geoPOM: A Heterogeneous Geoscientific 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

In this paper, we provide an overview of the geoPOM heterogeneous object system developed at the UCLA Data Mining Laboratory. The goal of this system is to provide object-oriented spatial data modeling capabilities and abstraction from heterogeneous spatial data repositories. geoPOM enables users to define spatial object types according to the geoPOM data model. geoPOM's data model is based on two standardization efforts: (1) the ODMG-93 data model, and (2) the OGIS object model defining standardized temporal-spatial object types. Internally, geoPOM maps user-defined object types to heterogeneous spatial data repositories, and utilizes their storage, search and spatial query capabilities. However, geoPOM is not a heterogeneous database management system (DBMS) in the classical sense, i.e. providing a run-time system and performing query processing outside the data repositories. Instead, geoPOM generates code for repository access, data creation, manipulations and queries to access spatial repositories at run-time directly.

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 exist, and a need to provide easy and uniform access to integrated data has become 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. Numerous GIS exist today that have syntactically and semantically heterogeneous data models describing basic spatial data types. Also, the definition and availability of spatial operators such as intersection, containment, union, etc. varies significantly from system to system. Some GIS provide only basic functionality such as storage types for file-based systems, others offer complete DBMS functionality with a data definition and query language, query optimization, spatial access paths, etc.. Thus, the semantic and functional gap between GISs is much larger than found in traditional heterogeneous systems today. Providing uniform access to heterogeneous spatial data repositories introduces a set of new challenges and system requirements such as efficient exploitation of capabilities provided by existing spatial data repositories, flexible mapping strategies, handling of large objects, and eventually tertiary storage support. However, research in this domain is still in its infancy.

At the UCLA Data Mining Laboratory, we are developing *geoPOM* (**geo**scientific **P**ersistent **O**bject **M**anager), a geoscientific object system capable of integrating heterogeneous spatial data repositories. The integrated system facilitates access to data in the repositories while maintaining the autonomy of the repositories, and without making copies of the data. geoPOM provides an object-oriented data model extended with a set of predefined geoscientific object types such as *feature*, *point*, *linestring*, *polygon*, etc. which are based on the OGIS object model [5]. Users are allowed to define their own temporal-spatial object types which are automatically mapped to spatial data repositories such as spatial DBMSs, file systems or mass storage systems, thus, making it possible to define spatial objects in a data repository independent manner. In addition, geoPOM provides an object-oriented query language with which users can pose queries to geoPOM objects. The primary focus of geoPOM lies in providing abstraction from heterogeneous spatial data repositories such that newly-defined geoPOM types are transparently mapped to available repositories, and existing legacy data is wrapped and made accessible through geoPOM types. In this paper, we do not focus on data integration, i.e. queries over objects stored in different repositories, or objects stored in multiple repositories.

In providing abstraction from heterogeneous data repositories, performance is an important issue. 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 thus, 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 a 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, performance loss is minimized.

geoPOM is developed as part of the OASIS project at the UCLA Data Mining Laboratory [20]. OASIS (Open Architecture Scientific Information System) provides a extensible, seamless environment for scientific data analysis, knowledge discovery, visualization, and collaboration based on CORBA [28]. OASIS provides a set of distributed geoscientific CORBA objects which can be accessed in a location- and platform independent manner. In OASIS, we use geoPOM to implement the persistent state of CORBA objects in a data repository-independent way, and thus, provide for high reusability of CORBA object implementation.

The remainder of this paper is organized as follows. Section 2 presents related work in this area. In section 3, we give an overview of geoPOM and its concepts. Section 4 describes the geoPOM data model. Section 5 shows an example. In section 6, we describe selected geoPOM components in detail, section 7 contains implementation issues, and we conclude with a discussion of remaining problems and future directions.

2 Related Work

Developing geoPOM, we focus on 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, i.e. the area of heterogeneous DBMS, and geographic and geoscientific information systems (GIS).

2.1 Heterogeneous DBMS

In the past decade, various approaches have been taken to address the issue of abstracting from heterogeneous data repositories. Starting in the mid 80s, most of the work focussed on integrating relational DBMS through multidatabase languages, multidatabase systems with varying degrees of coupling between the data repositories and the integrating system as well as the autonomy of the data repositories ([27, 4]). In recent years, approaches have been developed which use object-oriented data models and implementation techniques since the object-oriented programming paradigm provides powerful data modeling and new implementation strategies for subproblems in heterogeneous DBMS, e.g., schema mapping, transaction management, etc. A good overview of the work in this area can be found in [23, 6].

The first object-oriented, heterogeneous systems provided their own data model. Recently developed systems are based on the object-oriented data model proposed as a standard by ODMG ([9]). However, several extensions have been added by the 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 [8, 1, 18], for object identifiers that do not guarantee immutability during an object's lifetime (so-called weak identifiers) [8, 15, 19], and for weak references which are based on weak identifiers [8]. Also, a variety of new issues in the area of heterogeneous DBMS have been addressed such as mechanisms for object migration [24], object-oriented transaction management [12, 31], 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, scientific file format systems or video servers, does not exist. A notable first approach in this area, although not addressing GIS, is the Garlic project ([8, 7]). 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.

