OASIS: An Open Architecture Scientific Information System

Edmond Mesrobian, Richard Muntz, Eddie Shek,
Silvia Nittel, Mark LaRouche, and Marc Kriguer
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

Motivated by the premise that heterogeneity of software applications and hardware systems is here to stay, we are developing OASIS, a flexible, extensible, and seamless environment for scientific data analysis, knowledge discovery, visualization, and collaboration. In this paper we discuss our OASIS design goals and present the system architecture and major components of our prototype environment.

1 Introduction

In the course of research activities, a scientist would like to efficiently store, retrieve, analyze and interpret selected data sets from a large collection of scientific information scattered across heterogeneous computational environments, and to share the gleaned information with other scientific communities both nationally and internationally. Consider the following prototypical interaction of a scientist with a dataset[1]. Once a scientist has located the dataset of interest, a major undertaking in its own right, a specific portion of the dataset is retrieved. Typically the desired subset of the dataset is defined by a space/time window (e.g., area from $10°N$ Latitude $120°W$ Longitude to $50°S$ Latitude $120°E$ Longitude during the time period from Jan 15, 1979 to Dec. 25, 1980) and an attribute list (e.g., temperature, pressure). Next, a conversion program is executed to transform the data into a form (e.g., Hierarchical Data Format (HDF)) digestible by the analysis/visualization programs that are going to be utilized. Sometimes the analyst has to write the conversion program, which may require battling with poorly documented, locally created data formats[2]. This step is followed by the transmission of the dataset from the archive site to the user's local workstation (e.g. File Transfer Protocol (FTP)). The dataset is then injected into the desired analysis package (e.g., IDL, MATLAB, AVS) or into a local database. Here, the dataset might undergo filtering to remove noise, computation of parameter averages and cross correlations, extraction of data slices, etc. The results of these operations are then visualized. The analysis and visualization steps are iterated as necessary.

In summary, a scientist/programmer often has to battle with a plethora of computer systems, programs, protocols, and data formats. Heterogeneity is however a fact of life in dealing with computers. Hardware vendors continue to develop diverse platforms and operating systems; file systems, network software, etc. continue to proliferate. Vendors and research institutions continue to develop a large number of software tools, each of which satisfies part of what the scientist needs. Ironically, it is the availability of such a range of hardware and software that creates a major problem. More often than not, even within a single scientific group, a diverse set of platforms and tools are utilized. This diversity usually translates into time consuming and costly integration problems which were not anticipated. The problem is even more severe when attempting to share information or code between groups. We believe that software and hardware heterogeneity are here to stay. The players may change; today's state of the art systems may become tomorrow's legacy systems, network software, etc. continue to proliferate. The approach we advocate deals directly with heterogeneity rather than attempting to build a monolithic, all encompassing system that will replace everything else and therefore make heterogeneity disappear.

Motivated by the premise that heterogeneity is here to stay, we are developing OASIS (Open Architecture Scientific Information System), a *flexible, extensible, and seamless environment for scientific data analysis, knowledge discovery, visualization, and collaboration.* The central notion behind the proposed architecture is to provide software developers, as well as end-users, the logical abstraction that the entire computing environment is simply a set of objects of various types ("classes") as illustrated in Fig. 1 (the objects are at the core of system). An object class defines a type of object in terms of attributes (variables) and a set of operations ("methods"). For example, consider a scientist charged with the task of verifying an atmospheric dataset captured by the High Resolution Infrared Radiation Sounder (HIRS) and Microwave Sounding Unit (MSU) instruments flying aboard NOAA's polar orbiting meteorological platform with a model-generated dataset. Using the OASIS Catalog Browser, a scientist identi-

[1] A dataset is a unified collection of information. It represents information which has been collected as a part of a defined process like data collected by a satellite's remote sensors or produced by a simulation model.

[2] Archive formats like HDF and netCDF are alleviating the need to create new, proprietary data formats.

fies the HIRS2/MSU dataset object (a collection of multi-dimensional cellularly-gridded coverages). The network-addressible object reference can be used by a visualization package such as LinkWinds [1] or by scientific query analysis tool such as the Conquest Parallel Query Execution System [2]. The HIRS2/MSU object encapsulates the code necessary to retrieve subsets of data possibly stored in a file system or a data base system (DBS), or perform operations on the data. However, all HIRS2/MSU objects have a common interface, i.e., support a common set of operations. A "handle" or object reference is used to refer to the object and invoke these operations in a uniform manner, regardless of the location of the object or the implementation details. The point here is that the scientist **does not** have to remember, or be bothered with intricate, yet meaningless, information and can remain focused on the task at hand.

