

geoPOM: A Heterogeneous Geoscientific Persistent Object System

Silvia Nittel, Richard R. Muntz, Edmond Mesrobian
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024
{silvia, muntz, edmond}@cs.ucla.edu

Abstract

Lately, a need for uniform access to and integration of data stored in specialized, non-standard repositories such as GIS or multimedia storage servers has become apparent. In this paper, we provide an overview of a heterogeneous geoscientific persistent object manager (geoPOM) developed at the UCLA Data Mining Laboratory. GeoPOM provides users with the “illusion” of a single object-oriented spatial data store even though the data is actually stored in several different spatial data repositories, thus, allowing users to define and handle spatial data in a uniform manner. The geoPOM data model is based on the ODMG-93 standard for object-oriented data models, and the Open Geodata Consortium’s (OGC) standardization effort for temporal-spatial object types (OGIS).

1 Motivation

Distributed, heterogeneous data management is a prominent database research area due to the fact that a diversity of independent databases and file systems exists, and a need to provide easy and *uniform* access to integrated data has become everthmore apparent. In addition, despite database standards, a variety of heterogeneous data models exists today. Portability and/or automatic data model translation from, for example, an object-oriented data model to a relational data model, has become an important portability issue for applications that largely depend on interaction with a database system.

Both issues also play a significant role for applications that deal with heterogeneous, *non-traditional data repositories* such as geographic and geoscientific information systems (GIS) or multimedia data servers. Of particular interest to our research efforts are information systems which support the collection, analysis and generation of geoscientific data. Over the past two decades, a significant amount of geographic information has been collected, and the ne-

cessity exists to share, join and uniformly access this spatial data. Today’s spatial data is stored via numerous GISs that have syntactically, semantically and functionally heterogeneous interfaces and data models. In GISs, the definition and availability of spatial data types and operators varies significantly from system to system. Most GIS provide a rather specialized set of spatial operators; it is quite desirable to expose those operators through a heterogeneous system, rather than to require each repository to adhere to a common denominator of spatial functionality. Thus, the semantic and functional gap between GISs is much larger than that found in traditional heterogeneous systems. Unique characteristics of spatial data include sheer size and potentially complex structure (e.g. road networks). Therefore, data is computational-expensive, mandating high performance from a heterogeneous spatial object management system. Thus, providing uniform, integrated access to heterogeneous spatial data repositories introduces a set of new challenges and system requirements. However, research in this domain is still in its early stages.

At the UCLA Data Mining Laboratory, we are developing *geoPOM* (**geoscientific Persistent Object Manager**), a geoscientific object system capable of integrating heterogeneous spatial data repositories. GeoPOM is developed as part of the **Open Architecture Scientific Information System** (OASIS) project [21, 20]. OASIS provides an extensible, seamless environment for scientific data analysis, knowledge discovery, visualization, and collaboration based on CORBA [27]. In OASIS, we use geoPOM to implement the persistent state of geoscientific CORBA objects in a data repository-independent way.

GeoPOM facilitates access to data in repositories while maintaining their autonomy, and keeping the data in the repositories. GeoPOM provides an object-oriented data model which is extended with a set of predefined geoscientific object types such as *feature* and *coverage*, a rich set of geometry types and temporal-spatial reference systems necessary to model relevant information in the geoscientific domain. User-defined temporal-spatial object

types are automatically mapped to spatial data repositories such as a spatial DBMS or a scientific file archive format library, thus, making it possible to define spatial objects in a data repository-independent manner. Furthermore, geoPOM provides an object-oriented query language to pose queries to geoPOM objects. The primary focus of geoPOM lies in providing an abstraction layer from heterogeneous spatial data repositories by transparently mapping newly-defined geoPOM types to available repositories, and wrapping legacy data by geoPOM types. In this paper, we do not focus on objects whose state is stored across multiple repositories, query processing and concurrency issues.

Providing abstraction from heterogeneous data repositories introduces performance issues. While traditional heterogeneous data management systems are typically a DBMS in their own right, i.e. they consist of a run-time system, buffer management, etc., we want to minimize the access time for geoPOM objects, and avoid the two levels of access (heterogeneous system and actual data repository). Instead, the geoPOM compiler maps object type definitions and queries into concrete types and schemas of the target spatial data repository, and generates native data definition and access code which is linked into a client application. At run-time, a geoPOM application transparently accesses the spatial data repository directly using the generated code. Thus, the performance loss is minimized.

The remainder of this paper is organized as follows. Section 2 presents related work in this area. The characteristics of a heterogeneous geoscientific persistent object system along with new research directions introduced are described in Section 3. Section 4 presents the geoPOM data model. In Section 5, geoPOM's approach to integrate and abstract from spatial data repositories is described in detail with an example in Section 6. In section 7, implementation issues are discussed. Lessons learned while implementing and using geoPOM are described in Section 8, while Section 9 presents our conclusions.

2 Related Work

The primary focus for geoPOM's development is the implementation of a heterogeneous geoscientific object system. To the best of our knowledge, no such system exists today. However, two areas are closely related to our research efforts: the area of heterogeneous DBMS, and geographic and geoscientific information systems (GIS).

2.1 Heterogeneous Object-Oriented DBMS

Over the past decade, various approaches have been taken to address the issue of abstracting from heterogeneous data repositories. Starting in the mid-eighties, most of the work focussed on integrating relational DBMS through

multidatabase languages, and multidatabase systems with varying degrees of coupling between data repositories and the integrating system, as well as, the autonomy of the data repositories ([26, 4]). In recent years, approaches have been developed which leverage the benefits introduced by the object-oriented programming paradigm (powerful data modeling, new implementation strategies for problems in heterogeneous DBMS transaction management, etc.). A good overview of the work in this area can be found in [22, 6].

First generation object-oriented, heterogeneous systems provided their own data model, while recently developed systems are based on the object-oriented data model defined by ODMG [8]. However, several extensions have been added by different systems in order to make the ODMG model feasible as a common data model for heterogeneous DBMS; proposals have been made for adding view mechanisms [7, 1, 17], for object identifiers that do not guarantee immutability during an object's lifetime (so-called weak identifiers) [7, 14, 19], and for weak references which are based on weak identifiers [7]. Also, a variety of new issues in the area of heterogeneous DBMS have been addressed such as object-oriented transaction management [11, 30], and the integration of non-traditional DBMS through operational mapping [3].

