A Methodology for Updating Geographic Databases using Map Versions

Ally Peerbocus[1], Geneviève Jomier[1], Thierry Badard[2]

LAMSADE, Université Paris Dauphine, Pl. Mal de Lattre, 75775 Paris Cedex 16, France. Email: (jomier,peerbocus)@lamsade.dauphine.fr[1]
Institut Géographique National, COGIT, 2 à 4 Avenue Pasteur, 94165 Saint Mandé Cedex, France. Email: Thierry.Badard@ign.fr [2]

Abstract

This paper addresses issues concerning the exchange and integration of geographic data between producers and users. Once a producer has delivered a geographic database to a user, who then uses it as a reference for specific applications, the database may be updated on both sides. Consequently, the integration of future updates - delivered by the producer -in the user's geographic database is a complex operation due to possible conflicts between updates performed by both parties. The resulting database may become inconsistent and the user's added information may be lost. Users therefore need mechanisms to help them in the process of update integration. This paper provides a methodological framework for the updating of geographic databases. It relies on a multi-version GIS, allowing an automatic detection of conflicting updates between two map versions.
Keywords: geographic database, multi-version GIS, updates, map version

1 Introduction

Geographic Information Systems (GIS) are increasingly used in a large spectrum of applications. Since, implementing such systems is complex, users, generally, purchase reference geographic data from producers in order to set up their GIS. For instance, a transportation company purchases from a producer a geographic database representing the road network of a given region for its transport planning application. For the user, the database delivered by a producer serves as a *reference map,* on which to develop the application. Geographic data producers are responsible for producing and maintaining up-to-date databases, delivered to users on a regular basis. Meanwhile, users may need to add information on the

reference map or update the map to take into account real world changes, or information they are interested in; for instance bus lines and bus stops (see Fig. 1).

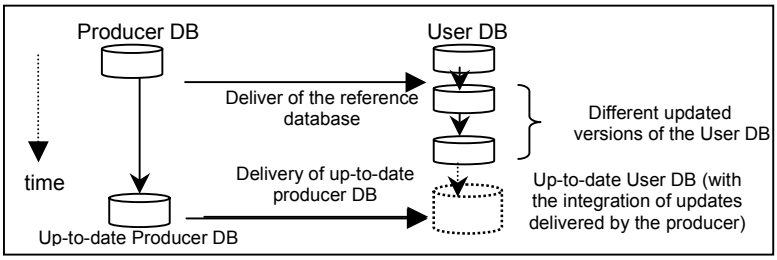


Fig. 1. Integration of updates

Consequently, the integration of the producer's updates in the user's database may result in conflicts with those already performed by the user as described in (Badard 1998). For instance, if the producer changes the location of a road, to increase information accuracy, the bus line and bus stops along the road must be changed too, otherwise the user's database is in an inconsistent state.

The first step for a proper integration of these updates requires the identification of the updated objects in the user's database as well as those in the producer's database in order to detect possible conflicts. At present, producers generally deliver a whole up-to-date database to the user. Even if the percentage of change is small between two updates, current GIS do not provide any mechanism for the extraction of updates between two versions of a database representing the same area at two different times (Raynal 1996). Several techniques have been proposed to achieve this purpose. They are based on the exhaustive comparison of all the objects in the two versions of the database (Badard 1998), relying on geographic data matching algorithms (Lemarié *et al* 1996). Such an approach, well adapted in a general context where no hypothesis on the data model is assumed, is based on complex algorithms and needs considerable effort to be implemented.

This paper proposes a mechanism for an automatic detection of conflicting updates performed in two different versions of a database. It is based on the version approach proposed in (Gançarski *et al* 1994). This paper is organised as follows: section 2 describes the context through an example and presents an overview of related work; section 3 details our approach and the way it is implemented; section 4 concludes the paper.

