7-25-2002

A real-time distributed analysis automation for hurricane surface wind observations

Sonia Otero
Florida International University

DOI: 10.25148/etd.FI15103037

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A REAL-TIME DISTRIBUTED ANALYSIS AUTOMATION FOR HURRICANE SURFACE WIND OBSERVATIONS

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Sonia Otero

2002

To: Dean Arthur W. Herriott
College of Arts and Sciences

This thesis, written by Sonia Otero, and entitled A Real-Time Distributed Analysis Automation For Hurricane Surface Wind Observations, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved

Yi Deng

Mark Powell

Raimund Ege, Major Professor

Date of Defense: July 25, 2002

The thesis of Sonia Otero is approved.

Dean Arthur W. Herriott
College of Arts and Sciences

Dean Douglas Wartzok
University Graduate School

Florida International University, 2002

DEDICATION

I jointly dedicate this thesis to the people who benefit (scientifically or not) from the use of this application, and to my sister, Dania, for just being there.

ACKNOWLEDGMENTS

I wish to thank the stimulating instruction of Dr. Raimund Ege and Dr. Yi Deng on their respective courses, which shaped my inclination within software engineering. Dr. Mark Powell and scientists from the Hurricane Research Division (too many to mention), who have devoted so much effort to this analysis model, deserve a most special recognition. This thesis hopes to bring prominence to their arduous work.

The dedication of my team colleagues, Nicholas Carrasco, Nirva Morisseau-Leroy, George Soukup, and Russell St. Fleur, was invaluable to achieve such degree of overall accomplishment. I would like to express my gratitude to Luis Amat, for being my study partner throughout the coursework and providing a well-oriented foundation for this project.

Finally, I cannot forget those friends who helped me maintain my mental and physical endurance during this lengthy but rewarding experience.

ABSTRACT OF THE THESIS

A REAL-TIME DISTRIBUTED ANALYSIS AUTOMATION FOR HURRICANE SURFACE WIND OBSERVATIONS

by

Sonia Otero

Florida International University, 2002

Miami, Florida

Professor Raimund Ege, Major Professor

From 1993 until 1999, the Hurricane Research Division of the National Oceanic and Atmospheric Administration (NOAA) produced real-time analyses of surface wind observations to help determine a storm's wind intensity and extent. Limitations of the real-time analysis system included platform and filesystem dependency, lacking data integrity and feasibility for Internet deployment.

In 2000, a new system was developed, built upon a Java prototype of a quality control graphical client interface for wind observations and an object-relational database. The objective was to integrate them in a distributed object approach with the legacy code responsible for the actual real-time wind analysis and image product generation. Common Object Request Broker Architecture (CORBA) was evaluated, but Java Remote Method Invocation (RMI) offered important advantages in terms of reuse and deployment. Even more substantial, though, were the efforts towards object-oriented re-design, implementation and testing of the quality control interface and its database performance interaction.

As a result, a full-featured application can now be launched from the Web, potentially accessible by tropical cyclone forecast and warning centers worldwide.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

1. INTRODUCTION

This thesis proposes the design and implementation of a distributed object application for the real-time analysis of quality controlled tropical storm surface wind observations. The analysis application will be integrated with a World-Wide Web and database based application that handles the quality control of those wind observations.

1.1 Mission of the application

Based on Powell, et. al. [7], analyses are produced by compositing all available observations relative to the storm center that is studied. Observations include Air Force and NOAA aircrafts, ships, buoys, Coastal Marine Automated Network (C-MAN) platforms and surface airways (airports). First, the data are quality controlled, and then processed to conform to a common framework for height (10 m), exposure (marine or open terrain over land), and averaging period (maximum sustained 1 minute wind speed). It takes several hours of collected observations to provide sufficient data to produce an objective analysis, which represents the mean state of the storm during the chosen time period. A typical 10-hour reconnaissance mission will yield two to three analyses. The primary product of each analysis is a streamline and isotach contour plot, designed to convey the location and strength of the maximum wind as well as the extent of hurricane force winds. Naturally, the analysis results help meteorologists determine the storm's most recent measured intensity and the extent of its damaging winds, which can, in turn, help them in issuing storm forecasts and warnings. Timely analysis results combined with geographic information on the area affected by a hurricane or tropical storm can help identify which locations suffer the most intense winds and severe storm surge. Early data acquisition should help emergency managers to better organize search, rescue and recovery operations shortly after the disaster has taken place. Given the importance of this information, some commercial and scientific communities have also expressed interest in accessing hurricane wind field data in a graphical or flat file format.

1.2 Origins of the application

The Hurricane Research Division (HRD), located on Key Biscayne, under the National Oceanic and Atmospheric Administration (NOAA), has been conducting real-time analyses of tropical storm surface wind observations since 1993 [5], on an experimental basis. The overall application that comprised a workstation-based quality control, a partially automated analysis process, and a graphical output was named WANDA (Wind Analysis Distributed Application). From a general perspective, the application's operation started with the fetching of data from a flat-file repository. FTP scripts regularly updated this repository to download near real-time data from the National Center for Environmental Prediction (NCEP) via the National Hurricane Center (NHC), located at the Florida International University campus. Secondly, the data were processed and quality controlled via a basic graphical user interface tightly tied to the format of those flat files. Then, the reviewed data were sent to the analysis server, and finally, the output was displayed and converted to a format that could be faxed or hard-copied to clients, such as NHC's hurricane specialists. WANDA (fig. 1) was logically divided in three independent subsystems: 1) Quality Control, 2) Analysis Automation, and 3) Output Generation. One could look at them as highly cohesive decoupled components.

1) Quality Control

This was the graphical user interface to WANDA, which resided on a workstation. Through a sequence of windows, the user selected a set of observations and an associated storm track, that were then displayed according to the geography of the selected storm. Several inspection tools were provided to the user to decide the validity of the data and thus make the final selection of a satisfactory set that should be analyzed.

2) Analysis Automation

The quality-controlled data and a storm track were passed through a series of Analysis subsystem components. Each component was distributed over two machines, a NeXTSTEP client and a VAX/VMS server containing the legacy analysis programs. WANDA used state machines to orchestrate all of the state transitions involved in the analysis automation. Included

in this automation was the automatic archival of all steps of an analysis for future research purposes. Any analysis could be traced back to its components results and data sets. The state automata approach ensured that the execution of an analysis component could only start if no errors had been encountered previously, which were reported accordingly.

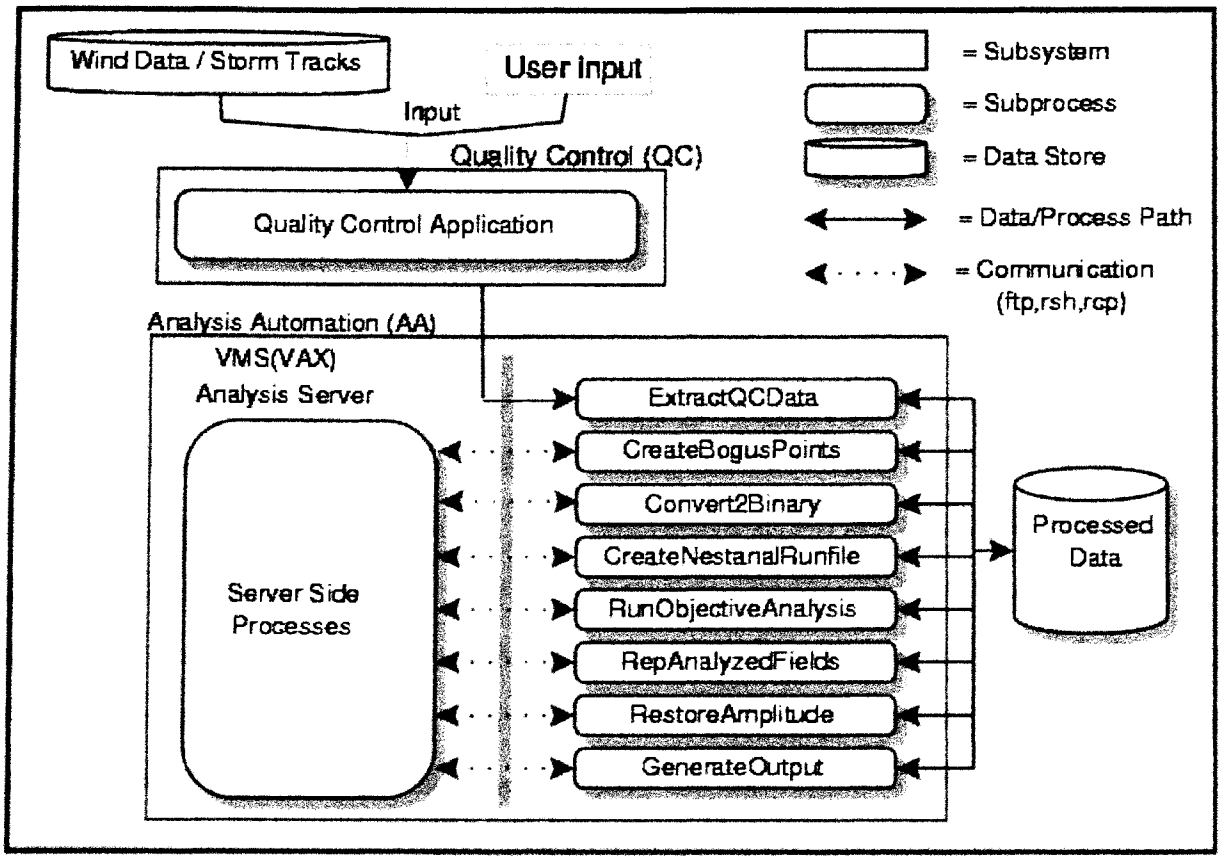


Fig. 1 Process and data flow diagram of the old WANDA

3) Output Generation

The Output Generation subsystem created a graphical representation of the wind fields. The implementation of this subsystem was done with an in-house graphics package that displayed an analysis product on the client workstation where the user could annotate and save it to an encapsulated postscript document.

1.3 Why was the old system obsolete?

One of the major drawbacks was the lack of a database common and accessible by all the subsystems. The use of a hierarchy of flat files made the application platform specific plus inhibited flexibility for manipulating data retrieval. Also, data integrity and security issues were raised due to the use of ftp, rsh and rcp scripts. The lack of portability was another problem, exacerbated by the fact NeXTSTEP and VAX/VMS operating systems are no longer supported. Regardless of the platform, however, one of the goals for WANDA was to become truly distributed and to be used on a continuous basis both at HRD and at NHC. Because HRD is located in Key Biscayne, a hurricane warning for Dade County would mean that the facility housing WANDA's primary resources would have to be shutdown. There is a need for maintaining redundant analysis and database servers at NHC as well, capable of performing the same exact asks described earlier. Ideally, both analysis and database servers would be running simultaneously for maximum capacity. Therefore, load balancing and concurrency factors need to be taken into consideration. The ability to reuse portable code is crucial in order to keep this redundancy as transparent as possible. I will attempt to describe the proposed WANDA's replacement, now named H*Wind, which was completely reconstructed, and was first released during hurricane season 2000, pressed partly by the fact that WANDA was also not Y2K-compliant.

H*Wind is made up by three main areas of development, aimed to improve the drawbacks listed above. They all have in common the exploitation of the Object Oriented paradigm:

- Incorporation of a database management system. This will greatly ease the retrieval and storage of all the data involved throughout all the phases of the application. H*Wind's database design was the topic of a Master's Thesis at Florida International University [10], whose author has remained H*Wind's database developer during this endeavor.
- The use of the platform independent object oriented programming language (OOPL), Java, for the Quality Control subsystem, in order to comply with the World Wide Web requirement. The

discussion about H*Wind's Quality Control subsystem is the subject of a Master's Thesis at Florida International University [11].

- The use of Distributed Objects (DO) technology for the Analysis Automation subsystem, for which this thesis will be mainly responsible. The source code that involves all the steps of an analysis run is already written and is maintained in FORTRAN 77 by expert meteorologists. It is a task that will remain managed by them, until or if a decision is reached to port the code to OOPL. As a result of this situation, the FORTRAN code needs to be wrapped in an OOPL (Java, to homogenize with the rest of the project as much as possible) that not only allows database connectivity, but more importantly, transforms each analysis step into an object that can then be used as a distributed object. The implementation will undoubtedly include the use of a CORBA-compliant (Common Object Request Broker Architecture) IIOP (Internet Inter-Orb Protocol) Object Request Broker (ORB), which is the Object Management Group's (OMG) well-established and widely adopted standard for object interoperability and communication. In addition, given the extended use of Java throughout both the Quality Control and the Analysis Automation subsystems, it would be perfectly sound to implement a version that uses Java's own distributed programming model, Remote Method Invocation (RMI), as it is recommended for Java-to-Java interprocess communication.

The FORTRAN programmers/meteorologists have already adapted the VAX/VMS code for UNIX execution. Analysis Automation is inherently constrained by its FORTRAN implementation, but once it becomes part of the distributed object infrastructure, through an Interface Definition Language (IDL) declaration, it can be freely invoked completely independent of the programming language, operating system or network to which it is tightly bound. This is especially important for HRD to achieve true application distribution. CORBA's features provide many DO advantages: transaction control, concurrency, and event notification. These characteristics take advantage of some well-known properties of OO, like encapsulation, inheritance and polymorphism.

Besides the development of the DO section of H*Wind, additional tasks involved completion of the analysis of the database schema for the archival of the Analysis Automation results, plus the

later connectivity to the database server. Also, a client GUI was provided for the acquisition of the Analysis parameters needed to run an analysis process. Simultaneously, the distributed Analysis Automation subsystem was integrated with H*Wind's Quality Control application and Product Generation processes, to eventually form an architecture as shown in figure 2.

This project used the iterative and incremental software engineering methodology. The plan was to constantly revise with the user if the requirements were being met after a certain amount of progress. H*Wind's project team is small and the contact with relevant key users is close and frequent, which allowed for the iterative development approach to work [12]. Upon completion of this thesis, the OO Analysis (OOA) and OO Design (OOD) documents will be submitted, along with sequence diagrams, activity diagrams and user scenarios. The UML notation is followed.

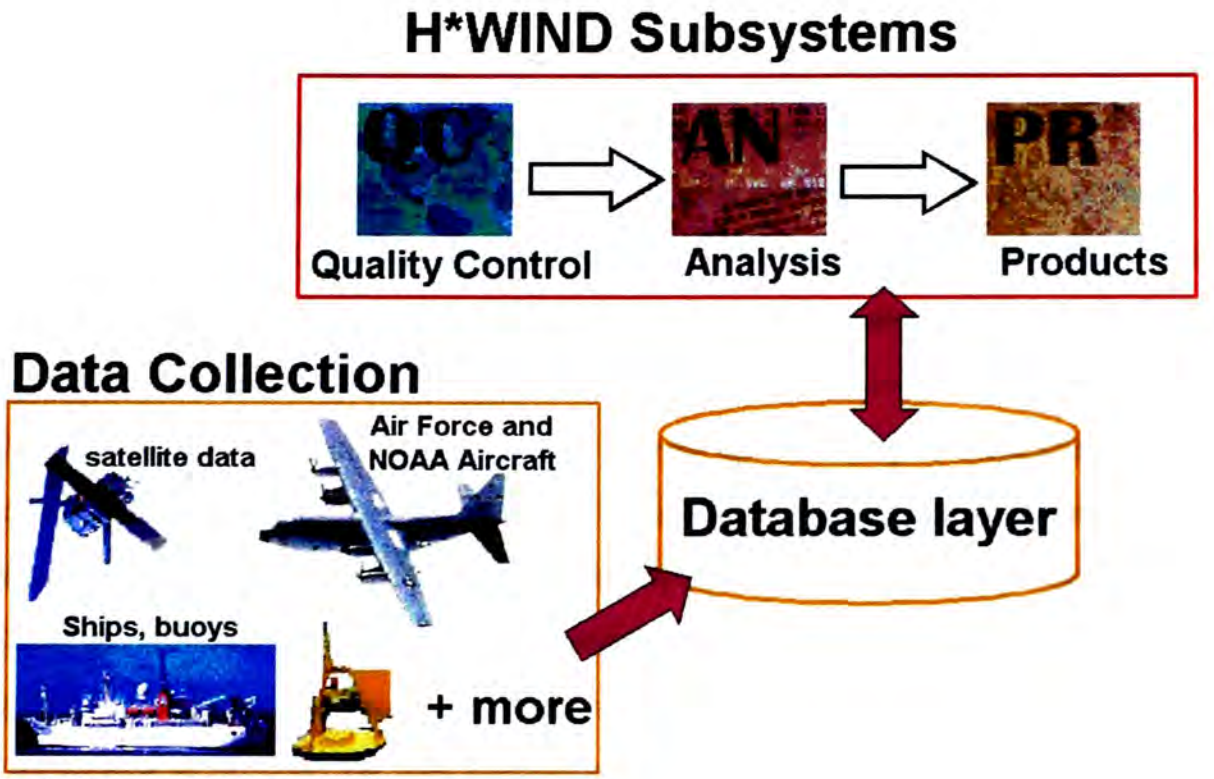


Fig. 2 Generalized view of H*Wind

2. DESCRIPTION OF THE HRD SPLINE ANALYSIS (HSA)

The analysis algorithm, cornerstone of this project, consists of a process of estimating the continuous spatial field of a physical variable from a set of discrete observational data [3]. For our purposes, the physical variables of concern are wind, pressure (or geopontential height above surface), temperature and relative humidity. In the ideal case, in which the domain of interest is densely covered by data of reasonable accuracy, all that is required may be mechanical interpolation of discrete data with some smoothing. However, for most meteorological observations, the data are collected by less-than-ideal number of irregularly placed stations; that is, an undersampled dataset. This analysis relies heavily on the judgements by a knowledgeable and experienced human analyst in order to ensure the resulting continuous field will be a reasonable approximation of the true data.

The analysis numerical method is based upon the Spectral Application of Finite Element Representation (SAFER) method, as explained in [2], [3] and [4]. In particular, a nested version of SAFER that allows the specification of different filters for each nest that depend on the scale of the features to be resolved; more dense sampling is needed in disturbed regions where atmospheric variability tends to be on smaller scales. Inner meshes of the nest focus on the wind structure of the eyewall including magnitude of highest wind and its distance from the center, while outer meshes cover the extent of hurricane and gale force winds. The whole area of interest is referred as the analysis domain.

