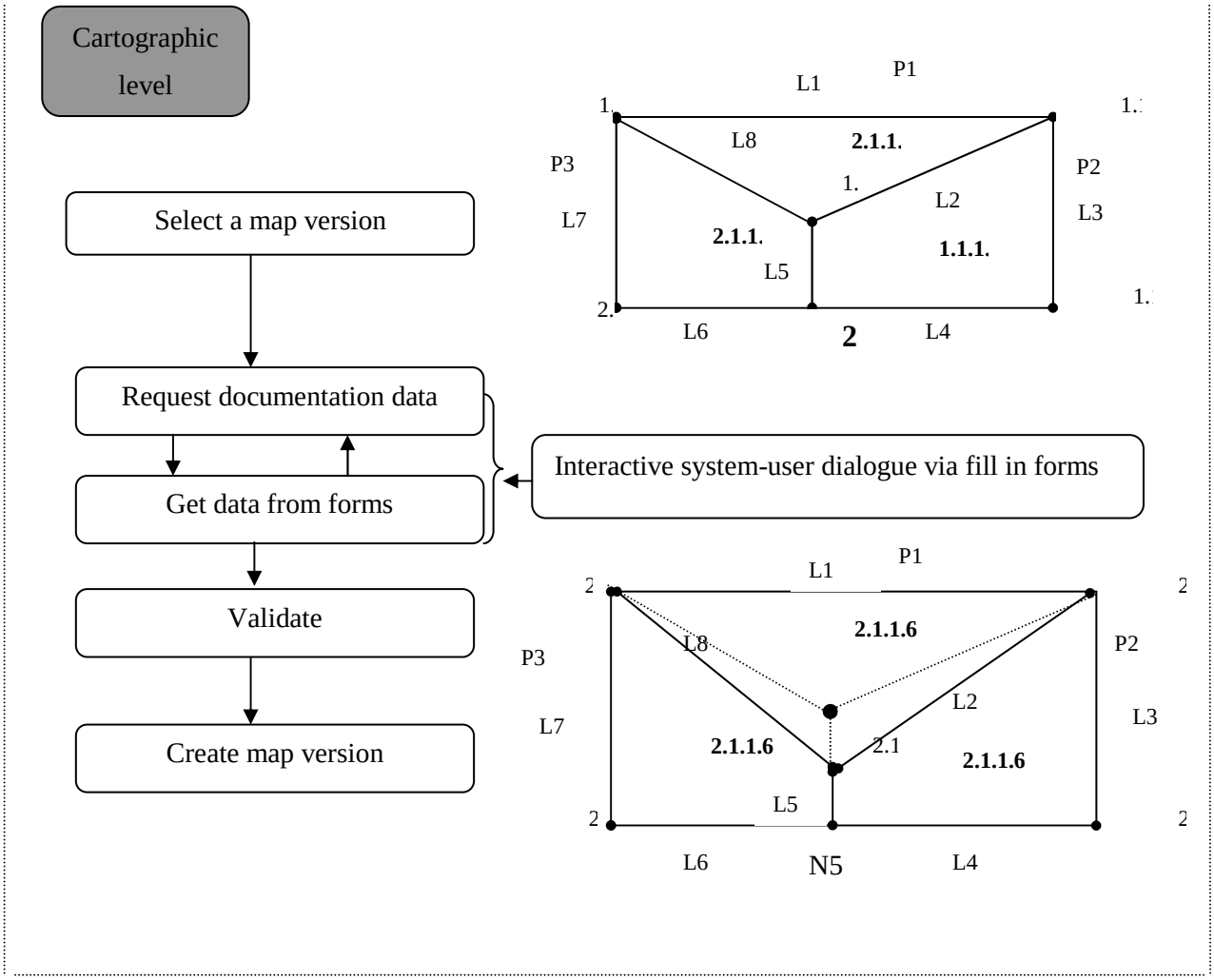


Figure 10. Workflow of actions requested from the user to document modifications (update mode)

Figure 11 shows the initial interaction screen for entering documentation. The menu bar is a MAPINFO menu (in French) extended with Version and Documentation sets of commands. Version commands allow manipulating and creating map versions. Documentation commands are issued to enter update and query modes.

Consider now that the user has chosen to document a map version. Figure 12 shows that the user has provided a date – 01/01/1996 - to indicate selection of the map version (*Selection d'une version de carte*). The user can choose any of the three documentation levels and four types of documents (respectively checklists at the left and right sides of the screen). The four types of documents are *why* (Pourquoi), *how* (Comment), *what* (Quelles – literally, which ones) and *other queries* (Autres). The text to the right helps the user select options within the query.