However, a heterogeneous system that provides application-specific types like geoscientific data types, multimedia data types or other scientific data types, and internally supports the optimized mapping to corresponding storage systems like spatial DBMSs, or scientific file format systems, does not exist. Notable first approaches in this area, although not addressing GIS, are the *Garlic* project [7] and different systems in the area of molecular biology (e.g. [23, 9]). *Garlic* employs the ODMG's data model and adds specific data types for multimedia applications which are mapped to a variety of multimedia storage systems. Systems in the area of molecular biology mostly provide an integrated environment for retrieval of molecular biology data from external on-line databases combined with the retrieval and management of local data.

2.2 Geoscientific Information Systems

Geographic information systems appeared in the early seventies. At that time, computer science and especially DBMS interest in GIS had been slow due in part to the inadequacy of data models to support the semantically rich geographic and spatial data types, the non-availability of comparatively cheap powerful, high performance processors and data storage devices. With the rise of non-standard data models like object-oriented and extended relational data models as well as non-standard DBMSs, development of GIS support has been one of the prominent application

domains for non-standard DBMS research [10, 29, 24]. Additionally, high-performance computers, high-speed, ubiquitous communication, and cheap disk and tertiary storage devices provided the enabling technology necessary for computer-aided geoscientific information systems.

Today, a variety of commercially-available GIS packages exist, most of which use file-based techniques for data storage. Many low-level spatial data repositories are available today. They are mainly used for platform-independent storage and retrieval of large scientific data sets and are available for different scientific domains such as meteorology, astronomy, for space mission data, etc. [12]. In recent years, relational and extended relational DBMS technology has been employed for data storage by GIS systems in order to support controlled data sharing between applications and efficient queries based on the combination of spatial and non-spatial attributes. However, in off-the-shelf products as well as in research prototypes, data storage for GIS is generally assumed to be a homogeneous, centralized DBMS [16, 25, 28] even though processing may be parallelized [10]. Furthermore, lack of a standard spatial data model hinders interoperability between GISs of different vendors today.

On the other hand, new challenges of scale are continually introduced by e.g. the EOSDIS (Earth Observing System Data and Information System) project. EOS is a collection of satellites to be launched by NASA starting in late 1998 to monitor global changes in the environment. It is anticipated that a fully operational EOS will generate about a terabyte of data per day. This information has to be stored, fused with other existing information and made available to interested scientists, and the public at large.

3 Integrating Heterogeneous Geoscientific Data Repositories

Since integrating and abstracting from heterogeneous spatial data repositories is a comparatively new research area, it is necessary to first define the characteristics of such a system, and then describe the requirements imposed and a set of new research challenges in this area.

3.1 Overview

The goal of a heterogeneous geoscientific persistent object system (POS) is to provide applications with the benefits of a *single geoscientific repository*, but without actually storing the data in a single spatial repository. Rather, a collection of diverse (spatial) DBMS or scientific file format systems (e.g. HDF or netCDF) store the data. However, the system provides the user with the “illusion” that the “same” spatial repository is always used, thus, achieving location

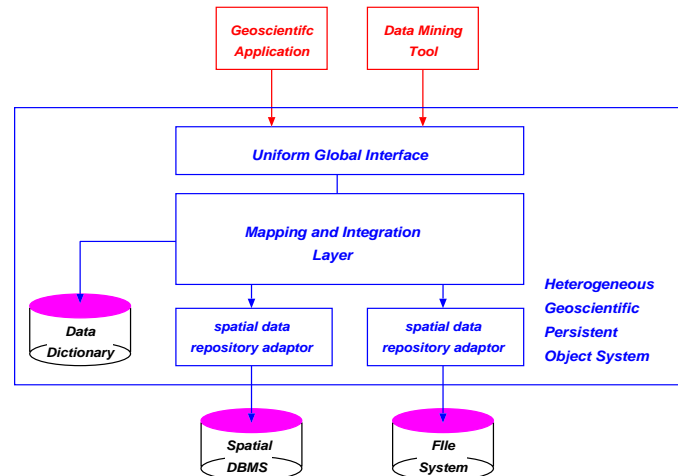


Figure 1. Architecture of a heterogeneous geoscientific persistent object system

transparency for the user’s objects. An overview of the architecture of a heterogeneous geoscientific POS is depicted in Figure 1.

Abstracting from the various spatial data models and APIs used by the participating spatial repositories, a heterogeneous geoscientific POS provides a *uniform* spatial data model that enables a client to define and query geoscientific information in a repository-independent fashion. This data model can be based on relational, extended relational or object-oriented data model techniques, and provides a predefined set of spatial data types such as *points*, *lines*, and *polygons* as well as a collection of spatial operators such as *intersection()*, *contains()*, *distance()*, *boundary()*, etc.. We will refer to this data model as the *common spatial data model* (CSDM). The CSDM is used to define new structures which are based on existing spatial data types. The CSDM’s query language enables users to express queries against spatial data in terms of POS types and apply spatial operators as part of the queries. Such queries are translated onto the query language of the repositories, thus, allowing users to perform spatial query processing over data stored in the component spatial repositories.

A collection of repositories is used to provide the actual storage of the data defined with the CSDM; a repository participating in a heterogeneous geoscientific POS typically provides some kind of spatial functionality.¹ A *spatial data repository* consists of the management software (e.g. the Illustra DBMS, or the HDF library) and the data that it manages.

¹Non-spatial repositories might also participate in a heterogeneous spatial POS; however, we assume that such a repository would be used for the storage of non-spatial data such as data dictionary information.

Types defined with the CSDM are translated to “types” defined in terms of the API of the spatial data repository in which the data is stored. The mapping involves creating an equivalent “type” defined in the syntax of the underlying spatial data repository’s API. For example, for a road object with a set of non-spatial attributes and a line string geometry object, a table consisting of equivalent attributes and the local “equivalent” geometric type is created. While an object type can be mapped to an equivalent table structure, it might not be possible to map a geometric type to an exactly equivalent type of an underlying repository. Similarly, requests for data creation, manipulation, and queries defined in the CSDM’s data manipulation and query language are translated into calls to the target spatial data repository’s API. Since the mapping of both type and manipulation commands to a spatial repository are repository type-specific (i.e. the repository type’s specific API and syntax), the mapping is performed by a *spatial repository adaptor*. This component “knows” how to create the equivalent spatial types and commands for a repository API. Information about types, data objects, and other administrative data are managed in the POS’s data dictionary.

