GE Energy

Smallworld Core Spatial Technology™ 4

Managing change in the world of spatial data – the long transaction

GER-4231 (10/04)

Managing change in the world of spatial data – the long transaction

Abstract

Early enterprise Geospatial Information Systems (GIS) projects often began as an expensive, tedious and labor intensive data capture exercise. Many of these issues were exacerbated by conventional GISs that treated spatial data in very much the same way as plain attribute data ignoring the subtle peculiarities of the geographic world. Businesses that used GISs built on traditional short transaction[1] access found that this approach slowed the data capture process resulting in missed milestones and increased costs. Unfortunately for the customer, these problems often did not end there: the short transaction model, stated simply, did not support many of the spatially-enabled business processes that enterprises demanded of their GIS installations.

From the outset, the GE Energy architects of the Smallworld platform recognized the limitations of short transaction-based databases and, consequently, set about the challenge of developing a new database technology designed from the beginning to efficiently support the spatial needs of the enterprise. This work led fifteen years ago to the successful introduction of a forward looking, highly optimized spatial database technology designed from the outset to support spatial processes and long transaction operations (changes to the database undertaken completed after a considerable period of time has elapsed). This bold strategy has not only been vindicated by great success in the enterprise GIS market, but has also been flattered by other GIS vendors who have now sought to re-engineer their products in an effort to catch up.

Check this out

Most enterprises approached data capture in a similar way.

First, they divided their operational territory into small geographic areas. Each area might correspond to some existing business region, but more often than not, it was simply represented as an amount of spatial data delineated by a rectangular bounds that one person could digitize in a reasonable period of time.

Next, to protect the master database and to prevent data capture users from trampling over each others work, any data already captured in an area was copied off line (checked out) either as a separate database or even as a plain flat file. This approach prevented the user from damaging the master copy of the data and also allowed work to be carried out independently of other users.

Finally, when data capture for the area was completed (a process that might take days or even weeks) the private copy of the data was then passed back to the master database (checked in).

This approach superficially sounds fine. It is only when the realities of GIS in an enterprise environment are fully appreciated that this approach becomes less appealing.

In most enterprises, spatial data is not simply decorative, designed to flatter a convenient map or embellish a lackluster presentation. This is important business critical information about infrastructure that is often the backbone of an enterprise's day-to-day operations. It is, therefore, imperative to the business to ensure that this data is of the very highest quality. Only high quality data can drive the best business decisions.

Since the operational data is stored in the master copy of the database, any quality assurance work has to be performed on the checked out data before it is checked back in. Quality assurance at its most primitive often involves nothing more than a visual inspection of the data (geography lining up with the original source, features connecting as expected and so on). However, since the view of the world is often constrained by the extent of the checked out area

[1] A database architecture in which changes to the data are available to all users immediately after the change has been completed (or committed).

even a trivial standard that ensures features line up correctly at the edges becomes an unnecessarily convoluted process. Features that straddle more than one data capture area complicated matters even further – just who is responsible for digitizing that major highway anyway?

Obviously, manual inspection to identify data capture issues is very labor intensive and very expensive. It is not surprising, therefore, that several automated techniques emerged to help out. These tools validated feature attributes, enforced business standards and automatically corrected small errors. More advanced tools performed business specific analysis on the data such as, for example, electrical correctness.

However, many of these tools could never be smart enough to see beyond the confines of a single isolated data capture area. This meant that some analysis were simply not possible: for example, it was not possible to determine if one part of an electrical circuit would result in an overload without the presence of the substation that fed it. This piecemeal view of the world was often compounded by the nature of the checked out data itself. Some gis vendors used cad-based drawing tools that, although allowing an accurate pictorial representation of the data, never captured the hidden, high value data (its connectivity and structure) preventing many useful analysis.

No more data, no more problem

With the end of the data capture phase most enterprises hoped that the problems they experienced early on were finally going to go away. To a certain extent this was a reasonable expectation: the vast majority of the data had been captured and what was left merely represented the effort associated with supporting day-to-day operations.

However, GIS vendors who cultivated this assumption failed to fully appreciate one of the most important business processes of enterprise GIS: design[2]. Design, at a simplistic level, represents a group of proposed changes that will be carried out by a worker in the field (for example, the construction of an electricity circuit to service a new school).

The spatial data held in an enterprise GIS is rarely static in nature. Sections of the infrastructure represented by this spatial data will be subject to periodic upgrade to sustain the highest level of service to the customers it supports. Businesses will also have to cater for additional customers by building new infrastructure.

Upgrading or augmenting network data is a non-trivial process. Many components on a network have complex interrelationships that need to be carefully considered before being connected. For example, an additional network to supply electricity to a new housing development will have to be carefully analyzed to ensure that it does not breach safety or operating standards.