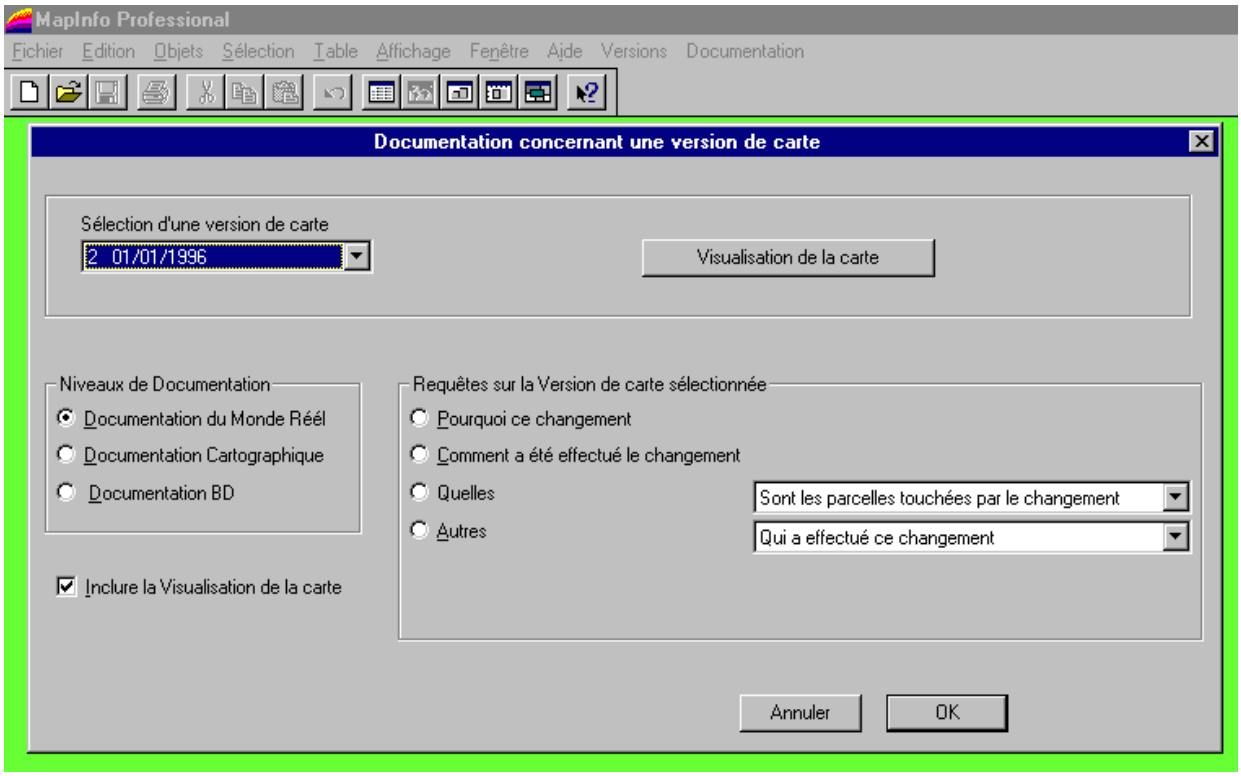


Figure 11. Initial screen for entering documentation (update mode)

The user can then select on a map the spatial items requiring updates, and indicate the kind of update. Figure 12 shows that the user selected «Thematic update » (Changement thématique) of three polygons (by clicking – see darker polygons). Figure 13 shows that the new values are being typed for some of the attributes of these polygons. After prompting for documentation, WHY-HOW-WHAT-WHO forms appear at appropriate moments, as indicated in Figure 14. The rest of this section is concerned with showing some details of the querying mode.

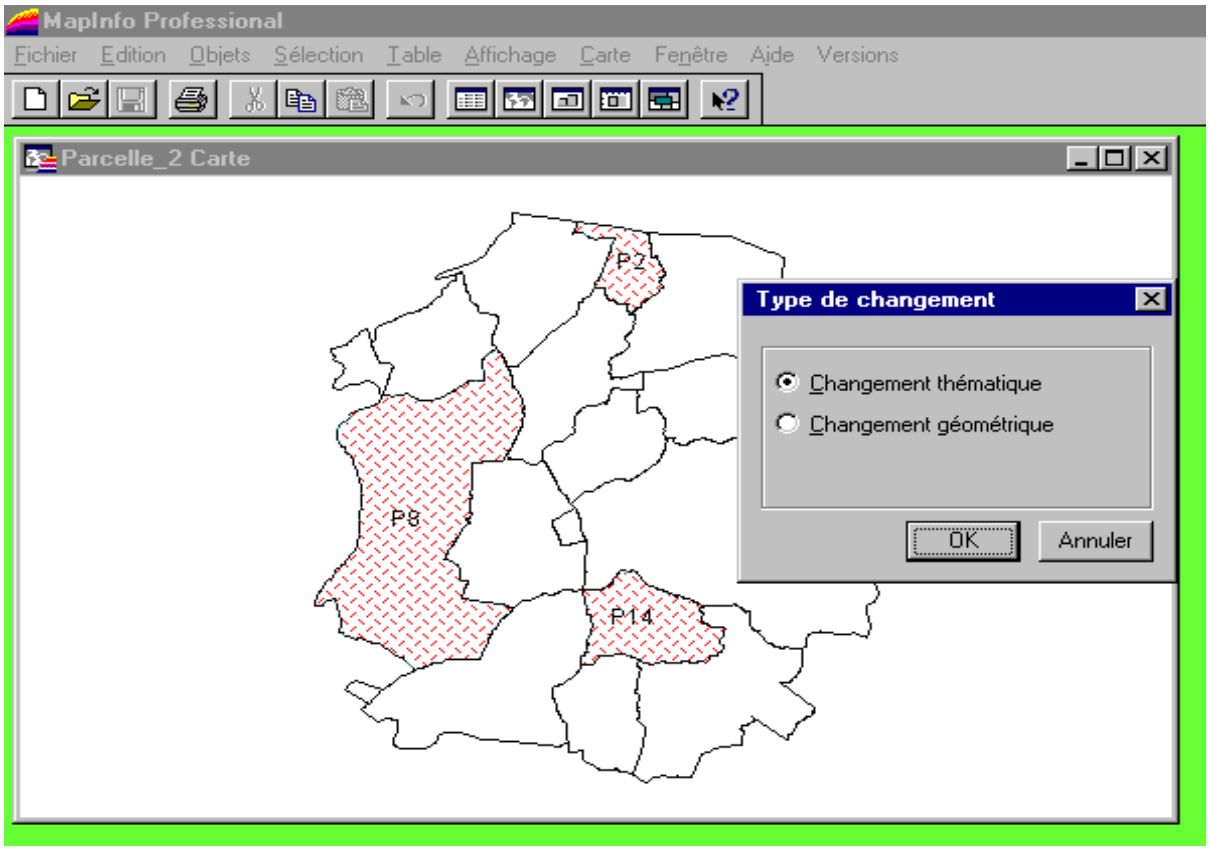
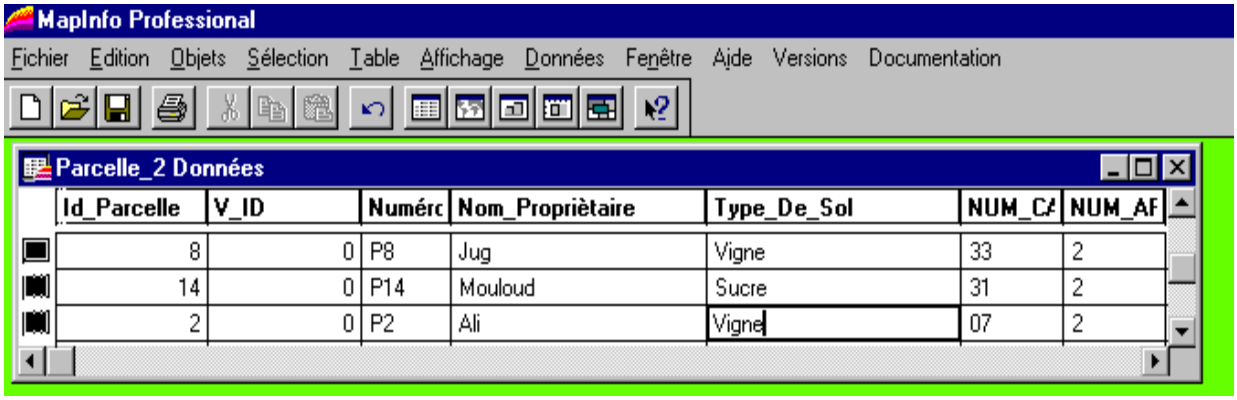