3.2 Requirements

Powerful Common Temporal-Spatial Data Model

A heterogeneous geoscientific POS should provide a CSDM that supports high-level abstractions allowing simplified modeling of geoscientific information. In geoscientific systems, two types of information are prevalent: *vector* and *raster* data. Vector data includes points, lines, polygons, etc. which represent real-world entities as (x,y) coordinates in a Cartesian or spherical coordinate space. Raster data is comparable to a photograph which is composed of cells (or pixels); each cell is filled with a value.

In addition to simple GIS object types, it is necessary to model *complex GIS entities* such as political boundaries (e.g., hierarchical structure with the levels of continents, countries, provinces, counties and cities). It is desirable that these entities and their relationships are modeled in a simple manner, and that the application of operators results in the retrieval of an entity and all its subobjects (e.g. retrieving information about a country, its counties, cities, etc.).

In addition to complex spatial structure, geoscientific data often has a *temporal component* such as the time a satellite raster image was recorded and represents the state of a certain geophysical phenomenon. These phenomena are often recorded in intervals to provide time varying data analysis. Furthermore, temporal data can be represented via different time models. For example, real-world time is used for measured real-world data, while an “artificial” model time (e.g. 30 day month) is used for model-generated data (e.g. general simulation models). A CSDM should enable a

user to represent the temporal dimension of spatial data in a variety of time models.

The CSDM should support the expression of powerful queries against the geoscientific information stored in the heterogeneous geoscientific persistent object system. Beside a declarative query language, a CSDM should provide support for spatial queries such as calculating the distance between two vector objects, length of lines, etc., proximity analysis such as the computation of a “buffer zone” around an object, raster image processing, network analysis, and others.

Performance

A heterogeneous geoscientific persistent object system introduces, by supplying a layer of software to abstract from a collection of spatial repositories, an indirection that, on the one hand, offers added functionality and ease of code portability, but, on the other hand, might introduce a performance penalty compared to using these GIS directly.

In this context, we define performance as the time to access, retrieve and manipulate information in a GIS. Clearly, using a heterogeneous geoscientific POS at coding time outperforms the traditional approach of writing (from scratch) an application which combines information from different GISs for a specific query or rewriting an existing application using a different GIS. However, since GIS processing is complex and time-consuming in itself, a heterogeneous geoscientific POS should introduce only minimal overhead at run-time. A high-performance heterogeneous system is desirable for data analysis and visualization tools, distributed GIS applications, and data mining efforts.

User-definable Optimization

System performance can be enhanced if user knowledge about data, data usage, and data layout is integrated into the system’s data management policies. For example, a user might have specific knowledge as to how a geodataset of a particular type is likely to be accessed and used; thus, his/her knowledge should be allowed to influence the storage layout of the dataset or the choice of spatial access paths. Furthermore, users might need to access the same datasets in different ways; e.g., a network analysis application might experiment with a data set consisting of a set of point, line and area features (such as census data for a city planning application) over a long period of time while rarely updating the database. For these applications, performance can be improved if the user can request that the relevant data is cached in memory. Another application might only access a small subset of the same data set; in this case, the performance is enhanced when only relevant data is retrieved from the underlying spatial repositories by the heterogeneous geoscientific object system. Thus, the

user should be allowed to choose strategies for caching/non-caching of data.

3.3 Problems

The integration and abstraction from *non-standard* data repositories such as GIS and spatial DBMS or multimedia storage systems introduces a set of research challenges which already exist in the area of traditional heterogeneous DBMS in a basic form, and also new challenges.

Capability-Based Usage of Spatial Repositories

Providing a heterogeneous spatial data management system is motivated by the existence of large amount of GIS data that one wants to share, join, and access uniformly, and the need to integrate specialized *services* of single spatial repositories, and make them available through a common interface. Thus, a heterogeneous system provides a larger suite of spatial functionality than any single participating repository through functionality-integration while repositories are used according to their capability. This capability-based usage of repositories introduces a new problem which can be divided into two aspects. First, independent of their specialized spatial services, spatial repositories offer functionality that can be “low-level”, as is the case for file-system based geoscientific file archive format systems (i.e. basic storage types and access operators), or “high-level” as provided by a spatial DBMS (e.g. spatial query language, storage optimization, etc.). Second, regarding the spatial functionality itself, there is a significant difference in availability and implementation of geometry types, spatial operators, coordinate systems, spatial reference systems, etc. offered today by spatial repositories.

When abstracting functionality from such a set of spatial data repositories, the goal is to preserve a high common denominator of the functionality provided by the individual spatial repositories. On the other hand, it is not a goal to implement a large amount of additional code on top of low-level repositories to “bridge” the functionality gap between the high-level, DBMS-like functionality of the CSDM and the rudimentary functionality of a scientific file format system, or to expect that each participating repository supports similar spatial functionality.

Mapping of Syntactically and Semantically-varying Spatial Data Models

The process of mapping the concepts of a CSDM to the spatial data models or APIs of the heterogeneous spatial data repositories can be divided into two parts: mapping the non-spatial part of the CSDM and mapping of spatial object types of the CSDM to the spatial API of the underlying repositories. For the first part, a set of solutions

has been proposed in the field of traditional heterogeneous data management systems such as mapping relational and object-oriented data models to relational, network and hierarchical data models [2, 18]. The second part, however, introduces a set of novel problems. Spatial data models or APIs vary significantly in the spatial concepts they offer. First, they differ in the availability of geometry types and their operators. This might necessitate the *approximation* of a spatial type of the CSDM by a simpler type supported by the spatial repository. However, this mapping might introduce inaccuracy. Inaccuracy is a novel problem to deal with in heterogeneous systems, since it challenges the premise that the heterogeneous system is able to hide the differences between underlying repositories. Second, the semantics of similar types in two systems might vary. Third, spatial repositories might not support all coordinate systems offered by the CSDM.

4 geoPOM Data Model

geoPOM’s data model is based on *object-oriented data model technology* for several reasons including its powerful modeling and abstraction capabilities (e.g. user-definable object types with type-specific operators, complex objects, and inheritance between object types), and the fact, that lately, an increasing number of new geoscientific applications have been based on object-oriented and distributed software technology such as CORBA, DCOM or Java. Therefore, providing an object-oriented data model avoids an impedance mismatch between applications and geoPOM. To enhance the reusability of geoPOM and its applications, we have based its data model on two standards: the proposed standard for object-oriented data models by ODMG, and the work done by OGC on the definition of a standard GIS object model as part of the Open GeoData Interoperability Specification (OGIS) [5].

