# A Moving Objects Database Infrastructure for Hurricane Research: Data Integration and Complex Object Management

Markus Schneider[1], Shen-Shyang Ho[2], Malvika Agrawal[1],
Tao Chen[1], Hechen Liu[1], Ganesh Viswanathan[1]

[1]Department of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611

[2]Center for Automated Research
University of Maryland
College Park, MD 20742

***Abstract*—Current web-based weather event and satellite data portals provide large amounts of data over a historical timeline. However, users of these portals often get access to data only through limited, pre-defined queries based on a strict set of criteria and event trajectories. Desirable capabilities, such as spatial-temporal analysis, efficient satellite data retrieval, and ad-hoc queries on trajectory data, are not available in these information systems and data archives. In this paper, we describe our current work on and progress in the development of a sophisticated moving objects database infrastructure designed primarily to allow ad-hoc querying of dynamic atmospheric events (e.g., hurricanes and storm systems) and the efficient retrieval of satellite (e.g, QuikSCAT, TRMM) measurements. In particular, we describe our progress in the integration of tropical cyclone event data and satellite measurements from different sources into a single moving objects database system for scientific users to perform ad-hoc queries and sophisticated spatiotemporal analysis. Moreover, we describe how a user can remotely connect her personal analysis software to the database system to perform flexible querying on tropical cyclone best track data and retrieve the associated satellite measurements. Finally, we show how complex objects like hurricane trajectories and massive satellite sensor trajectories with measurements can be effectively stored and handled in a database context using our novel iBLOB (Intelligent Binary Large Objects) concept and data structure.**

## I. Introduction

Weather event related information like hurricane best track data and the satellite data are currently maintained by various agencies in rather diverse file formats. The distribution of data repositories and the incompatibility of data formats lead to inconsistency among data, and more importantly, make the retrieval, combination, and analysis of data across different sources rather complex and inefficient. The web-based weather event information portals, data archives, and forecast services provide excellent subsetting and visualizations of weather events and satellite sensor measurements. However, they are restricted to certain data sources, and they provide only limited, simple and hard-coded query, retrieval, and analysis functionalities. On the other hand, database technologies support consistent data storage, efficient data retrieval, data indexing, ad-hoc queries, and complex analysis like aggregations. Thus, to bring database technology into handling weather event information is beneficial. The main objective of our project is to provide the NASA workforce with previously unavailable database management, analysis, and query capabilities that will advance the research and understanding of dynamic weather events and be based on weather data derived from the NASA mission sensor measurements.

In this paper, we describe our system from two perspectives, the data integration and the system architecture. From the data integration perspective, we describe the hurricane data integration from the various sources, the various normalization steps, and the construction of a *moving objects database* (*MOD*). We particularly describe the challenge how the terabyte scale data is properly migrated from HDF5 files into databases, and how we design the schema and organize the data with indexes for efficient spatial and temporal range queries. From the system architecture perspective, we describe in detail our system components which are the *Moving Object Software Library* (*MOSL*) and the *Spatiotemporal Query Language* (*STQL*). Further, we include a discussion about handling complex application objects in databases and our generic solution called $iBLOB$.

The outline of this paper is as follows. Section II reviews related work. In Section III, we outline the application and system objectives of our project. In Section IV, we describe the data integration and the main components of our system and their implementation. In Section V, we describe a MATLAB based application that is driven by our moving object database query output. Finally, in Section VI, we draw some conclusions and provide an outlook to our future work.

## II. Related Work

Our objective is to have a comprehensive moving objects database system that provides unified representations and rich functionalities for large amounts of tropical cyclone and hurricane data from various sources. However, the way how the hurricane data are currently handled is far away from

this goal. In this section, we explore the hurricane data from various data sources (Section II-A), review the approaches to handling complex objects in traditional, standard databases (Section II-B), and describe our concept of moving objects databases (Section II-C).

### A. Hurricane Data Sources

The hurricane data in which we are interested in our project falls into the two data categories of *cyclone best track data* and *satellite data*.

Three major agencies collect cyclone best track data, the *National Hurricane Center* (*NHC*), the *NOAA Hurricane Research Division*, and the *Joint Typhoon Warning Center* (*JTWC*). Data from NHC and JTWC are collected from satellites every 6 hours. The data from NHC relates to the North Atlantic and Eastern Pacific areas, while the data from JTWC refers to the southern hemisphere, the North Atlantic Ocean, and the Northwestern Pacific area. The best track data from NOAA is derived from center fixes (i.e., the location of the center of a tropical cyclone), which are obtained by flying aircraft over a hurricane. The flight frequency affects the amount of data for each hurricane. Data from NOAA complements the hurricane track data from NHC and JTWC, and therefore can reflect erratic motion of the hurricanes.