There is always an elapsed time between the moment data are being collected and the actual moment for which the analysis is performed. Even though meteorological conditions certainly change, it can be assumed that the features close to the storm center move with the hurricane. Therefore, to correct this space-time discrepancy, all observations are placed in a coordinate system that moves with the specified hurricane center, i.e., a storm-relative coordinate system.

To produce an analysis, the user specifies such basic parameters as the analysis domain, weights for each data source, pressure level, boundary conditions and filter wavelengths that govern the resolvable scales for the analyzed fields. The HSA operates only on surfaces of

constant pressure; specifically, it has 19 predefined levels (surface level plus 100 through 950mb in 50mb intervals). Nevertheless, there are many other modifiable parameters available to maximize flexibility given the subjectivity of the process, determined by observation density and overall meteorological conditions near and outside the domain boundaries. Several HRD scientists combined efforts to code this whole algorithm in FORTRAN, divided in five distinct programs:

1) Creation of bogus points - designed to minimize the ill effects of poor data distribution in the near-storm environment, by moving the area of maximum convergence closer to the storm center.

2) Dcopy - Conversion of input ASCII observations to an unformatted record.

3) Prenest - Prepares the input parameter files for Nestanal (next step).

4) Nestanal - Performs the nested objective analysis. The essential output is a 'KR' file containing the analyzed fields' spline coefficients.

5) Krdecode - Reads and processes a 'KR' file to extract the information necessary to produce plots.

Starting with the development of this thesis, important improvements have been externally attached to the analysis algorithm package via the output generation scripts and via real-time delivery to the Automated Tropical Cyclone Forecasting System (known as ATCF). Concretely, the output generation scripts (coded by an HRD mathematician) can now perform an enhancement on the 'KR' output file by more accurately exhibiting the actual storm's wind distribution relative to the real observed maximum wind and to the scientist's chosen storm center position, instead of relative to the analysis-determined maximum wind and storm center position. It reflects the scientist's choices. If no enhancement is performed, the analysis smoothing process has shown to slightly underestimate the higher wind speeds. For compounded benefit, the old black and white images still being produced have encountered the "competition" of several new color images at different zoom scales that better depict the situation in question (Fig.

3). These images also provide something very useful for the forecasters' assessment: the wind radii per quadrant, meaning the radius in nautical miles of the extent of 34-, 50- and 64-knot surface wind speed in the northwest, northeast, southeast and southwest quadrants with respect to the storm center. In the past, forecasters at the National Hurricane Center subjectively determined these measurements from looking directly at computer data images, not a very reliable or effective procedure in a deadline-constrained environment.

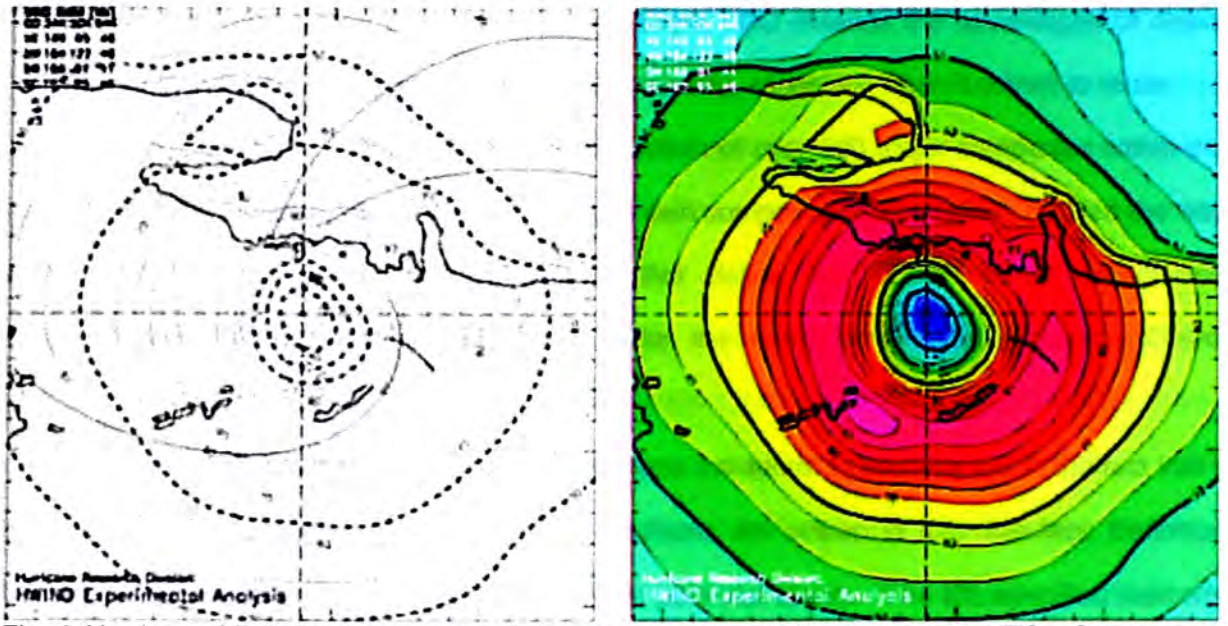


Fig. 3 Hurricane Michelle approaching Cuba on November 4, 2001 at 19:30 UTC. Color output more clearly conveys information.

Joining the ATCF system makes analysis results quickly available in a common format to major National Weather Service agencies and to Department of Defense tropical cyclone warning centers. The Automated Tropical Cyclone Forecasting System was developed by the Naval Research Laboratory (NRL) in 1998, designed to provide an organized framework of tools for the forecasting process by featuring global tracking capability, construction of messages, and dissemination of warnings [9]. For H*Wind, it serves a dual purpose: it provides a way to obtain tropical cyclone fix positions every six hours (which we timely ingest in our database) for as long an event is deemed relevant, and it provides a way of delivering our product: an objective, observation-based aid, which supplies values of intensity and wind radii per quadrant.

3. DISTRIBUTED OBJECTS TECHNOLOGY (DO)

Described as the ultimate client/server architecture, DO is oblivious to internal implementation details, address space, network distance, vendors, operating systems, and communication protocols, yet it enables object invocation with the transparency of a local access. This technology easily leads to the construction of autonomous loosely-bound components by encouraging the separation of user interface, process, and data; therefore, promoting collaboration and integration anywhere in the network. DO extends the advantages of object orientation (inheritance, encapsulation and polymorphism), especially when it comes to reuse.

The Object Management Group (OMG), a consortium of over 500 major companies within the computer industry (except Microsoft, which has its own competing model, COM), has led the way on the set of open standard specifications that define the TCP/IP based object bus communication infrastructure, encompassed under the name of CORBA (Common Object Request Broker Architecture).

How does CORBA achieve interoperability? Perhaps the secret to success lies in the fact that it creates neutral interfaces, not code. These interfaces are written in IDL (Interface Definition Language), announcing a component's services to potential clients. The IDL-specified methods can be bound to high-level languages (C, C++, Java, Ada, SmallTalk, etc.), responsible for the final implementation.

The core component of CORBA consists of an Object Request Broker (ORB), a self-describing object bus that provides the transparent messaging mechanisms among objects regardless of location or underlying system characteristics. By vendors offering CORBA-compliant ORBs, universal interaction is accomplished. This basically means that vendors obey the rules of the Internet Inter-ORB Protocol (IIOP), the common backbone protocol residing on top of TCP/IP.

There are several elements worth mentioning without describing low-level internal anatomy details. On the client side, IDL stubs define how to invoke a service on the remote server object, and take care of appropriately formatting the operation and its parameters into a message understood by the server (marshalling). A unique global identifier (repository ID) is assigned per

component and respective interface, with no name collisions across vendors. Server-side stubs (called skeletons) transfer control to the actual object implementation (servant) upon request reception. The server running environment is responsible for instantiating new server objects, assigning them an identifier (object reference), advertise their presence and balance its supply according to incoming client demands.

Commercially and freely, many ORBs have emerged. If one is willing to pay the price, one can obtain nice extra features, such as object self-discovery, location transparency or reliable transaction support for mission-critical components. In our case, in an environment of limited resources, these "luxuries" were out of the question and unnecessary to reach the goal. Java was the clear choice of programming language, and so I directed my efforts to develop object distribution with it.

Originally, Java introduced its own distributed friendlier-to-use model exclusively designed for Java-to-Java intercommunication, called Remote Method Invocation (RMI). Contemporaneously, the Java Development Kit, like any other vendor, provided its own CORBA/IIOP ORB implementation, known as 'Java IDL'. Later, a version of RMI over IIOP was released, making it CORBA-compliant by allowing access to remote CORBA objects; this combines the RMI-style Java interface with the much desired cross-language interoperability.

An important property that tilted the balance towards an RMI versus a pure CORBA approach was caused by the proliferation of firewalls on the Internet. Increasing security concerns did prompt AOML to install a firewall to protect its network a few years ago, and NHC is expected to install one in the near future. It is foreseeable that most client invocations will originate behind a firewall; thus, it is crucial to take this scenario into consideration within H*Wind's topology. RMI not only works on a pre-defined TCP port, which at least allows potential configuration in a firewall, but it can also be tunnelled through HTTP, a very common protocol already allowed in most standard intranet security policies as part of the outbound traffic. In this world of compromise though, this technique degrades performance due to additional overhead, but the security benefits seem to outweigh the disadvantages. The proposed CORBA 3.0 includes a

section called CORBA Firewall Security, intended to address the issues faced in order to provide a standard approach to handle controlled authorized inbound and outbound IIOP traffic through a firewall, while maintaining the great degree of dynamism CORBA is famous for when it comes to integrating enterprise-wide distributed applications.

CORBA applications and ORBs generally choose to launch objects at arbitrarily selected TCP ports, where any object is a potential server and client simultaneously, each one with its own interoperable object reference containing unpredictable host/port addressing information. This situation is difficult for a firewall to configure. There are several CORBA Firewall Security compliant commercial products on the market which provide an IIOP proxy with security access control per client and target object, HTTP tunnelling and even support IIOP over SSL (Secure Sockets Layer), the standard transport protocol for encrypted messages. However, due to budget constraints, these solutions could also not be contemplated in this project. With Java RMI (free), I can satisfactorily achieve the objective of creating a distributed object out of the Analysis subsystem.

4. APPLICATION REQUIREMENTS

As stated earlier, this thesis comprises one of the 3 subsystems required for the success of the overall project. Though the Analysis subsystem mostly exudes server behavior, there is a still a client side that needs to be satisfied, one that collects the multiple analysis parameters and offers visual aid to the scientist as to the location of the several domain meshes. This client portion must be integrated with the rest of the Quality Control subsystem, which is responsible for the selection and validation of observations and storm track fixes involved in the analyses.

Unfortunately, after some initial testing of the Quality Control subsystem, it was clear that it was far from being complete and trustworthy, something that deeply affected not only the progress of the Analysis subsystem but, more importantly, the success and life expectancy of the whole project. Therefore, the full implementation of the client application became a major unexpected requirement. I took the core packages developed by Luis Amat [11], and proceeded to convert the prototype he built into the full-grown application currently released. Several aspects needed completion or creation (as scientists discovered more needs), which should be considered an expanded and superseding list of user requirements specified for the initial prototype.

4.1 User requirements

1. Database interaction:
   a) Select observations from a database given a time range, exposure, pressure level, and ocean basin.
   b) Select fixes from a database belonging to an event, or to an already generated track.
   c) Store scientist chosen set of observations and fixes (as a track) to the database in the form of a quality control set.
   d) Load a preexisting quality control set from a database, and be able to modify any content, as it would be stored as a completely new quality control set leaving the original intact.
2. Prevent loading of duplicate observation and fix data into a quality control set.

3. Real-time features: display current world-wide events, automated querying for newly arrived data in a periodic or instantaneous fashion.

4. Manipulate track by edition, interpolation and extrapolation, obeying the following rules:

- Fixes can be extrapolated only to a time before the track's beginning fix or after the track's ending fix. It is valid to perform extrapolation beyond an already extrapolated fix.
- Fixes can be interpolated only to a time within the track's beginning and ending fix. It is valid to perform interpolation of an already interpolated fix.
- Interpolated and extrapolated fixes are not editable, but they can be removed.
- Edited, interpolated and extrapolated fixes are stored in the database as derived fixes from the original or causal fix.
- Relevant fixes marked as beginning, center or ending fixes cannot be removed.
- Removal of a fix results in recursive removal of those fixes dependent of this fix, meaning potential editions, interpolations or extrapolations, with their potential associated interpolations and extrapolations as well. Consistency is paramount.

5. Whenever a track exists, its date/time range prevails over the real platform date/time range. Only observations within a track's range are considered.

6. Per platform, visually identify the observation with maximum wind speed and the one with the most recent arrival time.

7. Display maximum wind speed value, location, date and time among all passed observations of the entire quality control set.

8. Certain satellite data sources are not able to provide wind direction measurements. For data detected from those sources, automatically apply a wind direction estimation algorithm based on the storm track. If no track is available, these observations are displayed with zero wind direction (i.e. facing North).

9. Add missing observation attributes (wind gust direction, wind gust speed, wind gust date, unadjusted wind direction, unadjusted pressure, unadjusted temperature), whose data were being collected, but had been neglected in the database schema and class definition. Display their values when using the observation inspection tool.

10. Ability to draw wind barbs with either adjusted or unadjusted wind speed/wind direction per platform.

11. Once observations have been loaded, be able to decrease the time span to a lesser amount of hours from the initially loaded time range, per platform.

12. Since it is possible to perform upper-surface analyses, provide the ability to visually constrain the display of observations to a certain pressure range.

13. Incorporate scientist-defined algorithms for wind surface adjustment. In addition to HRD's boundary layer default surface adjustment model, scientists want to apply other models to the unadjusted data, each one with its own rules. At the same time, provide a way to return to initial unadjusted values.

14. A tool for (un)flagging an observation with one mouse click. It behaves like a toggle switch: if the observation has a passed status, the mouse click converts it to failed, and viceversa.

15. A tool for (un)flagging certain platforms within a chosen map region. The user is prompted with all the platforms detected within that region, being able to select which platforms should actually be (un)flagged.

16. While inspecting observations data, distinctly mark the observation in question on the canvas. Also, be able to inspect multiple observations at an exact location (previously, only one could be inspected).

17. Establish the uniqueness condition among global events as name+date+type+basin. Though very unlikely, there is a possibility that two events with the same name and type started on the same day on different basins; thus, we need to account for that.

18. When saving a quality control set, let the user choose whether or not to store the associated analysis and whether or not to generate the pertinent ATCF file (only valuable while in real-time operation).

19. Print canvas.

4.2 Analysis subsystem requirements

A. From a client perspective

1. Once a track is entered, and therefore, a track's center fix is known, draw lightly on the canvas the location of the 5 default analysis domain mesh sizes. By the same token, provide a way to clear off the mesh drawing if desired.

2. A graphical user interface for customizing number of meshes and their location. Naturally, draw these meshes on canvas dynamically.

3. A graphical user interface for entering analysis parameters: type of analysis (wind, pressure, temperature, relative humidity or a combination of them), pressure level, whether or not to perform an enhanced version, whether or not to generate bogus points (if the presence of a background field is detected).

If using expert mode, additionally: specifications for generating bogus points and Barnes meshes, mesh filter wavelength.

4. Warn user if attempting to perform a surface analysis with some non-surface adjusted data and vice versa. The warning won't prevent analysis scheduling.

5. Ability to modify platform weights, since not all offer same scientific reliability.

6. To load a marine gridded field from a previous analysis. The user is presented with a list of analyses stored for the current event and whose center fix time dates within 24 hours of the current storm center. Once the user makes a selection, the marine gridded field is adjusted space and time-wise to current storm conditions, becoming a background field for the current set. If the user chooses a new storm center, the background field needs to be re-adjusted.

7. If a background field is detected, offer the possibility of not generating bogus points.

8. Offer the possibility of turning off the execution of an enhancement provided on the analysis output stages.

B. From a server perspective

1. Distributed, interoperable via IIOP for any other possible client application, and accessible from anywhere in the Internet.

2. Multi-user, meaning that simultaneous invocations must not conflict.

3. Ability to perform analyses on all basins worldwide. It is specially tricky around meridian 180, where the date line is located.

4. Ability to perform analyses of any combination of types: wind, pressure, temperature and relative humidity.

5. Determine, field by field, the necessary information for the creation of the proper database schema to be used for analysis storage.

6. Derived from the previous point, the natural consequence of database interaction:

a) To store each analysis parameters for future reproducibility.

b) To store each analysis results associated to the corresponding quality control set: amplitudes (original and enhanced); marine gridded field; wind radii for 35-knot, 50-knot, 64-knot and 100-knot wind speeds at each storm quadrant; wind maximi (observed in real-time and analysis-estimated); minimum pressure.

c) To query a marine gridded field associated to a given analysis.

7. Generate a file following ATCF (Automated Tropical Cyclone Forecast) format based on analysis results, to be readily available to other weather agencies.

8. Generate text content to be embedded into the final annotated output, which includes: platforms involved in analysis with their corresponding time range, mean height if any aircraft platform was involved, type of scientific adjustment performed, minimum pressure, value and quadrant location of the maximum wind observation, characteristics of the storm center position (if it is an extrapolation, state chosen storm speed and direction).

9. Make analysis results in the form of gridded files available for the modeling community.

With this preliminary list of functional requirements and notions of what is generally desired to achieve, the next step is to build a plan of action on how they should be combined to form a model with a more detailed description of the envisioned purpose of the whole system. The use case model promotes understanding of the system as it is easily related to reality and semantically close to the users. Due to the close interaction with this project's domain experts, HRD scientists, the natural strategy was to establish a constant dialog with them about the expected sequence of steps for each scenario, and how each of them could influence or interfere with other scenarios. Following a use case driven design, one can express several flows of transactions resulting from the interaction among the actors representative of the problem domain. For maintainability and documentation purposes, it turns out to be helpful and flexible for an iterative development cycle.

5. USE CASES

Here I present a general use case depicting the expected actions to produce an analysis, its derived products and database storage (fig. 4).