Figure 12. Creation of a new map version - thematic modifications



Id_Parcelle	V_ID	Numérc	Nom_Propriètaire	Type_De_Sol	NUM_C/	NUM_AF
8	0	P8	Jug	Vigne	33	2
14	0	P14	Mouloud	Sucre	31	2
2	0	P2	Ali	Vigne	07	2

Figure 13. Fragment of the thematic data table for the selected polygons (thematic modifications)

6.2 Query mode

In the query mode, the user first selects a map version and then performs queries concerning the documentation of either this map alone or this map and the precedent map version (i.e., to find out the differences). User interaction varies with the user's goals. Some queries require only clicking on menus or icons whereas others combine this interaction with typing of parameters (e.g., date, or author name). Finally, other queries are performed directly by interacting with cartographic displays. Let us simulate a typical user session.

The user starts a querying session by entering the *Documentation* module from an enhanced MapInfo menu – see Figure 12. This figure shows the initial screen for this module. Figure 14 displays the map version chosen, as well as subsequent interaction choices for querying the map. The Documentation menu, outlined, allows the user to choose among the following items (menu at the middle, from top to bottom): Choose a map version (*Choix d'une version de carte pour la documentation*), Indicate the objects modified in a map version, Real world documentation, Cartographic documentation, Database documentation, Differences between a version and other versions. The figure shows that the user selected to see the Documentation of the real world (*Documentation du Monde Réel*).