To meet OASIS design goals, we are exploiting the emerging distributed object management system (DOMS) technology as promoted by the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standard [3]. CORBA-compliant software is now available from many vendors and, now that a "on the wire" protocol standard for inter-ORB communication has been adopted (CORBA 2.0), interoperability between different vendors' implementation of CORBA on heterogeneous platforms will be supported. The basic idea of a CORBA-compliant DOMS is to merge the notion of location and platform transparency and object-oriented software technology. The object-oriented paradigm is well established as having a number of significant advantages in terms of building reusable, maintainable, extensible software. The idea then is to provide the abstraction to scientific applications, and in turn to scientists, that the "world" consists of a set of objects. These objects are designed to correspond to the needs of the applications, however shielding them from the actual physical realizations of the objects. Location and platform transparency means that scientific applications are provided with an abstraction in which they can operate on objects without regard for the objects' location, the hardware or software platform, etc. The bottom line is that applications are written in terms of this abstract world of objects and operations on objects.

The remainder of the paper is organized as follows. In Section 2 we outline our object model. Next, we present the OASIS system architecture and discuss the major system components. In Section 4, we present preliminary results, and conclude with a few remarks.

2 OASIS Object Model

In order to build the conceptual architecture shown in Fig. 1, one must define an object model for data objects that are to be manipulated by scientists (via applications) and managed by the OASIS environment. In some respects, this goal is ill-posed; a closed set of objects that cover (model) the enormous diversity of scientific data products employed by the science community does not exist.

To make the problem tractable, one typically selects an application area to restrict the set of objects to be defined, implemented and managed. In our project, we are concentrating on the support of a data model for geoscience data collected and produced from observations (e.g., Pathfinder datasets), Earth Observing System datasets at different processing levels, simulations (e.g., UCLA Atmospheric General Circulation Model), and derived products (e.g., high-level features extracted in data analysis runs).

Some efforts attempt to develop a completely new conceptual model and "world view" for geographical, spatio-temporal data and information systems. Alternatively, one may embrace a data model that has been accepted by the GIS community as its standard and develop an object and process model based on it. Unfortunately, an general accepted GIS data model does not exist. However, an effort is underway to alleviate this situation. The Open GIS Consortium has embarked on an effort to define an Open Geodata Interoperability Specification (OGIS) [4] for GIS data. The core of this specification is a standardization of basic geoscientific entities like e.g. datasets, features, or feature collections regarding their attributes and their operations. Over the past year and a half, we have been actively involved in defining OGIS.

Figure 2 presents our OGIS-inspired *interface hierarchy*. The Geometry branch provides the interfaces required to support a spatial object's geometric properties, be it a Point, LineString/Chain (a set of connected points), or a Grid, etc.. The GeoDataset-rooted interfaces are intended to provide behavior generic to all datasets (e.g. query support) as well as methods that are particular to the different spatial models (e.g. grid, image, vector). The non-spatial interfaces (Catalog, Mediator, etc.) provide facilities to gather and query spatial objects based on common attributes ("metadata").

Figure 3 depicts the OASIS *object hierarchy*. A GeoDataset is a type of object which is intended to represent geographic information. As with the more general dataset concept, we can think of different types of GeoDataset, each representing a different formal spatial model (e.g. vector, grid, image). Of particular interest to our research effort are objects designed to support model- or sensor-produced multidimensional regularly and irregularly gridded spatio-temporal data as well as traditional GIS objects such as polygons, line strings, etc. Multidimensional gridded datasets are typically quite large, representing tens or hundreds of megabytes of data, or more, are "heavyweight" enough to be represented as a CORBA object; the same cannot be said of fine-grained spatial objects.

Real-world features such as roads typically have geometric properties (centerline, extent) represented using line strings, envelopes, etc. and a set of attribute values. However, if one desired to model/instantiate an object for every road in North America, the number of objects can grow into the millions. One possible design strategy that we are investigating is the use of the container approach (FeatureBucket). The FeatureBucket is designed to provide storage and retrieval of features by value. All features in the bucket