Every agency maintains their own storage for the cyclone data, mainly in files. Different file formats are used for storing cyclone data in different agencies. NHC maintains so-called *hurricane best track files* for cyclone data, which is usually referred to as the *HURDAT* format. Center locations (latitude and longitude in tenths of degrees) and intensities (surface wind speeds in knots and central pressures in millibars) are recorded every six hours. Cyclone data in NOAA is stored in totally different formats like *USAF* and *HSA*. Every named hurricane has a center fix file and a track file. The latitude and longitude positions are output every two minutes to make the track file. As a result, the variety of data formats causes the inconsistency among cyclone data from various sources, and makes querying and combining cyclone data across different agencies rather difficult.

The other data category is satellite data. One can imagine a satellite data file as a sequence of snapshots of a moving object obtained from an orbiting satellite. The data type design for satellite data should be independent of a satellite mission. For satellite moving objects, we consider Level 2B wind field data from the *QuikSCAT* satellite and Level 2 precipitation data from the *TRMM* satellite.

The QuikSCAT (Quick Scatterometer) mission provides an important high quality ocean wind data set. The specialized microwave radar on the QuikSCAT satellite measures wind speed and direction under all weather and cloud conditions over Earth oceans. Near real-time wind data is available to the weather forecasting agencies from NOAA within three hours of observation. The *Tropical Rainfall Measurement Mission* (*TRMM*) is a joint mission between NASA and the *Japan Aerospace Exploration Agency* (*JAXA*) designed to monitor and study rainfall.

Data captured by these two satellite missions are stored in the same format called *Hierarchical Data Format* (*HDF5*). HDF5 is a data model, library, and file format for storing and managing scientific data. It supports an unlimited variety of data types and is designed for flexible and efficient I/O and for high volume and complex data. However, it does not provide ad-hoc query support for the data stored. Thus, searching for data that satisfies customized criteria is not possible in HDF5.

Therefore, a database system is a perfect choice for overcoming these issues. Moreover, with its built-in features like data abstraction, concurrency control, transaction management, and multi-user access control, a database system can support hurricane applications much better than a file management system.

### B. Approaches to Handling Complex Structured Application Objects in Databases

Traditional database management systems (DBMSs) are well suited to store and manage large, unstructured alphanumeric data. However, they offer very little support to handle large, structured application objects (such as spatial and spatiotemporal data) at a high abstraction level and to implement operations on them. *Binary large objects* (*BLOBs*) are the only means to store such complex objects. However, BLOBs represent them as low-level, binary strings and do not preserve their structure. As a result, this internal database solution turns out to be unsatisfactory.

As a result, most applications involving complex objects use one of two architectures to incorporate support for complex objects in databases. The first approach involves a *layered architecture* as shown in Figure 1a, in which a *middleware*, which handles complex application objects, is clearly separated from the *application front-end*, which provides services and analysis methods to its users, and from the relational database management system, which physically stores only the underlying primitive data in traditional relational tables. This approach has two main drawbacks. First, the application developer has to implement the middleware so that it is suitable enough to handle complex objects and support efficient operations on them. Second, all the benefits from the relational database technology get lost.

A largely accepted approach is to model and implement complex data as *abstract data types* (*ADTs*) in a *type system*, or *algebra*, which are then embedded into an extensible DBMS and its query language. This forms the basis of the second *integrated architecture* (Figure 1b), where the applications directly interact with the extended database system and use the ADTs as attribute data types in a database schema. However the ADTs have to be implemented using BLOBs for internal storage. Thus the information about the complex structure of an application object is not maintained at the database level.

We have developed a unique solution to this problem by supplying a novel extension to the integrated architecture approach, and by providing *type system implementers* with a high level access to complex structured objects. Our generalized framework (Figure 1c) consists of two components, the *type*