2.2 Geoscientific Information Systems

Geographic information systems appeared in the early 1970's. At that time, computer science and especially DBMS interest in GIS had been slow in part due 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 ([11, 30, 25]). Additionally, high-performance computers, highspeed, ubiquitous communication, and cheap disk and tertiary storage devices became enabling technology for computer-aided geoscientific information systems.

Today, a variety of commercially-available GIS packages exist, most of which use file-based storage techniques for data storage. Many low-level spatial data repositories are available which are mainly used for the platform independent storage and retrieval of large scientific data sets and are used in different scientific domains such as meteorology, astronomy, for space mission data, etc. [13]. 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 to support 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 mostly assumed to be a centralized DBMS ([17, 26, 29]). Additionally, flexibility is restricted by the underlying assumption that all data needed is stored inside a homogeneous DBMS even though it may be parallized [11]. Furthermore, lack of a standard 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 1996/1997 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 geoPOM Overview

Loosely speaking, the goal of geoPOM is to provide applications with the benefits of a single geoscientific object system, but without actually storing the data in a single spatial repository. Instead, any (spatial) DBMS or scientific file format system available can be used to store the data. geoPOM provides a logical abstraction that the same spatial repository is always used.

3.1 The Big Picture

The architecture of geoPOM is depicted in 1. At the top of the figure are applications like CORBA objects or C++ applications which use geoPOM to define persistent objects using the geoPOM data model. Queries on geoPOM objects are embedded in the C++ application code or the CORBA server implementation.

The *geoPOM Object Definition Language* (geoODL) is used to define temporal-spatial object types. The geoPOM type is mapped to a *spatial data repository*. We assume that the data repository provides temporal-spatial data model or a scientific file archive format API like the HDF [14] or netCDF library which are commonly used to store and access large geoscientific datasets. Thus, temporal-spatial geoPOM types are translated to "types" defined in terms of the "data model" of local data repositories. The mapping is performed by a *repository wrapper*. A repository wrapper translates between the geoPOM data model and the native data model of the repository, i.e. translates requests for geoPOM object creation, manipulation, and queries to the native spatial

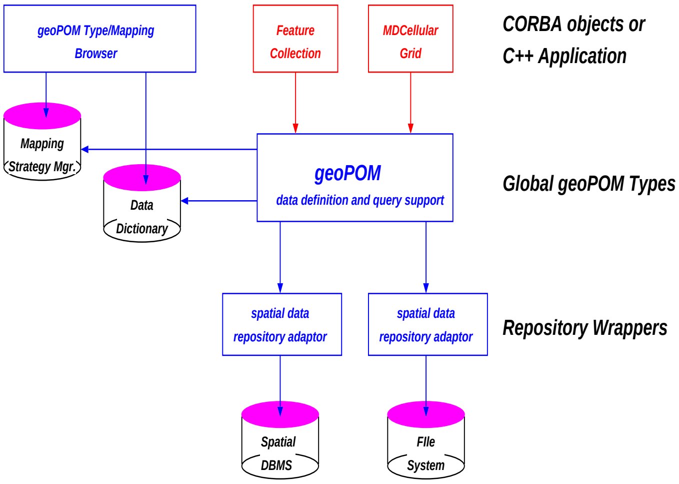

Figure 1: Architecture of geoPOM

repository. Information about the geoPOM types as well as mapping-specific information is stored in the geoPOM *data dictionary*. When a new geoPOM object type is defined, geoPOM implements a default mapping strategy for the object type based on available spatial data repositories, space limitations, repository functionality, and spatial characteristics of the type defined. However, to keep this mapping flexible, geoPOM supports a *rule-based mapping strategy manager* which contains rules about mapping-related strategies, and permits the definition and replacement of rules. To find out about existing types and their definition and/or about mapping rules, geoPOM supports a *Type/Mapping Browsing* facility as shown in Figure 1.

Query processing and data manipulation services are also provided by geoPOM. The state of geoPOM objects is manipulated via object methods, or via direct access of attributes. Queries are expressed in geoPOM's Object Query Language (geoOQL) which is an object-oriented extension of the SQL query language, and allows (spatial) method invocation and path expressions within select-from-where clauses.

3.2 Spatial Data Repositories

The goal of geoPOM is to provide clients with a storage abstraction layer above heterogeneous spatial data repositories. Our primary focus is on mapping types defined in the geoPOM data model transparently to heterogeneous spatial data repositories. However, for the storage of large scientific datasets, we support the wrapping of legacy data, i.e. existing files, and provide mapping of globally defined geoPOM types to legacy data. A data repository consists of two parts, i.e. the management software (e.g. the Illustra DBMS, or the HDF library) and the data that it manages. Based on the notation used in [8], we refer to the management software as the *repository type* and to a certain installation of the repository type as a *repository manager*. Similar, we refer to a particular data collection managed by a repository manager as a *repository instance*. To provide access to the data managed by a repository manager, a *repository wrapper* has to be provided for the repository type. A repository wrapper is general and provides for a common interface for a repository type.

