W5.4

Major restructuring of Ordnance Survey database to enhance British mapping

Brian Naughton, Worldwide Director of Product Management, Object Design

Introduction

A major partnership between Ordnance Survey and eXcelon Corporation is set to help businesses and public services gain faster and more flexible access to British mapping data.

Ordnance Survey is re-engineering its National Topographic Database - the computerised 'master map of Britain' and is nearing completion of the development of the Digital National Framework (DNF). The DNF has been built to provide a step-change in the ways that Ordnance Survey's customers and partners can access and use geo-spatial information.

The main aim of the DNF is to provide a consistent and maintained digitalized master map of Britain against which any geo-spatial information can be referenced, either through National Grid coordinates or unique identifiers. Data from the Digital National Framework is designed to be Internet- and customer-friendly, making it even easier and faster for public bodies and businesses to pick and mix the mapping and geographical information they need. To make this possible, Ordnance Survey has converted all 230,000 of the most detailed point-and-line Land-Line mapping tiles to a seamless data source containing 400 million self-contained individual objects.

Being able to serve geo-spatial data to customers is of vital importance for Ordnance Survey, since this is the fastest-growing part of their business. The Geospatial Object Server (GOS) is the database management software at the heart of the DNF solution. The GOS database uses the ObjectStore data management software from Object Design, a Division of eXcelon Corporation.

GOS Requirements

The GOS needs to provide full support for Ordnance Survey's stated e-business initiatives, in order to provide Ordnance Survey's customers with accurate, discrete GIS data in a format that they require, in a timely fashion, across the Internet.

- The GOS must provide an extremely high performance delivery platform that accommodates the required responsiveness to the entire customer base. The GOS must serve geo-spatial data covering the whole of UK at 1:10,000 and larger scales to client programs as fast as possible.
- The GOS must have the ability to readily scale to support the introduction of a larger number of customers.
- The GOS must provide full support for a very large (3 / 4 Terabyte) data set, without degradation of performance.
- 24x7 availability and backup. The GOS needs to provide a secure, robust, high-performance storage environment, that enables backup and other management operations during database operation, supports database growth over a wide range of capacities, and maximises data availability

- Full support for schema evolution or support for a dynamic schema in order that new features may be modelled readily.
- The GOS must support delivery of geo-spatial data in various data format including GML
- Ordnance Survey wanted a cohesive and responsive architecture and framework that could be reused in other Ordnance Survey initiatives.

Ordnance Survey realized that the complexity of the DNF and development of the GOS required an Object oriented approach. Ordnance Survey recognised that only by adopting an object-oriented, component-based approach would they be able to achieve the necessary flexibility in delivering data to their customers. The vast electronic map already contains details about more than 400 million features of the British landscape, all surveyed and stored in intricate detail - right down to the shapes of individual buildings.

Object-orientation provides a powerful abstraction mechanism based on encapsulation, information hiding and modularisation. These abilities allow software to be thought of as building blocks that can be assembled and re-assembled in order to meet user requirements. A consequence of the structuring principles supported by object-orientation is that it allows us to model the real world in meaningful terms as entities and relationships between entities. It therefore becomes easier to structure the system in the vocabulary of the geo-spatial domain.

The weight of opinion in the software development community and a view endorsed by Object Design is that object-orientation currently offers the best available method for handling large, complex systems. Object-orientation allows systems to be developed in a modular way so that extensions and refinements can be added without major disruptions to working components. It has been demonstrated in many situations to offer an excellent way of managing the risk of developing large systems.

Object-orientation also provides a consistent focus throughout analysis, design, and implementation so that we can use the same language and models to carry the analysis into design and then over into implementation so that object-oriented systems are more resilient, and are easier to maintain.

ObjectStore and the Geospatial Object Server (GOS) Solution

ObjectStore is the leading Object Oriented Data Database. The core of ObjectStore is a scalable and high-performance technology that leverages its object-oriented model and native support for Java and C++ to deliver unprecedented speed, reliability, scalability, and time-to- market demands required of mission-critical systems.

ObjectStore plays a key role in geo-spatial data management. It was selected for use in the GOS because of the following features:

- Direct support for objects and hierarchies of object
- High performance and scalability due to its CFA
- Transparent management of spatial data including flexible querying and indexing mechanisms
- 24*7 support

ObjectStore supports the storage of hierarchical information explicitly, e.g. as an object model, without the need to transform to a normalised relational structure that contains tables, rows and columns.