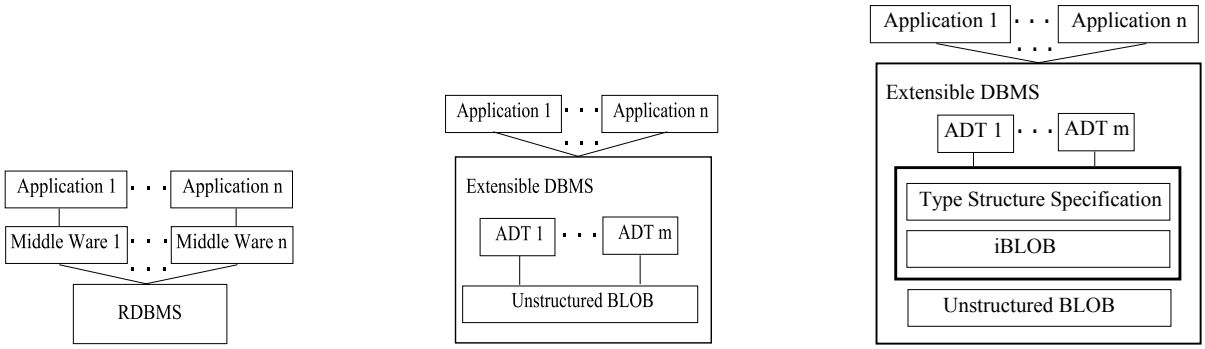


Fig. 1. The layered architecture (a), the integrated architecture (b), and our solution (c).

*structure specification* (*TSS*) and the *intelligent BLOB* (*iBLOB*) concept [1]. The TSS consists of algebraic expressions that are used by type system implementers to specify the internal hierarchy of the abstract data type. It is later used as metadata to identify the semantic meaning of each structure component. Further, as part of the TSS, we provide a set of high-level functions as interfaces for type system implementers to create, access, or manipulate data at the component level. To support the corresponding interfaces, we propose a generic storage method called *intelligent BLOB* (*iBLOB*), which is a structured binary array built on BLOBs but especially maintains the hierarchical information. It is "intelligent" because, unlike BLOBs, it understands the structure of the application object stored and supports fast access, insertion, and update to components at any level in the object hierarchy.

TSS and iBLOBs together enable an easy implementation of complex ADTs, including the spatiotemporal data types for *moving points*, *moving lines*, and *moving regions*. They bring back the focus on understanding the semantics of such objects and on performing scientific analysis. This is the case because type system implementers are released from the task of interpreting the logical semantics of binary unstructured data, and the component level access is natively supported by the underlying iBLOB.

### C. Moving Objects Databases

The idea behind *moving objects databases* (*MOD*) [2], [3], [4], [5], [6], [7] is to provide a system capable of representing moving entities in DBMSs and being able to ask queries about them. Moving entities could be *moving points* such as people, animals, and all kinds of vehicles like cars, trucks, air planes, ships, where usually only the time-dependent position in space is relevant but not the spatial extent. Moving entities with a spatial extent, for example, hurricanes, fires, oil spills, epidemic diseases, could be characterized as *moving regions*. Similarly, the moving front of a wild-fire or an army can be described by a *moving line*. Such entities with a continuous, spatiotemporal variation (in position, extent, and/or shape) are called *moving objects*. Since databases offer many advanced features such as transaction control, security and backup, and an SQL query interface for data management and retrieval, the integration of moving objects into databases can help leverage existing DBMS technology to enable flexible querying of complex moving objects.

## III. Objectives

We discuss the objectives of our project from an application standpoint in Section III-A and from a system standpoint in Section III-B.

### A. Application Objectives

The primary application objective of our technology is to support accurate and efficient ad-hoc query and retrieval of Earth science satellite sensor data for dynamic atmospheric events such as tropical cyclones. This is relevant to NASA missions and supports existing NASA Earth Observing missions such as AIRS, Calipso, CERES, Cloudsat, MODIS, QuikSCAT, and TRMM that measure dynamic processes. To support our objective, we have focused on the following aspects of research and system development.

1) Providing tools to facilitate efficient collection, storage, and management of "best track" and satellite data about tropical storm events. Such data sets range from a few megabytes to many terabytes, and can be in simple text files (flat files) or encoded into complex scientific formats such as HDF, NetCDF etc. as described in Section II-A.
2) Providing tools for ad-hoc dynamic event query and data retrieval. Such tools will allow scientists to perform flexible query and retrieval of satellite data for statistical analysis.
3) Supporting scientific analysis of retrieved satellite measurements. Novel statistical composite analysis tools will be developed and incorporated along with existing technology to assist scientists in justifying hypotheses and to facilitate knowledge discovery and data mining.

One of the design goals of our technology is to make it generic so that the solution can be easily incorporated into existing information gathering and analysis systems as well as systems supporting future missions. The core of the moving objects infrastructure we are developing is mission-independent and written at several levels of abstraction for

different levels of users, from an analyst view to a system developer view. Overall, this will help reduce implementation costs related to future missions while providing an efficient and uniform basis for data collection, integration, querying and complex analysis. In Section IV we describe our approach to achieve this objective.