The design of this infrastructure might take a few days for a small extension to the network to many months for a major project. The long transaction nature of design clearly fits awkwardly with the short transaction underpinnings of many conventional databases and leads to many of the same problems that surfaced during the data capture phase (new infrastructure design is conceptually nothing more than operational data capture). However, the key word here is operational. Design occurs on a day-to-day basis. A GIS architecture that complicates this important process is no longer an inconvenience delaying the deployment of an enterprise GIS, it is now a brake on the whole organization that increases costs and reduces operational effectiveness.

Designing for design

As well as its similarities to data capture, design also has it own unique characteristics that further

[2] This paper uses the term design but other equivalent terms include work order and job.

challenges the short transaction model. Designers often need to be able to switch dynamically between a set of views of the network structure as they work.

The as-built view

In this view the designer only sees the state of the infrastructure that is currently built.

The design view

In this view there is a clear graphical indication of the changes that are being made as a part of a design. This view is used to create a document that a crew will use to carry out work in the field. Objects to be added, replaced and removed are all usually marked with distinctive styles (for example, cross-hatching is often used to indicate objects to be removed). This view might contain temporary graphical changes to make it easier for the crew to interpret. One common requirement is to make a subset of existing objects invisible (if they are not relevant to the current work order). Another is to temporarily offset objects or to add temporary notes to make it easier for the crew to interpret the design.

The future as-built

In this view a selected set of pending designs are added onto the current as-built view to create a new view that represents the anticipated state of the network in the future after these changes have been made. No distinction is apparent between objects that currently exist in the field and those that are part of the selected pending designs. This allows analysis routines (for example, load flow analysis, customer revenue yields) to look at the impact of these future changes without the need for any additional special logic. Multiple future as-built views can be created with different sets of designs incorporated into each. They can either be created on demand or can be maintained permanently (for example, some organizations permanently maintain a view that includes all approved designs).

The multiple design view

Another common requirement is to be able to get a graphical indication of all the proposed work in an area (which may consist of data from many designs). Temporary graphical changes that have been made in an individual work order view might not be applicable in a multiple design view (in particular, if specific objects were made invisible in an individual design this would probably not make sense in a multiple design view). The multiple design view is often used while a user is working on a new design, to display other pending designs that may overlap or impact the current design.

Enter the long transaction

The Smallworld platform pioneered the long transaction model over fifteen years ago and this experience led to the most advanced spatial database technology available today. This is an important point: this technology was developed using a blank piece of paper. It did not start with a short transaction database and attempt to reverse engineer or bolt on this important functionality.

A quick introduction to a long transaction

A long transaction is a change in the database that might take several days, weeks or even months to complete. When a long transaction is started, a delta of the entire database is created called a version. Its parent is called the parent version and the parent of all versions is called the root or top version (or simply top, for short). The child version initially contains no data as it is identical to the parent version. As the user adds new data (for example, network infrastructure) or deletes existing data, the child version grows in size reflecting these changes (the data outside of the child version is unaffected). The underlying long transaction database manages things in such a way that the user effectively sees the data in the parent version together with the changes made in the child version. When appropriate, the user can periodically merge down

changes from the parent version to see updates made by other users. Also, when in a version, a user can preserve changes by committing them or undo changes by rolling back (milestones called checkpoints can also be created allowing previous states of the version to be revisited). When the long transaction is completed, the changes in the child version are then posted up to the parent version where they are applied to the existing data.

Conflicts resolved

Since the long transaction database can maintain information in terms of change, it can also identify conflicts that arise when merging and posting data by identifying data that has been changed both in the parent and child versions.

In the simple case where there are no conflicting changes, the end result of merging the parent's changes down to the child and posting the child's changes up to the parent is that the two versions are again identical and both contain all the changes.

In the more common situation where there are two or more child versions that are each modified from their unchanged parent, all the changes in the child versions can be combined by a series of merge and post operations.

However, if the same record (or structure) is changed in different ways in the parent and child versions (the common ancestor of the two versions), then there is what is called a conflicting change.

The merge operation detects any conflicting changes and can either resolve them automatically in favor of one specific version (parent, child or base), or, alternatively, can enable operator intervention. In this case a conflict viewer is activated which gives a highly detailed textual and graphical analysis of each conflicting change, and gives the operator the opportunity to override the default resolution for each one as required. This ensures quality and prevents data integrity problems.

