Incremental Update of Cartographic Data in a Versioned Environment

Marc-Olivier Briat, Jean-Luc Monnot, Thierry Kressmann
ESRI Inc.
Redlands, California, USA
mbriat@esri.com, jmonnot@esri.com, tkressmann@esri.com

ABSTRACT

ESRI is developing its GIS software to support cartographic representations inside a versioned geodatabase. Representations are usually described as a Digital Cartographic Model (DCM), as opposed to a Digital Landscape Model (DLM). A set of tools (including generalization operators) is involved in deriving the DCM from the DLM. One of the main benefits of the DCM representation mechanism is to support manual overrides, because there will always need to be human interventions. Thus, real cartographic data require a combination of automated and human work.

This paper covers the propagation of DLM updates to the DCM features and representations. The first challenge is to apply the same set of derivation rules using just update descriptions as input, instead of a whole new dataset. The second challenge is to prevent erasing the manual work of the cartographer, which represents much of the time spent to produce the final map. The paper describes how the database model is extended to keep track of the cartographer’s manual work.

1 INTRODUCTION

Geographic Information Systems are now built on top of database systems. This is necessary to manage increasing amounts of data, and serve multiple clients. There is also a strong push to try to organize the diversity of geographical data in more organized data models, where dependent information is correctly modelled and related to source data.

GIS have always been powerful tools to analyze and derive data. These capabilities, combined with a rigorous database structure, leads to workflows with strong automation capabilities. ESRI’s ArcGIS software, geodatabase, and the ArcGIS geoprocessing framework are good examples of achievements in this area.

Cartography and cartographic data modeling are also beginning to leverage GIS databases [Hardy & Kressmann 2005] and automation. There are two aspects to the automation problem. In the first aspect, DLM (Digital Landscape Model) data are derived for different scales using model generalization. In the second one symbolized data or cartographic effects (in the DCM or Digital Cartographic Model) are derived from the DLM [Lee & Hardy 2005].

Using traditional workflows, the map finishing process is often managed by Graphics packages like Illustrator, FreeHand, Mercator, etc. because of a perceived lack of tools inside the GIS. ESRI is working on an approach for ArcGIS where cartographic modelling can be achieved inside a GIS, built on top of a database, thus providing high quality rendering and editing tools in a system that leverages the traditional strengths of database technology [Eicher & Briat 2005].

Although automated tools and processes are important, most high-quality cartographic products will still require manual intervention. Both the generalization tools and the symbolization tools may produce graphic configurations that an expert cartographer will reject and rework. This is not going to reach 100% automation in the near future.

The challenge is then to design a system where both automated and manual work can coexist inside the same database. Usually, a workflow for the first distribution of a map or map product will mix automated steps (initial construction of the data) and manual steps (correcting some data), then automated steps again (creating “representations”, placing texts), and eventually finalizing the map (manually reworking the text positioning).

For the second release of a map or map product, running the same process again will erase the manual work, which represents most of the cost and time spent. Thus this is not an acceptable solution. When a list of only the updates is available as input, updating can be more elegant, however but the requirement to preserve manual work still exists.
2 SCOPE OF STUDY

This paper describes an attempt to manage this apparent contradiction that exists in a software system that provides both automation and manual refining tools.

The base assumption is that the system can store the result of some processing. This allows the user to modify the result of the process, and thus provides the freedom to override the graphic result. This is completely different from an all-on-the-fly approach, where results cannot be changed.

2.1 Foundation vs. Tools

This paper does not focus on a specific update problem, rather it aims to describe a generic framework. Because a diverse set of cartographic problems involve chained processes, a general approach centered on this concept can be more useful than a solution catered to a specific cartographic application.

![Diagram of deriving data](image)

Figure 1: Examples of deriving data

Probably every mapping organization has some ongoing work that fits this kind of workflow, especially for generalization, including deriving and updating data. Specific map products (aeronautical charts, nautical charts) will need specific process rules.

A database infrastructure is needed to manage propagation of updates. This is motivated by the fact that though the set of processes required by mapping organizations is very diverse, most processes require that the result be customized (at some level) by the cartographer. Currently this is often achieved in graphic-based, non database centered systems, external to GIS. There is a need for update capabilities for activities currently performed in these applications as well.

ArcGIS is a flexible system which allows developers to add their own custom functions. This is especially relevant when it comes to processing and updating. GIS users work with their own specifications and data models. A core set of functionalities is needed to address the update problem.