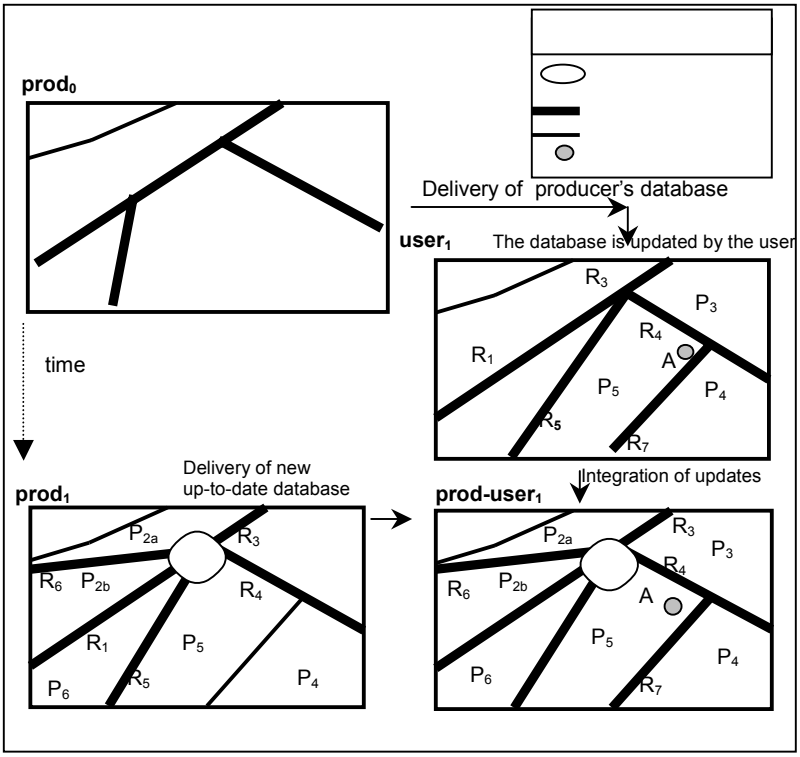
2 Context and Related Work

Before presenting an overview of related work, this section describes an example of exchange of geographic data between a producer and a user, illustrating conflicts between producer's and user's updates.

2.1 Context

The context used to illustrate the producer/user scenario is based on a road network application relying on Georoute®, a database produced by the IGN (the French National Geographic Institute) and dedicated to car navigation services. Fig. 2 depicts part of the producer's map, identified as $prod_0$, which is delivered in a geographic database to the user. The map shows the state of the modelled road network, identified as R_1 to R_5, and land parcels, identified as P_1 to P_6. Fig. 2 also represents the new up-to-date producer's map, identified as $prod_1$, reflecting the new state of the road network after:

1. the construction of a new road R_6, splitting parcel P_2 into P_{2a} and P_{2b};
2. the deletion of R_2;
3. the construction of a new roundabout, identified by P_7, at the junction of roads



R_1, R_3, R_4, R_5 and R_6 implying the update of all these road sections.

Fig. 2. The different states of the producer and user maps

On the user side, the initial geographic database has been updated to obtain a new version of the map, identified as $user_1$, different from $prod_1$. The updates performed in $user_1$ are illustrated in Fig. 2:

1. the update of roads R_1 and R_5;
2. the deletion of R_2;
3. the creation of a road section R_7 at the limit between P_4 and P_5;
4. the creation of an antenna, A, representing the user mobile phone company.

Finally, the map identified as $prod\text{-}user_1$ corresponds to what the user would like to obtain after the integration of updates from $prod_1$ and $user_1$.

In general, updates in a geographic database can be performed on both the schema and the objects. Updates on an object represent its evolution - i.e., creation, deletion, and thematic and/or geometric changes. Schema updates are required when new kinds of objects appear as bus-stops, mobile phone antennae which are specific objects for the user and whose representation is not provided in the producer's schema of the database. When the producer delivers the new up-to-date database to the user, ($prod_1$ in Fig. 2), the user must then consider this new information for an update to his current database ($user_1$ in Fig. 2). He cannot just replace the old reference database with the new one since the resulting database may be in an inconsistent state; for instance, if a road section is enlarged in $prod_1$, he should displace any antenna on this road. Besides, specific information added by the user may be lost and some of the updates may be in conflict with the producer's updates, like road R_7, which exists only for the user.

2.2 Sources of Conflicting Updates

Several situations may result in conflicts between updates separately performed by the producer and the user. First, conflicts may be due to the update of the same object by both actors, defined as a 1-to-1 update, such as parcels P_5, P_6, and roads R_1, R_5, updated by the producer and the user. Secondly, conflicts are very likely to occur in case of group updates like:

- 1-to-N update: one object is deleted and is replaced by several objects; e.g., the splitting of parcel P_2 into P_{2a} and P_{2b} in map $prod_1$,
- N-to-1 update: several objects are deleted and replaced by one object; e.g., the merge of R_1 and R_2 in $prod_0$ resulting in R_1 in $prod_1$,
- M-to-N update: several objects are deleted and in their place several other objects are created; e.g., the creation of the new roundabout P_7 by the producer in $prod_1$.