The ODMG model is the foundation of the geoPOM object model. Used without spatial types, geoPOM provides the functionality of a “plain” ODMG-based heterogeneous system. The main spatial types are derived from the ODMG base types (*Persistent Object*, *Collection*, *Iterator*, etc.), and can be used to derive “spatially-enabled” objects. Since the OGIS standardization is an ongoing effort, we have taken a snapshot of the OGIS model, and extracted a subset of types necessary for the geoscientific application domain to model feature collections with simple and complex geometry as well as geoscientific sets consisting of raster satellite images.

The basic modeling primitive in the geoPOM object model is the *feature*. The type *Feature* is derived from *Persistent Object*, and represents a real world or abstract (spatial) entity. A feature contains geometric and non-geometric properties (i.e. attributes or relationships), and a set of

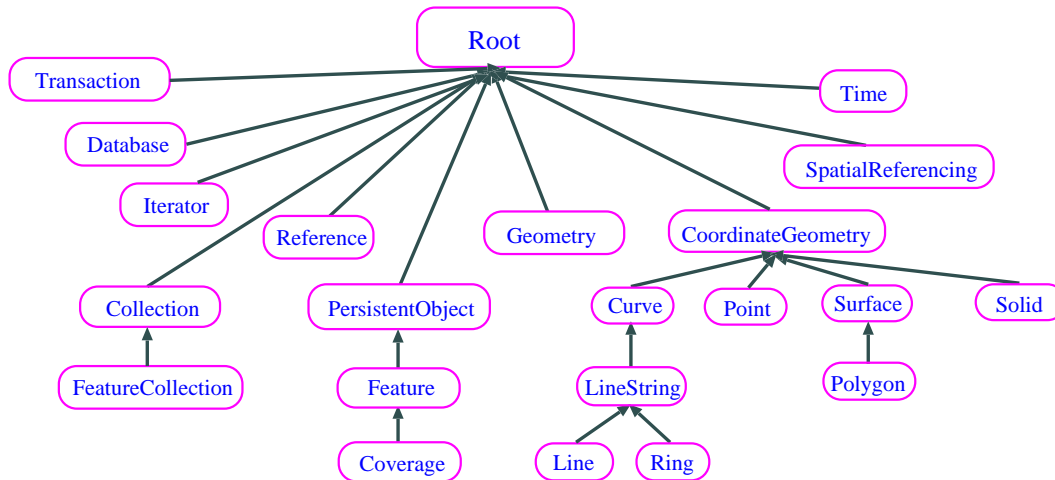


Figure 2. geoPOM's spatial data model

methods (e.g. a road is a feature). The second model primitive is the *coverage* which is derived from feature, and is an association of points, polygons or solids within a spatial-temporal domain with a value (of a certain type). E.g. point set coverages are used to represent raster images. A *spatial domain* consists of a *geometric description* such as a description of points, curves, surfaces etc, and includes a mechanism for mapping the underlying coordinate geometry to a *spatial reference system* (such as a coordinate system consisting of latitude and longitude coordinates). The *Feature* and *Coverage* type are the root object types for all spatial types (comparable to the *Persistent Object*). Aggregations of spatial objects are modeled by *feature collections*. Geometric object types support spatial operators such as *intersection()*, *contains()*, *distance()*, *boundary()*. More details of the spatial model can be found in [5]. Figure 2 depicts an overview of the basic geoPOM data model and its predefined spatial types.

GeoPOM provides spatial query processing services. geoPOM's query language (geoOQL) is based on the query language OQL of ODMG-93. GeoOQL is a superset of SQL92, and is based on select-from-where clauses. However, geoOQL also deals with complex (spatial) objects, and supports (spatial) method invocation and path expressions within select-from-where clauses. Spatial query processing is performed by invoking the spatial operators within a query. E.g., a query to a collection of cities would be expressed as “*select c.number_of_inhabitants from Cities c where c.extent.ContainedIn(area);*”. The object-oriented part of geoOQL supports the expression of complex queries; combined with the use of spatial operators, spatial query processing can be performed in a powerful fashion.

The concepts of the geoPOM object model support the definition of diverse collections of spatial objects (e.g. lists,

sets, bags, and arrays). In order to provide fast access to subsets of these temporal-spatial object collections, *spatial access paths* such as R-trees, Quadtrees, etc. are provided in addition to basic indexes (hash, B-trees) by geoPOM. The geoPOM object model consists of the geoPOM Object Definition Language (geoODL) and the geoPOM Object Manipulation Language (geoOML).

5 The geoPOM Approach to Integrating Heterogeneous Spatial Repositories

This section describes the approach employed by geoPOM to integrate and abstract from the functionality and varying capabilities of a collection of spatial data repositories. First, the architecture of geoPOM is presented along with the design and implementation decisions made to achieve a high-performance system. Second, the architectural aspects related to the varying capabilities of spatial repositories is described.

5.1 High-Performance Geoscientific Persistent Object Manager

One of the major goals of geoPOM is to provide high-performance geoscientific data management for heterogeneous spatial repositories. Two main architectural decisions were made to achieve this goal: first, the elimination of a set of indirections which are known from the “classical” architecture of heterogeneous systems ([26]), and second, the empowerment of users to choose several optimization parameters.

A full-blown heterogeneous geoscientific persistent object system is a very complex software system; several assumptions were made to make the problem more tractable.

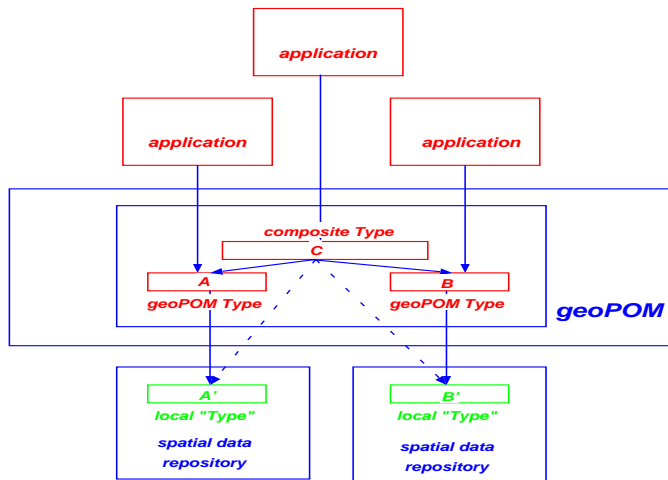


Figure 3. Type Mapping Architecture

First, the state of a geoPOM *object* is always mapped to a single spatial data repository, i.e. it is not spread across several repositories. However, a geoPOM *type* can have objects in several different repositories. Second, for the time being, queries can only be posed to a geoPOM object collection; i.e. full query processing is not supported yet.