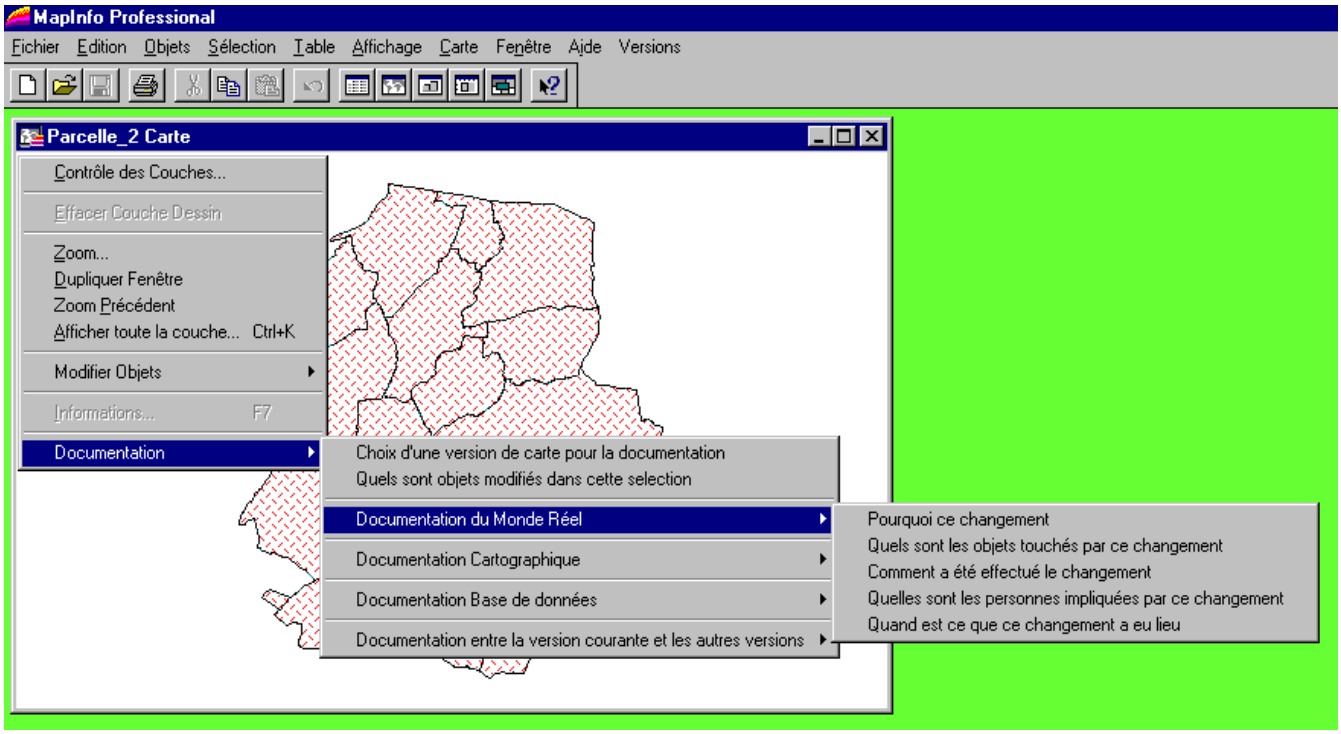


Figure 14. Query mode - real-world documents

Within this documentation item, the user can then select to see (leftmost menu, from top to bottom) the following: *Why* this change (Pourquoi ce changement), objects affected by the change (Quels sont …), *how* (Comment) the change was made, *who* (Quelles…)are the people involved in this change and *when* (Quand) this change occurred in the real world. Similar menus can be activated for the Cartographic and Database documentation levels, covering the same kinds of queries.

These queries concern documentation for an entire map version. Alternatively, the user can select objects within a map and ask to see which of these objects were changed from a previous version – see Figures 15 and 16. Figure 15 indicates with a different background *color* the user query (the parcels selected via clicks by the user). Figure 17 shows that polygonal objects P3, P2 and P6 were the ones actually modified within the selected set.