### B. System Objectives

The system objectives describe the goals of the project from a system and software architecture standpoint and include three components: (i) the Moving Objects Database (MOD), (ii) the Moving Objects Software Library (MOSL), and (iii) the Spatiotemporal Query Language (STQL).

From an application perspective, the *moving objects database* (*MOD*) that we are going to create is supposed to keep tropical cyclone and hurricane data provided by public sources and web sites in a centralized repository and make them available for querying to decision makers and application scientists. From a system perspective, it is a full-fledged database with additional support for spatial and spatiotemporal data in its data model and query language. In order to be able to add new functionality, we have the important requirement that a database system (DBMS) must be *extensible*.

From an application perspective, the *Moving Objects Software Library* (*MOSL*) provides the functionality in terms of spatiotemporal data types, operations, and predicates that can be deployed by decision makers and scientists in ad hoc queries and in database application programs (written in C++, Java, Matlab, etc.) to retrieve and derive tropical cyclone data. For that purpose, from a system perspective, MOSL provides a system of *spatiotemporal data types* together with a large number of *spatiotemporal operations* (e.g., *Intersection*, *Union*, *Difference*) and *spatiotemporal predicates* (e.g., *Inside*, *Meet*, *Disjoint*, *Overlaps*; *Enters*, *Leaves*, *Crosses*, *Bypasses*). A detailed description of the moving objects database technology can be found in [5]. It leads to many further scientific publications on this topic. In particular, MOSL provides *historical* spatiotemporal data types named *hmpoint*, *hmline*, and *hmregion* which can be integrated as attribute data types into extensible databases in a database-independent and application-neutral manner.

From an application perspective, the *Spatiotemporal Query Language* (*STQL*) provides the communication interface between the moving objects database for tropical cyclone data and the decision maker and/or scientist. This textual language enables users to pose ad hoc spatiotemporal queries on moving objects in general and on tropical cyclone data in particular and to obtain immediate response. Further, it is supposed to retrieve satellite data based on user queries.

All three components are to be integrated into extensible database systems. This leads to the planned system architecture shown in Figure 2. The components in red indicate the new components and their embedding into available DBMS.

## IV. System Design and Integration

In this section, we elaborate our system design and integration from two perspectives, the *data integration* and the *system architecture*. Section IV-A introduces the steps that are taken for integrating tropical cyclone data into our moving object databases (MOD). Section IV-B presents our system architecture in detail.

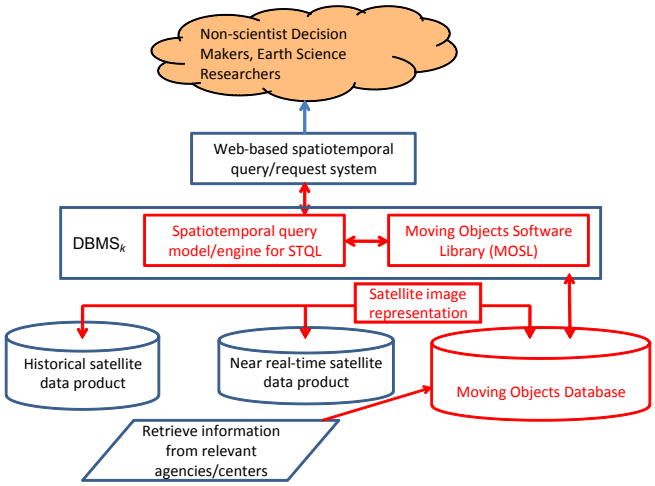


Fig. 2. The planned system architecture.

### A. System Components for Integrating Tropical Cyclone Data into the Moving Objects Database

The original hurricane data from varies sources are stored in plain text or complex file formats such as HDF. However, such kind of storage is neither effective nor efficient for users to perform queries on the movement of hurricanes. For example, the original QuickSCAT data is stored in tens of thousands of HDF files which are differentiated by time range, i.e., each HDF file stores the scanning result of a satellite in about two hours. Hence, it is only possible to perform queries within one HDF file each time by a particular application program. It would be much more beneficial to extract the data from the HDF files and populate it into a database. A query like "Find all time intervals between date 1 and date 2 when the wind speed exceeds 50 m/s at the location $(x, y)$" cannot be answered efficiently. Instead, users will need to find all HDF files between date 1 and date 2, and extract the data from all the files, filter the data by the condition of the query in each file, and union all results in the end. Therefore, an alternative approach is needed, which will enable the users to perform such query only once and get the correct result they want immediately. Our approach extracts the hurricane data from the original data sources and store them in the moving objects database. The users are then able to query hurricane data in the manner of spatiotemporal data types for moving objects. We call this task *data integration*.