The main task of geoPOM is to map geoPOM types to “types” defined in the API of a given spatial repository. Since the geoPOM types are known to users and are independent of a particular repository, they are called *global types*. From geoPOM’s perspective, the “types” a given spatial repository supports are known only by that the repository (and geoPOM) and are therefore called *local types*. The architecture of traditional heterogeneous DBMS consists of five intermediate levels of mapping between local and global types [26]. To avoid multiple mapping steps and the processing overhead introduced, we have simplified this architecture to three levels while still being able to achieve data integration and management of semantic heterogeneity (see Figure 3).

- The first level consists of the *local “types”* (i.e. relations or storage types) of the spatial data repositories in the geoPOM environment. They are described by means of the data model or API of the data repositories.
- A local “type”, i.e. a set of relations is transformed into a corresponding *global type*, i.e. a geoPOM type, expressed in terms of the geoPOM data model.
- The third level allows for enhancement, simplification or composition of geoPOM types via views. Views are also used to manage semantic heterogeneity.

Implementing two levels of types provides a direct mapping of a geoPOM type to a local “type”; using minimal

indirection, improved performance is achieved during runtime. While a geoPOM object is stored in a single repository, geoPOM *collection types* containing objects in different repositories and the *extension of a type* provide support for *data integration* across repositories.

The mapping of a geoPOM type to a local type (or vice versa) is divided into two parts: on the one hand, mapping the object-oriented concepts of geoPOM’s CSDM to the non-spatial part of the underlying spatial repository’s data model, and, on the other hand, mapping geoPOM’s predefined spatial types to the repository’s spatial functionality. The first part is based on [2, 18] in geoPOM.

The mapping of geoPOM’s spatial types and operators, however, is much harder to solve, in part due to the significant heterogeneity of spatial object models, especially for the geometric data types and their operators. Each geoPOM spatial type is mapped (via a particular implementation) to a corresponding spatial type of the different spatial data repositories; e.g. the *point* type of geoPOM’s CSDM is mapped to a *point* data type of Illustra or a *point feature* in ESRI’s SDE. Similarly, a geoPOM type’s operators are implemented using the corresponding operators of the spatial “types” of the underlying repository; i.e. a geoPOM spatial type “knows” how to map itself to the corresponding concepts of a supported spatial repository. However, the mapping of geometry, especially simple geometry such as e.g. a line string as part of a road object, is closely related to the mapping of the feature itself. Instead of treating the geometry object as a first-class object (i.e. being mapped to its own table structure and having an id), it is more efficient to combine the mapping of a simple geometry object with the feature object (i.e. append the geometry data type to the table of the feature). Therefore, in geoPOM the mapping of simple geometry is handled by the feature. The mapping of a *collection* of geometry objects, however, is managed by the collection type. GeoPOM only supports collections of geometry objects (rather than collections of references to geometry objects); therefore, the collection object acts as a container treating the geometry objects as its dependents, and creates e.g. a table with the set id as primary key, and the geometry data type as second attribute.

Similarly, legacy data is “wrapped” by geoPOM object types. For each legacy data set, mostly a relation, a set of relations or data files, a corresponding geoPOM type is defined. For simplicity, we assume that the same attribute names of the legacy data set are used for the (simple) attributes of the geoPOM object. For a set of legacy tables which are joined over foreign keys, a semantically-equivalent structure is chosen for the geoPOM type (using collections, references, or relationships). Here, the significant task is to define an equivalent structure of the geoPOM object type². The geoPOM compiler will generate all other

²The geoPOM type implementor has to be aware of the way geoPOM

necessary code to access and manipulate the legacy data set, so that only the correct geoPOM type has to be defined. It might be possible that a semantically equivalent type is represented differently in two (or more) repositories; here, for each repository an equivalent geoPOM object type is defined. The semantic heterogeneity is solved by defining a view over both types.

Object types defined via the geoPOM data model contain methods. We distinguish two types of methods for an object: *predefined methods* such as create, delete, activate, modify, and deactivate, and *user-defined methods*. Predefined methods are mapped to corresponding calls (or a combination of calls) provided by the spatial repositories. User-defined methods, however, are implemented as methods on the geoPOM C++ objects, and executed on instantiated C++ objects (in the application's process).

Besides a simplified internal architecture, geoPOM supports various optimization parameters that can be chosen by the user. GeoPOM objects are instantiated as C++ objects at run-time. However, it might not be optimal to load the complete state of a geoPOM object into memory; e.g. a user is interested in accessing certain temporal-spatial "slices" of a raster object rather than the entire sequence. Thus, when a user instantiated a geoPOM raster object, geoPOM does not load the object in its entirety into memory, but allows the user to retrieve only the object parts of interest. Spatial object collection are handled in a similar fashion; however, in this case the user is allowed to choose whether the elements of a collection should be cached in memory (thus, all elements are instantiated as C++ objects), or if elements should be instantiated on demand.

5.2 Capability-based Usage of Spatial Data Repositories

Providing an abstract layer over various specialized repositories requires the ability to handle dissimilar capabilities of spatial repositories. We distinguish mainly two types of varying capabilities: first, differences in the *data model and query capabilities*, and second, differences in the *spatial capabilities* a repository provides. For the time being, we assume that the missing (query) functionality of low-level repositories is implemented in the repository adaptor. In the section below, we describe the approach taken by geoPOM to handle varying *spatial capabilities* of participating spatial repositories.

In a traditional approach to heterogeneous data management, one assumes that all object types definable via the CSDM can be mapped to *each* participating data repository; thus, repositories are interchangeable, and can be used *transparently* by the integrating system. However, this is

maps object types to e.g. relational-based repositories.