A complete taxonomy of conflicts hindering the updating of geographic database is described in (Badard 1998).

2.3 Related Work

Several techniques concerning the detection of update differences between two geographic databases, modelling the same region, have been proposed. They are generally based on the comparison of the geometry (Devogèle 1998. (Badard *et al* 1999) proposes to isolate these differences by using the "geographic data-matching method", which goes through every database object in a region and computes the correspondence relationships between objects, from their geometry stored in the two versions. The resulting relationships can be classified, considering their cardinality: a) 1-to-0 or 0-to-1: an object of one database does not match with any object of the other one; b) 1-to-n or n-to-1, with n>0: an object of a database matches with one or several objects of the other one; c) n-to-m, with n>1 and m>1: several objects of a database match with several objects of the other one.

The correspondence relationships are then analysed and updated objects are classified according to the evolution they have undergone - the typology is defined in (Badard 1998b) -, and which can either concern the object level only, or both schema and object levels. Furthermore, new delivery modes dedicated to the exchange of updating information have been proposed (IHO96, Poupart-Lavoie 1997, Badard 1998b, Badard *et al* 2001) to help the integration process in databases.

Together with these methods for the detection of updates in geographic databases, propagation mechanisms of these effects have been proposed in a multi-scale database context (Badard 2000, Kilpelaïnen 1997, Uitermark *et al* 1998). In all these papers, no hypothesis is made about the data model used and only a general solution is provided. It's clear that detecting changes in the whole database requires tremendous efforts and sophisticated algorithms.

A proper updating of a user's database implies the preservation of the integrity of the map delivered by the producer. This means that users must perform their updates on versions of the reference map, and the comparison of the different map versions should be possible in order to detect changes between two map versions. However, as far as we know, in the geographic context only one technical paper of *SmallWorld* GIS (Easterfield *et al*) has focused on the management of version in GIS. Few implementation details, however, have been provided.

We propose a methodology for the updating of geographic databases called *Updating by Map Versions* (*UMV)*. It is based on the use of a multi-version geographic database as described in (Bauzer *et al* 1993), which supports the management of map versions. The detection of conflicting updates is based on the automatic identification of all the database objects, and not on comparisons performed on the geometry of geographic objects as proposed in (Badard 2000). The next section deals with the main features of the UMV methodology and describes how it is implemented.

3 Updating by Map Version Methodology

From here on, the producer database will be addressed followed by the user database as *producer-DB* and finally, the *user-DB*. Initially the *producer-DB* contains one version of the map representing the modelled geographic area. This version is identified as $prod_0$. The UMV methodology, comprised of four steps, is illustrated in Fig. 3 and detailed in the next sub-sections.

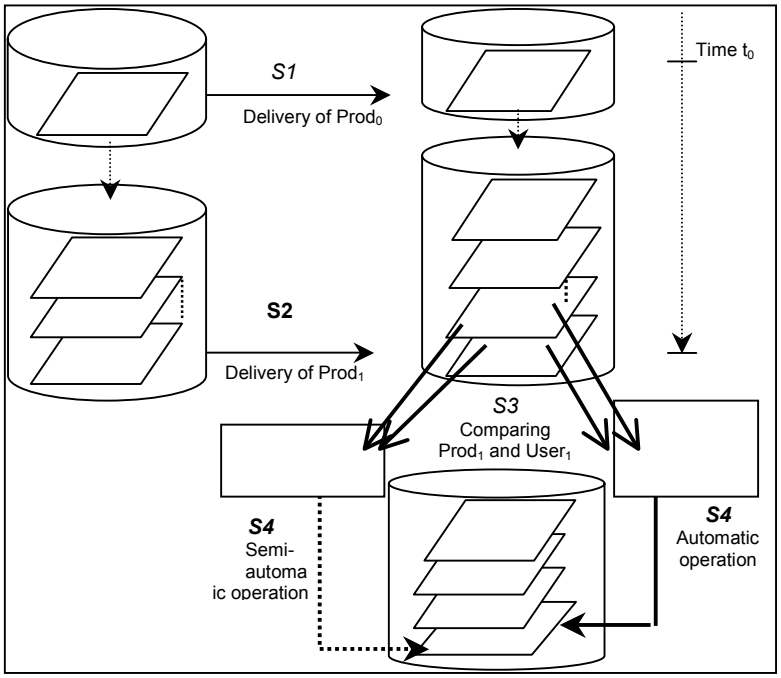


Fig. 3. The steps comprising the UMV methodology

These steps are:

S1. At time t_0, the producer delivers to the user the initial reference map version, $prod_0$. This map version is inserted in *user-DB*, and identified as $prod_0$. It serves as the reference map for the user. $Prod_0$ is preserved in the delivered state, frozen in both *user-DB* and *producer-DB*. Updates are performed on successive map versions generated from $prod_0$ on both sides. The newly generated map versions are identified as $prod_{0,i}$ in *producer-DB* and $user_{0,j}$ in *user-DB*.

S2. Then, at time t_1, an up-to-date database version is delivered by the producer to the user, identified as $prod_1$. The user's map version at this moment is identified as $user_1$. The delivered map version, $prod_1$, is inserted in *user-DB*.

S3. The map versions $user_1$ and $prod_1$ in *user-DB* are compared to detect conflicting (from an updating point of view) and non-conflicting data.

S4. Finally, using the strategy, discussed in section 3.3, a new map version is created, *user-DB*, to include part or all of the user and producer's updates. An automatic integration of data can be used for non-conflicting updates. A semi-automatic operation is needed to integrate the conflicting data and the consequences of the conflicts in the final user map version, identified as $prod\text{-}user_1$. For instance, the bus line of the user present in $user_1$ is moved to follow the new location of a road recorded in $prod_1$.

The UMV methodology is supported by a multi-version database (Gançarski *et al* 1994), which is introduced in the next sub-section. We assume that both the producer and the user have a multi-version geographic database. The underlying version mechanism is now detailed.

3.1 The Multi-version Geographic Database

Two levels are distinguished: the user level and the database level. At the user level, for external users, the multi-version database appears as a set of independent map versions, representing the same area, which coexist within the same storage space. This means that each map version can be managed (read and updated) separately and independently from the other map versions of the same region. A new map version is always generated or *derived* as a copy of an existing map version.

At the database level, however, one important feature of the multi-version database mechanism is that it automatically allows keeping track of all database objects that compose a consistent map version. Thus, several versions modelling the same real world object may coexist in the database. This gives origin to the concept of *multi-version object*, which is a "repository" of all versions of a given object – i.e., multi-version object O encapsulates the mapping between all different states of an object and the corresponding map versions. One of the main advantages of the mechanism is that it minimises storage occupancy and avoids redundancy while storing multiple map versions. The derivation relationships among the distinct map versions are recorded in a structure called *map version tree*. Updates to objects in one map version are handled without side effects on other map versions, due to an appropriate management of internal version identifiers. To obtain the value of an object in a 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 $\perp$, meaning it does not exist. When an object O in a map version v is involved in a group operation - splitting (e.g. the splitting of parcel P_2 into P_{2a} and P_{2b} in map version $prod_1$) or merging operation -, the link between O and the resulting object(s) is stored in a *genealogy graph* (Sperry *et al* 1999). 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 initialised 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 evolution of geographic objects.

The system uses the internal identifier of objects to follow the evolution of objects through time. Internal identifiers are managed only by the system, conversely to external identifiers, which are managed by users. For further details on this approach, the reader is referred to (Cellary *et al* 1990, Bauzer-Medeiros *et al* 1993, Gançarski *et al* 1994, Cellary *et al* 2000).

3.2 Illustration of the Multi-version Geographic Database

The top of Fig. 4 shows a part of the producer multi-version database corresponding to map versions $prod_0$ and $prod_1$ described in Fig. 2. For the sake of clarity, we ignore the intermediate map versions between $user_0$ and $user_1$ in *user-DB*, considering that the database is composed only of $user_0$ and $user_1$. Each multi-version object in the figure is represented by a table with two columns: *MV-id* (for multi-version identifier) and *Value*. Parcel P_6 has a different value for each of the two map versions: *valp6* in $prod_0$ and *valp6a* in $prod_1$. The Parcel P_2 in $prod_1$ has been split and replaced by parcels P_{2a} and P_{2b} as illustrated by the genealogy tree. Parcel P_1 has only one value, represented by *valp1*, corresponding to map version $prod_0$. According to the implicit sharing rule and the producer's map version tree, *valp1* is also the value of P_1 in map version $prod_1$.