ObjectStore stores data as defined by the C++ or Java client that is accessing it. That is C++ or Java objects are stored as exactly that within the ObjectStore database. This direct support for object-orientation drastically reduces the amount of code required to manage persistent objects, and improves application quality whilst reducing development costs and time.

The ObjectStore database employs a technology called the *Cache Forward Architecture™ (CFA),* to actually distribute and push the data out to the applications or application servers that are actually performing the

calculations on the data. ObjectStore's CFA is designed for maximum application performance through load balancing, cache affinity, transaction services, and overall component coordination management. This architecture allows distributed components to access data from local caches at in-memory speed, instead of sending requests across the network to a central server.

Geo-Spatial information represents geographic information as a hierarchical structure, maintaining relationships between geographic entities. A GIS model typically contains vast numbers of one-to-many relationships. For example 'Areas' consist of many 'Lines' and 'Lines' consist of many 'Points'. This complexity coupled with the addition of inter-related points of interest information in a normalised relational model would involve many table joins between these entities. This leads to significant degradation in performance.

ObjectStore's implicit understanding of hierarchical information directly supports the complex inter-related structure of the imported GIS information. ObjectStore stores and retrieves spatial data as objects, along with other related properties. In doing so, there are no mappings and no 'joins', present in all other relational database technologies. This is especially important with very large and complex data sets such as the GOS one.

Data and Object Model

Ordnance Survey decided that it was imperative that Object Oriented techniques be used in the modelling of the mapping data. The underlying database would therefore have to support models that use inheritance, polymorphism, encapsulation, and navigation of relationships between objects. It would also have to support models that involve aggregations of large numbers of objects and chains of aggregations.

ObjectStore directly supports the object models that have been designed, but the database structure and the object model have been designed to do much more than model spatial data and store it in ObjectStore. There were a number of other requirements, which were vital to the success of the project. The first was performance. The system was designed to be customer and Internet friendly, therefore performance was an important factor. In a competitive environment it is important to Ordnance Survey that they be able to respond to new and changing business requirements. In order to support this the system had to have a dynamic data model, which allows online changes and updates to the underlying data definitions. And finally the system must be able to provide 24x7 operation.

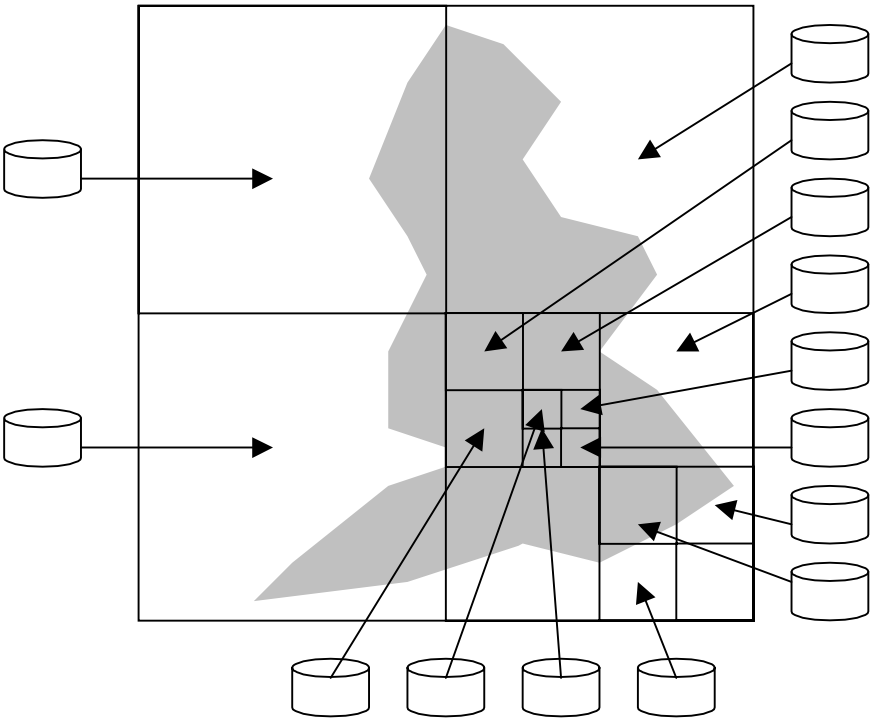
Database Structure

The volume of data for the Ordnance Survey has been estimated to be approximately 3-4 Terabytes. To effectively handle such a large volume of data it is necessary to *'divide et impera'* i.e. use multiple databases distributed across several machines. This has the following advantages.