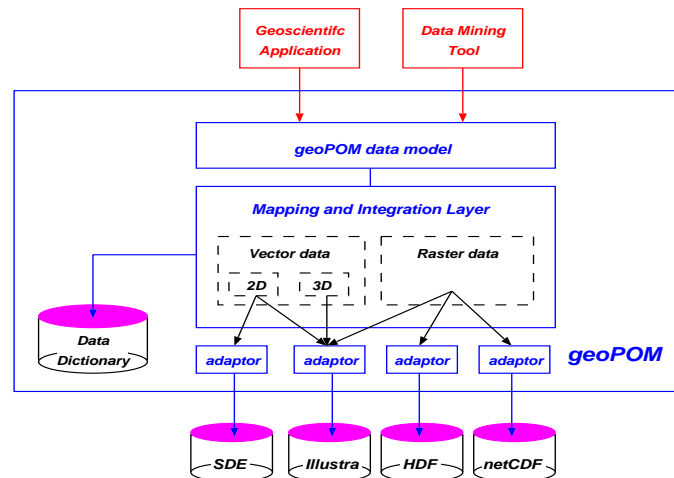


Figure 4. Capability-Based Usage of Spatial Repositories in geoPOM

not the goal for specialized repositories, since the repositories should be used according to the spatial capabilities they offer. Furthermore, the integration system provides a superset of functionality available at the single spatial repositories. However, the need for a semi-transparent use of repositories still exists in order to reduce the complexity of geoPOM.

The spatial capabilities of repositories can be described by the repository's support for vector data types, raster data types, coordinate systems, time-models, etc.. To make the use of spatial repositories (semi-)transparent to geoPOM, we classify *geoPOM's spatial object types* as well as the *capabilities of a spatial repository* by describing repositories via a collection of *functionality classes*, and *repository classes*. The spatial functionality of geoPOM's CSDM is divided into two main classes: vector and raster data. Each class is specialized; e.g. the vector data class is divided into 2-D vector data, and 3-D vector data. The number of subclasses is a function of the number of coordinate systems supported. Each geoPOM spatial type is assigned to one *functionality class*. Membership in a *repository class* indicates that a given repository provides support for the spatial data types described in the corresponding functionality class. E.g., the vector repository class of geoPOM contains all repositories that handle vector data. Each concrete spatial repository is assigned membership in one or more *repository classes*. We assume that all repositories within a repository class can be used interchangeably (see Figure 4).

geoPOM's spatial types and their mapping to relevant repositories are implemented within the geoPOM class library, and information about repository classes is used within it. A spatial geoPOM type can be integrated into any

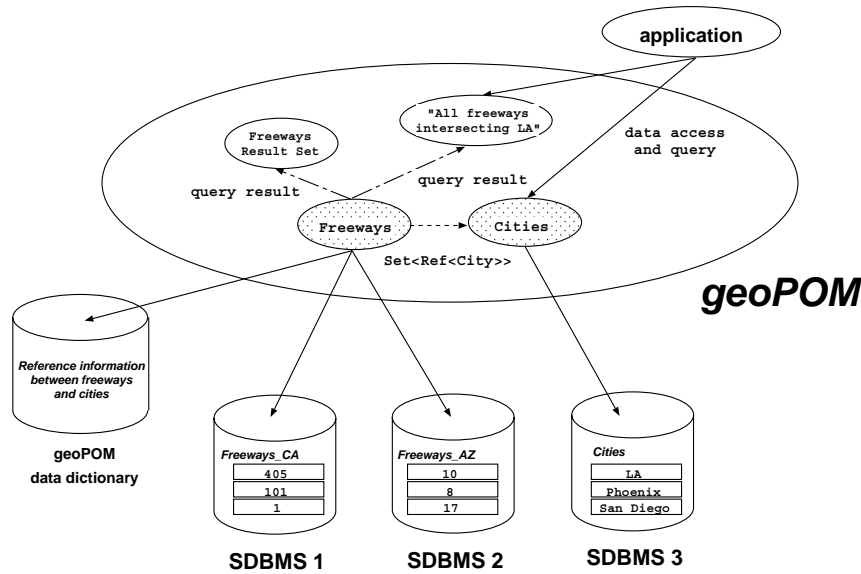


Figure 5. Example

user-defined geoPOM type; at compile-time of a geoPOM type, it is not defined which repository a user will choose to store the data (i.e. a repository is selected indirectly by the choosing a geoPOM database). Furthermore, the “real” type of a geometry object type such as *Point* can not be determined yet, since it depends on the instantiation of its spatial reference system object (e.g. 2-D or 3-D, Cartesian or spherical coordinate system). The necessary information, however, is available at run-time. GeoPOM uses information about repository classes to determine efficiently whether the (indirectly) chosen repository can be used for storing the spatial object.

Since geoPOM provides a superset of functionality that is normally available at any single spatial repository, the concept of repository classes also helps to ensure that a given installation of geoPOM with a selection of spatial repositories is indeed able to provide the full set of functionality. At installation time, the database administrator has to make sure that each repository class contains at least one member.

6 Example

Suppose a user wants to query and analyze freeway and city information across California and Arizona (see Figure 5). The user defines a geoPOM schema consisting of the types *Freeway* and *City*. In this scenario, the information happens to be scattered across three databases. The extent (i.e. the set of instances) of the type *Freeway* is spread across two different databases, one containing the freeways of California (managed by Illustra), while the other one con-

tains the freeways of Arizona (managed by SDE). The extent of *City* is stored in a third database managed by another Illustra installation.

Local Database Schemas

Before describing the geoPOM schema itself, it is instructive to take a quick look at the existing local schemas for freeways and cities. The schemas can be described by the following table creation commands (for simplification, the same schema for freeways is used for both repositories):

```

Illustra schema:

create table City (
  name      text not null unique,
  state     text not null,
  city_area Poly
);

create table Freeway (
  freeway_number int not null unique,
  freeway_geometry Path
);

SDE schema:

create table Freeway (
  freeway_number int not null unique,
  freeway_geometry Shape
);

```

A *Freeway* tuple contains the freeway number as the primary key, and its geometry is described by a linestring geometry in both cases (as a *Path* ADT in Illustra, and as a *Linestring* geometry (class derived from type *Shape* and instantiated at run-time) in SDE). A *city* tuple is described by its name and state (primary keys) and contains a polygon defining the city’s area.