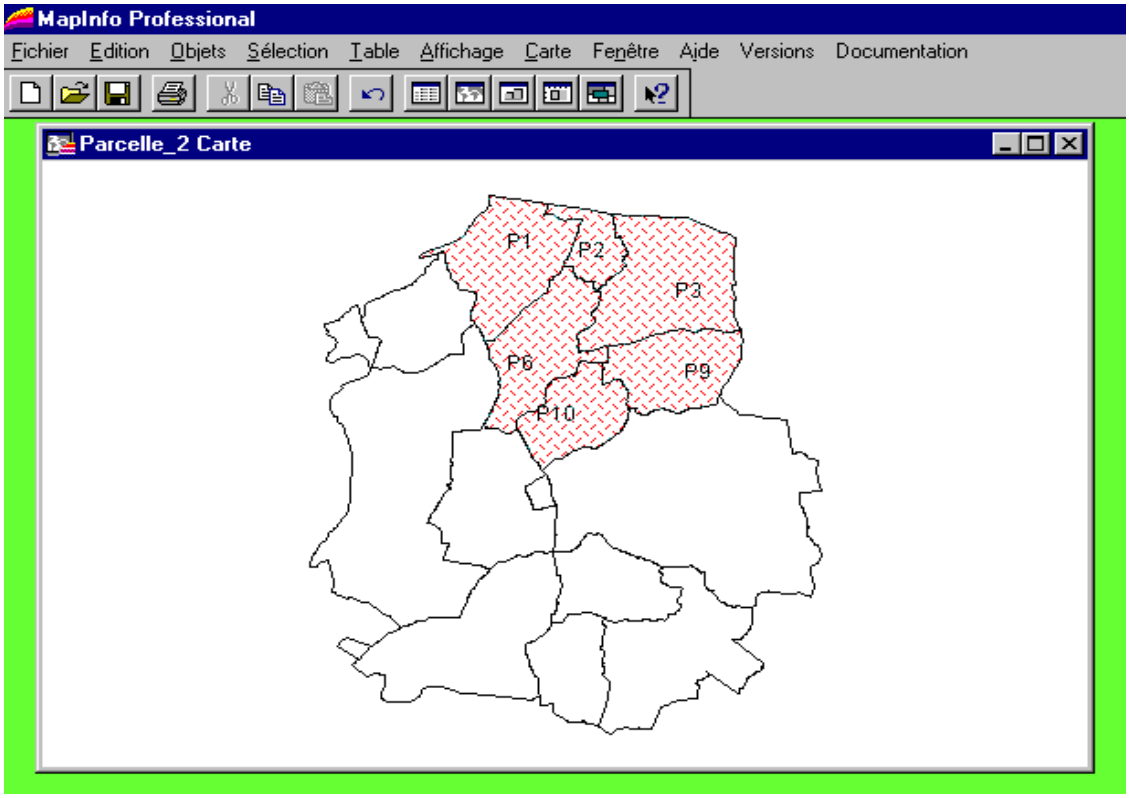


Figure 15. Query mode - selection of objects from a map version

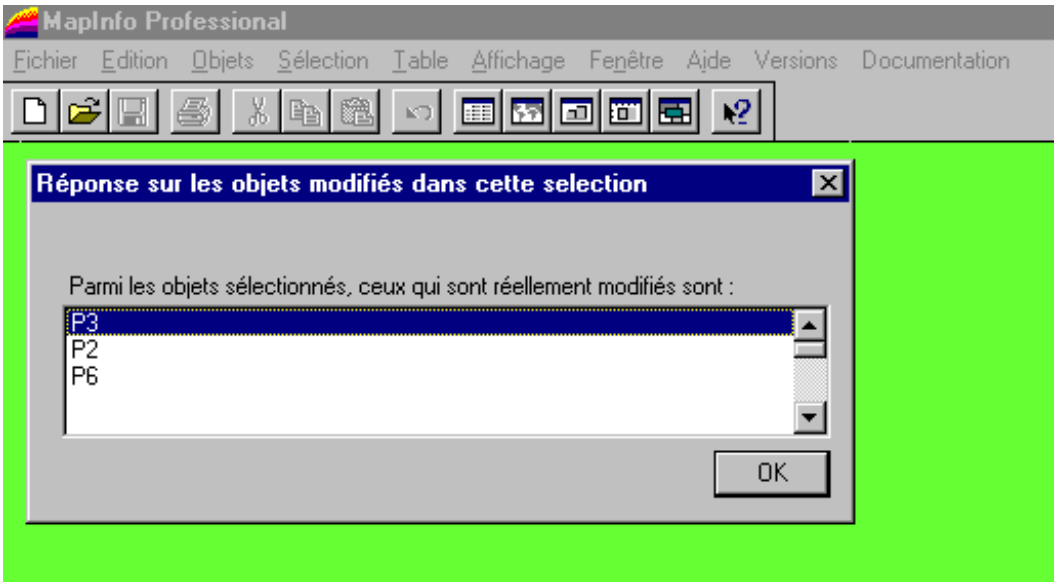


Figure 16. Query mode – information on which objects were changed

Now, the user selects Polygon P3 (see Figure 16) to find out more about the changes it underwnet. On clicking the OK button, Figure 17 appears. It describes textually the actual modifications of P3 in this version, as compared to its precedent map version. The information given is « The description of the map changes are the following – modification of parcels P2, P3 and P6 by downward displacement of node N3". Notice that this is the text entered by the person who performed the update, and it is assumed that the user is an expert who knows what to enter. This figure also indicates the name of the person who performed this specific update (Syphax) and the date in which this update was performed (07/09/2001).

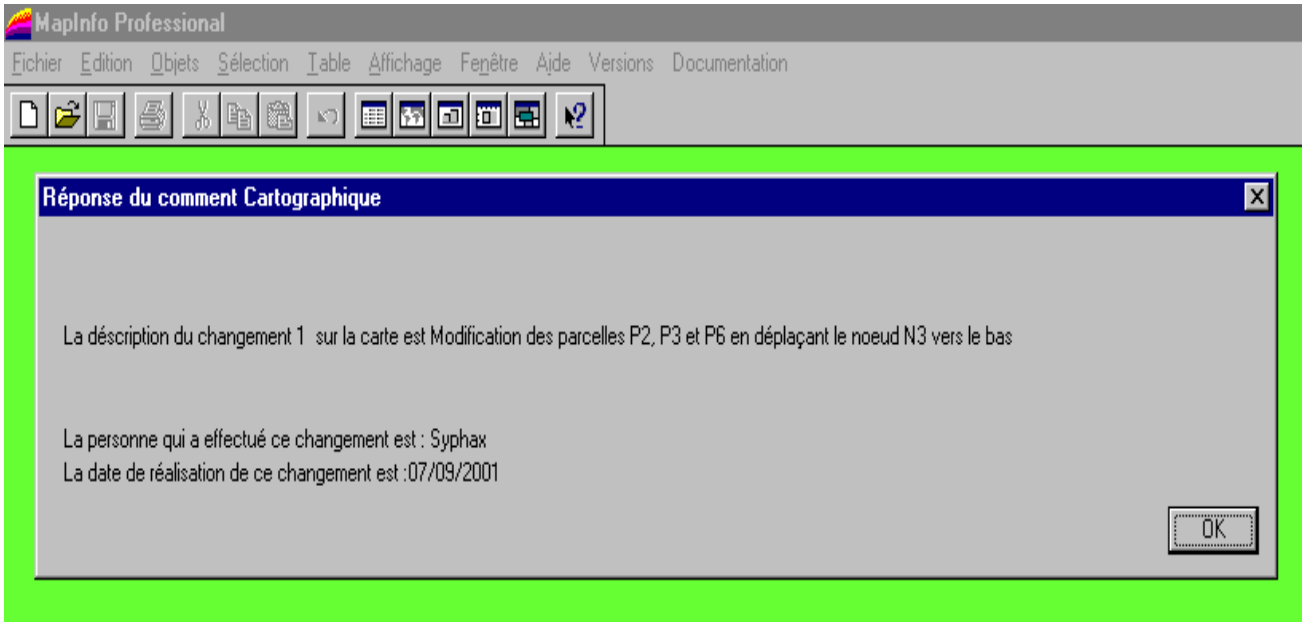


Figure 17. Query mode: textual description of how-, who-, and when-documents

Last but not least, the user may also find out more about updates by clicking on document buttons. Figure 18 shows an example of this kind of interaction. The textual information box labeled Documentation that covers part of the concerns all real world events that caused the map change. The box provides click-on buttons to two real world objects concerning this modification (2B1 and 3B1). Clicking on each of these buttons will give information on the corresponding objects. The bottom part of the Figure concerns information on 2B1, which happens to concern Polygon P2 – identifier, owner name, kind of soil, address. The text covering the map describes the following real world events: it informs that the event recorded is the sale of Parcel 3B1, owned by M. Samir, to M. Ali, owner of parcel 2B1. Furthermore, it explains the parcel was sold on 15/01/2001 and mortgaged by M. Syphax on 30/01/2001. Finally, it indicates that the people involved in the sales transaction are M. Idir and M. Ali. For more details on this prototype the reader is referred to [Pee01].