2.2 Workflow and processes

In ArcGIS, A workflow that chains processes is typically implemented by using a model managed by the geoprocessing framework.
A model contains processes, implemented as tools. There is a well defined order of execution, based on the links between the processes. One can clearly identify the input data and the output data. When the model is run the first time, the output structure is created, and its content populated, considering the whole input dataset. Once this is done, the model contains the description of the datasets and how they relate together in the sense that all the parameters are defined here.

We want to consider the incremental update problem in the context of the same geoprocessing model, where the same set of rules is used. The only difference is the type of input we consider. Instead of considering data contained by the dataset, we want to consider the flow of changes. This means that a process should have two modes: creating data from data and creating updates from updates.

When running in update mode, data are modified, but this is expressed as changes so that chained processes can propagate a consistent type of information. When dealing with manual updates (see below), the update is a clear object that can be validated or not.

### 2.3 Propagating the updates

The update workflow starts with an initial set of updates. This set is given to the first process, which produces a new set of updates as the result of its action. The two sets are combined, and this augmented set is then passed to the second process, and so on until all the processes are triggered.

One important property of an update is the last process that managed it. It is required that processes can be ordered in the model.

Updates can be simply ignored by a process, if they don’t relate to the process's input datasets. Updates can trigger a higher level object that extends the context of the update (this is up to the process). Care must be taken to avoid circular references.

### 2.4 Collecting the initial updates

The starting point of the incremental update process is the collection of update operations that describe the changes that occurred in the source data. Changes can be obtained in a variety of ways, and made available in a variety of formats:

- Data providers may be able to provide delta files that record new features, deleted features, and changed features. For instance, Ordnance Survey MasterMap is able to provide changes using GML. If not, tools may be able to extract the differences between two datasets sharing the same structure, providing change information in an equivalent format.
- If the data are managed by a system that supports versioning (for example an ESRI ArcSDE enterprise geodatabase), queries may be run on the parent and child version to extract the changes (ArcSDE versioning environment supports DifferenceQueries). Databases that support replication (master and slave databases) are also well suited to provide the necessary change records. ESRI’s implementation of geodatabase replication is based on versioning, which makes the two approaches similar, even if the actual database organization differs.
A GIS that supports database events may also be a candidate to record the changes, but this will require additional software to be run during the DLM editing phase, which is not desirable.

Regardless of how changes are collected, a well-defined structure must be identified to store the set of updates to be processed. An obvious solution to this problem is to use database tables for this. Database tables:

- Can persist the needed information
- Can be extended to attach additional management columns
- Can be populated by many tools if the storage format is public

In the prototype built to test the proposed framework for incremental updating, database events were used to populate the update table. In a more realistic use-case, two ArcSDE versions could be compared to generate the same information.

### 2.5 UI/Control Requirements

The authors have been involved in database-centered cartography for several years. A previous system (DataDraw) had a notion of processes (Agents) that were able to react on database events. This was an efficient way to trigger database changes that controlled graphic rules. However, when update logic becomes more complicated, one requires more control over basic update propagation. This occurs when:

- One update triggers multiple processes
- The output of a process is used as input by another process

In the earlier implementation, the result of a process always erased existing values, without control.

Including lessons learned from past experience, the following requirements were collected:

- The framework must provide control over the processing order of updates.
- The current state of the propagation of updates must be stored in the database. The update cannot be performed in just one edit session (it can last several days or weeks).
- Updates must be treated as entities that can be selected. One must be able to check their status in the context of the entire process.
- It is important to identify manual edits. A system that reprocess data automatically and erases manual work is unacceptable.
- A warning must be raised when an update conflicts with a manual edit. The cartographer should then be allowed to test, and then either accept or reject the result of the processed update.

### 3 CONCEPTS

#### 3.1 Description of the updates

The proposed system considers two types of data: tables (which include geographic feature classes) and relationships.

In the ArcGIS framework, dataset types can be identified by an unique ID. Individual dataset objects can also be identified by a unique key. Relationships offer the benefit of allowing a database to store information related to the logic of the derivation process. The update logic can be greatly simplified if the database model contains those relationships.

The types of updates we need to describe are creation, deletion and modification. As all the cartography is modeled as database elements in the ESRI representation project, there is nothing more to consider to manage cartographic updates. We assume that the graphic rules, which are part of the schema of the database, are not changed when updates are managed.

Description of updates can be more complex [Badard & Spery 2001], but the current approach is purposefully based on a description of updates supported by the underlying database, thus making sure that basic operations are natively supported.