geoPOM Schema

The corresponding geoPOM types are defined as follows:

```
class City : public Feature (
public:
    char*          name;
    char*          state;
    Polygon        city_area;

    City();
    City(int oid);
    ~City();

    // creates new object persistently in a geoPOM database
    void* operator new(size_t size, Database* database,
                      const char *typename, char* name);

    // deletes object from the database
    void delete_object();

    void activate();
    void deactivate();
    void mark_modified();
};

class Freeway : public Feature (
public:
    int            freeway_number;
    LineString     freeway_geometry;
    Set< Ref<City> > cities;

    Freeway();
    Freeway(int oid);
    ~Freeway();

    // creates new object persistently in a geoPOM database
    void* operator new(size_t size, Database* database,
                      const char *typename, char* name);

    // deletes object from the database
    void delete_object();

    void activate();
    void deactivate();
    void mark_modified();

    // user-defined methods
    float get_freeway_length();
};
```

The geoPOM types derive a set of create, activate and delete functions from *Feature* which defines their basic database behavior (the method code is generated by geoPOM). For the *Freeway* type, additional information is modeled such as the user-defined function *get_freeway_length()* and a set of references to cities which intersect or touch a freeway. The geometric attribute is defined via a *LineString* type which can be mapped, without introducing inaccuracy, to both the SDE and Illustra types. The geoPOM type *City* is defined in the same way. The additional reference information which relates freeways to the information stored in the city database is stored in geoPOM's data dictionary (each freeway object identifier is stored with a set identifier).

Querying the data

The user can query the extent of all freeways, and iterate through the result set, or select all freeways that intersect Los Angeles using the following code fragment:

In geoPOM, a query is represented as an object containing the query string. The function *oql_execute* is a free-standing template function which executes the query and

```
Ref<City> la;
```

```
OQL_Query q("select f from Freeway f where
            f.freeway_geometry.intersects(la.city_area)");
Set< Ref<Freeway> > freeways_of_LA;
oql_execute(q, freeways_of_LA);
```

copies the result set in the second parameter given as an argument, in this case a set of references to freeways. The function *intersects* within the query string is a predefined spatial operator for the *LineString* geometry. The query passes the query to the participating repositories and uses the Los Angeles area as an argument for the spatial predicate. The repositories' query engines make use of available spatial access paths to evaluate the query. The result set is instantiated as *Freeway* objects at the geoPOM level (i.e., the application level).

7 Implementation Issues

The initial geoPOM prototype provides a compiler that allows the user to define geoPOM types using ODMG syntax. The compiler generates the corresponding C++ class, i.e. the .h and .C files, and the code to create, activate, deactivate, modify, and delete geoPOM objects. The user writes user-defined methods only. Furthermore, geoPOM provides a C++ class library that contains the collection of ODMG base classes, i.e. *persistent object*, *database*, *transaction*, *reference*, *collection*, *set* and *iterator* and the described spatial types. The user-defined geoPOM types and the geoPOM class library are used to write geoPOM applications.

To provide the ODMG base classes to the user, the geoPOM class library internally contains the code to connect to different spatial repositories, manage sessions, translate between the CSDM and the spatial repositories' API, open databases and perform transaction management. In order to keep the implementation of the ODMG base classes separate from the actual spatial repositories, we designed the class library using the "bridge" design pattern [13]. Here, an object like *Database* delegates its implementation to an implementation object; thus, e.g. the body of the *open* method contains a call to the *Database* implementation object. The implementation object is an abstract object; at runtime, however, a concrete implementation object such as the *IllustraDatabaseImpl* object is instantiated which translates the input parameters and performs a call to open an *Illustra* database. Spatial repository adaptors are implemented as *implementation classes* for the geoPOM base types. Since all repository-specific code is encapsulated in the implementation classes, there is only one geoPOM compiler necessary to generate the geoPOM code. The cached and non-cached version of geoPOM collection classes are imple-

mented in a similar manner.

In the first version, geoPOM employs Illustra DBMS and its 2-D and 3-D Spatial Data Blade [16], and ESRI's Spatial Database Engine SDE [25]. In the area of low-level spatial data repositories, the HDF scientific file format [15] is employed.

8 Lessons Learned and Observations

The following summarizes the lessons learned while developing and applying geoPOM.

- *Heterogeneous spatial systems are domain-specific.* Designing and implementing geoPOM, it was noticed that if a heterogeneous spatial system is quite “generic”, it is likely to be of little use. A “generic” system supports a wide-range of common spatial functionality and data types via the CSDM which, on one hand, makes the heterogeneous system very complex and, on the other hand always contains “baggage” that is not necessary for a particular domain. Only the basic architecture of a heterogeneous spatial object system is domain-independent; a concrete heterogeneous spatial object system, however, should be domain-specific, i.e. provide a domain-specific choice of spatial CSDM functionality, such as base geometry types, coordinate and spatial reference systems, spatial operators, etc., as well as, a selection of spatial repositories. This provides for a complete and light-weight implementation, and enabled the addition of domain-specific optimizations.
- *Reimplement spatial functionality on the integration level.* Our initial assumption was to “push down” functionality, such as evaluation of spatial operators from the geoPOM level to the repository level and avoid any reimplementations of spatial functionality by geoPOM; therefore, spatial operators on a geometry type are implemented as wrappers which delegate the evaluation to a repository. However, first performance tests demonstrated clearly that this approach results in an unacceptable performance penalty. To achieve the necessary performance levels, spatial functionality should be reimplemented on the geoPOM level, and executed via the intermediate C++ objects after they have been extracted from the underlying repositories.
- *Design patterns proved helpful for architectural abstractions.* Implementing geoPOM, we used design patterns to design the internal abstractions of the geoPOM architecture. The design patterns helped to keep the main parts of geoPOM repository-independent and avoided code replication. Also, writing code to adapt to additional spatial DBMS was sig-

nificantly simplified; e.g. adding SDE to the system took around 3 days.

9 Conclusions

In this paper, an overview of geoPOM, a heterogeneous geoscientific persistent object system developed at the UCLA Data Mining Laboratory was presented. GeoPOM provides applications the abstraction from heterogeneous spatial data repositories. GeoPOM leverages the ODMG data model as well as the OGIS data model for the common spatial data model provided to the user. GeoPOM's data model provides support for modeling of complex temporal-spatial information. GeoPOM generates code that maps user-defined geoPOM types to spatial “types” of the local spatial data repositories. The generated code is linked into the geoPOM application, permitting the application to directly access the spatial data repositories in order to achieve minimal performance loss. Different problems that arise when integrating and abstracting from different specialized repositories were described and implementation strategies discussed. The strategy of classifying a CSDM's functionality and repository capabilities in order to use specialized repositories according to their capability was presented.