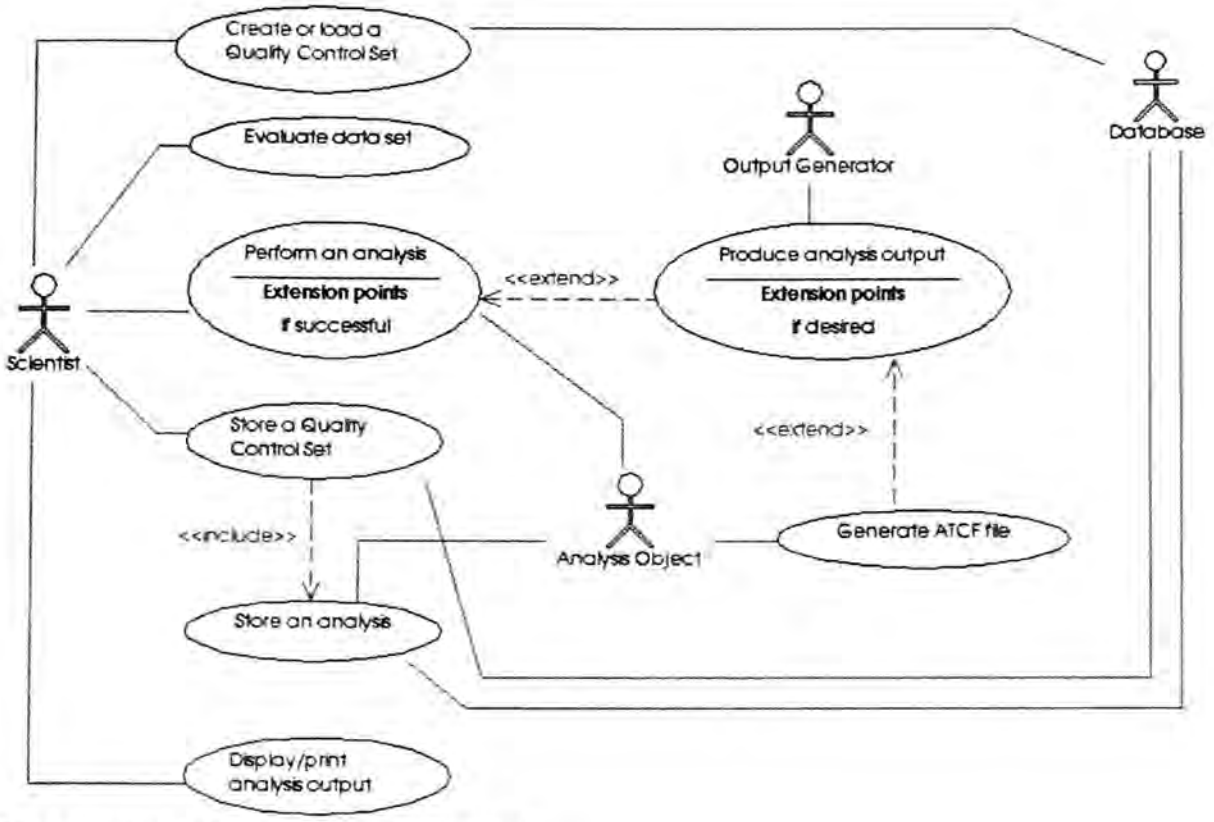


Fig. 4 General primary use case of H*Wind

Several actors interact for accomplishing the different tasks expected of the system. As one might expect, the role of the scientist takes a predominant place as the person who can request any of the major tasks, some of which are fulfilled by specialized software components. The extended use cases can only happen if the core use case occurs; in our case, the execution of a successful analysis denotes such a use case, which opens the possibility to generate an analysis output, whose results in turn allow the generation of an ATCF file, but none of these optional scenarios affect the goal of the core function. On the other hand, an inclusion use case must happen or be true before the encompassing use case can occur. In our example, even though a scientist could store a quality control set without an analysis associated with it, the rules stipulate

that it does not make any sense to store an analysis without its corresponding quality control set (for data consistency purposes); thus, the need to state this situation.

Details of steps inherent to the principal actions follow. As a general rule, the most regular "happy" path is described, with its possible alternatives. It is desirable to state pre- and post-conditions for later verification. To achieve a greater level of clarity and understanding, several activity diagrams are supplied, which express transitions among sequence of events when conditional and parallel behavior are relevant.

5.1 Create a quality control set

Pre-condition: The user selects to create a new set. The user is logged into the database.

Basic Path:

1. The application presents a list of active tropical cyclone events throughout the world.
2. Scientist selects one of these world-wide active events.
3. The system loads all the storm track positions (fixes) for that event, and sorts them in chronological order.
4. Scientist inspects this list of chronologically ordered fixes (a track) and decides the time range of interest.
5. Scientist can interpolate, extrapolate, manually add fixes or load a new track altogether.
6. Scientist proceeds to load observations from the database specifying a desired time range, exposure and pressure level.
7. From the list of data platforms shown to the scientist, he/she includes as many as desirable. An inclusion causes all the observations of a platform to be drawn on the canvas map. Since a track exists, all observations shown are constrained within the track time range, and displayed with storm-relative positions (as opposed to earth-relative).
8. Scientist decides to load a marine gridded field from a previous analysis, since the scientist considers there is an insufficient amount of raw observations.

9. The system presents the user with a list of all analyses stored whose storm track center time is within 24 hours prior to current storm center time chosen.

10. The user selects one of those analysis and the system retrieves its corresponding marine gridded field, which is automatically adjusted in time and position to the current storm conditions, as specified by scientist requirements.

11. Via tools such as observation scope, zoom, flagging, data inspection, distance calculation, the scientist achieves a desired level of quality control.

12. Scientist could repeat steps (4), (5), (6), (7), (8), (9), (10) and (11) as wished, in random order.

13. Scientist decides it is time to perform an analysis (see use case 5.3)

Post-condition: A valid quality control set is created in order to analyze it.

Alternative Paths:

a) Step 2 just shows one of the ways to load or create a track. Other ways are:

- to load a track from the database
- to load a track from a file
- to load individual storm track fix positions from the database
- to create individual storm track fix positions by manually typing the data.

All of the available methods to generate a track can be used interchangeably.

b) At step 6, if no track is present, all observations retrieved from the database are shown without time constraints.

c) Steps 7, 8, and 9 can only happen if a track exists, since one is indispensable to be able to adjust a marine gridded field to current storm conditions.

d) At steps 5, 6, and 7 the scientist can select to cancel the operation.

e) At step 6, if the system detects the user loaded flight-level data from aircraft, then the scientist is allowed to apply any of the surface adjustment algorithms offered, repeatedly.

Figure 5 describes the high-level overall possibilities of the Quality Control subsystem (called QCClient).

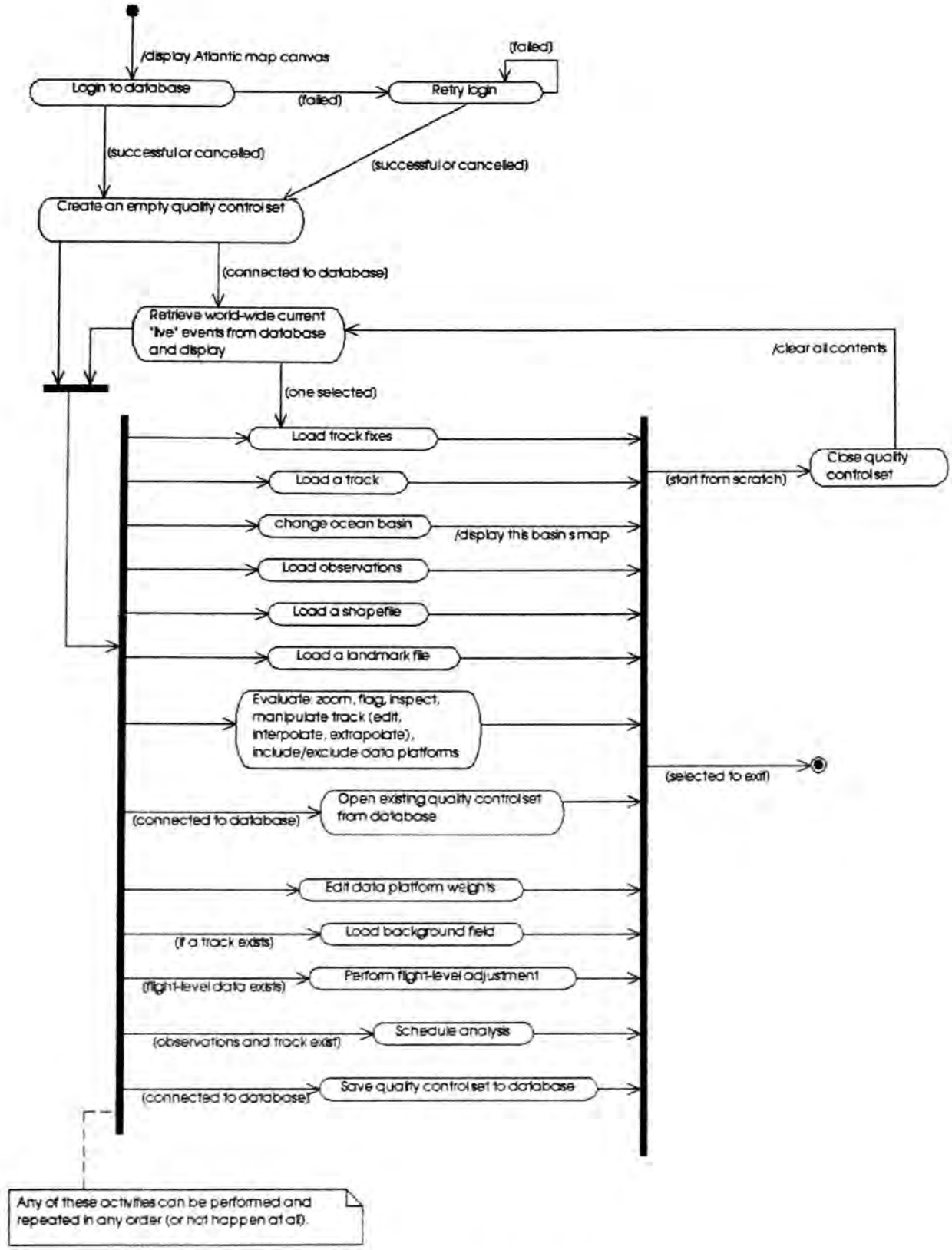


Fig. 5 General activity diagram of the Quality Control subsystem

5.2 Load a quality control set

Pre-condition: The user selects to load a QCSet.

Basic Path

1. The system presents the user with the following interface:

By default, it presents a list of events and a list of users associated to the QCSets stored for the Atlantic basin and current year, so that the scientist can narrow down his/her search. Obviously, a change in basin or year causes an update of the lists.

2. The scientist chooses at least an event (and optionally a user).

3. The system displays a list of all QCSets for that event (and user, if applicable) sorted in descending chronological order by storm center fix date and time. QCSets without a track are displayed at the end of the list. In general, scientists are interested on loading the latest QCSet of an event.

4. The scientist picks one QCSet.

5. The system loads the event information (name, ATCF code, date, category)

6. The system loads the track (if there exists one) and sorts the fixes in chronological order.

7. The system loads the observations.

Post-condition: The chosen quality control set is loaded and ready to be evaluated. The user is free to continue on step (8) from Use Case 5.1.

Alternative Path

- At steps (1), (2) or (3), the scientist can select to cancel the operation; therefore, safeguarding the contents of any previously loaded QCSet.

5.3 Perform an analysis (scientist's perspective)

Pre-condition: A track, an event and a non-empty set of wind observations are loaded.

Basic Path

1. Scientist selects the Analyze option from the QCClient application.

2. Scientist chooses 'Novice' mode.

3. Scientist enters analysis description:

a) exposure (land or marine)

b) time frame (operational or research)

c) type (a combination of wind, pressure, temperature and relative humidity)

d) registering a minimum pressure value

e) choosing whether or not to generate bogus points (applicable only if a background field is present)

f) choosing whether or not to perform output enhancement

4. Scientist selects a pressure level, ranging from surface (1070 milibars) to 150 milibars, in intervals of 50 milibars.

5. The QCClient application will dynamically redraw the canvas map with observations limited to the chosen pressure range, and warn whether there is an attempt to perform a surface analysis with non-surface-adjusted observations.

6. Scientist selects one of the predetermined analysis domain sizes, based on years of experience: 'Poorly Defined', 'Small', 'Medium' or 'Large'.

7. The QCClient application will display the location of the corresponding nested meshes on the map.

8. Scientist schedules an analysis.

9. The QCClient application invokes the execution of the analysis on the appropriate remote server as a distributed object.

10. The remote server runs through all the steps of the analysis process (use case 5.4).

11. At completion time, the QCClient application informs the scientist whether it was a successful run or not.

12. Scientist reviews analysis output.

Post-condition: An analysis has been performed providing a response about its final completion status. If analysis execution was successful, output files have also been generated.

Alternative Paths

- At any time before scheduling an analysis, the scientist can select to cancel the operation, or can trace back his/her steps.

- In all steps, if any input is incorrect, the system will prompt the scientist to correct it and will not allow proceeding unless the scientist does so.

- In all steps, if any input is incorrect, the system will prompt the scientist to correct it and will not allow proceeding unless the scientist does so.

- Starting at step 2:

2. Scientist chooses "Expert" mode

Continues with steps 3, 4 and 5 of basic path.

6. Scientist determines all parameters related to the generation of bogus points and Barnes meshes.

7. Scientist enters size of innermost domain mesh, in kilometers or degrees latitude.

8. Scientist establishes the number of meshes.

9. The system informs of the size of all meshes to be involved.

10. The system presents an editable table to modify the nodal interval, the filter wavelength and geographical coordinates of each domain mesh, and the outermost mesh boundary conditions for northern, southern, eastern and western flanks.

11. The system will visually display the location of the corresponding nested meshes on the map. If the scientist modifies the size of the meshes, so will they be displayed dynamically.

Continues with steps 8, 9, 10, 11 and 12 of basic path.

Figure 6 describes the choices to select analysis parameters.

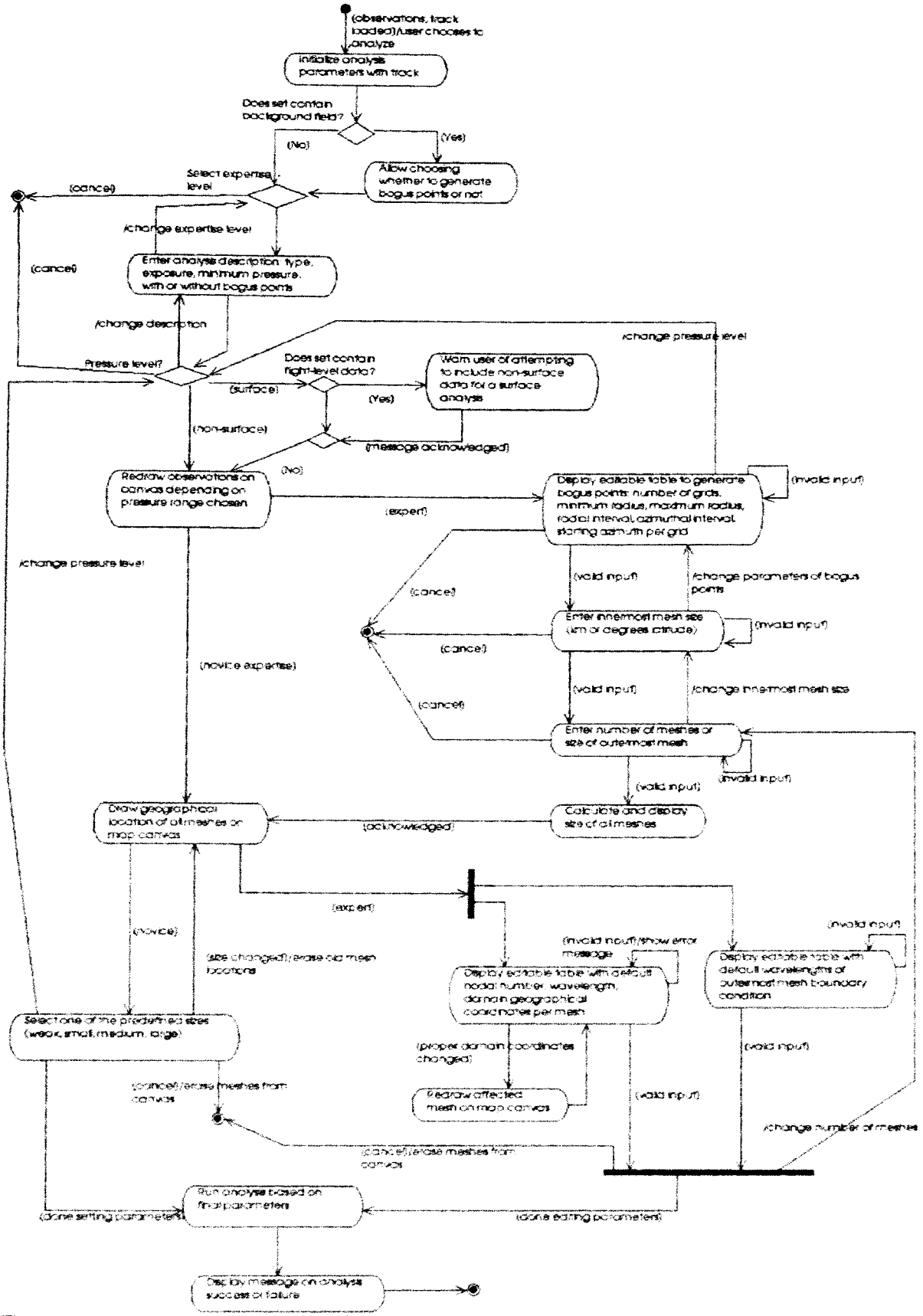


Fig. 6 Activity diagram to select analysis parameters

5.4 Perform an analysis (Analysis server's perspective)

Pre-condition: A client application invokes the analysis process via object distribution.

Basic Path (fig. 7)

1. The system creates a random-number-generated directory to be the area of current execution. (Several instances can run concurrently without any conflicts.)