There are three steps in the data integration process, as illustrated in Figure 3. In the first step, we extract the original data from the five different hurricane data sources, and store them into a *relational database*. We do not filter any data but store exactly what exists in the original hurricane data files. However, if a file data format is hierarchical like HDF, we have to flatten data hierarchies before we store the data

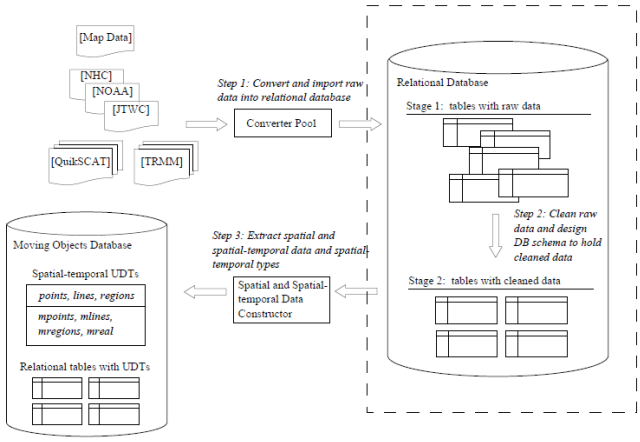


Fig. 3. An illustration of the various steps involved in hurricane data integration and MOD development

into a relational table. For each data source, we create a large table to hold all fields. In the second step, we perform a *cleaning* process. We set up data dependencies, carefully design the database schema by performing a normalization step to hold the cleaned data, filter unrelated data, and remove redundant data. After the second step, the data is still stored in a relational database but the original database size is reduced. In the third step, we convert the cleaned spatiotemporal data into moving objects of the spatiotemporal data types and store them into the moving objects database. We provide spatial and spatiotemporal data constructors. The moving objects database supports spatial data types such as *point*, *line*, and *region* as well as spatiotemporal data types such as *hmpoint*, *hmline*, and *hmregion* for historical moving points, historical moving lines, and historical moving regions. In addition, we provide a data type *hmreal* for historical moving real data. For example, the eye (trajectory) of a hurricane can be stored as a value of the type *hmpoint*, and the time-varying wind speed of a hurricane can be stored as a value of the type *hmreal*.

There are some challenges in the data integration process. The first challenge is the complexity of the HDF files. The original QuickSCAT and TRMM data are stored in HDF format, which cannot be extracted directly. We solve this problem with the help of the Java API "hdf-java" provided by the HDF group. The hdf-java software package helps us read the contents of the HDF files. We store the data into intermediate text files and make the Oracle loader to load the text files into tables. The second challenge is how to query the large satellite data tables. Since each of the satellite tables occupies several terabytes, we must find an efficient way to query such big amounts of data. For this purpose, we build spatial and temporal indexes on the tables. We also create partitions of tables and organize them by time range so that a query on a specific time instance or period is performed on a small number of partitions only, and time range queries extending over different partitions can be performed in parallel.

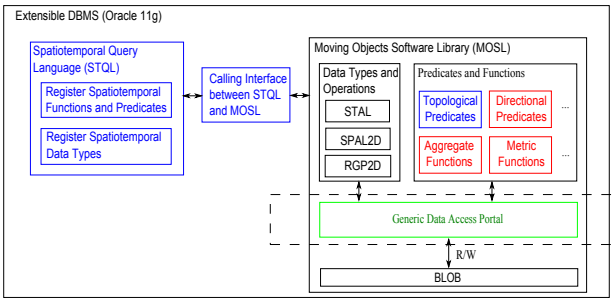


Fig. 4. A more fine-grained architecture for MOSL.

### *B. System Architecture for Storing and Retrieving Complex Structured Application Objects in Moving Objects Databases*

The three major components in our system are the *Moving Objects Database* (*MOD*), the *Moving Objects Software Library* (*MOSL*), and the *Spatiotemporal Query Language* (*STQL*). In Section IV-A we have introduced the system component MOD and described the steps to integrate various data sources into the MOD. In this section, we introduce in detail the other two components. The overall system architecture is presented in Figure 4, which also shows the connections between different components.