Acknowledgements

We sincerely acknowledge support from NASA EOSDIS grant NAGW-4242, and we thank Wei Wang for her helpful comments on the paper. We also thank Jiong Yang for his implementation work on the geoPOM compiler.

References

- [1] A. Abiteboul and A. Bonner. Objects and Views. In *Proc. of SIGMOD, Denver, Colorado*. ACM Press, 1991.
- [2] T. Barsalou, N. Siambela, A. Keller, and G. Wiederhold. Updating Relational Databases through Object-Based Views. In *Proc. of the ACM SIGMOD Conference, Denver, Colorado*. ACM Press, 1991.
- [3] E. Bertino, M. Negri, G. Pelaggati, and L. Sbatella. Integration of heterogeneous database applications through an object-oriented interface. *Information System*, 14(5):407–420, 1989.
- [4] M. Bright, A. Hurson, and S. Pakzad. A Taxonomy and Current Research Issues in Multidatabase Systems. *Computer, IEEE Computer Society*, 1992.
- [5] K. Buehler and L. McKee. Introduction to Interoperable Geoprocessing. Technical Report OGIS TC Document 96-001, OGIS Project Technical Committee of the Open GIS Consortium, Inc, 1996.
- [6] O. Burkhres and A. Elmagarmid, editors. *Object-Oriented Multidatabase Systems*. Prentice Hall, Englewoods Cliffs, New Jersey, 1996.

- [7] M. Carey, W. Codey, L. Haas, P. Schwarz, M. Arya, W. Cody, R. Fagin, and et.al. Towards Heterogenous Multimedia Information Systems: The Garlic Approach. Technical Report RJ-9911 (87291), IBM Research Division, 1994.
- [8] R. Catell, T. Atwood, J. Duhl, G. Ferran, M. Loomis, and D. Wade. *Object Database Standard: ODMG-93, Release 1.2*. Morgan Kaufmann Publishers, San Mateo, California, 1994.
- [9] A. Chen, A. Kosky, V. Markovitz, and E. Szeto. Developing and Accessing Scientific Databases with the OPM Data Management Tools. In *Proc. of Int. Conference on Data Engineering (ICDE-97)*. IEEE Computer Society Press, 1997.
- [10] D. DeWitt, N. Kabra, J. Luo, J. Patel, and J. Yu. Client-Server Paradise. In *Proc. of 20th Intl. Conference on Very Large Databases (VLDB), Santiago, Chile*, 1994.
- [11] W. Du and A. Elmagarmid. Quasi Serializability: A correctness criterion for global concurrency correctness in interbase. In *Proc. of the Int. Conference of Very Large Databases (VLDB)*. ACM, 1989.
- [12] S. D. Formats. *Web site for Scientific Data Format Information FAQ*. <http://fits.cv.nrao.edu/traffic/scidataformats/faq.html>, 1996.
- [13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.
- [14] M. Haertig. An Object-Oriented integration Framework for Building Heterogeneous Database Systems. In *Proc. of Conference on Semantics of Interoperable Database Systems, Lorne, Australia*. IFIP, DS-5, 1992.
- [15] HDF. *Hierarchical Data Format*. <ftp://ftp.ncsa.uiuc.edu/HDF>, 1996.
- [16] Illustra. *User's Guide, Version 3.2*. Informix, Inc., 1996.
- [17] M. Kaul, K. Drosten, and E. Neuhold. Viewsystem: Integrating Heterogeneous Information Bases by Object-Oriented Views. In *Proc. of 6th Int. Conference on Data Engineering (ICDE), Los Angeles*. IEEE Computer Society Press, 1990.
- [18] M. A. Keller. Updating Relational Databases Through Views. Technical report, Dissertation, Computer Science Department, Stanford University, 1995.
- [19] W. Kent, R. Ahmed, J. Albert, M. Ketachi, and S. M-C. Object identification in multidatabase systems. In *Interoperable Database Systems (DS-5), (A-25)*. IFIP, 1993.
- [20] E. Mesrobian, R. Muntz, E. Shek, S. Nittel, M. LaRouche, and M. Kriquer. OASIS: An EOSDIS Scienc Computing Facility. In *International Symposium on Optical Science, Engineering, and Instrumentation (SPIE), Conference on Earth Observing System, Denver, Colorado*. IEEE Computer Society, 1996.
- [21] E. Mesrobian, R. Muntz, E. Shek, S. Nittel, M. LaRouche, and M. Kriquer. OASIS: An Open Architecture Scientific Information System. In *Workshop on Research Issues in Data Engineering (RIDE 96), New Orleans*. SPIE, Volume 2820, 1996.
- [22] E. Pitoura, O. Bukhres, and A. Elmagarmid. Object Orientation in Multidatabase Systems. *Communications of the ACM*, 27(2):141–195, 1995.
- [23] B. Rieche and K. Dittrich. A Federated DBMS-Based Integrated Environment for Molecular Biology. In *Proc. of the 7th Int'l Working Conference on Scientific and Statistical Database Management, Charlottesville, Virginia*. IEEE Computer Society, 1994.
- [24] H.-J. Schek, H.-B. Paul, M. Scholl, and G. Weikum. The DASDBS Project: Objectives, experiences, and future prospects. In *IEEE Transactions on Knowledge and Data Engineering*, volume 2. IEEE, 1990.
- [25] SDE. *User's Guide, Version 2.0*. ESRI, 1995.
- [26] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *Communications of the ACM*, 22(3):183–236, 1990.
- [27] R. M. Soley, editor. *Object Management Architecture Guide (2nd Edition)*. Object Management Group, 1992.
- [28] M. Stonebraker. Sequoia 2000 - a Reflection on the First Three Years. In *Scientific and Statistical Database Management Systems, Charlottesville, Virginia*. IEEE Computer Society Press, 1994.
- [29] M. Stonebraker and L. Rowe. The Design of Postgres. In *Proc. of SIGMOD 86*. ACM Press, 1986.
- [30] A. Zhang and E. Pitoura. A view-based approach to relaxing global serializability in multidatabase systems. Technical report, TR CSD-93-082, Purdue University, West Lafayette, 1993.