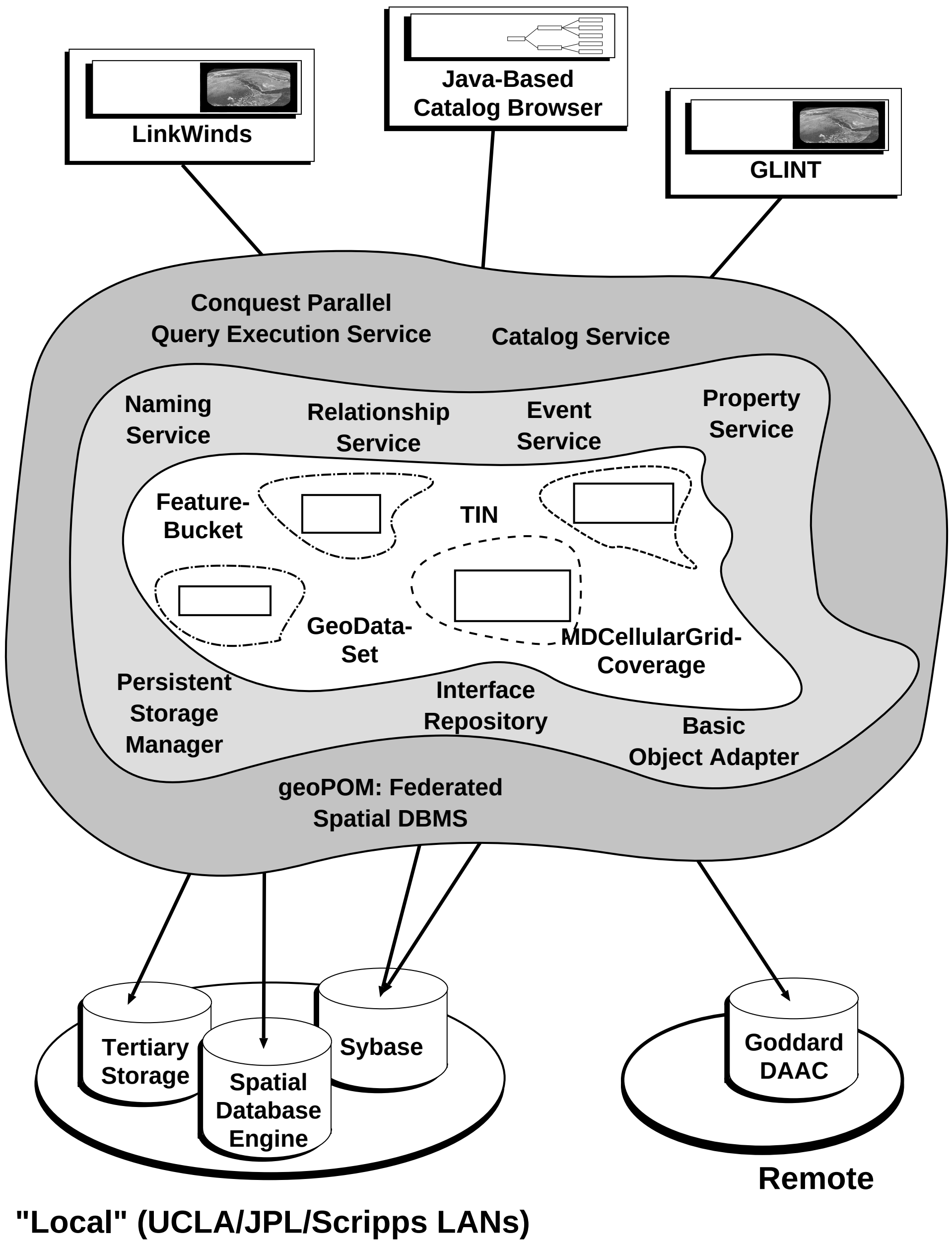

Figure 1: Conceptual architecture of OASIS, a flexible, extensible, and scalable environment for scientific data analysis, knowledge discovery, visualization, and collaboration. Central to the architecture are the scientific, spatio-temporal objects accessed by applications via a distributed object management framework.

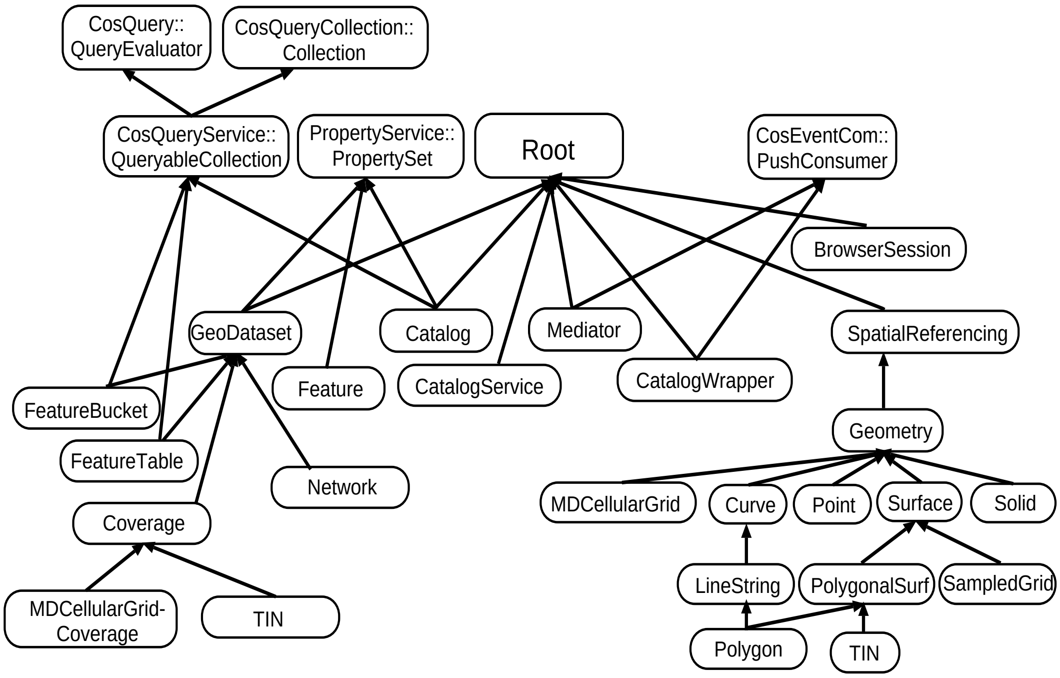

Figure 2: OASIS's OGIS-inspired interface hierarchy.