The basis of our system is an extensible DBMS with an interface for new *user defined data types* (*UDTs*) and *user defined functions* (*UDFs*), and a storage type (BLOB) that is suitable for holding large application objects with user defined complex structures. We have explored the commercial and public domain databases and decided to take Oracle 11g as the underlying extensible DBMS for the implementation of our system.

The first step is to include the Moving Objects Software Library into Oracle. The implementation of the MOSL can be further divided into three components, which are the *type system implementation*, the *generic data access portal*, and the BLOB access interface. Figure 4 presents the three major components of the MOSL.

The *type system implementation* component refers to the implementation of spatial and spatiotemporal data types like regions and moving points, and the necessary operations for manipulating the data stored. Major packages that implement the data types and operations are the *Spatiotemporal Algebra* (*STAL*) package, the *Spatial Algebra 2D* (*SPAL2D*) package, and the *Robust Geometric Primitives 2D* (*RGP2D*) package. In addition to the data type packages, we also include packages that contain the implementations of spatial and spatiotemporal predicates and functions into the MOSL. The predicates and functions that are implemented in our system are *topological predicates* like *is_cross* and *exist_prior*, *directional predicates* like *exist_northeast* and *exist_west*, *metric functions* like *getLifeTime*, *aggregate functions* like *avgWindSpeed*, and other functions like *atInstance*.

The type system implementation is a rather high level implementation whose focus is on the design of internal data structures of application objects and on the design of

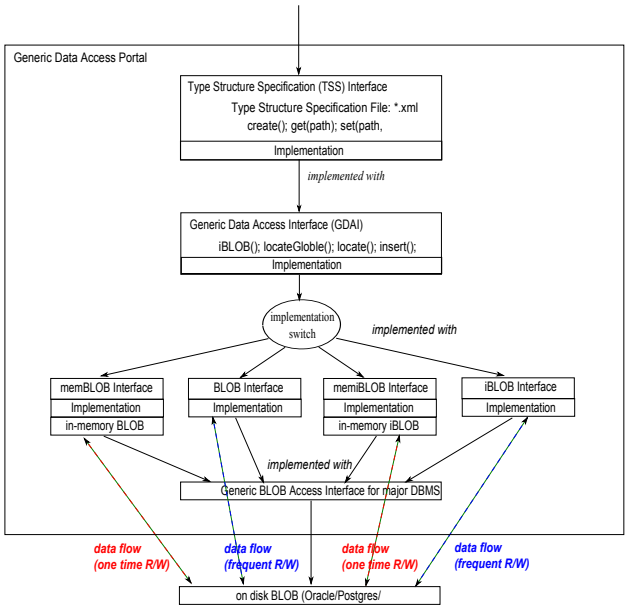


Fig. 5. Detailed flow chart of the implementation of the generic data access portal.

efficient algorithms for operations. On the other hand, the actual data of different types have to be stored in BLOBs as the only DBMS data type for storing large objects of varying representation length. They are accessed through the functions provided by the *BLOB access interface*. The *BLOB access interface* supports byte level data access and does not recognize the objects stored. Thus, it is a rather low level implementation. In order to free the high level type system implementation from the direct low level byte manipulation, we introduce an additional component called the *Generic Data Access Portal* (component in the dashed box of Figure 4) that lies between the type system implementation and the BLOB access interface and abstracts from low level details in BLOBs.

A detailed implementation flow chart of the generic data access portal is shown in Figure 5. The generic data access portal can be divided into three components with decreasing closeness to the type system: the *Type Structure Specification Interface* (*TSS*), *the Generic Data Access Interface* (*GDAI*), and the various *implementations of a structured BLOB*. We take a bottom up fashion to explain these components in detail.

From an implementation perspective, the direct interaction with a BLOB involves byte level manipulation and is a rather tedious task. Further, complex objects like regions and moving points are highly structured objects, and the implementation of functions and operations usually takes advantage of such structures. Thus, there is a gap between the high-level structured objects and the low-level unstructured BLOBs. In order to bridge this gap, we have designed a data structure called the *Intelligent Binary Large Object* (*iBLOB*). In general, the iBLOB is a structured BLOB that provides higher level data access methods (compared to BLOBs) that allow users to retrieve, insert, and update objects at the component level. For example, instead of manually locating the starting byte of a segment that belongs to a region, and instead of reading the segment byte by byte, one can load the segment directly from the region by using an iBLOB. We implement different versions of an iBLOB due to efficiency and scalability issues. For applications that involve only small size objects that fit into main memory, a main memory iBLOB version works best in terms of efficiency. However, for applications that involve large size objects that do not fit into main memory, loading the entire object into main memory fails. Hence, functionalities that can load only needed components into main memory are required. As a result, a scalable on-disk iBLOB works best. To handle all different implementations of the iBLOB and to provide users a generic interface for accessing them, the *the Generic Data Access Interface* (*GDAI*) is added for this purpose. However, the GDAI is still closer to the underlying storage system, the iBLOB, and it provides the *same* interface for the implementation of *different* type systems. Thus, the type system implementor might still find it not intuitive to use. This is where the *Type Structure Specification Interface* (*TSS*) comes into play.