7 Conclusions

Documenting changes in a spatiotemporal database is an important feature which has, however, been neglected so far by the research community. This paper focused on documentation management in this context at many levels of abstraction. We introduced two orthogonal notions. The first one is the notion of *level of changes* in a spatiotemporal application. We considered three levels: real-world level, cartographic level, and database level, all corresponding to various user contexts: real world concerns (e.g., a surveyor), cartographic presentation (e.g., city planner), and database management (e.g., DBA). Changes may be introduced at each level. The second notion introduced here is the *type of documentation* associated with changes. Beside classical metadata, which has been a common notion in information systems for years, we introduced other types of documents associated with changes, namely *what-*, *how-*, and *why-documents*. Each of them may encompass classical metadata such as *when* and *who*. We exemplified our approach through an application centered on urban development.

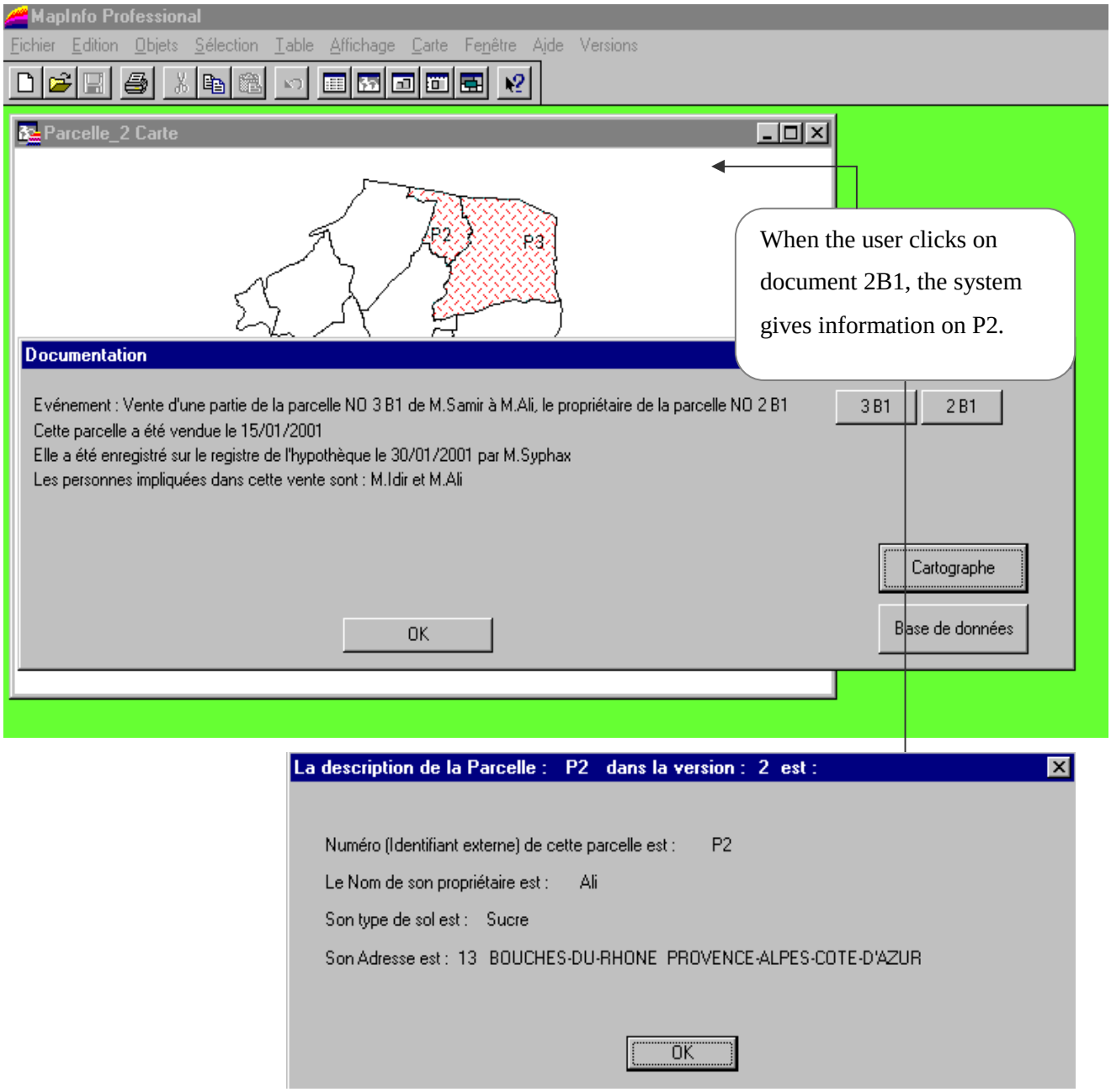


Figure 18. Real-world documentation of objects displayed on the map

In our approach, why-, how- and what-documents are stored as complex multiversion objects in the database. They are associated with geographic objects in order to document the reasons, the procedures, and the originators of changes. Our framework can be extended to accommodate as many kinds of user and document fields as needed. It caters to a real user need: to better understand the evolution of real world phenomena. Dynamic documentation can also be used to support exchange of geographic data, and is thus a step towards interoperability.