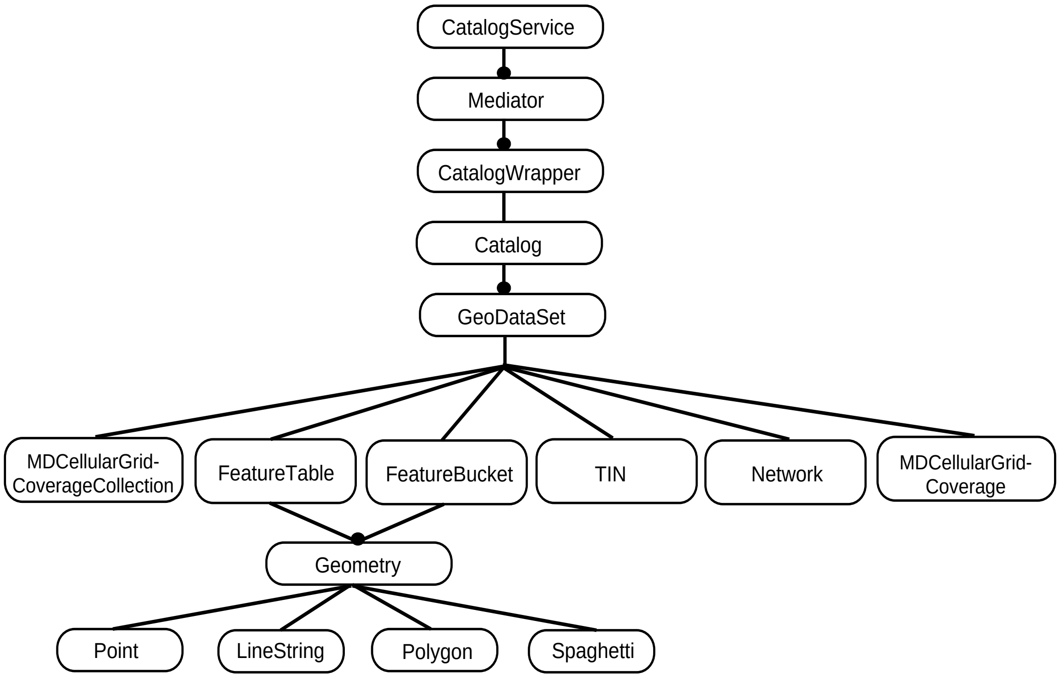

Figure 3: OASIS's OGIS-inspired object hierarchy.

have the same attributes, but may have differing geometric properties (Point, LineString, Spaghetti, Polygon, etc.). Queries are expressed using SQL along with spatio-temporal operators. Results are delivered to the requesting client as value structures via an iterator. The FeatureTable is a container GeoDataset object with slightly differing properties. First, all features must have the same geometric representation and attribute set. Second, methods provide the ability to send and retrieve feature information as CORBA objects (geometry objects).

3 System Architecture

In Fig. 1, we depict a conceptual environment for scientific data analysis, knowledge discovery, visualization, and collaboration based on the marriage of object-oriented programming paradigm with a distributed object management system. At the heart of the system lies our object hierarchy which acts as the conceptual/semantic glue linking our applications with scientific data. Surrounding this core is OMG's common Object Request Broker (ORB), Basic Object Adaptor (BOA), and Interface Repository (IFR). The next layer contains a host of Common Object Services. The latter two layers provide the physical machinery and services necessary to develop a distributed, object-oriented computing environment. The final layer of this processing onion contains the query, storage, and catalog services provided by our research effort. Applications that are operating on top of this distributed computing substrate include: the Java-based Catalog Browser, GLINT and LinkWinds. The latter two applications are legacy applications developed at JPL under different funding auspices. In this effort, we are collaborating with colleagues at JPL to extend these systems in order to embrace distributed object technology.

3.1 Distributed Object Management System

Our development effort is based on SunSoft's CORBA-compliant NEO software. We have participated in SunSoft's Early Developer Release Program of NEO (previously known as DOE) and have been a beta test site since March 1994. NEO enables users to access information location independent in the network as discrete, self-contained "objects". An object, a combination of code and data, may handle a single function, or it may encapsulate an entire existing application and its data. Once again, it should be noted that NEO provides interoperability with other vendor's CORBA environments due to the recently adopted CORBA-2, Universal Networked Objects specification.