The spatial data repositories participating in geoPOM offer diverse functionality which can be low-level like the HDF API or sophisticated like a spatial DBMS. Low-level repositories only provide basic storage types and access operations, while high-level spatial repositories such as spatial DBMS also support a query language, storage optimization, spatial access paths, etc.. Thus, e.g. queries to geoPOM can be delegated to a spatial DBMS and executed in the repository, while they can not be sent to scientific file format systems, and have to be processed outside the repository, i.e. in the repository wrapper. However, it is not the goal to implement all missing functionality of spatial DBMS on top of low-level spatial repositories. On the other hand, low-level spatial repositories provide specialized functionality that may not be offered by spatial DBMS, e.g. fast processing of large multidimensional arrays. Therefore, we use spatial repositories according to their *capability* in geoPOM. The functionality offered by geoPOM is a superset of functionality supported by various specialized spatial repositories. However, to make the handling of these repositories transparent to geoPOM, we introduce a classification of repositories types so that each repository class maps to a certain class of the geoPOM functionality. The semantic and functional gap within a class, e.g. the class of spatial DBMS, might still be significant. This gap must be bridged by the repository wrappers.

The contents of a geoPOM database are presented to end users as a set of geoPOM types expressed in terms of the geoPOM data model. Internally, these types are mapped to corresponding types or - in DBMS terminology - schemas of the repositories that are used for actually storing the data. The geoPOM architecture follows the architecture proposed in [27], but instead of five levels, we only support three levels which we describe bottom-up (see Figure 2); this is also describes the way lagacy data is wrapped and integrated. The first level consists of the *local schemas* of the spatial data repositories in the geoPOM environment. These schemas are described by means of the local data repositories such as the Illustra data model and its spatial extension. The local schemas are transformed into corresponding *global schemas* expressed in terms of the geoPOM data model, i.e. the geoPOM object types. The global schemas represent the second level. Most of the time, geoPOM applications will use the global schemas of the second level. The third level consists of *enhanced*, *simplified* or *reshaped* schemas of the second level. The enhanced, or reshaped schemas are defined via views over the global schemas of the second level. Since we allow geoPOM users to define global schemas independently from each other, global schemas, i.e. geoPOM types, might be defined redundantly and non-consistent. The view mechanisms used for defining the third level also provides for the integration of redundant, non-consistent geoPOM types.

geoPOM is not a full-blown heterogeneous spatial DBMS, but rather provides spatial repository abstraction for geoscientific applications. Therefore, we made the following assumptions: first, a geoPOM type is always mapped to a single spatial data repository only, thus, the state of a geoPOM object is not stored across several repositories. Second, queries can be posed to one geoPOM object type's extent only. This provides for selection of subsets based on predicates applied to an object's attributes and queries. The query is sent to the responsible repository wrappers which maps the query to a query to the local spatial data repository. The query is executed within the spatial repository using only the available spatial operators, or within the wrapper in case the spatial data repository does not provide sufficient functionality. However, no query processing is performed outside the repositories or wrappers.

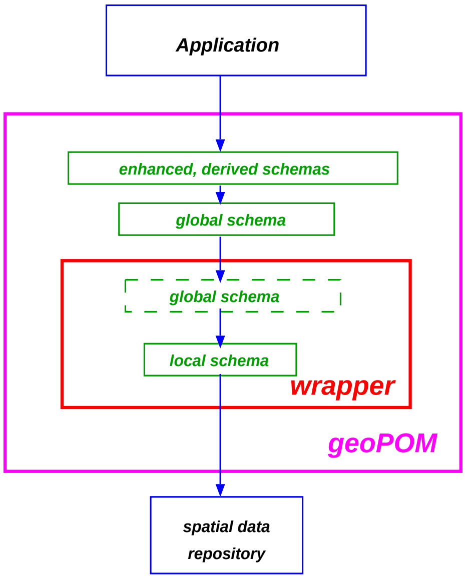

Figure 2: Schema Architecture of geoPOM

4 geoPOM Data Model

geoPOM's data model is based on object-oriented data modeling techniques which have proven to be useful in geoscientific environments [10]. Based on an object-oriented data model, geoPOM provides a set of predefined temporal-spatial types which are used like primitive types for the definition of user-defined temporal-spatial object types.

4.1 *Basic ODMG Model*

Since our primary research interest with geoPOM is to investigate issues involved in providing storage abstractions for geoscientific objects, and access to heterogeneous spatial data repositories, we use the object-oriented data model proposed as a standard data model by ODMG-93 [9] as the basis for the geoPOM data model.