2. The system writes the input files expected.

3. The system invokes all legacy programs in the proper order.

4. The system cleans up any secondary files unnecessary for next step (to save disk space).

5. The system executes the scripts responsible for generating the postscript output.

6. The system annotates the Postscript output files and converts them to a graphical format easily accessible via an Internet web browser.

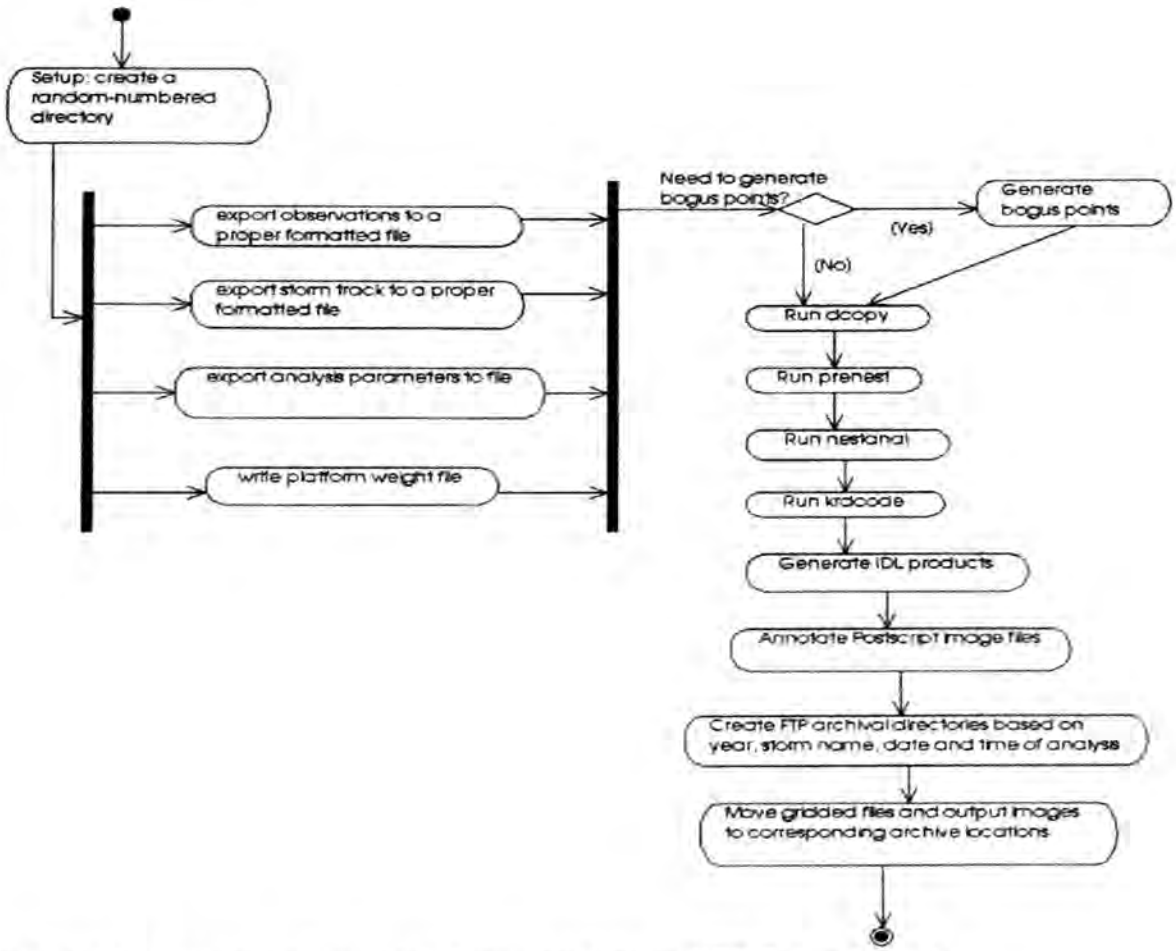


Fig. 7 Activity diagram for the steps taken to produce an HRD Spline Analysis

7. The system archives the graphical files.

8. The system returns a success or failure status to client application.

Post-condition: The remote server sends analysis completion status to client. If successful, analysis output files are available for review by the scientist.

5.5 Store a quality control set to database

Pre-condition: A track, an event and a non-empty set of wind observations are loaded, where all observations originate from the database.

Basic Path (fig. 8)

1. The scientist selects to save a set from the QCClient application.

2. The system presents a panel informing whether a successful analysis has been performed or not. In the case of a valid analysis, the user is able to choose whether or not to generate an ATCF file.

3. The scientist did perform a successful analysis and chooses to generate an ATCF file for it.

4. The scientist proceeds with database committal, which consists of:

a) Saving event information, such as event name and level of cyclone development.

b) Evaluating the status of each observation (passed, flagged, edited):

b.1) if it is a non-edited passed observation, its database id is associated to this QCSet.

b.2) if it is an edited observation, it must be first inserted in the database to obtain an id, and then associate that id to this QCSet.

b.3) if it is a flagged observation, then its database id should be associated to the set of failed observations of this QCSet.

In the case of edited and flagged observations, a reason is attached as to why they were modified from the original observation.

c) Associating all track fixes with this set id. These fixes may have diverse origin:

c.1) If a fix originated from the database, then use that given id.

c.2) If a fix was manually entered by the scientist, then it must be first inserted into the database to obtain a unique id.

c.3) If a fix was interpolated or extrapolated, then it is inserted as an edited fix of another fix, since they can only be created based on other fixes. (Bear in mind, that the scientist is free to create unlimited interpolations and extrapolations of already interpolated or extrapolated fixes.)

5. The database responds with a unique identification for this quality control set.
6. Store an analysis (see Use Case 5.6).
7. The client application informs scientist about the success or failure of this operation.

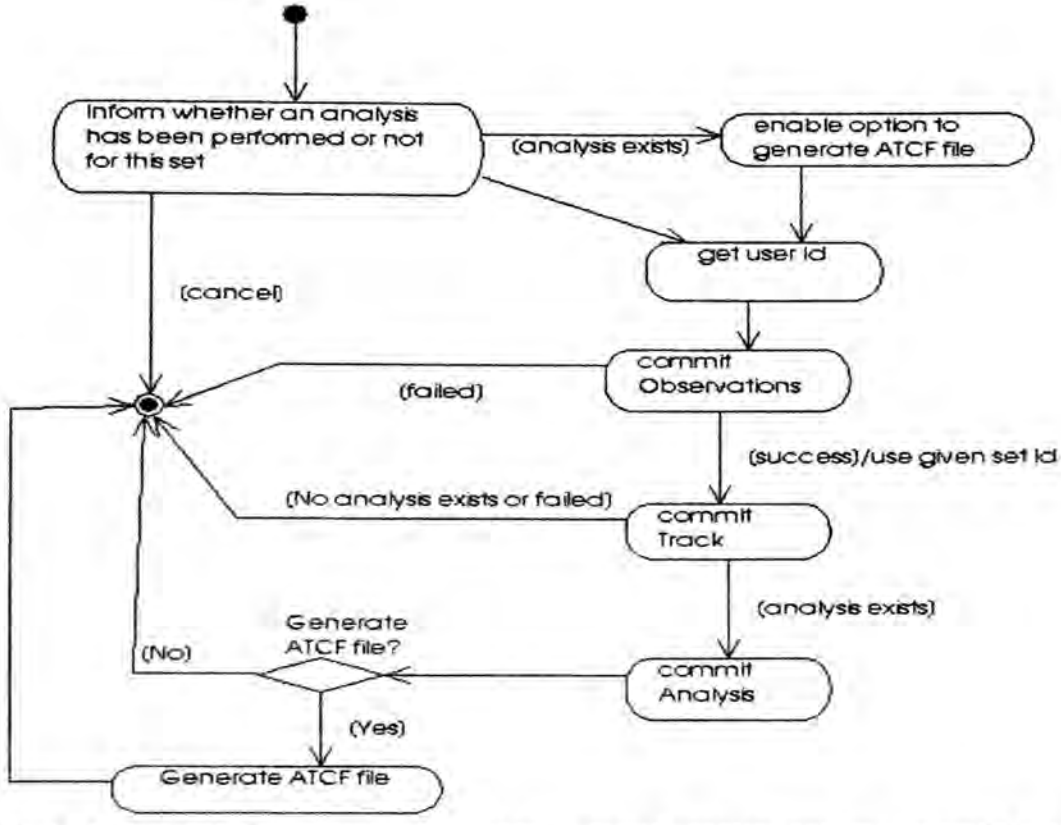


Fig. 8 High-level activity diagram for storing a quality control set and potentially the analysis associated with it

Post-condition: A quality control set is stored in the database.

Alternative Path

- At step 2, the scientist can select to cancel the operation.

- At step 2, if no analysis has been performed prior to storing a quality control set, then step 6 does not occur.

5.6 Store an analysis to database

Pre-condition: A quality control set has been stored in the database; therefore, the system knows of its unique database id. The scientist performed a successful analysis.

Basic Path

1. The client application invokes the analysis remote distributed object to start a thread for committing to database several components of analysis results: krdfile, enhanced amplitude file, marine gridded field, and wind radii.

2. The client application invokes the analysis remote object to generate the corresponding ATCF file, and publish it for the Joint Typhoon Weather Center (JTWC).

Post-condition: An analysis and derived products are stored in the database associated with the corresponding quality control set.

Alternative Path

- If no generation of ATCF was chosen, then step 2 does not take place.

5.7 Flight-level surface-adjustment based on eyewall tilt corrections (designed by Jason Dunion and Mark Powell, HRD)

Pre-condition: At least one flight-level data platform (AirForce or NOAA aircraft) is included in the quality control set and a track exists.

Basic Path

1. The system calculates the radius of maximum wind (RMW) at each relative quadrant of the storm center with respect to the storm motion direction. In case a RMW could not be found in a certain quadrant, apply the following rules:

For no RMW found in	Use RMW found in (preferred order)
Front Right (FR)	FL, RR, RL
Front Left (FL)	RR, FL, FR
Rear Right (RR)	RL, FR, FL
Rear Left (RL)	FR, RL, RR

2. The system presents the 4 preliminary RMWs (nautical miles) and a default sea surface temperature (Celsius degrees) to the user, who has the freedom to edit them.

3. The user accepts parameters and continues.

4. For each passed flight-level observation with 650 < pressure < 1010 milibars:

a) create an edited observation associated to the original one, where the changes will take place.

b) calculate the distance from its storm relative position to the storm center fix position.

c) calculate the ratio of this distance over the RMW obtained in this observation's relative quadrant.

d) if the pressure is between 650 and 750mb and the ratio is < 2.0, then a mean boundary layer wind is computed based on a specific polynomial function. In all cases, a comment is set to indicate whether the eyewall tilt has been applied or not.

e) if the resulting mean boundary layer wind speed (boosted or not) is < 55 m/s, then surface adjust it using HRD PBL program [1]; otherwise, an empirical derived adjustment of 0.85 based on GPS dropsonde data is applied to estimate the maximum 1-minute sustained wind at the 10 meter level.

5. The canvas map is updated, by showing the edited version of these observations, and by distinctly pinpointing those wind observations where the RMW per quadrant was found.

Post-condition: All evaluated flight-level observation wind speeds are surface adjusted. The adjustment type is noted for that platform.

Alternative Path

- At step 3, user might decide to cancel the operation.

6. OBJECT-ORIENTED ANALYSIS

Gathering requirements and eliciting use cases is an intrinsic part of determining what the system must do, the essential preoccupation of a project's analysis phase. In addition, there is certain modeling involved to identify the classes that fundamentally belong to the application, as well as to express their relationships. For the most part, the basic analysis and design of the Quality Control prototype already provided a good infrastructure. The main classes and their associations had been well identified and established, and were well organized in meaningful packages in [11]. Based on the requirements, clearly this project involves a complex graphical user interface, data structures holding consistent information over the life span of use cases (with their natural applicable operations), and actions or processes to be performed in response to the behavior and state of the two aforementioned elements. A good effort had been made to classify the identified objects of this problem domain into objects responsible primarily with presentation, persistent information, or behavior characteristics; a strategy that has continued and intensified throughout this undertaking. Current trends in object-oriented software engineering, such as the Model-View-Controller (MVC) pattern, signal this architecture to be conducive for creating systems that are better prepared for potential changes, which I have found to be the engine that decides where to split functionality.

A good problem analysis should define a stable, robust, and extensible structure, resilient to the inevitable common and significant changes the overall system will be subject to on functionality and user interfaces as time goes on. The key to stability is to maintain changes as local as possible, affecting as few classes as possible. The heavy and extensive testing applied to the prototype with real data revealed certain facts that were not contemplated originally, different associations that more accurately described the problem domain, new classes that led to more efficient shared functional implementation, and undoubtedly new operations. Even though there have been widespread implementation modifications, they have not notably disturbed the overall OO analysis, and yet they have proven the benefits of the principle of locality.

The complete final class diagrams of the Quality Control subsystem (Appendix A) are the product of an evolution process of more than two years of several iterations of the development life cycle.

Since the object-oriented analysis and design of the Quality Control subsystem has already been comprehensively explained in [11], I will concentrate on the idiosyncrasy of the Analysis subsystem in this thesis.

The job was divided in several sections. One entailed defining the most obvious classes, such as the ones carrying the weight of holding analysis parameters, analysis results (KR file) and analysis products (enhanced amplitudes, wind radii, ATCF, etc.). After a long time of scientists being used to a hardcoded inflexible operational procedure, they were disengaged of the inner workings of the FORTRAN programs. Bits and pieces were consolidated from a few knowledgeable individuals (Fig. 9).

In the case of ATCF issues, I was able to contact Charles Sampson [9], a major developer of the system. In order to maintain good synchronization between the experts and myself, I have kept a close approximation to their structural arrangement.

Another section was dedicated to define the Spline Analysis steps. Recalling, the code consists of five separate programs or sequential steps. From a behavioral point of view, clearly, each one of them is considered a different class, but all of them share same functional phases: 1) setup, 2) run, and 3) cleanup. Therefore, these steps and any other potential additional ones can be grouped under an abstract parent class called 'AnalysisStep' (Fig. 10). The 'setup' phase is intended for those actions involved in preparing the input files needed for the current step, the 'run' phase is responsible for the actual execution, and the 'cleanup' phase manages deletion of non-essential files generated during this step to deaccelerate storage space shortage.