In Fig. 1, the NEO facilities are shown as the light shaded blob encapsulating the scientific objects. The first blob next to the scientific objects contains the core CORBA environment, i.e. the ORB, the IFR which keeps track of available object types in the environment, and the BOA. Central to this layer is the ORB which acts as an intermediary between objects and clients, locating and delivering messages to requested objects. The ORB employs the BOA to start up the server process for a requested object (or to create a new thread inside an existing server process). The next blob contains the CORBA Common Object Services. To build object servers as well as client applications, we are using the NEO Workshop, a powerful, C++ API to ORB, BOA, IFR and CORBA's Common Object Services.

3.2 OASIS Services

The main services provided by our effort appear in the outer (dark grey background) area in Figure 1.

Conquest Parallel Query Execution Environment

The primary goal of our current HPCC [5] research effort is to demonstrate the applicability of information systems for geophysical databases to support cooperative research in earth and planetary science projects. The Conquest Parallel Query Processing System [2] was designed to execute scientific queries expressed as dataflow diagrams. A novel scientific data model was developed to facilitate exploratory data analysis and data mining and to encompass scientific observation and simulation data so that it can be used as a canonical data model in a heterogeneous scientific query processing environment. In addition, the data model and its associated algebra capture significant semantic and structural properties, and hence provide the basis for exploiting these properties during the parallelization and optimization of complex scientific queries.

A scientific query, visually represented as a dataflow diagram (Fig. 4) is expressed by the user using a scripting language. An interpretor automatically translates the query into an equivalent Conquest algebraic expression and sends it to the Conquest Query Manager. The Query Manager, upon receipt of a query, sends the query to the rule-based Query Optimizer which optimizes, parallelizes, and transforms the algebraic expression into a parallel Query Execution Plan (QEP). The plan is then forwarded to the Conquest Query Execution Server for evaluation. In the mean time, the Query Manager sets up a connection to the Visualization Manager and initializes it in preparation for the data stream returning from the Query Execution Server as the result of the query. The design of the Query Execution Server is based on the Volcano extensible query execution engine [6]. Conquest extends Volcano with support of a scientific data model encompassing relational data, scientific data fields, and multidimensional array data. We have implemented generic algebraic operators in this data model as well as application-specific operators to support scientific studies. The executor adopts Volcano's hybrid demand- and data-driven dataflow paradigm, and supports different forms of query parallelization through various support operators.

It is natural to ask how the Conquest's field model is related to the OASIS data model. The field model and OASIS object model share some very fundamental concepts such as fiber bundles (the separation of value and index spaces) and space bundling (ability to represent a hierarchy of spaces). However, the OASIS

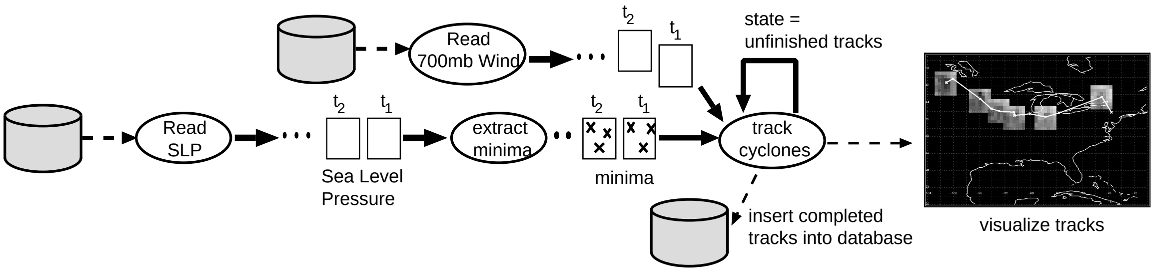

Figure 4: Data flow representation of the cyclone tracking query.

object model has a more behavioral aspect to it since data objects have behaviors, while the field model comes from a more database point of view in that it captures more of the properties of data (e.g., layout) and algebraic operations (e.g., how data can be partitioned for parallel processing) so that this knowledge can be exploited during query optimization.

Catalog Service

One of our project goals is to provide one stop shopping for scientific data, be it data created and stored locally, or data located at remote data repositories such as NASA's EOS Distributed Active Archive Centers (DAACs)[3]. In this scenario, scientists pose queries via the OASIS Catalog Service infrastructure (Fig. 5).