The basic modeling primitive in the ODMG-93 object model is the object. An object can consist of a set of *attributes*, *relationships* with other objects, and a set of *methods*. The values of the attributes and relationships define the object's state. Values of attributes can be atomic, structured or references to other objects. Relationships are 1:1, 1:n or n:m with other objects. An object can also contain collection-valued attributes such as bags, lists, sets, and arrays. The set of methods supported by an object type defines its *behavior*. Object attributes can be accessed directly or via user-defined object methods to be read or updated. Each object is uniquely identified by an object identifier. Objects are strongly typed, and their characteristics are described by their *object interface*. The model also supports inheritance between types ([9]). The ODMG-93 object model also defines a query language, i.e. the Object Query Language (OQL), a declarative query language usable to query objects based on their interfaces. OQL is a superset of SQL92, and based on select-from-where clauses. However, OQL also deals with complex objects, and allows method invocation and path expressions within select-from-where clauses.

The ODMG object model consist of the Object Definition Language (ODL) and the Object Manipulation Language (OML). The object model is language-independent, but only a mapping to C++ and Smalltalk is defined by ODMG. We are using the C++ binding of the ODMG object model. Here, the ODL/OML part of the ODMG object model is expressed in a class library providing classes and functions to implement the concepts found in the ODMG object model.

4.2 *Extensions to the ODMG Model*

The ODMG model is designed for a homogeneous, non-distributed DBMS environment. Using geoPOM, we are dealing with additional problems, i.e. access to heterogeneous spatial data repositories, spatial repositories that are based on relational DBMS technology or even file systems, and access and integration of legacy data sources in a distributed environment. Therefore, we added the following extensions to the basic ODMG data model:

- **Weak object identifiers:** At the global geoPOM level, objects are identified by an object identifier. It would be desirable to provide strong object identity for geoPOM objects, i.e. an object identifier is immutable during the life time of an object. On the other hand, we also want to provide fast access to corresponding data objects in the spatial data repository; therefore, we use the native "identifier" as a part of the geoPOM object identifier. However, most of the participating spatial data repositories are based on (extended) relational database technology "identifiers" which are the values of key attributes that might change during the life time of tuple representing an entity. Therefore, we are using *weak identifiers* which were introduced in several other heterogeneous system approaches based on object-oriented technology ([23, 8, 15]). Weak identity means that objects are identified uniquely, but are not necessarily immutable, i.e. the identifier might change during the lifetime of an geoPOM object.

- **Spatial-indexed collections**: geoPOM provides a set of predefined temporal-spatial object types which are used to define higher-level, user-defined spatial object types and spatial object collections. In order to provide fast access to subsets of these temporal-spatial object collections or for temporal-spatial queries over several sets of collections, we provide in addition to ODMG's basic access path types like hash functions, or B-trees *spatial indexes* like R-trees, Quadtrees, KD-trees etc. These access paths are used within the definition of collections.

- **Object-oriented view mechanisms**: In order to extend, simplify, or reshape existing types, and/or for the unification of redundant global types with slightly different semantics, we are adding a concept of object-based views to the ODMG model which does not provide a view mechanism today. Recently, this shortcoming was noticed in several areas, and proposals for view models have been made. We will base the geoPOM view model on the model used in the Garlic project [8].

4.3 *geoPOM's Predefined Temporal-Spatial Types*

The geoPOM data model provides a set of predefined spatial spatial object types. As in the OODBS domain, several temporal-spatial data models have been proposed during the past few years. However, a standard GIS data model does not exist, although, an effort is underway to alleviate this situation. The Open GeoData Interoperability Specification (OGIS) [5] is working on the definition of standard GIS object model. In order to provide broad usability of geoPOM, we have based the temporal-spatial part of the geoPOM data model on the OGIS object model.

The goal of OGIS is to provide well-known types and common aggregates as basic building blocks. OGIS defines two basic geodata types, i.e. *features* and *coverages*. Features represent real world or abstract entities, and are associated with a spatial (or spatial-temporal) domain, for example, roads. A coverage is an association of points, polygons or solids within a spatial-temporal domain with a value (of a certain type). 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 geometry to a real world *spatial reference system* (such as a coordinate system consisting of latitude and longitude coordinates). Considering a feature and a coverage as an abstract object, a spatial object is a concrete object and e.g. represents a spatial feature with its specified geometry. A spatial object can also be an aggregation of features, i.e. a so-called *feature collection. GeoDatasets* are structured aggregations of spatial objects and represent higher-order collections of entities.

OGIS also defines geospatial operators for geospatial objects. In traditional object models, the definition of methods is a part of the object type definition. In geoscientific processing, however, many operations such as intersection etc. depend on the classes of more than one object. Therefore, OGIS supports a model in which the operations are not necessarily part of the objects, but are objects themselves and can be selected on basis of the classes of the participating objects.