Each update refers to the dataset ID and object ID of the affected feature, and it also contains information about which fields are modified. Feature geometry is stored in a column, and changes are considered like changes to any other field
type. That is, feature geometry receives no special treatment in the model, even if it is a major input for the derivation and update algorithms.

The implementation of the deletion updates must carry all information about the feature being deleted, since the feature is assumed not to be in the database when it is time to process the deletion update. A process will certainly need the spatial extent of a feature to be able to rework an area. Attributes values are useful too. In an aggregation process based on similar attribute values, you need to combine both geometry and attribute to identify the derived feature that needs to be updated.

Implementing relationships as part of the model helps to analyze which derived feature must be changed. The basic information associated to a relationship is then the foreign key values.

3.2 Manual edits

As previously mentioned, an important system requirement is to be able to combine automated and manual work. A conflict between automated and manual edits may occur when the result of a process tries to modify data that have been modified manually.

Update conflicts occur when:
- An object cannot be deleted because it contains manual edits
- A field cannot be modified because it has previously been modified manually

When data are modified, events are triggered in the geodatabase. If a modification corresponds to the output of a process, it is registered in a table that contains all the potential conflicts. This is memorized in the database. Later, when a process tries to update some data, this table is queried to check if there is a conflict. If not, the modification is validated and the data is changed. Alternatively, if there is a conflict, the modification is not validated and the source update is considered as “not processed”. (Additionally, the state of the update is unchanged in the table of updates).

A user interface is provided that allows the cartographer to review all updates that have been blocked because of conflict, to identify the manual edit that is the cause of conflict, and choose from the follow list of actions:
- Allow the process to overwrite the manual edit. If the result is still not acceptable, the user has to perform their manual edit again, but this will now be based on up-to-date data.
- Ignore the update and consider the manual edit as still valid. The update is no longer considered in conflict for this process and can be processed by the next process in the model.

4 EXAMPLE

The above principles were tested in the context of processing tools dedicated to automate an advanced cartographic representation of an overpass in a transportation network (e.g. roads) (Fig.3)

![Basic GIS representation](image1.png)
![Desired representation with masking and parapet effect](image2.png)
![Intermediate polygon used to mask the “below” road feature and to draw the parapet.](image3.png)

Figure 3: Modeling an overpass

This graphic effect involves two processes:
- Creating the polygon defined by the intersection of the road outlines and creating relationships between this polygon and the “above” and “below” road features.
- Creating the parapet effect based on the polygon shape.

4.1 Applying updates step by step

A series of screen captures from the system shows the successive state of the data as triggered by an update to the geometry and position of the “below” road (Fig. 4).

![Initial state](initial-state.png)

<table>
<thead>
<tr>
<th>Initial state</th>
<th>Integration of the change to the road geometry</th>
<th>Result of the first process (updating the polygon)</th>
<th>Result of the second process (updating the parapet)</th>
</tr>
</thead>
</table>

Figure 4: Successive steps when processing the updates

At each step, the database is in a well known state. Events currently waiting to be processed are recorded in a table and accessed in a user interface that allows the user to see the relation with processes (Fig. 5, “Class” column). All modifications take place in an edit session, and individual transactions can be undone if necessary.

![Updating table](update-table.png)

Figure 5: User interface for reviewing updates

4.2 Situation involving manual edits

Figure 6 shows the set of steps for processing updates when manual edits are present.

![Step 1](step-1.png)

Step 1: Initial state of the representations as created by automated processes

![Step 2](step-2.png)

Step 2: After some parapets are removed manually for clarity
Step 3: After an update changes the size of the center line, processes automatically update all of the unaffected intersections.

Step 4: User reviews the “Updates errors” tab and chooses to remove the manual conflict (see tooltip).

Step 5: The new representation of the affected intersection can now be created by automatic process based on the updated street center line.

Figure 6: Steps for processing updates when manual edits are present

4.3 Algorithm Implementation

A small set of attributes control the overpass effect:

- The shape geometry of the roads
- The symbol used to draw the road (modelled by an integer field). A wider line symbol changes the polygon shape.
- The columns that identify the level of the road (below or above)

Without elaborating on the details, it is obvious that some updates can be processed quite simply (updating the width of a road), while other can be trickier, such as:

- Adding a road may require a check to see if it participates to an already existing overpass
- Changing road level attributes may reverse the overpass. In this case, can the same mask polygon still be used?
- Changing the shape of a road may require that two existing overpasses be merged.

Working on this overpass example made clear that:

- It is necessary to be able to access the whole set of updates instead of working on a one by one basis. This is especially true when the process involves spatial relations.
- Since the derivation logic and the update logic use the same low-level functions to calculate the resulting data, it makes sense for the same component to contain/manage both functions. This is one of the key point that underlies the work described in this paper.