The bottom part of Fig. 4 shows a part of the user's multi-version database. Road section R_7 and the antenna A have only one value, *valr7* and *valA* respectively, corresponding to map version $user_1$, meaning that they have been created in $user_1$. Parcel P_1 has only one value, $valp_1$, for map version $prod_0$, thus its value in map version $user_1$ is implicitly shared with that in $prod_0$. Parcel P_6 has two values, *valp6* in map version $prod_0$ and *valp6x* in map version $user_1$, because it has been updated in map version $user_1$.

Notice that the identifier of new objects created in $prod_1$ or $user_1$ must not be in conflict. This can be the case if the same identifier is used in $prod_1$ and $user_1$ to represent two different real world entities. To prevent this, the identifier of new objects is prefixed with the name of the database in which it is created. For example, the identifiers of P_{2a} and P_{2b} in Fig. 4 are in fact *prod-DB*.P_{2a} and *prod-DB*.P_{2b}. For simplicity sake, this does not appear on the figure.

This sub-section has described the main principles of the multi-version approach. In reality, the geometry of some geographic objects may be represented by complex objects. As such, updates are carried out on the elementary objects that compose the geometry (Peerbocus *et al* 2001).

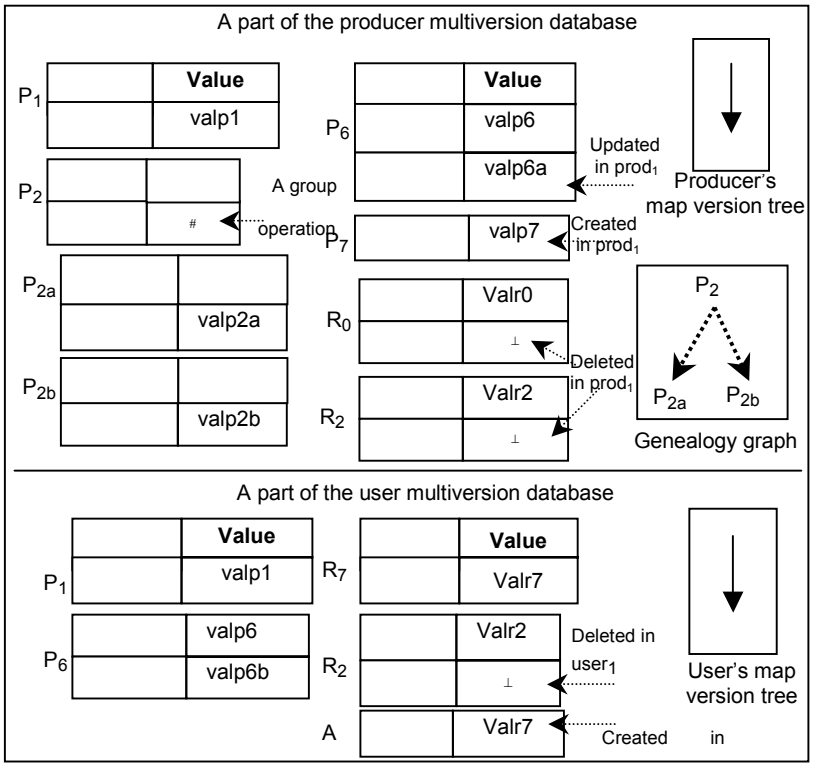


Fig. 4. Part of the multi-version databases

3.3 Delivery of Producer's Updates

On delivery, the producer's updated map version $prod_1$ is inserted in *user-DB* to enable the detection of conflicts between the producer's and the user's updates. This insertion operation is performed automatically as follows:

1. The system first modifies the map version tree to include map version $prod_1$ as derived from $prod_0$; $prod_1$ and $user_1$ become alternative map versions, both derived from $prod_0$, as illustrated in Fig. 5.
2. The system then verifies for each object in the delivered map version $prod_1$ whether the object has been updated or created by the producer - i.e., it has a value explicitly associated with $prod_1$. If so, the system inserts the value corresponding to $prod_1$ in the corresponding multi-version object in *user-DB*.

Finally, the multi-version database is composed of multi-version objects having values corresponding only to $prod_0$ and/or $prod_1$ and/or $user_1$ (see Fig. 5). For instance, parcel P_6 has three distinct values corresponding respectively to map

versions $prod_0$, $prod_1$ and $user_1$, parcel P_1 has only one value for $prod_0$, shared implicitly by $user_1$ and $prod_1$, and so on.