Using a number of databases and database servers to manage access to the data supports scalability, both in terms of the size of data and the number of users of the system. The distributed nature of the system gives the ability to scale a larger number of smaller machines rather that a single powerful database server. Because functionality of the system is distributed across a number of processes, it means that any failure will only result in partial loss of functionality. And finally because the system is comprised a number of completely decoupled processes it also makes it possible to incrementally update parts of the system, for example with new software releases.

The map data is stored in multiple ObjectStore databases. Each database has two areas associated with it. A *nominal area*, which is the fixed area of the Great Britain covered by the database. This is a rectangle of fixed size, that does not overlap with the nominal areas of other databases. The geometry involved is justified onto a millimetre grid and any point is either in one database or in another. Consequently, databases have no geometric points in common.

The other area associated with a database is its *populated area*, which is initially identical to the nominal area of the database. As objects that overlap the nominal area of the database are inserted this populated area can grow so that it always bounds the farthest extremities of all the objects contained within the database.



1 Diagram of Databases and their Nominal Areas

This strategy ensures that each feature is only stored once in the entire set of databases. Storing each feature once guarantees that each query process operates on a unique set of objects.

Consequently, the distributor process doesn't need to handle duplicates and accumulate state, this allows the system to scale to large queries and a higher number of users.

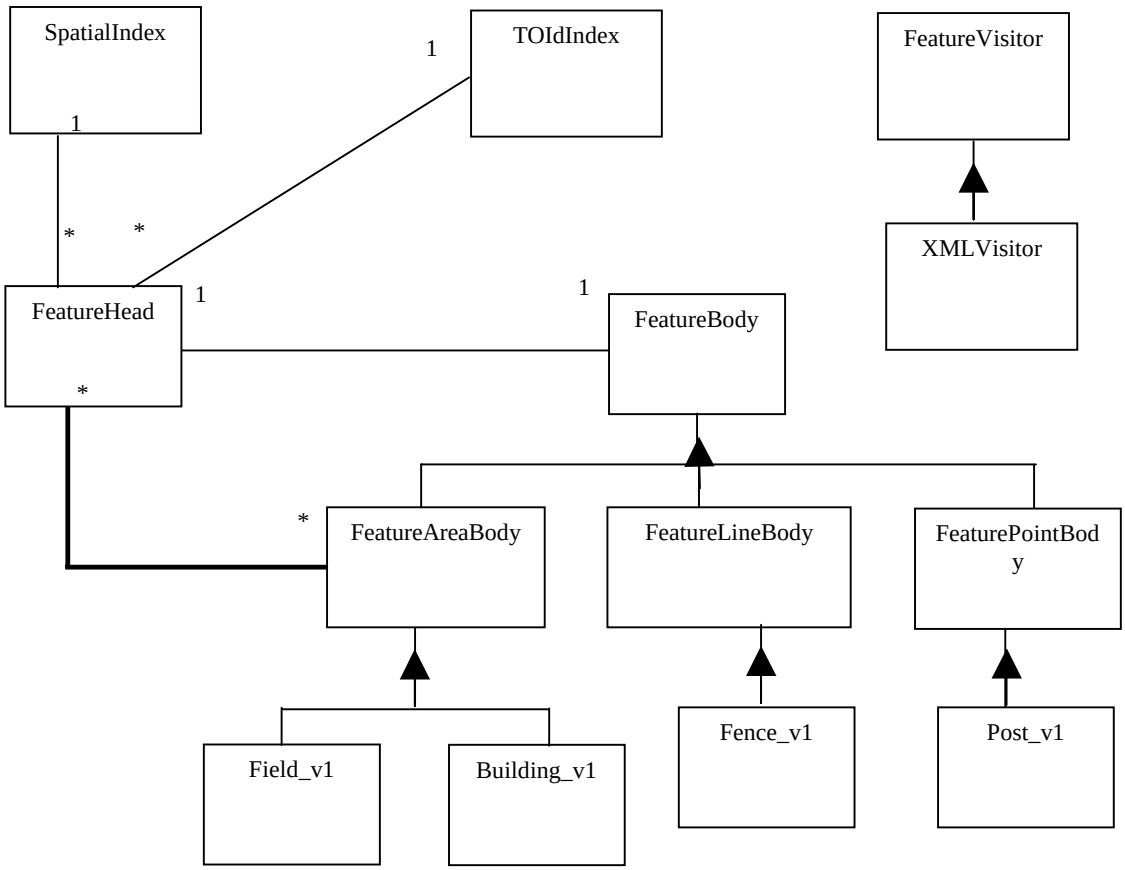
The overall configuration of the GIS components is stored in a configuration database. The configuration database records what database files exist, what machines they are on, and the content of the databases. This is used as an indexing mechanism to the data. The query splitter/splicer uses the configuration database to route queries to the appropriate databases. Therefore when a query is processed the databases that need to be accessed to perform that query can be quickly identified.