This proposal was implemented in a prototype at the University of Paris at Dauphine (Paris IX), within the LAMSADE laboratory. Both the development and testing of this version were followed by experts from the French National Cartographic Institute (IGN). Some of the interface facilities were included to adjust to their experience in terms of real user needs. One of the nice features of this prototype is its seamless integration with the underlying GIS (MapInfo), which does not require the user to quit the GIS environment to either document data updates or to query map versions.

Even though this paper is centered on geographic urban development applications, the rationale used here can be applied to other spatiotemporal domains – e.g., environmental planning. What is needed in each case is a good understanding of users' needs in order to determine the relevant change levels and the necessary documentation to be associated at each level. This kind of requirement analysis is an important trend for future work.

There are, moreover, several other kinds of need for change documentation. Another important issue, briefly mentioned in Section 3, is that of cartographic production. Geographic data are provided to several kinds of clients (e.g., power company, street maintenance authorities, cadastral offices), who update portions thereof according to their needs. At the end, these data need to be consolidated, to provide a global view of the world (and, for cartographers, a single map version). This consolidation requires analyzing the individual updates, often containing conflicting information. Here, again, proper dynamic documentation will help manage these inconsistencies. We are currently considering this kind of integration.

Acknowledgements. The work reported in this paper was partially financed by a bilateral cooperation program CNPq/Brazil and CNRS/France. French grants also include help from the Université Paris IX. Brazilian financing was complemented by research grants from CNPq and FAPESP, and by the PRONEX/MCT Project on Advanced Information Systems (SAI) and the CNPq WebMaps project. The authors gratefully acknowledge the help of M. Badard from IGN and Mouloud Hedjar for his work in developing the prototype. Part of the work reported here has appeared in [PMJ01].

8 References

[AIL98] Ailamaki, A., Ioannidis, Y., and Livny, M.: Scientific Workflow Management by Database Management. In *Proceedings 10th IEEE International Conference on Scientific and Statistical Database Management*, 190-201, 1998

[AHV95] Abiteboul, S., Hull, R., and Vianu, V.: *Foundations of Databases.* Addison-Wesley, Reading, MA, 1995

[BA94] Böhm, K. and Aberer, K.: Storing HyTime documents in an object-oriented databases. In *Proceedings of the third Iinternational Conference on Information and Knowledge Management (CIKM)*, 26-33, 1994

[BAD98] Badard, T.: Towards a generic updating tool for geographic databases. In *Proceeding of the International GIS/LIS Conference*, 352-363, 1998

[BFS00] Buneman, P., Fernandez, M., and Suciu, D.: UnQL: A query language and algebra for semistructured data based on structural recursion. *The VLDB Journal*, 9(1)1-110, 2000

[BJ93] Bauzer-Medeiros C., Jomier G.: Managing Alternatives and Data Evolution in GIS. In *Proceedings of ACM Workshop on Advances in GIS (ACM/GIS)*, ACM Press, 1993

[BJKS01] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. TIMECENTER Technical report TR-66

[CACS94] Christophides, V., Abiteboul, S., Cluet, S. and Scholl, M.: From structured documents to novel query facilities. In *Proceedings of the ACM Conference on the Management of Data* (ACM SIGMOD), 313 – 324, 1994

[CD00] Cassati, F. and Discenza, A. Supporting Workflow Cooperation Within and Across Organizations. In *Proceedings of the International ACM Symposium on Applied Computin*g, Vol. 1, 196-202, 2000

[CDF+00] Cattaneo, F., Di Nitto, E., Fuggetta, A., Lavazza, L. and Valetto, G.: Managing software artifacts on the Web with Labyrinth. In *Proceedings of the International Conference on Software Engineering*, 746 – 749, 2000

[CJ90] Cellary, W. and Jomier, G.: Consistency of versions in object-oriented databases. In *Proceedings of the Very Large Database Conference (VLDB)*, 1990

[CJ00] Cellary, W. and Jomier, G.: The Database Version Approach. *Networking and Information Systems Journal*, *Hermes*, Vol. 3, No 1, 177 - 214.

[CR94] Christophides, V. and Rizk, A.: Querying structured documents with hypertext links using OODBMS. In *Proceedings of the ACM European Conference on Hypermedia Technology*, 186-197, 1994

[CT95] Claramunt, C. and Thériault, M.: Managing time in GIS: an event oriented approach. In *Recent Advances on Temporal Databases*, Clifford, J. and Tuzhilin, A. (eds). Lecture Notes in Computer Science No. 1021, Springer Verlag, Berlin/Heidelber/New York, 23-42, 1995

[DL96] Dattolo, A. and Loia, V.: Collaborative Version Control in an Agent-based Hypertext Environment. In *Information Systems*, 21(2):127-145, 1996

[DELS99] Dourish, P., Edwards, W. K., LaMarca, A. and Salisbury, M.: Presto: An experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction*, 6 (2), 133 – 161, 1999

[DEL+00] Dourish, P., Edwards, W. K., LaMarca, A., Lamping, J., Petersen, K., Salisbury, M., Terry, D.B. and Thornton, J.: Extending document management systems with user-specific active properties. *ACM Transactions on Information Systems*, 18 (2), 140 – 170, 2000

[EGK+01] Eick, S.G., Graves, T.L., Karr, A. F., Marron, J.S., and Mockus, A.: Does Code Decay? Assessing the Evidence from Change Management Data. *IEEE Transactions on Software Engineering*, 27(1), 2001

[FGN00] Forlizzi, L., Güting, R.H., Nardelli, E., and Schneider, M.: A Data Model and Data Structures for Moving Objects Databases. In *Proceedings of th. ACM Conference on the Management of Data (SIGMOD),* 319-330, 2000

[FLP+03] Fileto, R., Liu, L., Pu, C., Medeiros C. B. and Assad, E.: POESIA – an Ontological Workflow Approach for Composing Web Services in Agriculture. In *The VLDB Journal*, 2003.