The Catalog Service is loosely modeled after the TSIMMIS system [7]. Mediators are employed to realize global query schemas and provide query support via their query language (e.g. ODL, SQL). Upon receipt of a query, a Mediator decomposes the query and farms out sub-queries to its participating catalogs, which are encapsulated via Catalog Wrappers. A Catalog Wrapper translates the query from the global query schema and language to the target catalog's local schema and query language. The catalog then processes the query and returns an iterator object (e.g. database cursor) to the Wrapper. The Wrapper, in turn, passes an iterator object back to the Mediator. However, this iterator returns tuples in the Mediator's global schema. To do so, the Wrapper's iterator issues a request to the catalog iterator for the next tuple, translates the tuple from the catalog's schema into the Mediator's global schema, and returns the resultant tuple. The major reason for this "lazy" evaluation of query results is to enable query results to be returned to the user as necessary, thereby avoiding potentially unnecessary processing overhead (especially on large query result sets).

geoPOM: Federated Spatial DBMS

OASIS scientific objects need to store their state in a storage system, i.e. a file system or database management system (DBMS). However, binding the object implementation to a specific storage system makes it non-portable. For example, the state of an OASIS grid object may in fact be stored in different file formats like netCDF, HDF, etc.. One way to solve this problem is to implement different versions of the grid object (same interface) each supporting a different file format. However, if a new file format is added, a new implementation of each grid object class has to be developed. Alternatively, one can provide an abstraction of the physical storage and let the object delegate the storage of its state to a persistent object manager which handles the actual mapping to a concrete storage system. Now, if new file format or storage system is added, only the persistence object manager has to provide a new adaptor to it.

The OMG CORBA specification adopted a specification for such a system called the Persistent Object Manager (POM) [8]. The POM provides the functionality of a federated object-oriented DBMS, however without supporting object methods or a query language. This functionality is not sufficient for a geoscientific environment where COBRA objects like feature buckets need to store their fine-grained state inside a DBMS and to recreate it in main memory by posing spatial queries to the DBMS, i.e. support of a (spatial) query language is necessary. Furthermore, loading flexibility and storage optimization has to be provided for huge objects like grid objects. Therefore, we extended POM for a geoscientific environment and the resulting system, named geoPOM, comprises the functionality of a federated spatial DBMS.

geoPOM uses an object-oriented data model based on ODMG's ODL and OQL as the common data model. Furthermore, geoPOM's common data model provides spatial types like points, polygons, or multidimensional arrays, and spatial operators for these types. Fine-grained spatial objects are mapped to spatial DBMSs like Illustra or ESRI's SDE, while large grained objects like multi-dimensional grids can be mapped to the corresponding types of spatial DBMS, if available, or to files using archive formats like HDF or netCDF. Object types not including spatial characteristics are stored in an object-oriented or relational DBMS. While geoPOM is considered to be the basic persistence manager for all CORBA objects in the OASIS environment, we are focussing primarily on spatial types in the development of geoPOM.

[3] Currently, we have catalogs scattered across UCLA and JPL, and plan to add the Scripps Institute of Oceanography and Goddard Space Flight Center's DAAC in the near future.

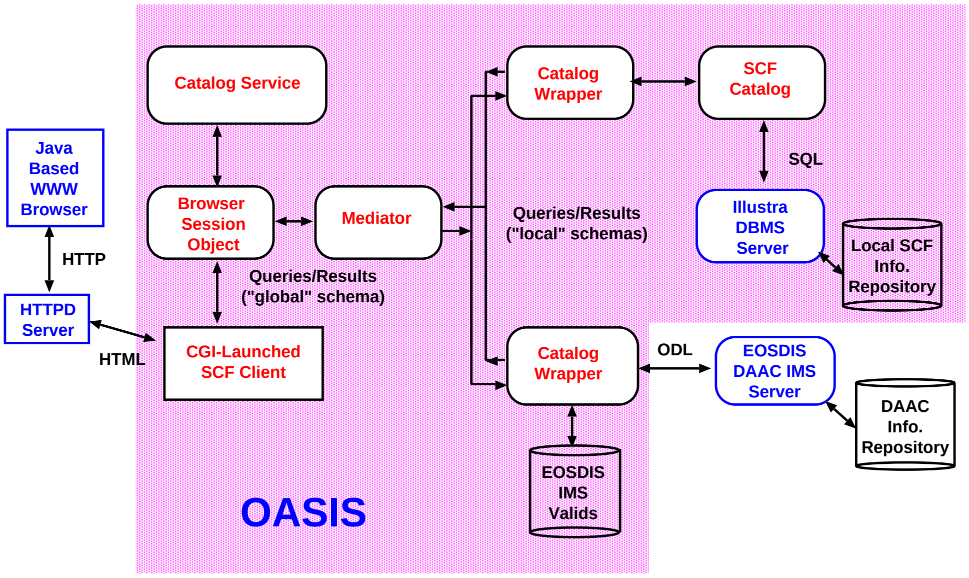

Figure 5: OASIS Catalog Service System.

3.3 OASIS Applications

Catalog Browser

The Catalog Browser provides a graphical user interface for locating scientific datasets, derived products produced as the result of previous queries, etc.. The Browser is written using SunSoft's Java language and C++ CGI-bin programs and executes via Java-enabled Netscape 2.0. The Browser can be invoked in a stand-alone mode or launched from an application such as LinkWinds or GLINT (see Fig. 1). Upon invocation, the scientist is asked to choose from a set of available keyword sets (i.e. global schemas). This information is provided to the Browser by the Catalog Service. Once the global schema is chosen, the Browser interacts with the Mediator which is responsible for evaluating queries on the the schema. The Mediator provides the Browser with schema information (keywords, valid values, etc.) necessary to build the appropriate query form.