The configuration database also maintains a map of databases and spatial identifiers. This means that to select a particular object given a spatial identifier, the configuration will return the location of the database that contains the identifier. Once the correct database has been identified the object can be found via a single lookup operation.

Data Model

Central to the storage model are Ordnance Survey's topographic features. Typically these would be real-world objects that are the subject of surveys, but they could also be objects that are strictly internal to the Ordnance Survey.

The diagram below (2 GOS Storage Model) shows the main elements of the GOS class diagram.



2 GOS Storage Model

Each feature is modelled using the Bridge Pattern being split into two parts: a head class and a body class. These classes are called FeatureHead and FeatureBody.

The design of the body class has been optimised to represent the varying behaviour of different geometric constructs. For example it is possible to represent constructs such as Area, Line and Point. Attributes to represent the geometry of the construct and to define the

behaviour of the object are defined for each geometry construct. The head class contains a unique identifier called TOId. It points to the body associated to that head.

The big advantage of this approach is that if the type of a body object needs to be changed only the head object that references it needs to be updated. A sub-class of an existing body class can be created to add new behaviour.

The GOS foundation classes implement the visitor pattern to support the processing of features by programs such as the query processor. The FeatureBody provides an interface that accepts an FeatureVisitor.

This pattern makes it easy to add new functionality, particularly the provision of new output formats, without changing the existing persistent classes. An FeatureVisitor sub-class is written for each product produced from the GOS. Visitors can also be used to sweep the database to look for data quality problems, and to migrate features from an old FeatureBody sub-class to its replacement etc.

One of the most significant strengths of this model is with respect to performance. The combination of the definition of FeatureBody and FeatureHead and because ObjectStore allows objects to be physically clustered together reduces the number of pages fetched from the server when an FeatureBody is accessed. This has a high impact on performance.

Data Compression

One of the issues facing the GOS project was the sheer size of the data set. Compression of data both in storage and during transmission between processes provides major performance enhancements. Using techniques, including the use of string tables and bit-field compression, the data in the database has been compressed to approximately 300 Gigabytes from an original data set of 3 to 4 Terabytes.

Dynamic Data Model

Ordnance Survey needs to be able to respond to new and changing business requirements. The impact on the GOS system is that it will be necessary to make changes to the underlying data model and also update the processes that access the data. This is an important issue, especially when the data set is as large as the GOS one.

This process is made possible and manageable at Ordnance Survey because of the distributed nature of the system and because of the underlying object model.

To change the feature type of a feature, the FeatureBody instance is replaced with an instance of another sub-type. This is a very efficient process because there is only ever one persistent pointer to any FeatureBody – the one held in its FeatureHead. Thus replacing the FeatureBody is a straightforward process. And because the GOS consists of a number of decoupled processes, the system can be updated in an incremental way without effecting other parts of the system.

Data Navigation and Indexing

One of the key features of the GOS is the way in which the indexing strategy is implemented. The GOS indexing implementation represents a unique solution in the geo-spatial world.

The complexity and size of the information that needs to be managed and delivered in a real time environment such as the GOS, requires very sophisticated spatial indexing mechanisms that can easily be expressed directly in C++ models and held within ObjectStore. These indexing mechanisms can be critical to system performance particularly as the size of the database increases. Using ObjectStore enabled the implementation of bespoke and highly efficient indexing classes, almost impossible using a relational database. The development of such indexes is not possible with relational technology; if the indexes that are supplied with a relational database are not fast or flexible enough updates to the underlying database engine are required.