The long transaction shows its form

The long transaction approach has some key advantages:

- In a child version the user sees a contiguous view of the world, not a piecemeal one. This makes it easier to capture and align new data. Data capture can also be organized by feature rather than by geographic area allowing greater working flexibility.
- Quality assurance is easier as data can be seen in a broader context.
- Analysis tools can operate on more complete views of the data.
- A user can have access to more up-to-date data simply by merging it down from a more up-to-date parent version.
- Changes made in a version only affect that version protecting the operational data at all times.
- Changes made in a version can easily be discarded after being posted to the parent version to save space when the database is compressed.
- Changes in a version that represent an important piece of work (for example a design) can be kept for long periods for analysis or auditing purposes. Competing designs can even be evaluated before the best is chosen.

The grass isn’t greener on the other side of the fence

Recent implementations of version management have all used what is termed shallow version management in this paper. In this approach, versioning was not an intrinsic part of the database design. Instead, it is implemented by storing records belonging to all versions in the same relational table, together with additional key information relating to the version(s) that a given record applies to. Queries

perform additional processing to determine which records apply to the current version. To make this transparent to the application, queries are usually done against table views, rather than physical tables. The attraction of this approach is that it can be done on top of a conventional database. A significant drawback of this approach is that it requires significantly more server processing than the deep version management approach. Another potential drawback is the relative immaturity of most of these implementations (it is easy to demonstrate shallow versioning, but it is much harder to produce a system that provides good performance and scalability). Other problems with the shallow versioning approach tend to result from a failure to fully appreciate the demands of enterprise GIS:

- Routine administrative tasks such as garbage collection and database compression suddenly become overly complex processes that need to be carefully managed.
- The underlying complexity of the shallow versioning approach adds a processing overhead that scales less well (even medium size enterprises often require thousands of versions to be managed simultaneously).
- Enterprises often have good operational business reasons to want to keep the data in more than one version (and not just in the operational or top version). Shallow versioning tends to require that all operational data in sub-versions be combined with the top version (this is the version that is then used operationally).
- The realities of enterprise gis inevitably mean that the data model will need to change from time to time to support new business processes. Shallow versioning only manages the data and not the data model making it difficult to archive versions that have legacy data models.

Technology built on expertise and experience

The expertise and experience gained over the last fifteen years of meeting the spatial needs of enterprise GIS has led to the most established mechanism for supporting long transactions: deep version management.

Bluntly, deep version management has to be implemented as part of the fundamental design of the database: it cannot simply be added in later as an afterthought.

At its heart it works by the low level data structures that make up database tables, to share disk blocks between different states or deltas of the data at different points in time. These individual states are used to store the individual versions mentioned earlier. This important advance means that there is no processing overhead at all associated with deep version management. When in a particular version, the database only sees data in that version and does not require any additional processing to filter out other data.

This idea can be extended further to a persistent cache that is in essence a demand driven distributed database that works very well for applications where geographic subsets of the database are generally accessed in regional offices as is typically the case with many spatial applications. This leads to significant scalability and cost advantages. For example, sites with thousands of concurrent update users working against a single database are now a practical proposition using this approach. Another feature of this approach is that it is also possible to version manage the data model in addition to data, which is very useful for development and testing purposes.

A dedicated, long transaction database also allows other novel uses of versions beyond the basic merge and post functionality. For example, the sideways merge is an operation that allows changes from one version to be applied to another version elsewhere in the hierarchy instead of restricting changes to being

propagated only to immediate parent or child versions, which is what traditional merge and post operations do. This technique can be employed by design applications allowing the creation of future as-built views or multiple design views by copying data from all the relevant design versions into a single design.

Despite the fifteen year investment in long transaction technology, the Smallworld platform continues to move forward supporting the emerging demands of enterprise GIS.

For example, as a result of globalization and consolidation, some very large enterprises have emerged that require literally tens of thousands of designs. Storing each in its own version is theoretically possible, but often requires management processes or hardware that is prohibitively expensive (for example, traversing the version hierarchy is obviously slower because it is more populated). In this context, businesses prefer to be able to archive designs to conserve it resources and improve manageability.

A simple approach might involve archiving a version by converting it to a simple flat file and storing on disc. This is a problematic approach to say the least: recreating the version will probably be slow and unable to handle changes in the data model that might happen over time.

Being built on a database designed from the start to support long transactions provides an unprecedented opportunity to provide elegant solutions to real customer challenges such as this. Recently introduced technology provides built in support to easily archive versions and return them to the operational database again when required without the need for lengthy translations. This process is transparent to the design application requiring only modest changes to take account of this new functionality. Versions are reused as required when archived data is needed to be brought back on line minimizing the size of the version hierarchy resulting in reduced disc space, faster backups and database compression.

Conclusion

With over fifteen years of experience pioneering the use of long transactions in the gis enterprise, the Smallworld platform's highly optimized database has matured into the most advanced technology for storing and managing spatial data.

Designed from the outset to store spatial data and support the business critical applications that today's enterprise demands, Smallworld database technology delivers tangible success in performance, scalability, flexibility and reduced cost.