Figure 6 presents an example query form. In this example, the scientist can specify a time range, a spatial extent (by drawing a bounding box over the map), and select parameters of interest. Figure 7 shows the detailed information retrieved presented to the scientist as the result of the example query. The net result of a scientist's interaction with the Browser is the location and selection of distributed objects (i.e., datasets). The object references of the selected objects can then be passed back to the launching application, or can be passed to an application invoked through the Browser.

GLINT

GLINT is a tool developed by Scripps Institute of Oceanography and JPL's Data Exploration Laboratory that provides a means to study data from geographic subregions of a global dataset as well as interactively comparing multiple data parameters. The application is designed to support the analysis of both point (Level 2) data and gridded (Level 3) data. GLINT is written using Research System Inc.'s Interactive Data Language (IDL), and is being layered on top of OASIS.

LinkWinds

The Linked Windows Interactive Data System (LinkWinds) is a prototype visual data exploration system developed at JPL. LinkWinds employs a graphical spreadsheet paradigm to support access, display, and analysis of large, locally stored data sets. Running under UNIX, it is an integrated multi-application execution environment allowing the dynamic interconnection and control of multiple windows containing a variety of displays and manipulators. As part of our prototype effort, we have enhanced LinkWinds to employ the OASIS API to locate and access distributed, heterogeneous datasets.

4 Preliminary Results

During the past year, we have developed a prototype of OASIS using SunSoft's NEO environment. We have prototyped the object hierarchy depicted in Fig. 3, and implemented the MDCellularGridCoverage and MDCellularGridCoverageCollection for gridded data and the FeatureBucket and FeatureTable for lightweight GIS objects.

The MDCellularGridCoverage family of objects supports representation of and access to gridded data stored in HDF and NetCDF data files. The collection object provides a convenient representation of a gridded dataset's parameters stored (tiled) across numerous data files. To elucidate access times to gridded data via these objects, consider the following test case.

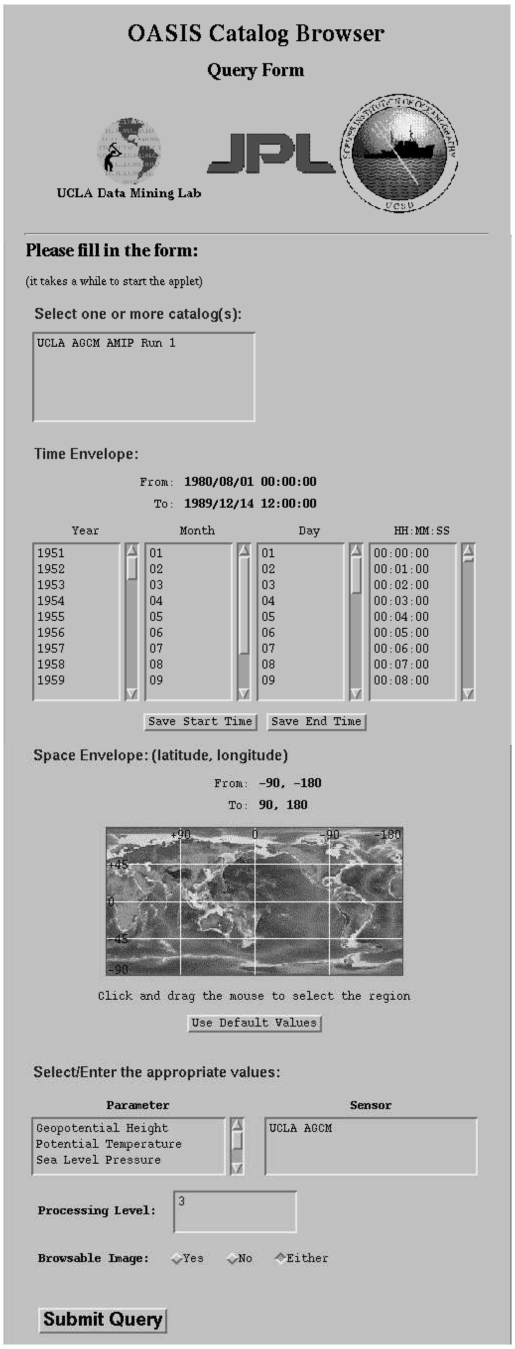

Figure 6: Example of a query form produced to support EOSDIS IMS schema.