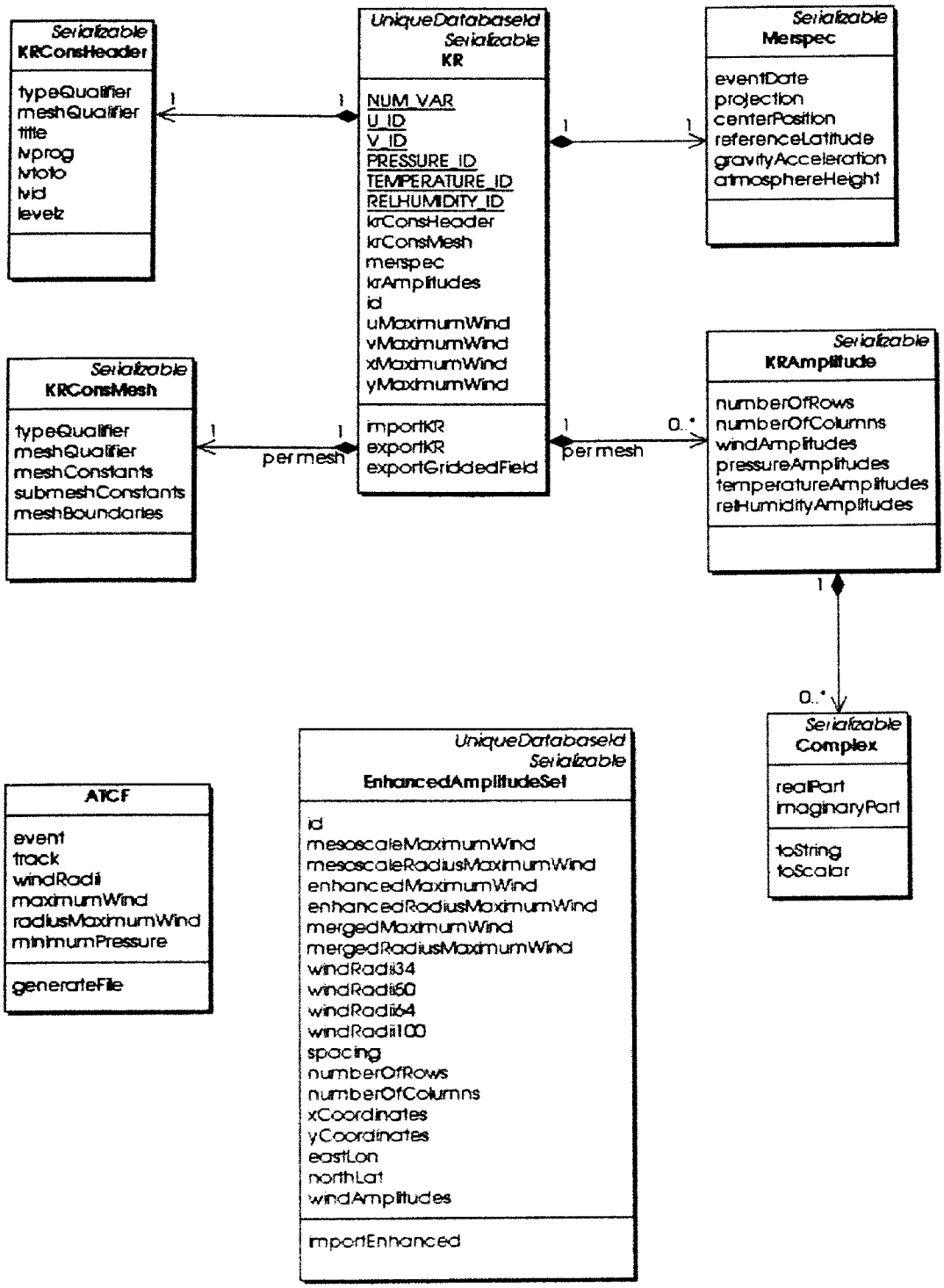


Fig. 9 Class diagram of analysis results and derived products

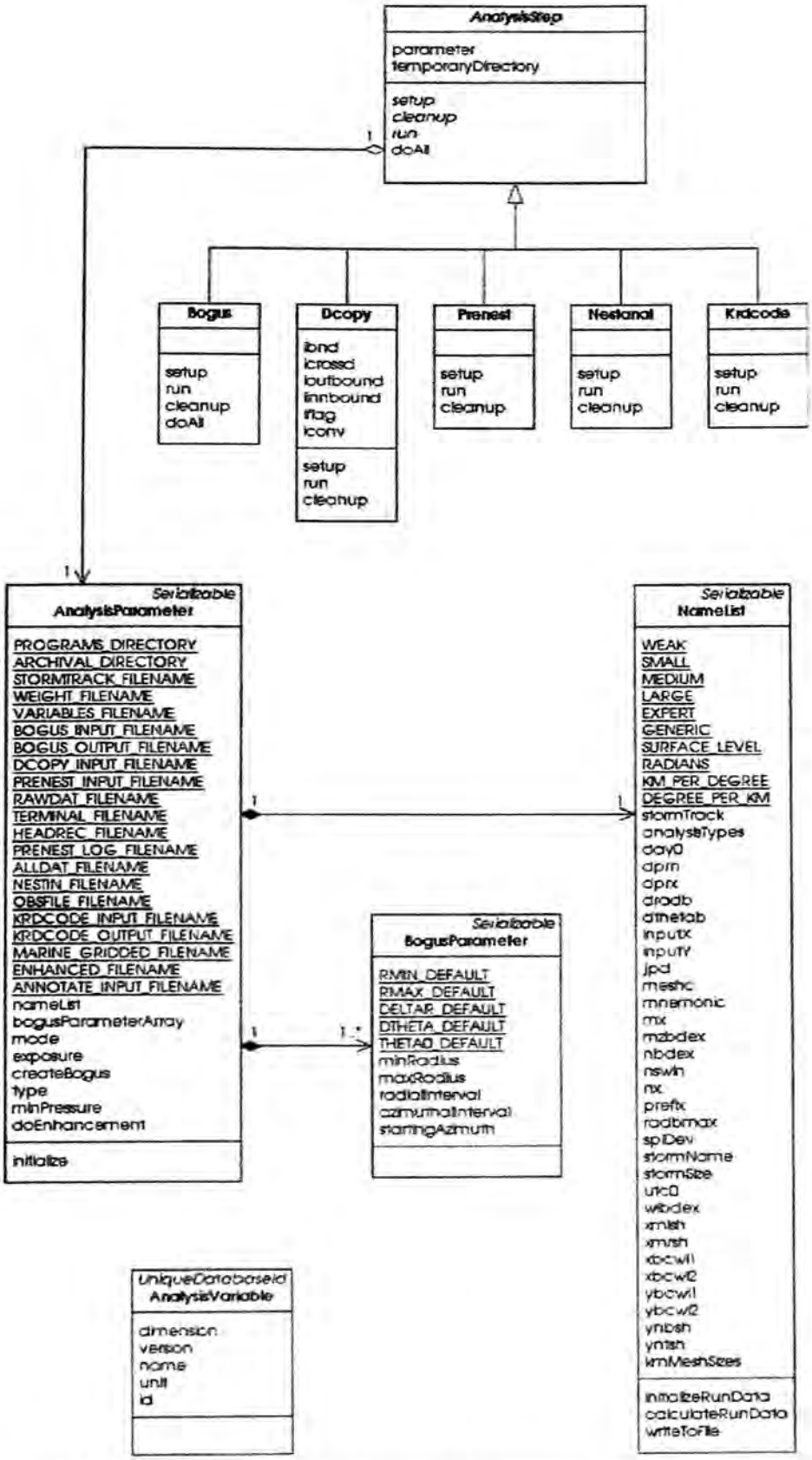


Fig. 10 Class diagram of analysis steps and related classes

A third section dealt with the different user interfaces in charge of capturing unambiguous valid input for editing platform weights ('WeightEditor'), loading a background field ('MarineGriddedFieldQueryWizard') and entering analysis parameters ('AnalysisWizard'), as well as the drawing of the location of the spline analysis meshes based on the parameters ('AnalysisArtist', fig. 11).

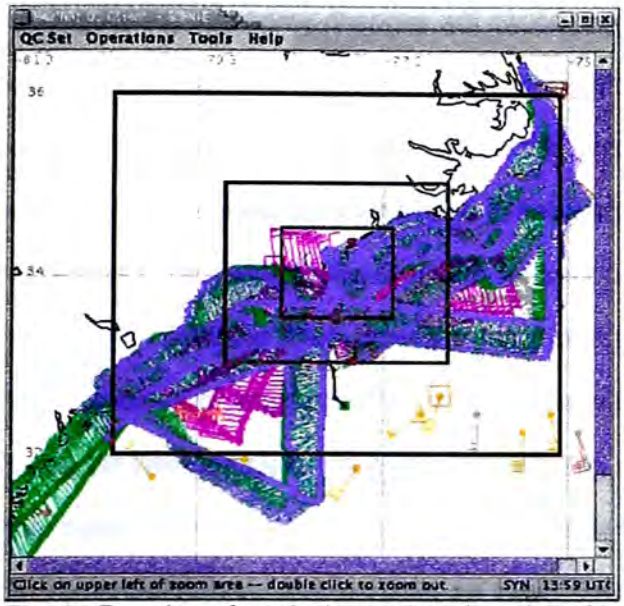


Fig. 11 Drawing of analysis meshes (landfall of hurricane Bonnie, 1998)

The integration of these graphical interfaces into the Quality Control subsystem mainly constituted the inheritance from an abstract class already present ('Artist', 'Wizard') or the transformation of a class into a more generic one to accommodate the cohabitation of an existing evident extension of it and a new arrival ('Inspector', superclass of 'WindObservationEditor' and 'WeightEditor'). See Appendix A.

7. OBJECT-ORIENTED DESIGN

This phase is a refinement of the analysis, geared to get in touch with reality, to adapt to our implementation environment, until it is straightforward to write source code from it. My experience has been that there is a "gray" or "blurry" area during the transition from analysis to design. One is easily tempted to start adding complexity to the analysis model, without distinguishing that some changes are caused by a logical change in the system and others are a consequence of the implementation environment. In a project influenced by notable factors such as DBMS and object distribution, certain sections of the analysis model will remain rather abstract and informal because decisions need to be postponed for the design. It is recommended to keep an analysis view which reflects all the work focused on capturing the essentials, and change it if new logical relations arise among classes due to new properties, but we should stop when we find ourselves changing it to adapt to the environment. The conceptual view of the analysis, being far less complex, will assist in reasoning when to incorporate changes, always remaining the basis of the design model, which is just one specialization, an approach for a certain implementation [14]. Not precisely an exact science, but that is where gaining experience comes into place.

The closer we work our way to the actual source code, the greater the diversion from the original prototype.

7.1 Issues raised on the Quality Control subsystem

A. Spatial filtering

Even though the application is oriented to deal with one storm in a specific ocean basin, parameters in observation queries to the database did not take into account this welcomed constraining factor, and therefore, were inefficiently retrieving observations for all basins within only date/time limits. This imposed unnecessary processing work on the quality control client application, and caused user complaints due to poor response time. In collaboration with the database specialist in our development team, the situation was corrected. Currently, we are in

the process of taking this concept a step further: two years of use have shown that scientists are really only interested on a geographic area of around 10x10 degrees latitude centered on the chosen storm track center position, for even greater tuning. The database is in the midst of a significant upgrade; we hope to offer this spatial filtering capability for the 2003 hurricane season.

B. Duplicate data

Lack of checking for loading duplicate observations and storm track positions created incoherent quality control sets for visualization and HSA purposes. Different causes could produce this situation. To begin with, although the real-time data collection subsystem tries to go the extra mile to prevent malformed or meaningless data from insertion in the database, duplicate bursts of data are occasionally ingested. Another instance could occur when a user, free to load observations as many times as desired, loads for a time and space frame previously chosen. It is imperative that the user interaction flexibility should never compromise the consistency of the quality control set; therefore, proper checking was added via verification with the method 'isSame(...)', which percolates down to the most basic constituents.

C. Optimized searches

The original strategy of using hash tables as data structures was improved. They maintain loaded observations classified per data platform description, per date/time combination, and per a unique 5x5-degree world region based on their earth-relative and storm-relative positions. Each table facilitated quicker searches or more targeted operations throughout the hundreds of potential observations. Previous methods that inefficiently looped through the whole list of observations were replaced with methods that exploited these tables. As a matter of fact, this global list turned out to be totally unnecessary. Drawing observations really meant drawing observations of those included platforms (thus, the method 'drawPlatform(...)' in class WindObservationArtist); (un)flagging a group of observations on the canvas really meant (un)flagging observations within a mouse-selected geographic area enclosing one or more of these 5x5-degree regions (thus, the method 'getAreaIndexList()' in class GlobalArea). A nice optimization will be to come up with hashtable keys denoting smaller regions, not only because

the user will be dealing with a more limited area of interest in the future, but also because practice tells us group operations tend to affect smaller geographic sections.

D. Efficient lookup tables

A lookup table was added to keep information of all data platforms after the first database observation query, searchable by platform description. This is useful to obtain access to the default weights assigned to all platforms, so these values can be presented in the weight editor dialog window (attribute 'allDatabasePlatforms' in class WindObservationSet).

Another lookup table was added to keep track of the observation station names given their station database ids. Database observation queries return the station id where the observation was taken, but the station name is really the meaningful piece of information to the scientist while inspecting the collected field values of an observation ('stationTable' in class HwindObservationSet).

E. Establish correct database storage

Regarding a storm track, two hash tables were added to manage the potentially multiple interpolations and extrapolations a storm track fix position could be subjected to. The key is the causal storm fix position, with an associated value of the newly derived fix ('interpolatedFixTable' and 'extrapolatedFixTable' in class 'Track'). These tables are especially crucial at the time of properly storing a track in the database, whose schema was constructed to ensure these kinds of derivation relationships are not lost, and are indeed traceable. Figure 12 shows the steps to store a track to the database. Detailed, concise activity diagrams allow expressing a deterministic problem in a finite state machine fashion, proving an invaluable tool for the developer during implementation.

Regarding observations, figure 13 shows the steps to store passed and flagged observations to the database, given the fact that edited observations may also be included.

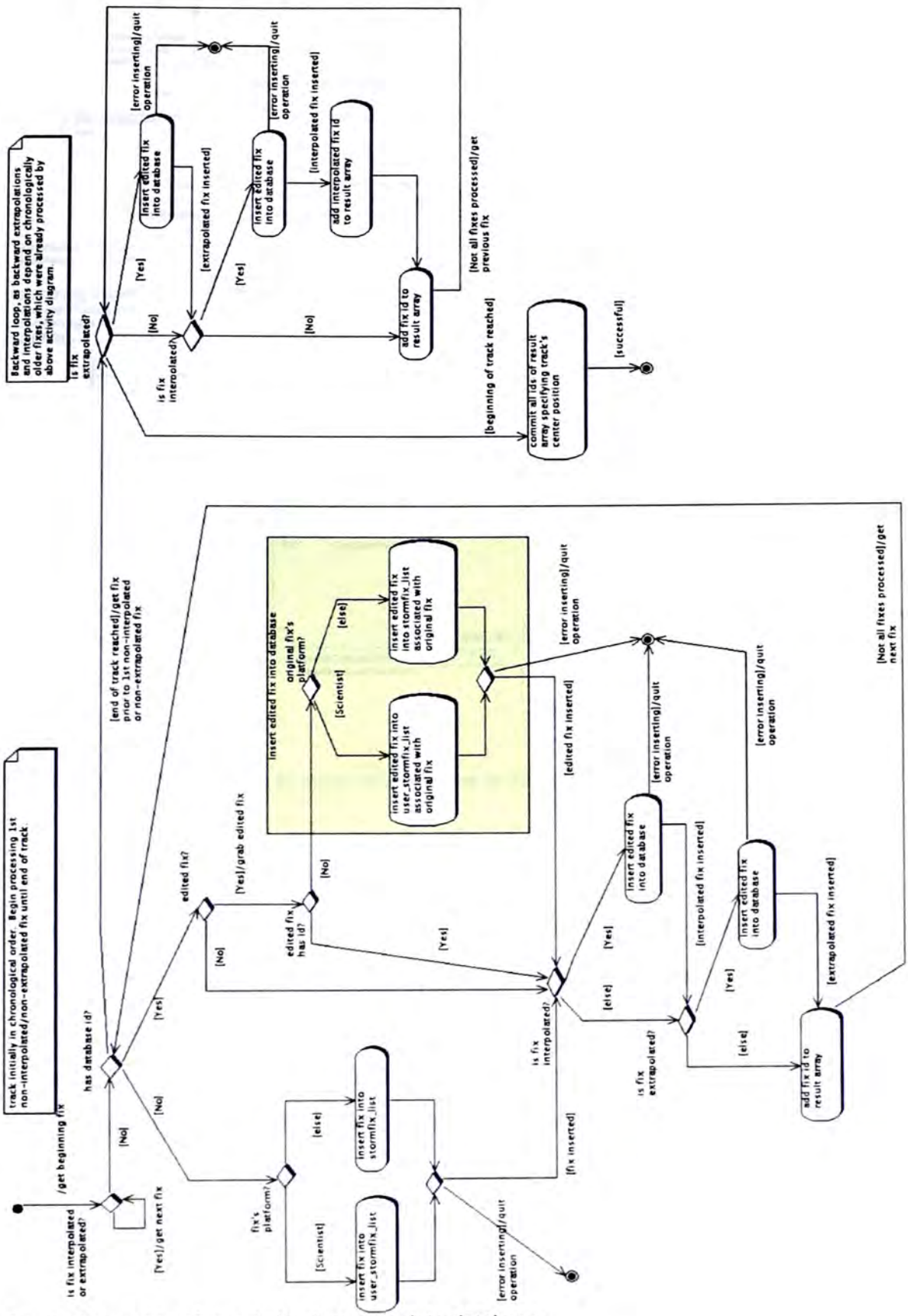


Fig. 12 Sequence of events to store a track to database

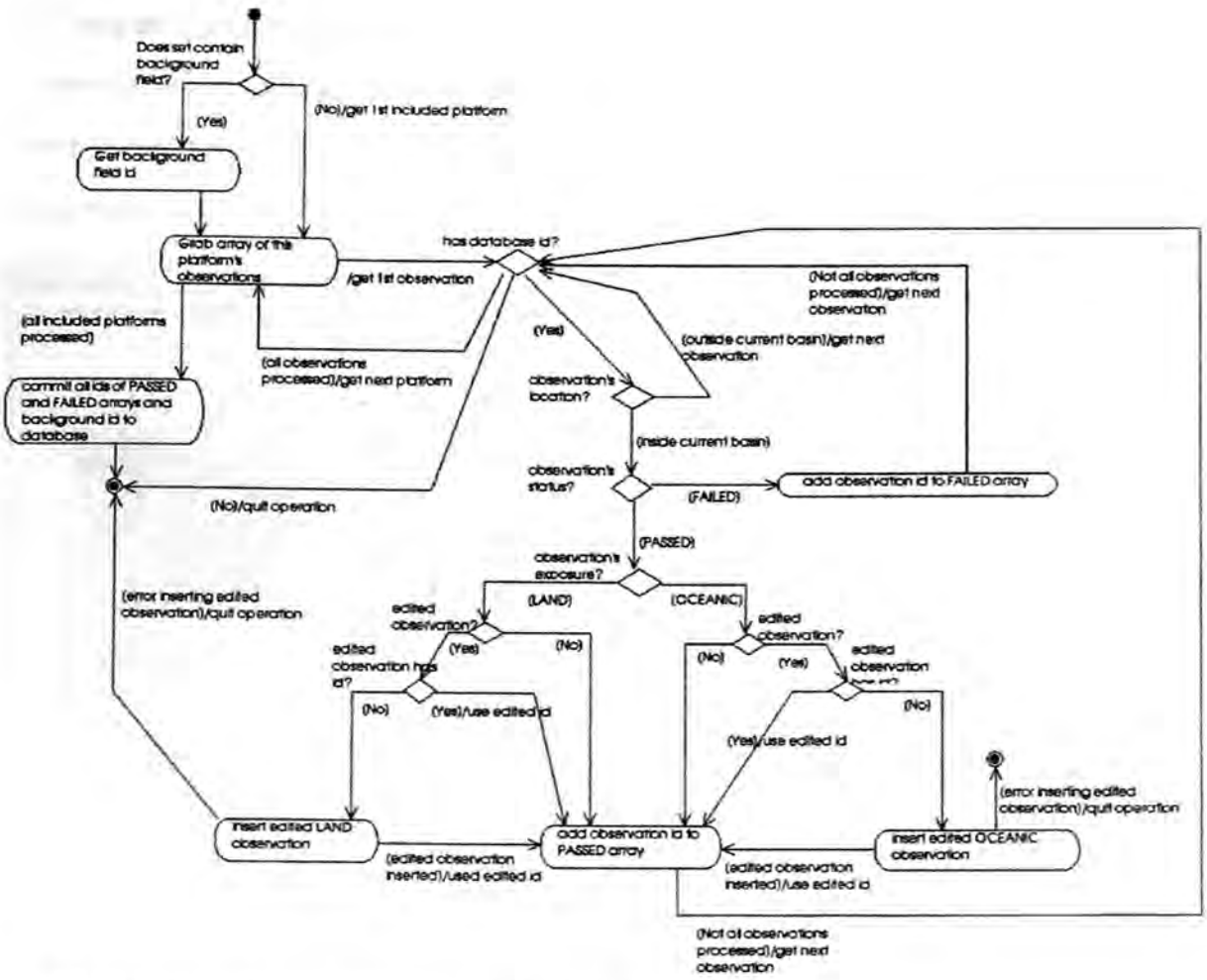


Fig. 13 Sequence of events to store observations to database

F. Enhance observation inspector tool

This tool works by displaying individual information about a selected wind barb on the canvas. Its prototype version had failed to consider the fairly common likelihood that multiple observations could be located at the same geographic position over a typical time range (moored buoys, land stations). In addition, in case an observation's wind speed or direction was modified, a more efficient method was devised to repaint it without performing a total canvas redraw. Furthermore, when the maximum wind observation per platform was either modified or flagged, a new maximum wind (universal and per platform) was searched among the passed status observations, always keeping paramount the correctness premise.