The next step consists in comparing, in *user-DB*, the user's and the producer's map versions, $user_1$ and $prod_1$, for the detection of possible conflicting and non-conflicting updates.

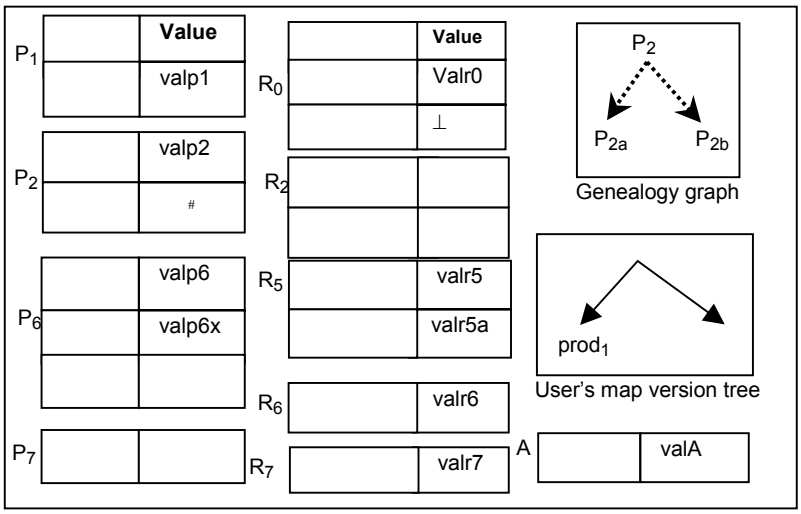


Fig. 5. Part of the user's database after the insertion of $prod_1$

3.4 Comparison of Updates and Detection of Conflicts

When the system compares the values associated with $prod_1$ and $user_1$ for the different multi-version objects in the database, the two following situations are possible:

Case 1. An object has the same value in both map versions. This can occur because the object has not been updated - i.e., the multi-version object contains only one value for $prod_0$, shared implicitly by $prod_1$ and $user_1$. Alternatively this may happen when the object has been updated or created in both the *prod-DB* and *user-DB*, and the values are equal - e.g., road section R_2 which has been deleted in map versions $user_1$ and $prod_1$. In these cases, there is no conflict.

Case 2. The object has different values in the two map versions, $user_1$ and $prod_1$. This situation is possible in the following cases:

a) the object has been updated or created in both $user_1$ and $prod_1$ and the two values are different; e.g., parcel P_6 has value *valp6* in $prod_0$, *valp6x* in $user_1$ and *valp6a* in $prod_1$, and *valp6x* and *val6a* are different. Here, the two updates or creations are in conflict.

b) the object has been updated either in $prod_1$ only or in $user_1$ only. The update corresponds to one of these operations:

- a creation: roundabout P_7 and road R_6 have been created in map version $prod_1$, and antenna A in $user_1$.
- a deletion: road R_0 is deleted in $prod_1$ and still exists in $user_1$.
- an update of its value: road R_5 has value *valr5a* in $prod_1$ and value *valr5* in $user_1$ (implicit sharing with $prod_0$).
- a group operation: the value of parcel P_2 in $user_1$ is *valp2*, implicitly shared with $prod_0$. Its value in $prod_1$ is denoted by #, meaning a group operation which is explained by the genealogy graph of P_2: it has been split into two new parcels P_{2a} and P_{2b}. These two parcels have only one value in $prod_1$ (corresponding to their creation). Conflicts exist in these cases and, for each object, its value in the new map version to be created in the *user-DB* depends on the user's decision.

These different situations can be visualised on the map by using special colouring of the object, revealing non-conflicting and conflicting updates and the types of conflicts.

This section has focused on updates relating to objects only. For schema updates a similar procedure is adopted [BCJ98]; e.g., antenna if it exists only in *user-DB*.

3.5 Proposed Strategy for Updates Propagation

The previous steps of the UMV methodology help the user in the visual detection of conflicts both at schema and object levels. Moreover, it supplies information concerning the types of evolution underlying the different updates. Now, the remaining step concerns the propagation of the detected updates in the user database.

This step needs an appropriate strategy, which may depend on many factors of the application concerned - e.g., knowledge about the underlying topology of the spatial objects (Egenhofer *et al* 1994, Badard *et al* 1999). For instance, the user may decide or not to favour his update in place of the producer's one in case of conflict. This choice may affect the user's added information, which may need to be readjusted. It is, therefore, wiser to use already proposed strategies such as (Badard *et al* 1999, Badard 2000, Kilpelaïnen 1997, Uitermark *et al* 1998), where the propagation problem has been thoroughly studied. The final map version of the user after the propagation of updates may contain the updates of the producer as well as those of the user. Suppose that the final user's map version is $prod\text{-}user_1$ as illustrated in Fig. 2.