Record #1 :

Object reference = 1:2:color:47892:03baea8600008027:0

parameter	Potential Temperature
sensor	UCLA AGCM
browse	Yes
processing level	3
minRange	177.85
maxRange	310.994
unit	Kelvin
Time Envelope	1980/08/01 00:00:00 GMT to 1989/12/14 12:00:00 GMT
Space Envelope	(200, -86, -180), (850, 86, 175)
Dimensions	4 X 44 X 72 (Elevation, Latitude, Longitude)
Time Steps	6842 steps
Step Delta	0 d, 12 h, 0 m, 0 s
DataSet Size	346807296 bytes

Please click on the image for animation effect:
(for the first 20 time steps)

Save this dataset? Yes No

To save the marked datasets, click the "Save" button.

Save

Figure 7: Example of detailed information retrieved for a dataset selected via a query.

A year's worth (sampled twice a day) of sea level pressure data (single precision floating point) generated by UCLA AGCM model with a 4° Latitude by 5° Longitude grid cell resolution is 9250560 bytes ($730x44x72x4$). Each grid "slice" is therefore 12672 bytes. In this example, the dataset was stored in a HDF file located on the same machine as the MDCellularGridCoverage object, and the client was run on a different Sparc 20 Model 71 on the same LAN (10baseT). Given a query to retrieve the entire year's worth of data, the object returns an iterator to the result set. A user can ask for a slice at a time or a n slices at a time. The times below are the total elapsed time to query and retrieve the entire dataset. Asking the iterator object for a slice of data at a time (i.e. 730 CORBA/object calls) took 38.45 seconds. Asking for 64, 128, and 730 slices at a time took 21.88, 23.30, 24.76 seconds, respectively.

The FeatureBucket and the FeatureTable provide a container for lightweight GIS objects. In the current implementation, both objects store the "states" of their feature elements (i.e. points, polygons) as records in the Illustra DBMS. We use Illustra's Spatial Blade (abstract data types and methods) to define the lightweight spatial types, i.e. their temporal and spatial attributes, their meta data and user-defined attributes. Feature access is optimized using R-trees for the spatial attributes. Preliminary performance analysis shows that creating 100 polygon features, each polygon consisting of 30 points, takes about 94 seconds while performing spatio-temporal queries takes around 1 second. FeatureTable objects have similar performance characteristics. The major difference between the two object types is the time taken to retrieve features from their query result set iterators. A next call on a FeatureBucket's iterator returns a feature as a value structure in 0.02 seconds. A FeatureTable's iterator returns a geometry object (a full-blown CORBA object) in 0.1 seconds. We plan to further investigate the performance of these objects by running the FeatureBucket and FeatureTable object servers inside client process walls and by employing other spatial DBMSs for feature storage.

The current implementation of the Mediator, Catalog Wrappers, and Catalogs all support the same data model (i.e. translation from a global schema to a Catalog's local schema is straightforward, requiring only a mapping from Illustra SQL data types into CORBA data types). Efforts are underway to integrate EOSDIS's IMS Catalog located at the Goddard Space Flight Center. This catalog supports a proprietary query language and communication protocol. The Catalog Wrapper for this catalog will actually perform the required query and schema translation into NASA's Object Definition Language, send the request to the GSFC DAAC, and perform the inverse mapping of query results.

5 Conclusions

In this paper, we presented the goals, functionality, architecture and preliminary results of OASIS, a flexible, extensible, and seamless environment for scientific data analysis, knowledge discovery, visualization, and collaboration. OASIS offers scientists as well as application developers a view of the world as a collection of distributed, location- and platform independent objects. This view is achieved for application developers through the usage of well-known object interfaces based on OGIS, and object implementations based on CORBA. The view of the world as location-independent objects is offered to scientists via high-level applications such as the OASIS Catalog Browser which provides query facilities to locate geodatasets, as well as visualization and analysis tools which directly deal with OASIS data objects.

References

[1] A. S. Jacobson, A. L. Berkin, and M. N. Orton. Linkwinds: Interactive Scientific Data Analysis and Visualization. *Communication of ACM*, 1994.

[2] E.C. Shek and R.R. Muntz. The conquest modeling framework for geoscientific data. Technical Report Technical Report 940039, UCLA Computer Science Department, Oct 1994.

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

[4] K. Buehler and J. A. Farley. Interoperability of Geographic Data and Processes: The OGIS Approach. *StandardView, ACM Perspectives on Standardization*, 2(3):163–168, 1994.

[5] R.R. Muntz, L. Alkalaj, D. McCleese, C. Mechoso, J. Skrzypek, and C. Zaniolo. Data analysis and knowledge discovery in geophysical databases. NRA-92-OSSA-2, 1992.

[6] G. Graefe. Volcano, an extensible and parallel dataflow query processing system. *IEEE Trans. Knowledge and Data Engineering*, 6(1):120–135, 1994.

[7] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information source. In *Proceedings of the 100th IPSJ, Tokyo, Japan*, 1994.

[8] Object Management Group. IBM/JOSS Object Services, Persistence Service Specification. Technical report, OMG TC Document Number 93.11.3, 1993.