G. Changes in GeographyView

Added table to display all the world-wide current events (fig. 14). The selection of one of these events forces the creation of a new quality control set, since a new default track is automatically formed with all the fixes associated to this event.

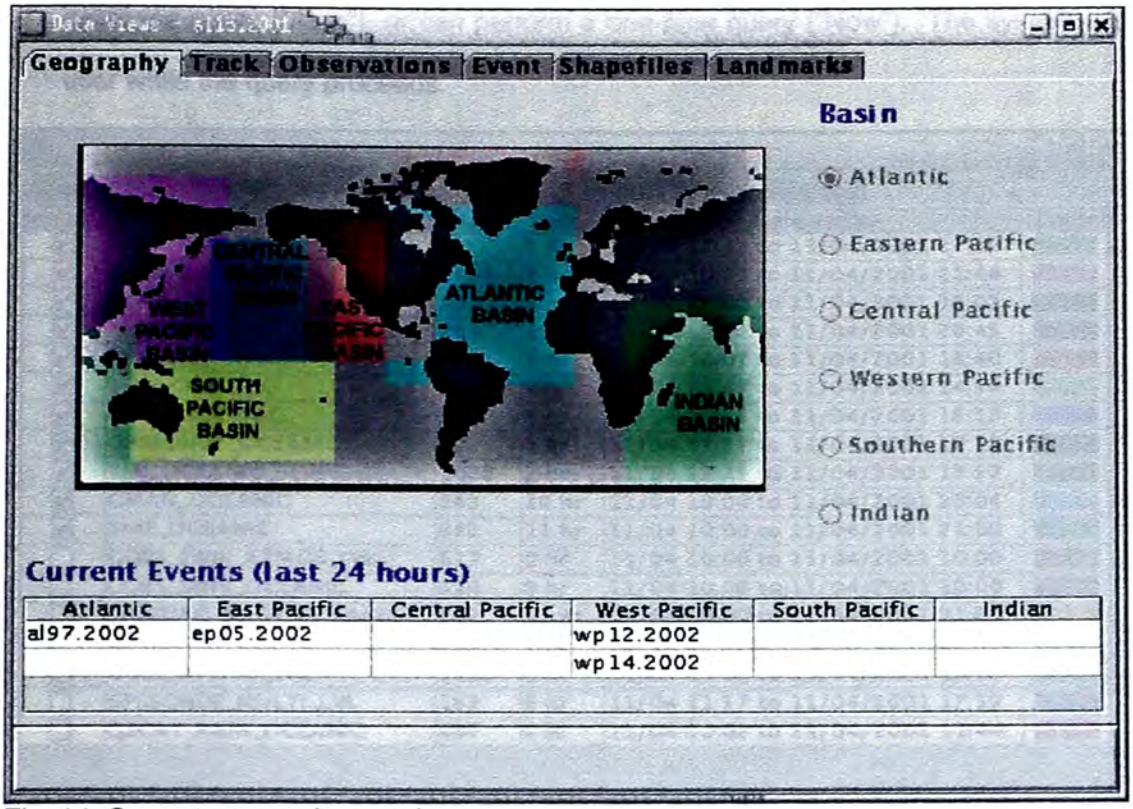
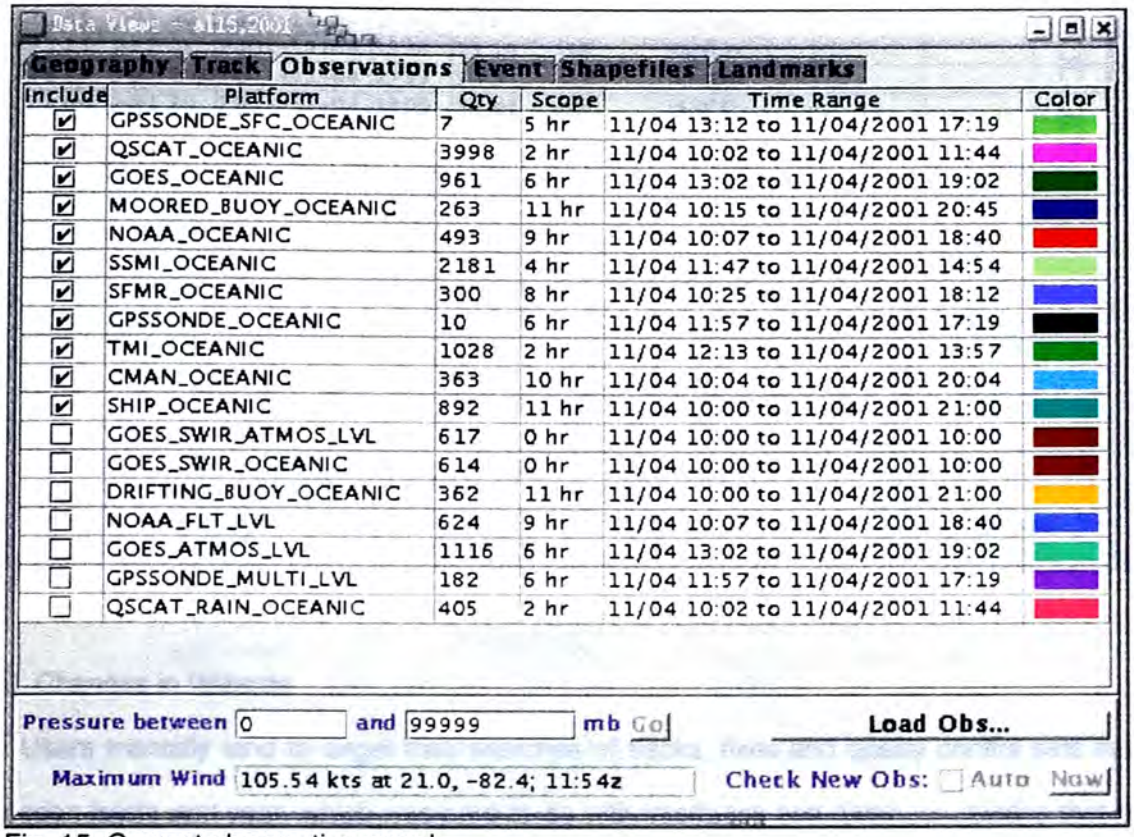


Fig. 14. Current geography panel

H. Changes affecting WindObservationView (fig. 15)

- Added 'Scope' column to WindObservationView, to constrain the time window of a certain platform. The 'Quantity' column gets adjusted accordingly.
- Re-designed the algorithm to generate not so redundant platform colors.
- Displayed the location, time and value of the maximum wind speed observation of all included platforms.
- With the advent of upper-level analyses, so came the necessity to offer the ability to restrict drawing to observations within a certain pressure level range.

- In real-time operations, it is highly desirable to obtain the latest observations and fixes up until it leaves enough time to run an analysis. If there is less than a 6-hour difference between the current UTC time and the latest arrival time among all observations, the two methods for "Checking New Data" are enabled. The user can activate a 10-minute periodic database query ("Auto"), or can perform a one-time query ("Now"). The system informs the user while the query proceeds.



Include	Platform	Qty	Scope	Time Range	Color
☑	GPSSONDE_SFC_OCEANIC	7	5 hr	11/04 13:12 to 11/04/2001 17:19	
☑	QSCAT_OCEANIC	3998	2 hr	11/04 10:02 to 11/04/2001 11:44	
☑	GOES_OCEANIC	961	6 hr	11/04 13:02 to 11/04/2001 19:02	
☑	MOORED_BUOY_OCEANIC	263	11 hr	11/04 10:15 to 11/04/2001 20:45	
☑	NOAA_OCEANIC	493	9 hr	11/04 10:07 to 11/04/2001 18:40	
☑	SSMI_OCEANIC	2181	4 hr	11/04 11:47 to 11/04/2001 14:54	
☑	SFMR_OCEANIC	300	8 hr	11/04 10:25 to 11/04/2001 18:12	
☑	GPSSONDE_OCEANIC	10	6 hr	11/04 11:57 to 11/04/2001 17:19	
☑	TMI_OCEANIC	1028	2 hr	11/04 12:13 to 11/04/2001 13:57	
☑	CMAN_OCEANIC	363	10 hr	11/04 10:04 to 11/04/2001 20:04	
☑	SHIP_OCEANIC	892	11 hr	11/04 10:00 to 11/04/2001 21:00	
☐	GOES_SWIR_ATMOS_LVL	617	0 hr	11/04 10:00 to 11/04/2001 10:00	
☐	GOES_SWIR_OCEANIC	614	0 hr	11/04 10:00 to 11/04/2001 10:00	
☐	DRIFTING_BUOY_OCEANIC	362	11 hr	11/04 10:00 to 11/04/2001 21:00	
☐	NOAA_FLT_LVL	624	9 hr	11/04 10:07 to 11/04/2001 18:40	
☐	GOES_ATMOS_LVL	1116	6 hr	11/04 13:02 to 11/04/2001 19:02	
☐	GPSSONDE_MULTI_LVL	182	6 hr	11/04 11:57 to 11/04/2001 17:19	
☐	QSCAT_RAIN_OCEANIC	405	2 hr	11/04 10:02 to 11/04/2001 11:44	

Fig. 15. Current observation panel

I. Changes in TrackView (fig. 16)

- Added 'Source', 'Pressure', 'Height' and 'Edited' columns.
- Added an "Update Track" capability to let the user make all then necessary changes to the track and inform the system when it wishes to update the map. Prior, any change detected in the track resulted on an automatic observation check and map redrawing; somewhat annoying and counterproductive.

- In real-time operations, it is highly desirable to obtain the latest observations and fixes up until it leaves enough time to run an analysis. If there is less than a 6-hour difference between the current UTC time and the latest arrival time among all observations, the two methods for "Checking New Data" are enabled. The user can activate a 10-minute periodic database query ("Auto"), or can perform a one-time query ("Now"). The system informs the user while the query proceeds.

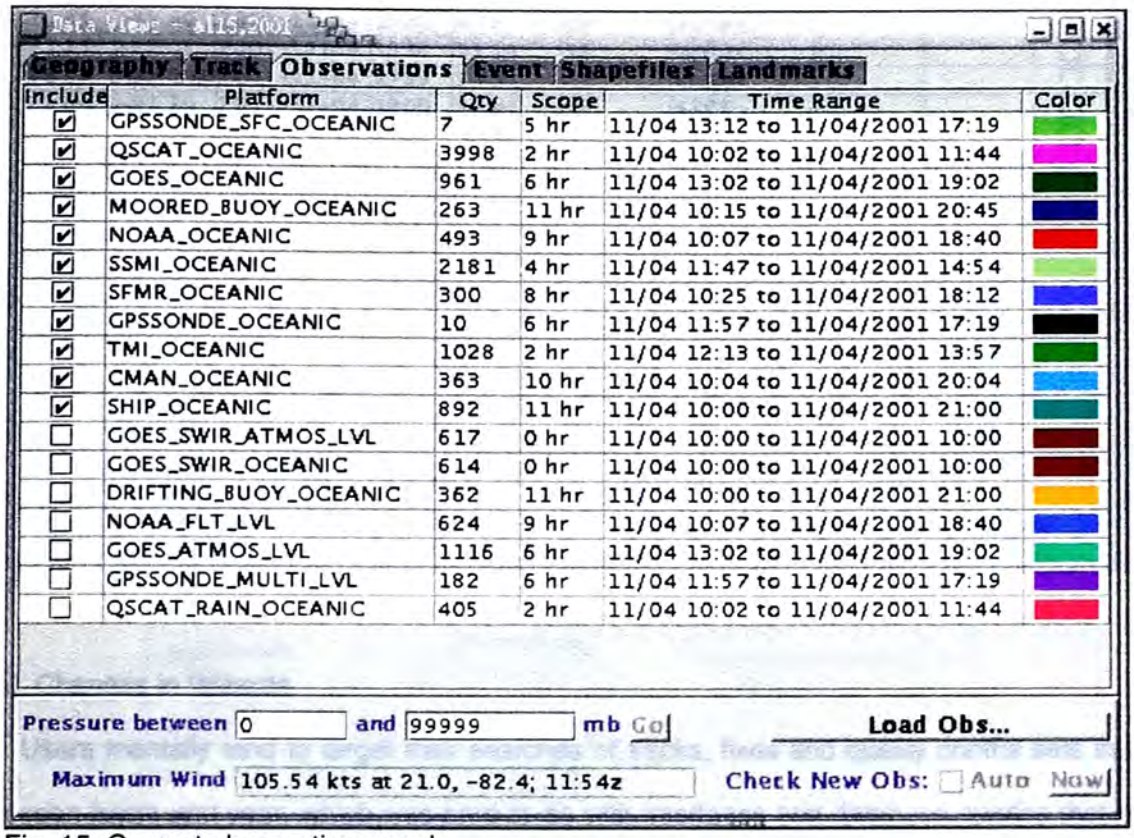


Fig. 15. Current observation panel

I. Changes in TrackView (fig. 16)

- Added 'Source', 'Pressure', 'Height' and 'Edited' columns.
- Added an "Update Track" capability to let the user make all then necessary changes to the track and inform the system when it wishes to update the map. Prior, any change detected in the track resulted on an automatic observation check and map redrawing; somewhat annoying and counterproductive.

- To better visualize the location of the begin, center and end fixes, fonts in those rows are colored with green, blue and red, respectively.

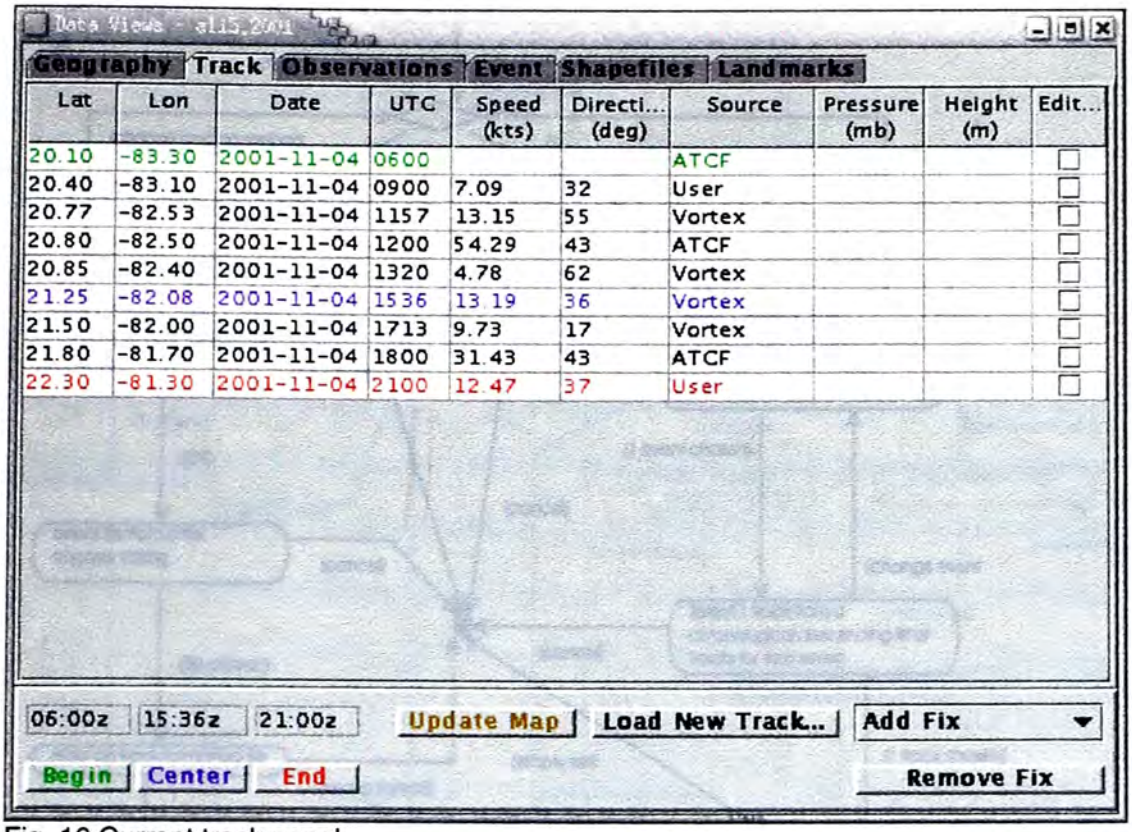


Fig. 16 Current track panel

J. Changes in Wizards

Users mentally tend to target their searches of tracks, fixes and quality control sets in terms of ocean basin and year, which was hard to do with interfaces and database queries that offered a growing list of items as years went on. A common graphical interaction panel and new queries were designed to solve this issue. A good example is to present the process of querying a track (fig. 17).

Initially, the criterion for querying observations was per platform, but it was more intuitive to query per exposure (marine or land), and later by pressure range with the introduction of flight-level data (fig. 18). Panels in WindObservationWizard were restructured. These new criteria had the extra advantage of improving database query performance, since making one database call is better than making multiple ones for answers of equal size.