Since the OGIS standardization effort is an ongoing effort, we have taken a snapshot of the OGIS model, and extracted a subset that is significant for the first cut of the geoPOM data model. This subset defines object types for *time*, *spatial referencing*, *geometry* and *geometry types* like points, polygons, etc. The subset also contains the object types *feature*, *coverage* and *feature collection.* Figure 3 depicts an overview over the geoPOM data model, and presents the basic geoPOM classes that are used to define application-specific object types.

5 Example

In this section, we use the geoPOM data model to describe the schema for two basic types of geoPOM objects, i.e. *fine-grained temporal-spatial objects* like points, polygons, etc., and *large-grained temporal-spatial objects* like multidimensional arrays. To describe both, we use the example of cyclone track extraction from a geoscientific data set. In this example, a geoscientific data set consists of a set of remotely-sensed satellite images which record the sea level pressure over a certain area of the earth. This data set is used as a basis for extracting cyclones and cyclone tracks by a data mining system like [21]. A cyclone is a local minimum in a sea level pressure field represented as a point in a three-dimensional space at a certain time and a pressure value. A cyclone track is a sequence of such points in time.

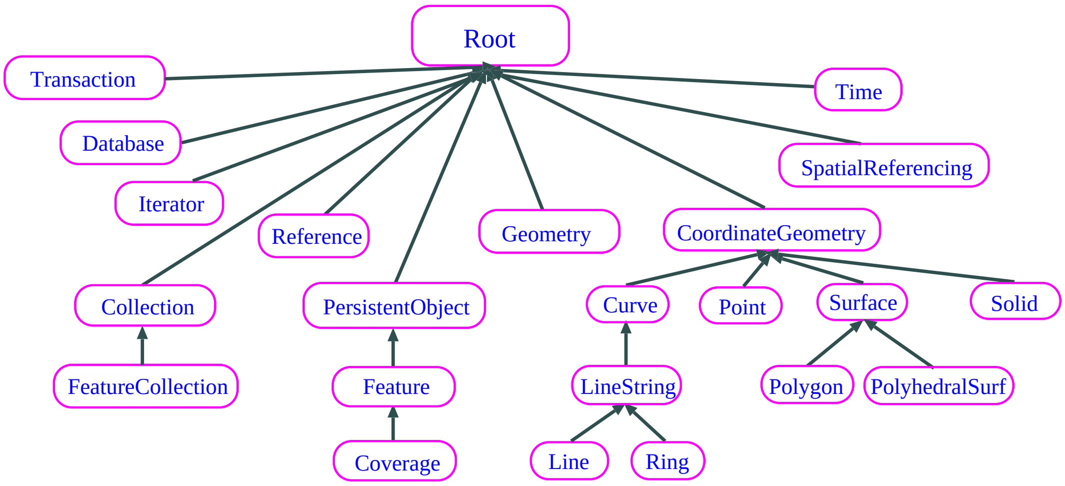

Figure 3: geoPOM interface hierarchy

Figure 4 depicts the geoPOM types involved in this schema. First, the definition of the *Cyclone* type is depicted. A Cyclone consists of a spatial-temporal location in a 4-dimensional space (space and time) and a sea-level pressure values for the location. The type *CycloneTrack* consists of a list of these values as described above, the start and end time, and the spatial box covered by the *CycloneTrack*. The track length and the minimal and maximal pressure value are additional attributes.

Each CycloneTrack belongs to a *CycloneTrackCollection*. A *CycloneTrackCollection* contains all *CycloneTracks* that are extracted from a specific data set with a certain data mining algorithm. The *CycloneTrackCollection* also records the time and spatial envelope and the number of *CycloneTrack*s contained in the collection. Additionally, a spatial index is defined over the collection elements so that the temporal-spatial query method defined on the *CycloneTrackCollection* can be executed efficiently. The *CycloneTrackCollection* contains two relationships: first, a bi-directional relationship with the *CycloneTrack* objects that are elements of the *CycloneTrackCollection*, and second, a bi-directional relationship with the scientific data set from which the *CycloneTrack*s were extracted.

The schema of the type *GeoScientificDataset* is an example of the second group of typical geoPOM object types, i.e. large-grained spatial objects. The *GeoScientificDataset* is a multidimensional array and contains a rather simple interface, i.e. the temporal and spatial envelope and a method to execute temporal-spatial "slice calls" on the object. To iterate through the retrieved values, a *Iterator* is returned.