The *Type Structure Specification Interface* (*TSS*) is closest to the type system, and it provides the interface for the implementation of any type system. The type system implementor can use a simple grammar provided by TSS to describe the designed representation structure of a data type, and the TSS translates the description and makes calls to the underlying GDAI functions to properly organize the storage of the data. Therefore, it is like a wrapper around the GDAI that provides functions that are more meaningful to the type system implementor.

Beside the MOSL, another important component is the *Spatiotemporal Query Language* (*STQL*) that provides an SQL like query language for spatial and spatiotemporal data. In this component, we register data types and the functions to the DBMS and create links between the MOSL and the registered types and functions. The implementation of the STQL is very specific to the underlying DBMS because different DBMSs have different mechanisms for registering user defined types and user defined functions.

## V. An Application driven by Hurricane Database Query Output

A major strength of our moving objects database infrastructure is that it is an extension to existing, well-established database systems (e.g. Oracle, PostgreSQL). Hence, ODBC (Open Database Connectivity API) or JDBC (Java Database Connectivity API) drivers can be easily used to connect and communicate with our hurricane database server.

In this section, we demonstrate a Matlab implementation of a satellite data retrieval and subseting application driven by query output from our hurricane database (see Figure 6) focusing on the database connection and query. The data retrieval and subseting functionalities are discussed in [8], and the detailed algorithms are described in [9].

The Matlab-based satellite data retrieval component is connected to the hurricane database using the command

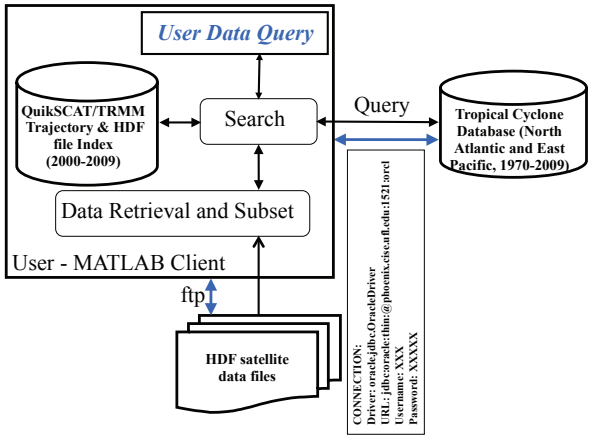


Fig. 6. Satellite data retrieval and subseting software driven by Hurricane database query output.

| Year | Hurricane | Number of QuikSCAT snapshots | Search Index Time (sec) | Retrieval and Subset Time (sec) |
|---|---|---|---|---|
| 2000 | Gordon | 12 | 46.60 | 121.37 |
| 2001 | Allison | 13 | 73.50 | 155.06 |
| 2003 | Isabel | 21 | 93.70 | 227.21 |
| 2004 | Charley | 10 | 38.52 | 80.31 |
| 2004 | Gaston | 13 | 45.45 | 135.24 |
| 2004 | Ivan | 31 | 131.76 | 341.09 |
| 2004 | Jeanne | 24 | 109.57 | 278.92 |
| 2005 | Cindy | 11 | 46.74 | 107.65 |
| 2006 | Ernesto | 18 | 70.50 | 159.22 |
| 2008 | Hanna | 23 | 81.44 | 196.61 |

Fig. 7. Performance for multiple tropical cyclone QuikSCAT data retrieval driven by the STQL query "Find all tropical cyclones that crossed Maryland State between 2000 and 2009".

```
conn = database('Database',
                'username',
                'password',
                'oracle.jdbc.OracleDriver',
                'jdbc:oracle:thin:
                 @phoenix.cise.ufl.edu:
                 1521:orcl')
```

from the Matlab Database Toolbox.