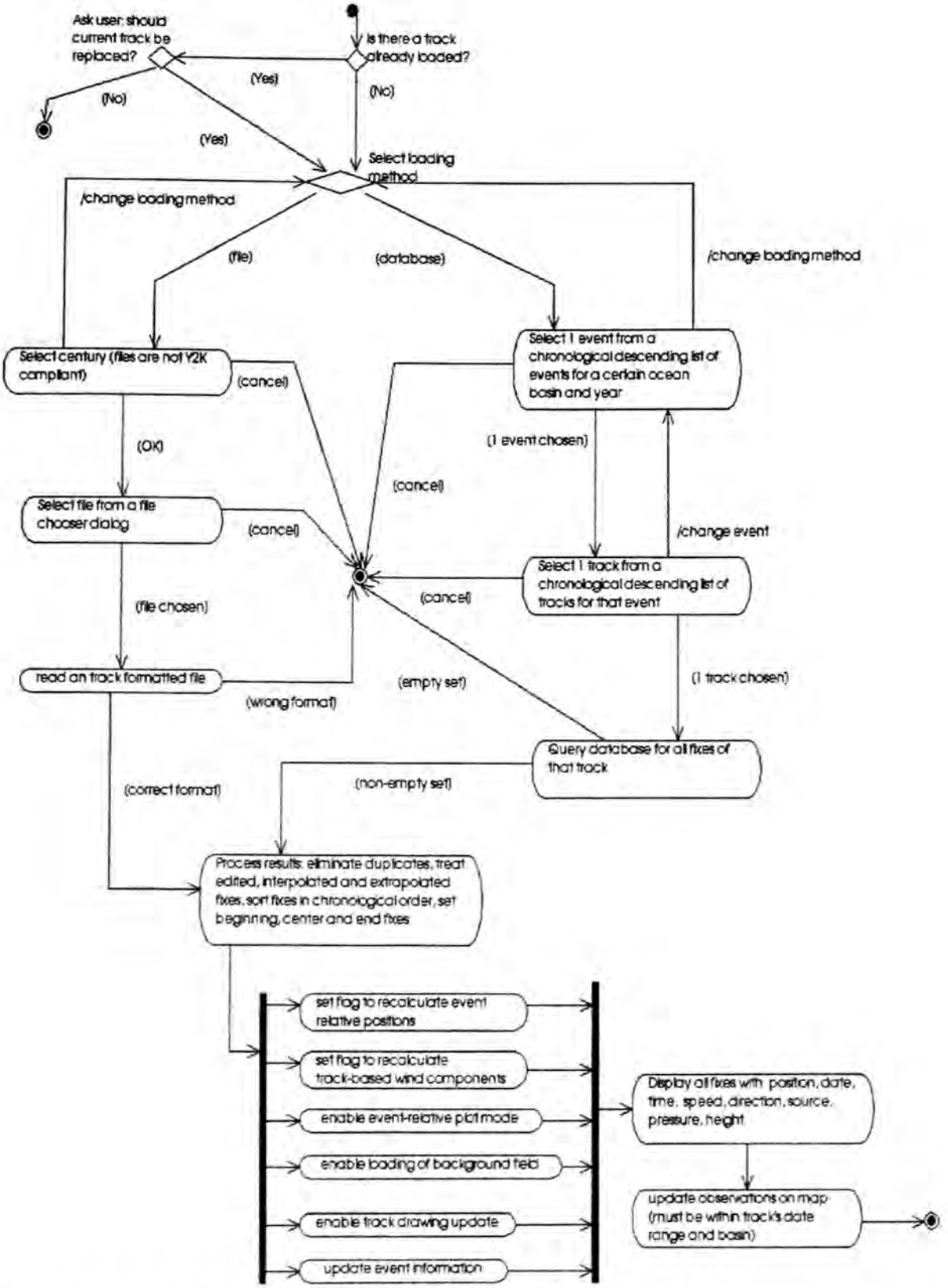


Fig.17 Sequence of events to query a track from database

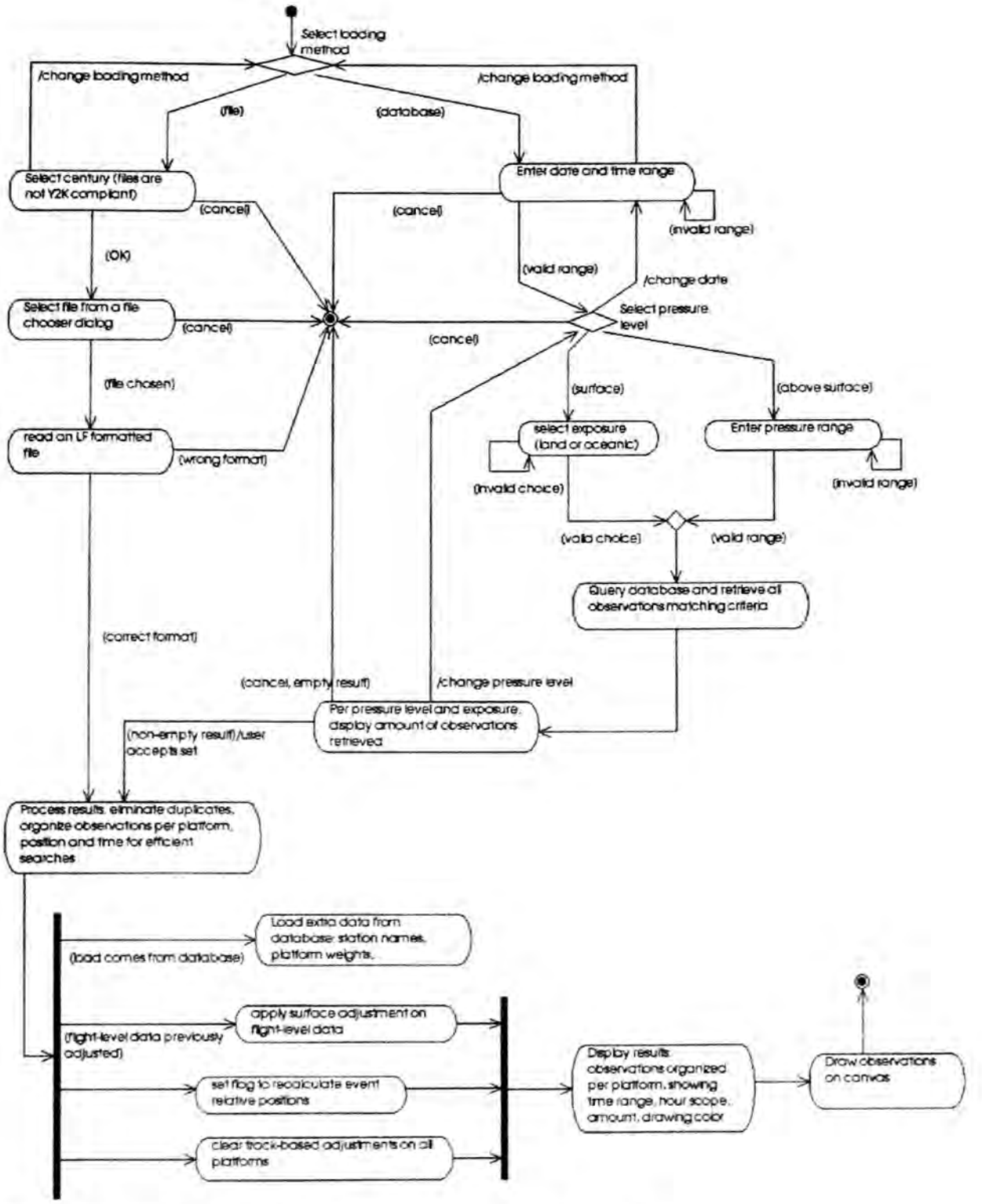


Fig.18 Sequence of events to query observations from database

7.2 Issues raised on the Analysis subsystem

A. Operating system

The FORTRAN legacy code is compiled for a Sun Microsystems Sparc architecture machine, with an operating system Solaris 6 or above.

B. Choice of Java's Remote Method Invocation

The foremost needed task was to prove the feasibility of object distribution with legacy code. Design is the phase for experimentation. The release of Java 2 Standard Development Kit (SDK) introduced a new improved mechanism to remote object invocation via RMI, one in which an instance of the server object did not have to run all the time; rather, its implementation could be registered with an RMI daemon ('rmid') and its stub or remote reference with 'rmiregistry', RMI's naming directory service. Rmid provides a Java Virtual Machine (JVM) from which other JVM instances can be spawned "on demand". Figure 19 shows the consequence of this approach.

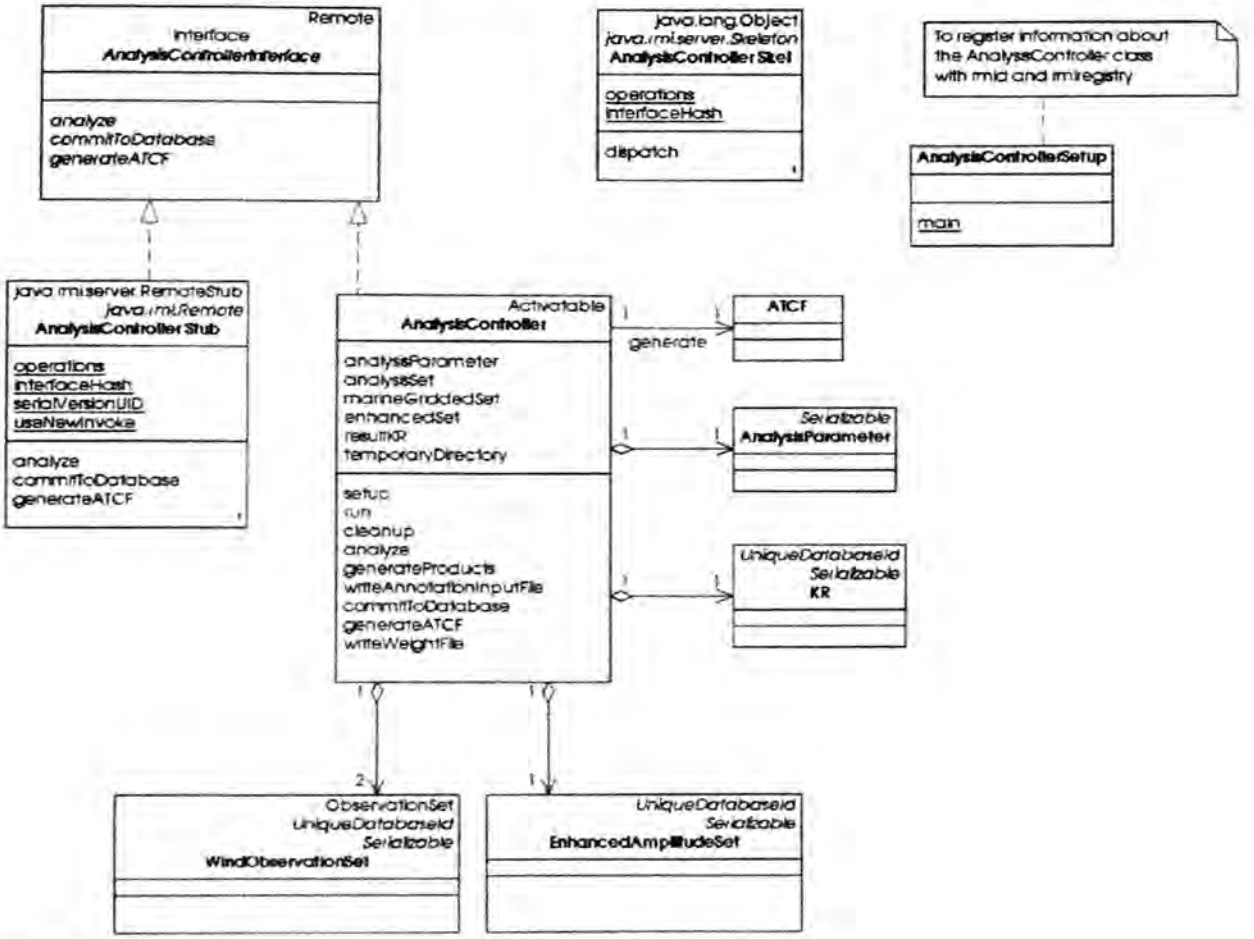


Fig. 19 Class diagram of the analysis distributed object

The interface 'AnalysisControllerInterface' must extend Java's Remote interface and declare our beloved operations: schedule an analysis given the necessary parameters, store an analysis in the database associated to the respective quality control set, and generate an ATCF file. The class 'AnalysisController' implements this interface and must extend from Java's abstract class 'Activatable'. The registration job occurs in the 'AnalysisControllerSetup' program, which must be executed every time the implementation is updated. A client finds a remote reference by looking up a registry with a URL of the form *rmi://hostname:port/ServiceName.*

C. Code reuse

As a subsystem being integrated with another, code reuse is sought after. One of Java's RMI notable advantages is the ability to pass objects as parameters across the network (just by implementing Java's 'Serializable' interface); therefore, needless to say, the same classes that make up a quality control set are the ones sent to the analysis server object. Figure 20 depicts the sequence of steps that occurs at each server object invocation.

In order to preserve CORBA's integration capabilities with other potential applications, the analysis IDL interface relies on basic data types (byte, integer, boolean) as arguments to operations, as shown below:

```
module AnalysisController {
  interface AnalysisInterface {
    typedef sequence<octet> Data;

    void initialize();
    void setBogusInput(in Data input);
    void setObservationInput(in Data input);
    void setStormTrackInput(in Data input);
    void setPrenestInput(in Data input);
    void setWeightInput(in Data input);
    boolean analyze();
  };
};
```

D. Database integration

Database storage is straightforward upon completion of the OO analysis phase. The resulting database schema supersedes that found in [10].

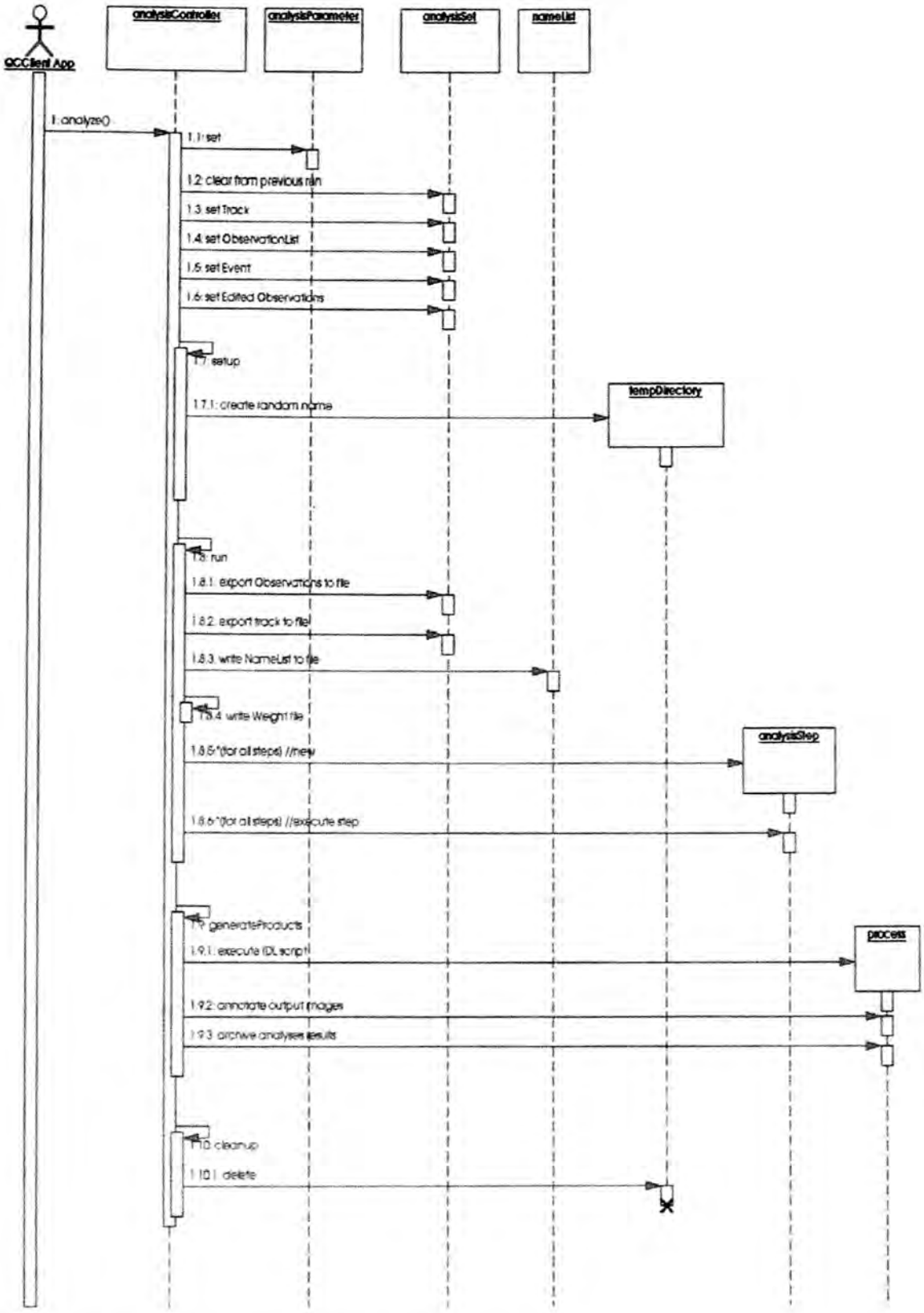


Fig. 20 Event sequence diagram of the analysis process

E. Concurrency conflicts

The HSA nature of dealing with files posed a greater challenge: each analysis step expects to act on equally named input files whose contents are tailored for each specific run, and these files are expected to be located on the filesystem directory where the program responsible for a certain step was initially executed. The danger of this situation is that two or more simultaneous users scheduling an analysis would find themselves with unreliable adulterated output, due to a blend of randomly overwritten input files, as each step takes an unpredictable amount of time of completion. In the past, it was easy for the small pool of scientists at HRD to always ensure a single processing environment. But, undoubtedly, this situation is intolerable in a global distributed context, where flexibility is a must. To solve this conflict, a random number generator was employed as a means to name a temporary directory for each scheduled analysis, and force it to be that analysis' execution environment throughout all steps. Even though it is extremely unlikely that the number generator produces repeated output, in order to guarantee absolute conflict avoidance, an existance directory check is performed for each name generated. If it exists, then the previously used directory is removed and recreated from scratch, thus allowing proper analysis execution and safe results. This method guarantees unique products for multiple users interacting with the Analysis engine at any given single time.