The next type definition contains an example for a *derived type*. The derived type is defined as a view over the type *CycloneTrack*. For example, we define the derived type *Africa_CyloneTrack* which extent contains all cyclone tracks of the base class *CycloneTrack* that fulfill the predicate that their location is within the area of Africa. A new extent can be defined to execute the view one time and store the *CycloneTrack*s in a new extent. On the other side, the data does not need be stored in which case the cyclone track set is materialized at access time. The view contains only one attribute, *cyclones*, which is defined as the equivalent attribute of the base class.

```
Fine-grained Spatial Objects:
struct Cyclone
{
    Point location,
    Date time,
    Float pressure
}

interface CycloneTrack(extent CycloneTrack): persistent
{
    attribute List<Cylcone> cyclones;
    attribute TimeEnvelope time_env;
    attribute Envelope spatial_env;
    attribute Long track_len;
    attribute Float min_pressure;
    attribute Float max_pressure;
    relationship CycloneTrackCollection belongs_to inverse CycloneTrackCollection::elements;
}

interface CycloneTrackCollection (extent CycloneTrackCollection): persistent
{
     attribute String name;
     attribute Long cardinality;
     attribute TimeEnvelope time_env;
     attribute Envelope  spatial_env;
     relationship GeoScientificDataset source;
     relationship List<CycloneTrack> elements inverse CycloneTrack::belongs_to;
     void create_index(on:elements[t:r_tree]);
     void get_cyclones_in_time_space_box(in TimeEnvelope time, in Envelope env,
                                         out Set<CycloneTrack> result);
}

Large-grained Spatial Objects:
interface Iterator:transient
{
     MDArray next(out GridInfo grid,out TimeStamp ts);
     MDArray next_n(in Long n,out GridInfo grid,out TimeStamp ts);
     boolean more();
}

interface GeoScientificDataset(extent GeoScientificDataSet):persistent
{
    attribute TimeEnvelope time_env;
    attribute Envelope spatial_env;
    void get_data_in_time_space_box(in TimeEnvelope time, in Envelope env,out Iterator iterator);
}

Derived Object Type:
mapping Africa_CycloneTrack
{
    origin   CycloneTrack:CycloneTrack  cyclone_track;
    def_ext  Africa_CycloneTrackCollection as
             select
             from cyclone_track
             where Contained_In(env, (38.50, -37.80, -20.00, 61.30));
    def_attr cyclones as cyclone_track.cyclones;
}
```

Figure 4: Example for geoPOM schema definition

6 Integrating Heterogeneous Spatial Repositories with geoPOM

geoPOM consists of various software components to provide applications with a spatial access layer that abstracts away the underlying spatial data repository utilized to store the actual data. This capability also provides a mechanism to support legacy spatial data. The specifics of a spatial repository type are encapsulated in a repository wrapper while the other modules of geoPOM are generic.

geoPOM provides the *geoPOM Object Definition Language* (geoODL), and the *geoPOM Object Query Language* (geoOQL) which enable users to define, manipulate and query geoPOM object types. geoPOM type definitions are compiled and inserted into the *geoPOM data dictionary*. Beside geoPOM type definitions, the data dictionary manages information about available data repository types, concrete data repositories, and statistics like the capacity of a repository. The actual mapping of geoPOM object types to a spatial data repository is performed by a *repository wrapper* described below.

6.1 *Wrapping Spatial Data Repositories*

geoPOM contains a set of *repository wrappers* each of which encapsulates the specifics of a repository type and generates repository type-specific schema creation, data access, and manipulation code. geoPOM is designed to be extensible and supports the addition of new repository wrappers without affecting existing geoPOM applications. A *repository wrapper* is responsible for the mapping of a geoPOM type to a given spatial data repository's corresponding datatype and for mapping a user query expressed using geoOQL into the query language supported by the repository. Each wrapper knows how to map two kinds of information: first, how to map the basic ODMG object model to the data model of the repository, and second, how to map the geoPOM's spatial object types to the temporal-spatial types or API of the underlying data repository.

Today, DBMS-based spatial data repositories are typically implemented using relational or extended relational DBMS technology. Therefore, mapping geoPOM object types to the DBMS-based repository interface is performed in two steps: first, mapping the ODMG base types like *persistent object* to the base type *relation*; and second, mapping the geoPOM spatial types to the spatial types of the repository. In recent years, several strategies for mapping objects to relations have been designed and implemented. We based the geoPOM mapping strategy on the approach proposed in [2].

The generic mapping of geoPOM's spatial object types, operators and access paths is much harder to solve in part due to the significant heterogeneity of spatial object models. geoPOM provides a set of predefined spatial object types which are used to build complex, user-defined spatial object types. For each of the predefined spatial object types, we must define a a mapping of the object to a spatial object/functionality provided by participating spatial data repositories. The mapping is then reused to map user-defined spatial object types. Early on in the geoPOM's design stage, we determined that geoPOM's data model has to provide a high common denominator of the functionality supported by the individual spatial data repositories. Therefore, we used the OGIS spatial object model [5] as a basis for the spatial part of the geoPOM data model, since it supports a rich set of functionality for the representation and manipulation of fine-grained vector data, and for coarse-grained raster data. It is typical, that during the integration and abstraction from various specialized repositories, the chosen repositories are used according to their *capability*. However, to make their use transparent to geoPOM, we classify *spatial object types* of geoPOM as well as the *spatial repository functionality*, i.e. define a set of *functionality classes*, and *repository classes*. For each *functionality class*, we define the *repository class* that is best equipped for mapping the spatial types of the *functionality class* to the functionality of the spatial repositories available in the *repository class*. A spatial data repository belongs to one or more *repository classes*. Within a repository class, all repositories can be used interchangeably, i.e. the use of a repository is *transparent* to geoPOM. Therefore, each *repository class* has to contain at least one repository so that it can be guaranteed that the complete functionality of the geoPOM common data model is available to the user.