There are two basic queries in the GOS each supported by dedicated index structures within the GOS. One supports spatially based queries and the other supports queries based on unique identifiers associated to each head object.

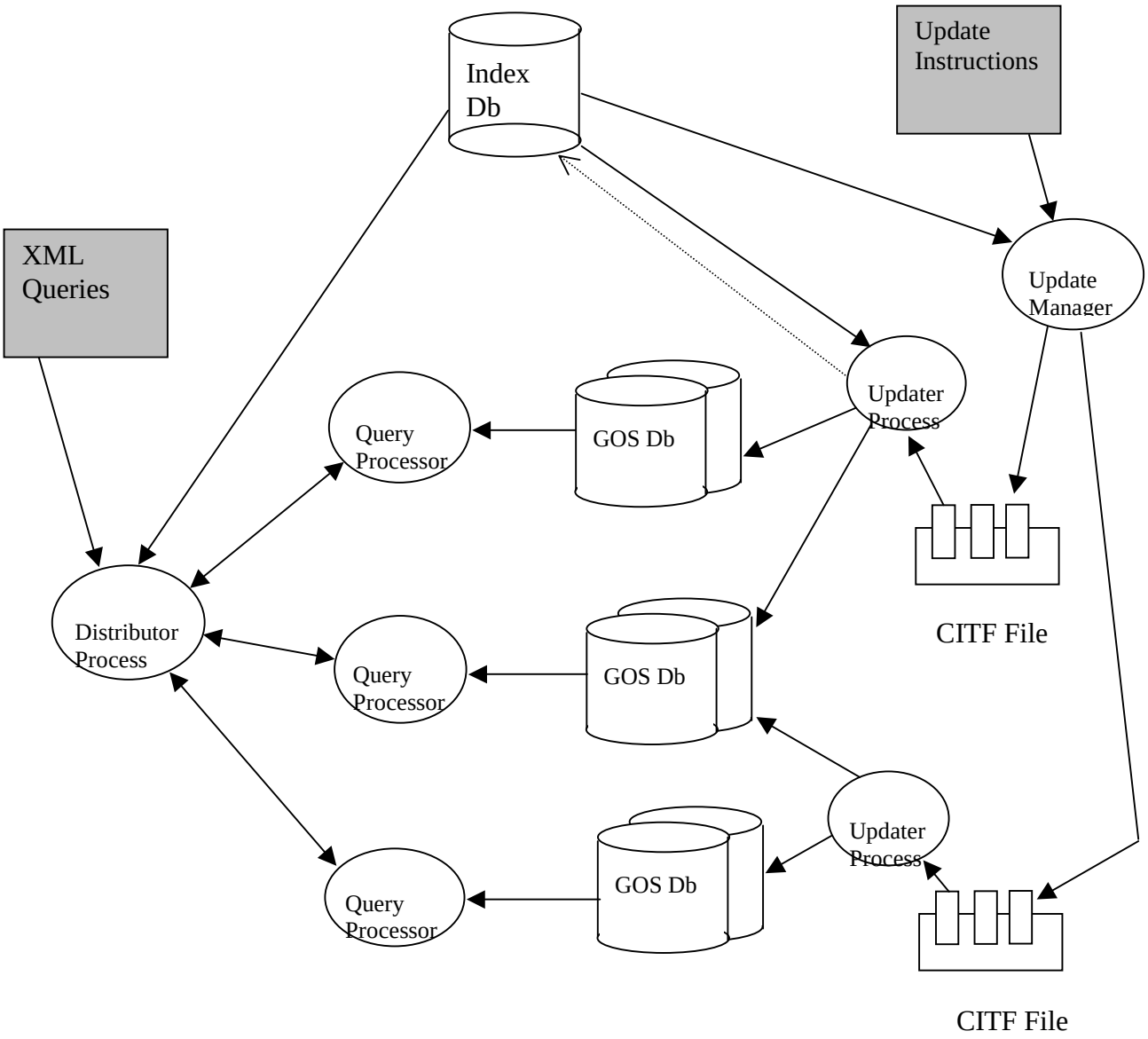
The spatial index defined at Ordnance Survey is a refinement of the well-established quad-index commonly used in GIS application. This refinement combines very well with ObjectStore in that once an object has been placed in the database, the index structure is created immediately to reflect that physical location. Further updates to the index do not require any movement of objects that already exist in the database. This eliminates the need to copy objects and update object pointers during index update.