To obtain the map version $prod\text{-}user_1$, the user first derives a map version identified as $prod\text{-}user_1$ from $prod_1$ (see Fig. 6), since $prod_1$ contains a large part of updates that the user needs to represent in his map version. Thus, all objects in the new map version are shared implicitly with $prod_1$; e.g., parcels P_1, P_6 and road

sections R_5, R_6. Next, he includes in *prod-user*$_1$ his specific updates: the road section R_7 and the antenna A. Fig. 6 shows part of the state of *user-DB* after the creation of the final map version *prod-user*$_1$, resulting from the merging of *user*$_1$ and *prod*$_1$. In the user's database, the value, *valr7*, of the road section R_7 in *prod-user*$_1$ is the value coming from *user*$_1$ and it is shared explicitly with *user*$_1$ as illustrated in Fig. 6. The situation is the same for antenna A that the user preserves in *prod-user*$_1$. In case, an object in *prod-user*$_1$ has a value different from that in *prod*$_1$ and *user*$_1$, for instance road section R_4 in Fig. 6, the system creates a new entry for this value, *valr4b*, which is associated with *prod-user*$_1$.

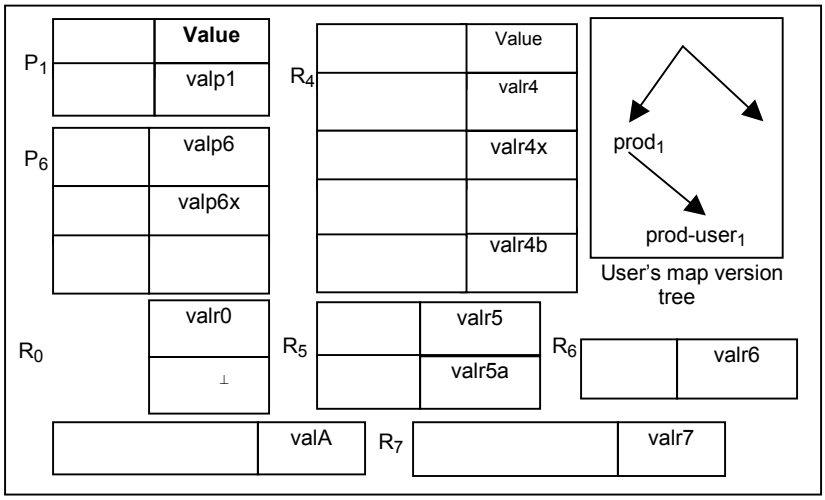


Fig. 6. Part of *user-DB* after the integration of data from *prod*$_1$

Finally, from *user-DB* of comprising of four map versions, the system reads the value of an object O in *prod-user*$_1$, as follows: if *prod-user*$_1$ appears in the multi-version object, then the value of O is the one corresponding to *prod-user*$_1$ (see Fig. 6). Otherwise, if *prod-user*$_1$ does not appear in the multi-version object, then the value of O is shared implicitly with the value of the nearest ancestor, obtained from the user's map version tree, which is either *prod*$_1$ (e.g., P_6, R_6), or *prod*$_0$ (e.g., P_1); finally, if only *user*$_1$ appears in the multi-version object, then O does not exist in *prod-user*$_1$.

After this propagation step, the user will work on new versions of *prod-user*$_1$ for further updates, whereas the producer uses new versions of *prod*$_1$. Thus at time t_2, $t_2 > t_1$, a new operation of integration of updates may take place: the new up-to-date map version, *prod*$_2$, is inserted in the user's database and compared to the current user's map version, *user*$_2$, which has been created by derivation and updates from *prod-user*$_1$. A new map version *prod-user*$_2$ is created, integrating updates coming from *prod*$_2$ and *user*$_2$. To help users in understanding updates performed in the different map versions for integration purposes, the updates should be documented as depicted in (Peerbocus *et al* 2001).

4 Conclusions

The focus of this paper was on how to help the exchange of updating information between a geographic data producer and a user. The main advantage of the UMV methodology is that it allows the automatic detection of updates whereas existing techniques require an exhaustive retrieval within the different versions of the database. The UMV methodology responds as well when updates are delivered on a given frequency as for real time updates. The UMV methodology can also be applied in a general context where there is a need of exchanging geographic data between any two users or between a user and a producer.