However, spatial repositories that directly utilize formatted file/disk storage, typically provide a dramatically more limited set of functionality. In these cases, missing functionality such as queries over a set of geoscientific data sets, has to be performed outside of the repositories by their repository wrappers. In the long term, other solutions might be possible to describe the capabilities of a repository; this would enable geoPOM to decide whether the query can be sent directly to the repository or to determine how the query can be decomposed into a set of simpler query plans that can be executed by the repository, with the results combined by geoPOM. This general strategy can be utilized by geoPOM to support repositories that have limited functionality without requiring the modification of spatial repository wrappers.

6.2 *Flexible Mapping Strategies*

Each spatial repository wrapper implements a default strategy for mapping the geoPOM base and user-defined types to its target spatial data repository. The default approach is to use a set of mapping strategies to map geoPOM's base types such as the ODMG and spatial types and for user-defined object types. However, for performance reasons it might be useful to override a default mapping strategy for: (1) user-defined types, (2) geoPOM base types used to define a user-defined type; or (3) a geoPOM base type. Therefore, geoPOM contains a *rule-based mapping strategy component* that allows a user[1] to define rules for mapping of geoPOM types a specific repository data types. For example, a default strategy for mapping fine-grained spatial geoPOM object types includes the definition of how a geoPOM object type which consists of base attributes and a set of spatial objects is split into two tables. At access time, the two tables are joined via a foreign key condition. For performance reasons, the user might redefine the mapping so that the object type is mapped to only a single table.

Providing support for mapping of user-definable types through the mapping strategy component also provides for *wrapping of legacy data*. In this case, we first identify the schema to be exported by the data repository, and then define the corresponding geoPOM object type; the mapping between both types is declared via the mapping strategy component. The geoODL compiler automatically generates code to access, manipulate and query the already existing native data type(s) and its data via the geoPOM object type.

We assume that the mapping strategy manager is mainly used for coarse-grained spatial objects like multi-dimensional arrays, satellite rasters images, etc.; for these geoPOM object types, the geoPOM mapping strategy manager allows the definition and redefinition of storage layout rules for these large data objects. In particular, the geoPOM mapping strategy manager supports rules to define the declustering of the object's data into regular shaped tiles. The user is allowed to define parameters like the tile size, or overwrite the rule to apply a certain compression algorithm before the tiles are stored.

7 Implementation Issues

The initial geoPOM prototype provides a C++ interface to create, manipulate and query geoPOM objects. geoPOM provides a C++ class library that contains the set of ODMG base classes, i.e. *persistent object*, *database*, *transaction*, *reference*, *collection*, and *iterator* and the described spatial

---

[1] In this case, the user is considered as the database administrator.

types. Each geoPOM type inherits the behavior of the ODMG *persistent object* (e.g., the ability to instantiate, activate, deactivate and delete an object). A geoPOM type is defined in ODMG syntax and compiled by the geoPOM compiler. It creates the corresponding C++ class definition and inserts the geoPOM type definition in the data dictionary. A wrapper, which is actually a compiler, retrieves the geoPOM type definition from the data dictionary and generates code, i.e. a set of C++ methods containing the interaction with the native data repositories. The C++ class definition, contains user-defined methods as well as the geoPOM base classes, is then compiled with the application. At run-time, the repositories are accessed directly.

The first version of geoPOM employs a set of high-level and low-level spatial repositories. In the area of high-level spatial data repositories, we use the Illustra DBMS and its 2-dimensional and 3-dimensional Spatial Data Blade [17], and ESRI's Spatial Database Engine SDE [26]. In the area of low-level spatial data repositories, we employ the HDF file format [16], and the netCDF file format [22].

8 Conclusions and Future Directions

In this paper, we presented an overview of geoPOM, a heterogeneous geoscientific object system developed at the UCLA Data Mining Laboratory. geoPOM provides applications the abstraction from heterogeneous spatial data repositories. geoPOM leverages the ODMG data model as well as the OGIS data model. The data modelis mapped to participating spatial data repositories using repository wrappers. geoPOM allows users to define, manipulate and query spatial object types; it automatically generates code that maps between the geoPOM types and the local spatial repository's data types. The code is linked into the application, permitting the application to directly access the spatial data repositories in order to achieve minimal performance loss. In geoPOM, we tackle different problems that arise when integrating and abstracting from different specialized repositories. Specialized repositories are used according to their capability to avoid having to implement the missing funcitonality outside of the repositories.