After the software is successfully connected to the database, we execute the STQL query for the database connection *conn* with the query output stored in the cell array *cur*.

```
cur = exec(conn,
['SELECT m1.storm_name
  FROM   tropical_cyclone m1,
         tropical_cyclone m2
  WHERE  exist_overlap(m1.track, m2.track) = 1
         AND m2.name = NATE']);
```

In the above STQL query, all tropical cyclones that occurred during the life time (computed by the spatiotemporal predicate "exist_overlap") of Hurricane Nate in the North Atlantic Ocean in 2005 will be stored in the cell array *cur*. The trajectories corresponding to the tropical cyclone names stored in *cur* will be used by the software to retrieve satellite data.

Another more sophisticated STQL query is as follows.

```
cur = exec(conn,
['SELECT m.storm_name
  FROM   STBL_USMAP r,
         MTBL_NHC_BESTTRACKS m
  WHERE  r.state_name= Maryland AND
         is_cross(m.trajectory,r.geoshape) = 1
         AND m.start_date >=
             to_date(01-JAN-2000)']);
```

In this query, all tropical cyclones that crossed (computed by the spatial predicate "is_cross") Maryland State between 1 January 2000 and 31 December 2009 (currently, we have tropical cyclone data from 1 January 1970 to 31 December 2009) is stored in *cur*.

Using the satellite data retrieval and subseting algorithms described in [9], we performed a *QuikSCAT L2B* satellite data retrieval such that the returned data is a collection of circular regions defined by *1* degree radius from the eyes of the tropical cyclones returned by the above query. The query output list consists of 10 hurricanes/tropical cyclones between 2000 and 2009, and 175 QuikSCAT snapshots are retrieved. A breakdown of the index search and retrieval computational time for each output hurricane is shown in Figure 7. Since the number of QuikSCAT snapshots is proportional to the hurricane lifespan, the search and data retrieval time is proportional to the hurricane lifetime. It takes slightly more than 30 minutes to complete this user data request that includes (FTP) retrieving and uncompressing QuikSCAT data from the PODACC data center.

## VI. Conclusions and Future Work

In this paper, we describe the data integration and the system architecture of a moving objects database for ad-hoc querying and retrieval of atmospheric events and their associated satellite measurements. Future work includes additional functionalities for all system components and the continuation of the development of the software library MOSL as well as the design of the query language STQL. Further objectives are the optimization of the storage and indexing for satellite data, the exploration of additional scientific analysis operations, and the implementation of functionalities to support spatial queries based on both the satellite measurements and the hurricane data.

## VII. Acknowledgments

This work was carried out at the University of Florida, Gainesville, and at the University of Maryland, College Park. It was funded by the National Aeronautics and Space Administration (NASA) Advanced Information Systems Technology (AIST) Program under grant number AIST-08-0081.

## REFERENCES


[1] T. Chen, A. Khan, M. Schneider, and G. Viswanathan, "iBLOB: Complex Object Management in Databases through Intelligent Binary Large Objects," in *3rd Int. Conf. on Objects and Databases*, 2010, pp. 85–99.

[2] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis, "Spatio-temporal Data Types: An Approach To Modeling and Querying Moving Objects in Databases," *GeoInformatica*, vol. 3, no. 3, pp. 269–296, 1999.

[3] M. Erwig and M. Schneider, "Spatio-temporal Predicates," *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, vol. 14, no. 4, pp. 881–901, 2002.

[4] L. Forlizzi, R. Güting, E. Nardelli, and M. Schneider, "A Data Model and Data Structures for Moving Objects Databases," in *ACM SIGMOD Int. Conf. on Management of Data*, 2000, pp. 319–330.

[5] R. Güting and M. Schneider, *Moving Objects Databases*. Morgan Kaufmann Publishers, 2005.

[6] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. J. N. A. Lorentzos, M. Schneider, and M. Vazirgiannis, "A Foundation for Representing and Querying Moving Objects," *ACM Trans. on Database Systems (TODS)*, vol. 25, no. 1, pp. 1–42, 2000.

[7] J. A. C. Lema, L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider, "Algorithms for Moving Objects Databases," *Computer Journal*, vol. 46, no. 6, pp. 680–712, 2003.

[8] M. Schneider, S.-S. Ho, T. Chen, A. Khan, G. Viswanathan, W. Tang, and W. T. Liu, "Moving Objects Database Technology for Ad-Hoc Querying and Satellite Data Retrieval of Dynamic Atmospheric Events," in *Earth Science Technology Forum*, 2010.

[9] S.-S. Ho, W. Tang, W. T. Liu, and M. Schneider, "A Framework for Moving Sensor Data Query and Retrieval of Dynamic Atmospheric Events," in *22nd Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, ser. Lecture Notes in Computer Science, vol. 6187. Springer, 2010, pp. 96–113.