5 VERSIONING

The tables added to the database to manage the updates are versioned tables. This means that all the edits performed when processing updates can be isolated in a version, allowing multiple cartographers to be working on the same database [ESRI 2004]. There are two strategies to reconcile the update work from multiple cartographers/cartographic editors:
- Wait for all the updates to be processed and then allow a reconcile and post operation
- Allow data reconciling during the update process

The first option is simple, because all the events are no longer needed when the update phase is done. Basically it is only necessary to reconcile the DCM data, ignoring the event tables.

The second option however needs more attention because we need to carry over the updates being processed and the manual edits. Reconcile conflicts can occur during the reconcile operation, and the final state of the database must still be consistent, regarding the relation between the database objects that describe the updates and the DCM data. In other words, “No updates can be lost”. One possibly acceptable basic rule is to give precedence to manual edits over automatically derived data. With this approach, nothing is lost during the reconcile operation. There are however decisions to be made if two manual edits produce a reconcile conflict.

Managing the reconcile operation with tables that describe updates as been examined in detail, but nothing has yet been prototyped to prove that it is valid.

6 CONCLUSION

A framework for the propagation of updates has been proposed in this paper that provides services to help manage the coexistence of manual work within a mainly automated workflow. Although the strategies outlined in this paper could be applied to generic derivation problems, not many algorithms are available, and few databases are currently built this way. Model generalization or label placement [Murad 2005] are bigger challenges for the incremental update. However, the framework that has been experimented is orthogonal to research in specific incremental update functions.

The current work at ESRI on developing software that supports the storage of cartographic representation information inside a geodatabase is a good basis for exploring the incremental update problem. This is because the cartographic representation system makes use of geoprocesses to automate the generation of feature symbolization—thus employing automated processes, while also providing for intervention through manual editing—thus introducing the very real possibility of update conflicts.

Speaking to the challenge of deriving cartographic data through automation, it should be mentioned that the automated symbolization processes discussed in this paper (e.g updating an overpass) are thought to be less complex than those involved in generalization. This makes these processes more easily isolated and chained, and it makes a system based on these processes an easier starting point for studying and implementing an incremental update system for cartographic data. Automated symbolization processes benefit cartographers and administrators by saving human work, and, adding the incremental update component enables a system to exist where editors can manually edit data and still be able to gracefully accept incoming updates at a future point. This comes only with the benefit of having all the data managed in a central database, which is a fundamental assumption of this framework.

7 NOTES

1. The sample data is swisstopo VECTOR25, copyright Swiss Federal Office of Topography.
2. Thanks go to the ESRI team working on database cartography, representation and overrides, and also to Paul Hardy and Cory Eicher from ESRI for their comments.
3. This paper is a forward-looking document, and the capabilities it describes are still under development. As such, it is intended to give guidance as to likely future direction and should not be interpreted as a commitment by ESRI to provide precise capabilities in specific releases.

8 REFERENCES

- Badard T & Spery L, “Problèmes liés à l'échange et à l'intégration des mises à jour dans les bases de données géographiques”
• Hardy P & Kressmann K. 2005. “Cartography, Database and GIS: Not Enemies, but Allies!”, 22nd ICA Conference Proceedings, A Coruña, Spain
• Lee D & Hardy P, 2005 “Automating Generalization – Tools and Models”, 22nd ICA Conference Proceedings, A Coruña, Spain
• Murad-al-shaikh M, 2005, “Professional Labeling and Annotation Techniques with ArcMap” 22nd ICA Conference Proceedings, A Coruña, Spain

9 BIOGRAPHY

Marc-Olivier Briat has been working in the GIS industry for more than 15 years. He graduated from the French Ecole Polytechnique in 1986, then from ENSTA in 1988 (option System Analysis).

He started developing a GIS on Macintosh in the company Klik Development as technical lead and cofounder. This GIS (MacMap) was used in the IGN GeoRoute production line. A second version of the software enhanced the cartographic capabilities and was used in the TOPO25 production line.

He then cofounded in 1996 the company Hemispheres (later renamed Alida) with Thierry Kressmann, focusing on cartographic capabilities on top of a database, implementing the DataDraw solution on top of GeoConcept, Oracle and ArcGIS.

In 2002, ESRI acquired the Alida company. He moved to ESRI in Redlands, California, as developer lead for the Representation project, which implements the Alida technology inside ArcGIS.