A prototype of a multi-version geographic database has been developed using MapInfo® in the LAMSADE Laboratory, University of Paris Dauphine. It requires the implementation of the version mechanism in the geographic database, which must be managed by a version manager. It allows the representation of the different states of geographic objects. All changes are documented. The prototype allows the retrieval of updated geographic objects between any two map versions of the multi-version geographic database and provides the user with the associated change documentation (Hedjar 2001).

The integration of the updates and propagation of their effects in geographic databases requires handling all the spatial relationships between entities in an effort to preserve consistency or added information. Several research works have set up tools for the retrieval of these relationships necessary to the updating of geographic databases. Ongoing researchers (IGN) investigating the development of a formalism and a model for the design of geographic databases, which are easier to maintain. The UMV methodology thus appears as a key element of a global methodology for the design of easy-to-update GIS.

References

Badard T (1998a) Towards a generic updating tool for geographic databases. In: GIS/LIS'98, Fort Worth, Texas, pp 352-363

Badard T (1998b) Extraction des mises à jour dans les Base de Données Géographiques. Revue Int. de Géomatique 8(1-2):121-147

Badard T, Lemarié C (1999) Propagating updates between geographic databases with different scales. In: Innov. in GIS VII:GeoComputation, London

Badard T (2000) Propagation des mises à jour dans les bases de données géographiques multi-représentation. Ph.D. thesis, Univ. Marne-la-Vallée, France

Badard T, Richard D (2001) Using XML for the exchange of updating information between GIS. In: CEUS 25. Elsevier, Oxford, pp 17-31

Bellosta MJ, Cellary W, Jomier G (1998) Consistent Versioning of OODB Schema and its Extension. 14-èmes Journées BDA, Hammamet, Tunisia

Bauzer-Medeiros C, Jomier G (1993) Managing Alternatives and Data Evolution in GIS. In: ACM Workshop on Advances in GIS. Arlington,Virginia

Cellary W, Jomier G (1990) Consistency of versions in object-oriented databases. In: VLDB. Brisbane, pp 432-441

Cellary W, Jomier G. (2000) The Database Version Approach. Networking and Information Systems Journal 3(1): 177-214

Devogèle T (1998) Le processus d'intégration et d'appariement des BD géographiques. Ph.D. thesis, Univ. Versailles-Saint Quentin, France

Egenhofer MJ, Clementini E, Di Felice P (1994) Evaluating inconsistencies among multiple representations. In: 6th SDH. Edinburgh, UK, pp 901-920

Easterfield ME, Newell RG, Theriault DG ('No Date') Version Management in GIS-Applications and Techniques. Smallworld technical paper no. 4

Gançarski S, Jomier G (1994) Managing Entity Versions within their Context: a Formal Approach. In: DEXA'94. Athens, LCNS no. 856, pp 400-409

Hedjar M (2001) A prototype for documenting spatiotemporal evolution. Research report, DEA 127, LAMSADE, Univ. Paris-Dauphine, France

International Hydrographic Organisation (1996) IHO transfer standard for digital hydrographic data. Publication S-57, Edition 3.0

Kilpelaïnen T (1997) Multiple representation and generalisation of geo-databases for topographic maps. Finnish Geodetic Inst., 124, 51-711-212-4

Lemarié C, Raynal L (1996) Geographic data matching: First investigations for a generic tool. In: GIS/LIS'96. Denver, Colorado, pp 405-420

Peerbocus MA, Bauzer Medeiros C, Jomier G, Voisard A (2001) Documenting Changes in a Spatiotemporal DB. In: XVI BSDB. Rio

Poupart-Lavoie G (1997) Développement d'une méthode de transfert des mises à jour de données à réf. spatiale. M.Sc. , Univ. Laval, Québec

Raynal L (1996) Some elements for modelling updates in topographic database. In: GIS/LIS'96. Denver, Colorado, pp 405-420

Sperry L, Claramunt C, Libourel T (1999) A Lineage Metadata Model for the Temporal Management of a Cadastre Application. In: Int. Workshop on Spatio-Temporal Models and Languages. Firenze, Italy

Uitermark H *et al* (1998) Propagating updates: Finding Corresponding objects in a multi-source environment. In: 8th SDH. Vancouver, pp 580-591