[Fons01] Fonseca, F. *Ontology-driven Geographic Information Systems*. PhD Thesis, University of Maine, USA, June 2001

[FMM02] Furuta, R., Munson, E. and Maletic,J. Editors. *Proceedings of the 2002 ACM Symposium on Document Engineering*, ACM Press, 2002.

[GJ94] Gançarski S., and Jomier, G.: Managing Entity Versions within their Context: a Formal Approach.In *Proceedings of Database and Expert Systems Applications Conference (DEXA)*, 1994

[GML02] OpenGIS.: Geographic Markup Language GML 2.0. http://opengis.net/gml/01029/GML2.html. OGC Document Number: 01-029, February 2001. (As of March 2002)

[Grice02] Grice,R. Some reflections on the Emergence of a Profession. ACM Journal of Computer Documentation, 26(3):126-129, 2002.

[GV98] Günther, O. and Voisard, A. Metadata in Geographic and Environmental Data Management. In *Managing Multimedia Data: Using Metadata to Integrate and Apply Digital Data*, W. Klas, A. Sheth (Eds.), McGraw-Hill, San Francisco/New York, 1998.

[LAN92] Langran, G.: *Time in Geographic Information Systems*. Taylor and Francis, London, 1992

[MapInfo02] The MapInfo GIS Software home page. URL: http:www.mapinfo.com, 2002

[MC96] Moran, T.P., and Carroll, J.M.(eds.): *Design Rationale: Concepts, Techniques and Use.* Laurence Erlbaum Associates, 1996

[Pee01] Peerbocus, M.A. *Management of spatiotemporal evolution in a geographic database*., Phd thesis, University of Paris at Dauphine (Paris IX), France, December 2001

[PEU94] Peuquet, D.J.: It's about time: A conceptual framework for the representation of the temporal dynamics in geographic information systems. *Annals of the Association of American Cartographers*, 84(3), 441-461, 1994

[PJB02] Peerbocus, M.A., Jomier, G. and Badard, T.: A methodology for updating geographic databases using map versions: *10th International Symposium on Spatial Data Handling* , 2002.

[PMJ01] Peerbocus,M. A, Medeiros, C. B., Jomier, G. and Voisard, A. : Documenting Changes in a Spatiotemporal Database. *Proc. XVI Brazilian Database Symposium*, 10-24, 2001

[PT00] Pfoser, D., and Theodoridis Y. : Generating Semantics-Based Trajectories of Moving Objects, In *Proceedings of the International Workshop on Emerging Technologies for Geo-Based Applications*, 2000

[PT98] Pfoser, W. and Tryfona, N.: Requirements, definitions and notations for spatiotemporal application environments. *Proceedings of ACM Workshop on Advances in GIS (ACM/GIS)*, ACM Press, 1998

[RAB+00] Roddick, J., Al-Jadir, L., Bertossi, L. et al.: Evolution and change in data management – Issues and directions. *ACM SIGMOD RECORD* 29(1), 21-26, 2000

[RS95] Rusinkiewicz, M. and Sheth, A.: Specification and Execution of Transactional Workflows. In W. Kim, (ed.): *Modern Database Systems. The Object Model, Interoperability and Beyond*, ACM Press, 592-620, 1995

[RSV01] Rigaux, P., Scholl, M., and Voisard, A.: *Spatial Databases – With Application to GIS.* Morgan Kaufmann, San Francisco, 432p. 2001

[SCL99] Sperry, L., Claramunt, C. and Libourel, T.: A lineage Metadata Model for the Temporal Management of a Cadastre Application. In Spaccapietra, S. and Pissinou, N., (eds.): *Proceedings of the International Workshop on Spatio-temporal Models and Language*, 1999

[SS98] Schmidt, K. and Simone, C.: Taking the distributed nature of cooperative work seriously. In *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, 295-301, 1998

[SWCD] Sistla, A.P., Wolfson, O. Chamberlain, S. and Dao, S.: Modeling and Querying Moving Objects. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE)*, 1997, 422-432

[TRY98] Tryfona, N.: Modeling Phenomena in Spatiotemporal Applications: Desiderata and Solutions. In *Proceedings of the 9th International Conference on Database and Expert*

Systems Applications (DEXA). Lecture Notes in Computer Science Springer Verlag, Berlin/Heidelberg/New York, 1998

[UOM+99] Uitermark, H. van Oosterom, P., Mars,N. and Molenaar, M. Ontology-Based Geographic Data Set Integration. In Proceedings of the Spatio-Temporal Database Management, International Workshop STDBM'99, Edinburgh, Scotland, September 10-11, 1999, Proceedings. Lecture Notes in Computer Science, Vol. 1678, Springer, 1999

[VCP99] Valsecchi, P., Claramunt, C. and Peytchev, E.: OSIRIS: An International Operable system for the Integration of Real Time Traffic Data within GIS. In R. Laurini (ed.), *Proceedings of the 1st Telegeoprocessing Conference*, 40-46, 1999

[VBJ00] Voisard, A., Medeiros, C. and Jomier, G.: Database Support for Cooperative Work Documentation. In *Proceedings of the 4th International Conference on the Design of Cooperative Systems*, Sophia-Antipolis, 2000

[WOR94] Worboys, M. F.: A Unified Model for Spatial and Temporal Information. *The Computer Journal*, 37(1):25-34, 1994