We plan to extend geoPOM in new directions such as support for object migration and tertiary storage. Since one of the common object types provided by geoPOM is a grid object type, which can be very large in size, support of this object type requires the use of tertiary storage in order to provide sufficient, inexpensive storage space. Therefore, one of the future development areas for geoPOM will be support of tertiary storage within a heterogeneous spatial object system. This includes extension of the strategy component for the definition of tertiary storage layout and staging rules, and data model extension for user-defined hints for data movement from and to tertiary storage.

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.M. 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. 14(5):407–420, 1989.

[4] M.W. Bright, A.R. Hurson, and S.H. Pakzad. A Taxonomy and Current Research Issues in Multidatabase Systems. 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.A. Burkhres and A. Elmagarmid, editors. *Object-Oriented Multidatabase Systems.* Prentice Hall, Englewoods Cliffs, New Jersey, 1996.

[7] M. Carey, W. Codey, L.M. Haas, and et al. Querying Multimedia Data from Multiple Repositories by Content: The Garlic Project. In *Proc IFIP Working Conference on Visual Database Systems, Lausanne, Switzerland.* IEEE Computer Society, 1995.

[8] M. Carey, W. Codey, L.M. Haas, P. Schwarz, M. Arya, W.F. Cody, R. Fagin, and et.al. Towards Heterogenous Multimedia Information Systems: The Garlic Approach. Technical Report RJ-9911 (87291), IBM Research Division, 1994.

[9] R.G.G. Catell, T. Atwood, J. Duhl, G. Ferran, M. Loomis, and D. Wade. Object Database Standard: ODMG-93. Morgan Kaufmann Publishers, San Mateo, California, 1994.

[10] M. Coyle, S. Shekhar, D.-R. Liu, and S. Sarkar. Experiences with Object Data Models in Geographic Information Systems. Technical Report TR 95-10, Department of Computer Science, University of Minnesota, 1995.

[11] D. DeWitt, N. Kabra, J. Luo, J.M. Patel, and J.B. Yu. Client-Server Paradise. In *Proc. of 20th Intl. Conference on Very Large Databases (VLDB), Santiago, Chile*, 1994.

[12] 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)*, 1989.

[13] Scientific Data Formats. *Web site for Scientific Data Format Information FAQ.* http://fits.cv.nrao.edu/traffic/scidataformats/faq.html, 1996.

[14] B. Fortner and D. Ilg. Draft HDF-EOS Primer for Version 1 EOSDIS, white paper. Technical report, Hughes Applied Information Systems, Inc., 1995.

[15] M. Haertig. An Object-Oriented integration Framework for Building Heterogenoeus Database Systems. In *Proc. of Conference on Semantics of Interoperable Database Systems, Lorne, Australia.* IFIP, DS-5, 1992.

[16] HDF. *Hierarchical Data Format.* ftp://ftp.ncsa.uiuc.edu/HDF, 1996.

[17] Illustra. *User's Guide, Version 3.2.* Informix, Inc., 1996.

[18] 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.

[19] W. Kent, R. Ahmed, J. Albert, M. Ketachi, and Shan. 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. Kriguer. OASIS: An Open Architecture Scientific Information System. In *Workshop on Research Issues in Data Engineering (RIDE 96), New Orleans.* IEEE Computer Society, 1996.

[21] E. Mesrobian, R. Muntz, E.C. Shek, and J.R. Santos. Exploratory data mining and analysis using Conquest. In *Proc. of IEEE Pacific RIM Conference on Communications, Computers, Visualization, and Signal Processing.* IEEE Computer Society, 1995.

[22] netCDF. *Network Common Data Form.* http://www.unidata.ucar.edu/packages/netcdf/, 1996.

[23] E. Pitoura, O. Bukhres, and A. Elmagarmid. Object Orientation in Multidatabase Systems. *Communications of the ACM*, 27(2):141–195, 1995.

[24] E. Radeke and M.H. Scholl. Framework for Object Migration in Federated Database Systems. In *Proc. of Int. Conference on Parallel and Distributed Information Systems (PDIS), Austin, Texas*, 1994.

[25] H.-J. Schek, H.-B. Paul, M.H. Scholl, and G. Weikum. The DASDBS Project: Objectives, experiences, and future prospects. In *IEEE Transactions on Knowledge and Data Engineering*, volume 2. IEEE, 1990.

[26] SDE. *User's Guide, Version 2.0.* ESRI, 1995.

[27] A.P. Sheth and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *Communications of the ACM*, 22(3):183–236, 1990.

[28] R. M. Soley, editor. *Object Management Architecture Guide (2nd Edition).* Object Management Group, 1992.

[29] M. Stonebraker. Sequoia 2000 - a Reflection on the First Three Years. In *Scientific and Statistical Database Management Systems, Charlotteville, Virginia.* IEEE Computer Society Press, 1994.

[30] M. Stonebraker and L.A. Rowe. The Design of Postgres. In *Proc. of SIGMOD 86.* ACM Press, 1986.

[31] 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.