Implementing an effective index strategy for spatial object identifiers was one of the main issues at Ordnance Survey, considering the large amount of data managed by the GOS. Standard indexing techniques would have consisted in massive index structure. A different strategy has been adopted at Ordnance Survey. It consists in storing ranges of identifiers rather than each identifier explicitly. The implemented solution occupies the minimum number of memory pages.

The GOS Process Architecture

The GOS process architecture is divided into two major activities:

- Those concerned with processing queries and serving out the results.
- Those concerned with managing data loading and updating.



3 The GOS process architecture

Processing Queries

To serve the necessary extensibility and interoperability requirements the query engine provides a modular query interface that can be extended dynamically to accommodate both new data formats and communication mechanisms without affecting the underlying geo-spatial storage.

Such flexibility, scalability, and the fully distributed architecture have been achieved by using ObjectStore's support for distribution and component based computing.

ObjectStore has actually been designed to support distributed component based computing. This means that as concurrent performance needs increase more components can simply be added to the middle tier to scale performance response.

Via ObjectStore's CFA, information is cached at the client accessing the information. Therefore it was possible to architect a distributed middle tier of components accessing information in parallel at in memory speeds.

The query engine architecture involves multiple query processes, each one working on a subset of data.

The Distributor process is responsible for receiving the queries in the form of GML, identifying which query processes are needed to execute the query, forwarding the query to the other process, called Query process, and combining the results when returned. This process is also called 'Splitter/Splicer'.

The Query process is responsible for executing the query, and there are multiple Query processes. Each one executes queries against a small number of databases each covering a unique area of the country. This has been possible since component based computing and ObjectStore support Routing. Routing associates specific components within the middle tier with particular functionality or information, and any requests that pertain to that functionality or information is automatically routed to the component.

This means for example that regional information could be partitioned onto specific individual data stores, each one simply dealing with requests for information relevant to its geographic region.

By regionalizing access to the database, queries that span large regions can be performed across a distributed environment. For example a query that spanned a number of regions will firstly be performed at each specific region, and the result set from all involved can then be combined. This obviously has the effect of being able to perform large queries across a number of machines or components.

The main advantage of this technique is that queries against the GOS are distributed among several processes and can run in parallel. These query processes are distributed across many CPUs and so genuine parallelism results.

Without the ability to efficiently distribute the processing, a centralized server based architecture would require very high-end specification hardware in order to simply run the query. With the added demands of multi-user and concurrent environments, the administration of such a server-based architecture in itself requires a very specialized skill set. All these factors contribute to a high cost of ownership, and poor performance.

Data Loading and Updating

This process has again been designed to maximize performance of the GOS, and ensure that whilst updates are occurring to Ordnance Survey's data there is no degradation to other parts of the system.

The Updater Process has been designed so that multiple instances of the process can run in parallel, each being responsible for a different area of the country and each having their own data queue. ObjectStore enabled the adoption of this approach since its distributed architecture allows the addition, update, and deletion of specific data/services without affecting any other deployed data/ services.

This would have been impossible using a relational technology, since its intimate nature means that changes to existing applications may mean intensive restructuring and application (re)development.

Conclusions

The work done at Ordnance Survey has led to the development of a highly scalable and flexible system. ObjectStore has played a key role in the development process of the GOS.

ObjectStore's direct support for objects and hierarchies of objects, and its easy management of spatial data, has meant it has been possible to design an object and data model in a very effective and flexible way. ObjectStore has also enabled the development of a unique and highly effective indexing solution. Scalability and high performance have been achieved by taking advantage of ObjectStore's CFA and inherent support for distribution. To summarise the key features of the system are

An indexing mechanism which provides fast data access and is flexible so that it can be updated in the face of new data access paths

A dynamic data model which means that the model can be updated in response new business requirements

A distributed process architecture, supported by ObjectStore and CFA, resulting in high performance and scalability

The solution implemented at Ordnance Survey represents a significant development for GIS applications. Never before have they had to support such high demands in respect to real time performance and extensibility.