F. Unanticipated problems

An unforeseen consequence of the legacy code (not fully documented) was the discovery that data platform ids considered by HSA are deeply hardcoded into Prenest and Nestanal programs, accepting a maximum of 24 platforms, where 2 are reserved (bogus points and background field). One of the expected input files is called the 'weight file', which is simply a listing of several columns, the most important being the one specifying a data platform id and another specifying its weight (real number in the range of [0,1]). Initially, thinking that as long as I complied with the file format, the contents could be dynamic, I naively used the same platform id stored in the database, which resulted in observations (bogus points and background field). One of the expected input files is called the 'weight file', which is simply a listing of several columns, the two

most important being the one specifying a data platform id and another specifying its weight (real number in the range of [0,1]). Initially, thinking that as long as I complied with the file format, the contents could be dynamic, I naively used the same platform id stored in the database, which resulted in observations of those platforms to be completely ignored or misinterpreted. The database already contains data belonging to more than 22 platforms, and the number keeps growing. Project resources were not available to allow an HRD scientist to modify the cumbersome FORTRAN programs to accept diverse observation platforms. Therefore, the solution was to establish a correspondence table of weights to observation platform ids, as shown in following table.

weight	maps to id
1.0	1
0.95	2
0.9	3
0.85	4
0.8	5
0.75	6
0.7	7
0.65	8
0.6	9
0.55	10
0.5	11
0.45	12
0.4	13
0.35	14
0.3	20
0.25	21
0.2	22
0.15	23
0.1	24
0.05	25
0.025	26
~ 0	52

8. IMPLEMENTATION

Every single Java class of the original prototype has undergone thorough revision over a three-year period. With so many repair choices, I decided to attack in a prioritized order, giving urgency to the most basic tasks, many of them intertwined in substance and chronology:

1. Establish reliable communication with the database (via Java Database Connectivity, JDBC) for both insertion and querying of observations and storm track fix positions. This is indispensable for real-time data collection.
2. Impose the extension of the Java class 'DefaultTableModel' to control all editing manipulations, for all tabular forms of input data in the graphical user interface. This allowed Instances of creating edited observations and storm track fixes to be recorded truthfully.
3. Provide a basic AnalysisWizard for scheduling an elementary (only surface wind) analysis of a quality control set, without expert parameterization.
4. Enable storage of a quality control set (QC set). Poor performance under JDBC required that the database developer provide an equivalent implementation using SQLJ (SQL embedded in Java), which tremendously reduced QC set storage the time from hours to minutes.
5. Convert all components of the graphical interface (buttons, checkboxes, panels, drop-down lists, frames, etc.) into Swing components, a lightweight version of the window toolkit that guarantees same look-and-feel regardless of the JVM being used, vital for global deployment.
6. Apply the most complicated and flexible of Java's window manager layouts, GridBagLayout, to all GUI window panels. Positions and sizes of all GUI components (which were hardcoded) are now relative to the size of their container, the top one being relative to the size of the monitor screen. Again, this neutralizes visualization issues in any environment.
7. Study the memory consumption of the application, which in my opinion, abused of excessive object instantiation, when the same procedure could be accomplished by reusing existing variables. A total of around 35MB spared.

8. Lower the resolution of the contour maps up to an acceptable level to accelerate their drawing.
9. Evaluate scenarios in which a total canvas redrawing is not necessary, such as when changing the color of a platform, or when editing one specific observation.
10. Incorporate the drawing symbols to distinguish, per platform, the observation with the highest maximum wind speed (a triangle) and the observation with most recent arrival time (a square). See figure 21. If any such observation happened to be flagged by the user, a new one had to be found and drawn among the rest of passed observations.

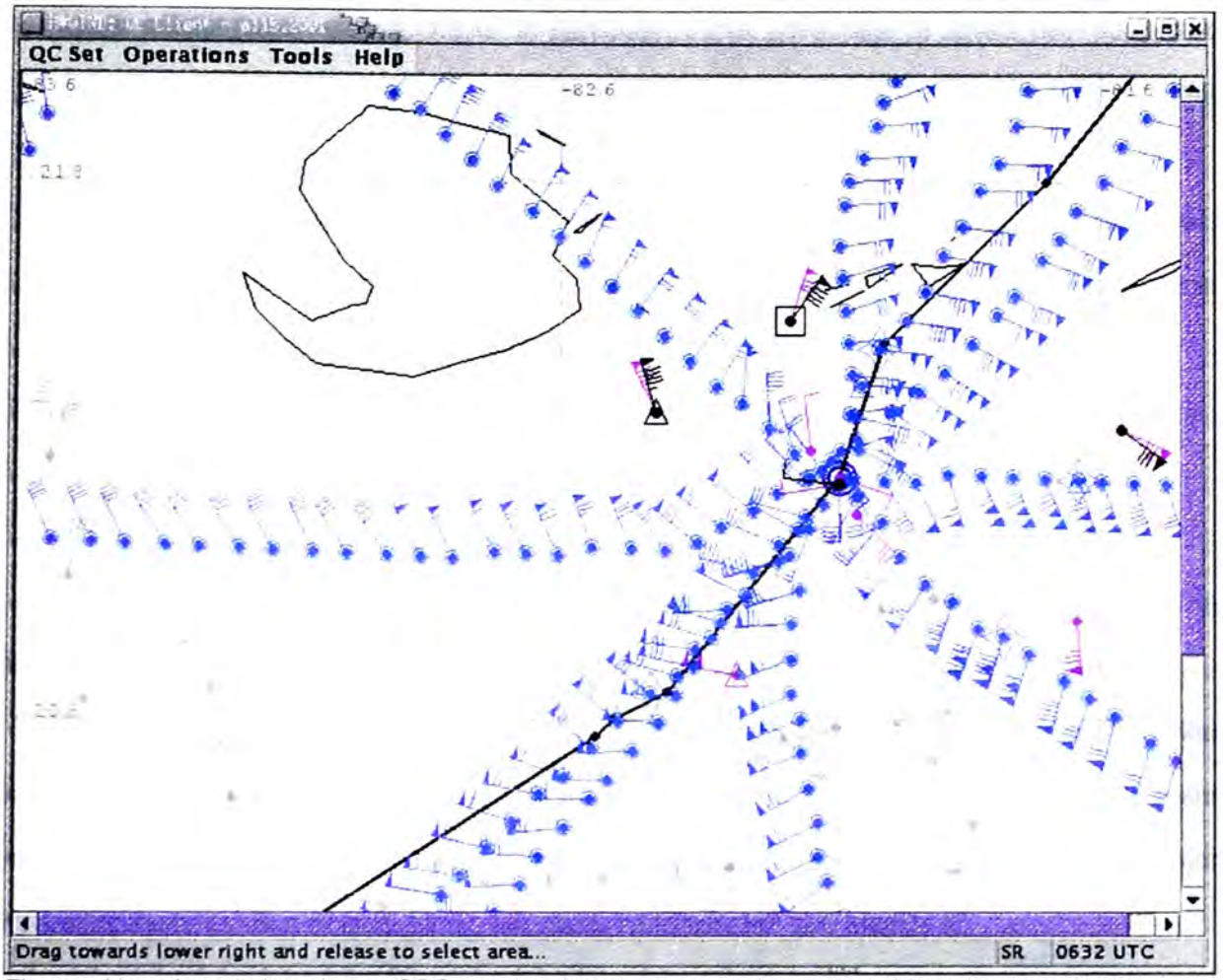


Fig. 21 Aircraft path (blue) and GPS sondes (black, magenta) on storm Michelle, Nov. 2001

11. Expand AnalysisWizard to add panels to input analysis expert parameters, as indicated in use case 5.3, including mesh drawing. The ease for selecting above-surface pressure levels or non-wind types of analysis improves the efficiency of streamlining analysis procedures.

12. Enable multiple-observation inspection per unique geographic position.
13. Storage of analysis results in database via SQLJ.
14. Translate the Mean Boundary Layer model from C to Java, fundamental part of most scientist-designed surface adjustment algorithms. Directly associated with this, implement Dunion-Powell's flight-level surface adjustment algorithm. Notice in figure 21 the star-shaped polygons indicating those observations of maximum wind speed per storm-relative quadrant.
15. Add the possibility to store and load a quality control set containing a background field.
16. Along with the evolution of the Java Standard Development Kit (SDK), update source code to latest official major release, 1.3.x. In particular, I took advantage of Java's own supported sorting procedures instead of using original bubble-sort algorithm. The classes that constitute the ordering key implement the 'Comparable' interface.
17. Take advantage of separate threads of execution whenever concurrent activities behooves. For example, the user is able to continue fine-tuning data quality control while an analysis is in progress or during a check for newly arrived data.
18. Pervasive error checking, corrections of logic and miscalculations.

An accomplished application without easy vast deployment capabilities could very likely fall in the darkness. One of the top priorities of this project is to provide global application access.

Java Web Start, designed to launch full-featured applications from a Web browser, and provide centralized software management, satisfies the requirement of Internet-wide deployment. Users only need to install once a Java Virtual Machine and Java Web Start, both available for all major operating systems: Sun Solaris, Linux, Microsoft Windows, HP-UX, MacOs X.

If the resources are not present in the client or are in need of update, Java Web Start takes care of transparently downloading all the necessary resources (archived in JAR files). Otherwise, the application is launched immediately, by just clicking on an HTML link. The first-time activation is lengthy, but the benefits of availability of a highly interactive quality control client to weather

forecast agencies, world-wide, outweigh the disadvantage of initially waiting several minutes to download the most current version of the application.

By default, Java Web Start will run an application in a secure sandbox or restricted environment, but H*Wind needs access to local resources (disk and network), to perhaps load observations from a file, connect to our database server (which is not the same host JAR files originate from), or to invoke a remote object via RMI. Code signing is an important security feature of Java Web Start. Digital code signing guarantees that JAR files have not been tampered with since they were last signed. Java Web Start will not run an application if it detects a signature compromise; thus, users can trust the application's source. Java 2 SDK includes a 'jarsigner' tool to sign JAR files. In addition, another advantage of Java Web Start is accommodation of firewall proxy settings, since it is designed to work closely with HTTP traffic.

H*Wind's official launching pad resides in http://cat5.nhc.noaa.gov.

9. OBJECT-ORIENTED TESTING

Exhaustive testing has been an integral part of this thesis, but it usually is the most conspicuous loss on the material covered by technical literature, with just a brief mention. It is fair to say that the authors' expertise serves a much better purpose focused on the more critical sections of object-oriented analysis and design. But, I can certainly attest in favor of the premise that granularity or modularity imposed by object orientation is indeed a major benefit of this methodology. Furthermore, encapsulation serves an extra purpose of localizing and confining runtime errors and exceptions, which helps maintain execution stability by not compromising unaffected sections. Though possibly underrated, the consequences of these characteristics greatly help a programmer's morale in solo mode during a lengthy project, by being able to provide partial but effective working versions of the system with tangible evidence of progress.

I cannot stress enough the importance of testing with real scenarios and data starting at the very early stages of the project, due to the exponential growth of intertwined sequences of execution caused by the interaction between the user and the system. Object oriented testing should start as soon as a few objects collaborate. Object message-driven interaction with its non-linear unpredictable behavior clearly and simply demands fierce integration testing from the bottom-up. Without the desire of diminishing OO well-known advantages, there are several factors that can complicate matters: separation of specification (interface) and actual implementation, and inheritance. Interfaces can be supported by multiple implementations, and subclasses can operate in a different context by having inherited methods and attributes overriding the parent definition. Testing yielded changes on certain inheritance relationships where the parent class included too much specialization for some of its subclasses, and therefore, the actual commonality needed to be streamlined. Testing revealed much optimization in terms of searches and explosive memory allocation. In summary, the massive repetition of the software life cycle proved successful, whether it affected all or a subset of the stages. Though I have not kept an exact count, I estimate I must have performed over a hundred developmental iterations in the last

three years, of varying breadth and depth. Each one has been pivotal for continuous rational refinement on subsequent ones.

The distributed nature of this system in four subsystems takes integration testing a step further, in a more complex multi-threaded environment. Delaying serious thorough testing and debugging to more advanced stages of implementation quickly reaches an unmanageable situation in which much more time and resources would need to be devoted to achieve a given level of reliability.

One of the best testing techniques was to follow the use cases extracted during the analysis phase. Use cases provide a basic infrastructure as a systematic pattern for a task that tends to be approached randomly and superficially. Use cases explain exactly what should happen, pre- and post-conditions explain what the state should be before and after each main activity. They are all an excellent source to check that indeed the user requirements have been met, almost in a contractual form. Each use case can be considered a unit for the integration testing of all its involved objects, as an independent branch ideally designed for the developer to concentrate on its correctness by exhaustively attacking that single issue from many angles and preventing unacceptable behavior (i.e. bugs). It is basically an exercise in tightening the rules of each method in each activity to yield exact expectations under universal circumstances. Following this strict policy will facilitate and encourage reuse of classes, which is another main objective of this object-oriented project. As a result, since other use cases will undoubtedly depend on previously tested units (even if only partially), the amount of testing remains relatively constant but progressively less in proportion to the growth of functionality.

Another ingredient for success has been the close relationship of collaboration between the developer and a group of committed scientists (primary users), pursuing the realization of their vision. User requirements were elicited with a great deal of detail and constant feedback was received on improvements, errors, and help on setting priorities. The software has become a much more robust, powerful, and diversified product with much more potential than originally envisioned.

10. SUMMARY AND FUTURE WORK

The transformation of H*Wind application is very much a reality at the Hurricane Research Division, which has witnessed a series of distinct accomplishments with every release. The application received the "Best Java Implementation" award from the High Performance Computing and Communications (HPCC) NOAATech 2000 conference, held in Silver Spring, Maryland, in October 1999, and the "Best Technology Transfer to Operations" award at NOAATech 2002, in October 2001. In 2001, this application was rated by the National Hurricane Center as its highest priority research tool to be transferred to their forecast operations, and so we are involved in a two-year transition effort. The feedback of hurricane specialists and forecasters is being integrated in the application. Strong interest has been received from the Central Pacific Hurricane Center and the Department of Defense Joint Typhoon Warning Center. I can safely state decisions regarding adherence to a database, to object-orientation via Java, and to distributed objects technology have proven very suitable for achieving the project goal.

The next frontier for H*Wind is to become compliant with the Java 2 Enterprise Edition (J2EE) architecture. Currently, the quality control client is too "fat" for the Web, meaning it contains the built-in logic to control user interaction, to perform requested algorithms and to access database and remote objects. The client application is not a trivial size download, and users still suffer from noticeable network performance degradation depending on the bandwidth and/or distance from H*Wind's production database and analysis servers.

With the incorporation of a middle-tier application server, presentation, scientific logic, and access to remote objects and data can be clearly separated into independent components. Thus, the complex graphical user interface would be converted into a light-weight applet (just for mouse/keyboard handling), and the routing of user requests would take the form of Java servlets, which would dispatch the actual processing to Enterprise Java Beans (EJB), responsible as well for the access to remote objects and databases. The aim will be to follow the appropriate J2EE design patterns. J2EE containers provide services and resources (transaction, security, deployment, naming, distribution) that allow applications to be flexibly customized. Benefits of

this approach include greater system performance, availability, scalability and manageability. All that clients need is a web browser, resulting in a significant reduction of network traffic. One of the most positive advantages is that the vast majority of code already proficiently tested can be reused. A potential outcome will be to make tropical cyclone observations and analysis accessible to developing countries.

BIBLIOGRAPHY

[1] Powell, M. D., 1980: "Evaluations of diagnostic marine boundary-layer models applied to hurricanes". Monthly Weather Review, vol. 108, no. 6, 757-766.

[2] Lord, S. J. and J. L. Franklin, 1987: "The Environment of Hurricane Debby (1982). Part I: Winds". Monthly Weather Review, vol. 115, No. 11, 2760-2780.

[3] Ooyama, K. V., 1987: "Scale-Controlled Objective Analysis". Monthly Weather Review, vol. 115, No. 10, 2479-2506.

[4] DeMaria, M., Aberson, S. D. and K. V. Ooyama, 1991: "A Nested Spectral Model for Hurricane Track Forecasting". Monthly Weather Review, vol. 120, No. 8, 1628-1643.

[5] R. W. Burpee, S. D. Aberson, P. G. Black, M. DeMaria, J. L. Franklin, J. S. Griffin, S. H. Houston, J. Kaplan, S. J. Lord, F. D. Marks, Jr., M. D. Powell, and H. E. Willoughby, 1994: Real-time guidance provided by NOAA s Hurricane Research Division to forecasters during Emily of 1993. Bulletin of the American Meteorological Society, 75, 1765-1783.

[6] Powell, M. D. and S. H. Houston, 1996: "Hurricane Andrew's Landfall in South Florida. Part II: Surface Wind Fields and Potential Real-Time Applications". Weather Forecast., 11, 329-349.

[7] M. D. Powell, S. H. Houston, L. R. Amat, and N. Morisseau-Leroy: "The HRD Real-time Hurricane Wind Analysis System". 8th US National Conference on Wind Engineering Conference Proceedings, 1997.

[8] Franklin, J. L., 1994: "Documentation for the HRD Spline Analysis Programs Prenest and Nestanal". Internal Document available from NOAA/AOML/HRD, 4301 Rickenbacker Causeway, Miami, FL 33149.

[9] Sampson, C. R., A. Schrader, "The Automated Tropical Cyclone Forecasting System (Version 3.2)", Bulletin of the American Meteorological Society, 81, 1231-1240.

[10] N. Morisseau-Leroy, "Atmospheric Observations, Analyses, and The World Wide Web Using a Semantic Database", Master Thesis, School of Computer Sciences, Florida International University, Miami, FL, 1997.

[11] Amat, Luis R. Jr., "A Realtime Internet Based Quality Control Application for Hurricane Surface Winds", Master Thesis, School of Computer Sciences, Florida International University, Miami, FL, 1998.

[12] E. Yourdon, "Object-Oriented Systems Design: An Integrated Approach", Prentice-Hall, Englewood Cliffs, NJ, 1994.

[13] R. Orfali, D. Harkey, J. Edwards, "The Essential Distributed Objects Survival Guide", John Wiley & Sons, New York, 1996.

[14] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, "Object-Oriented Software Engineering, A Use Case Driven Approach", Addison-Wesley, 1996.

[15] Schneider, G., Winters, J. P., "Applying Use Cases: a Practical Guide", Addison-Wesley Object Technology Series, 1998.

[16] Krutchen, P., "The Rational Unified Process: An Introduction", Addison-Wesley Object Technology Series, 2nd edition, 2000.

[17] Fowler, M., "UML Distilled", Addison-Wesley Object Technology Series, 2nd edition, 2000.

[18] Muller, P., "Instant UML", Wrox Press Ltd., 1997

[19] Vinoski, S., "New Features for CORBA 3.0", Communications of the ACM, vol. 41, no. 10, October 1998.

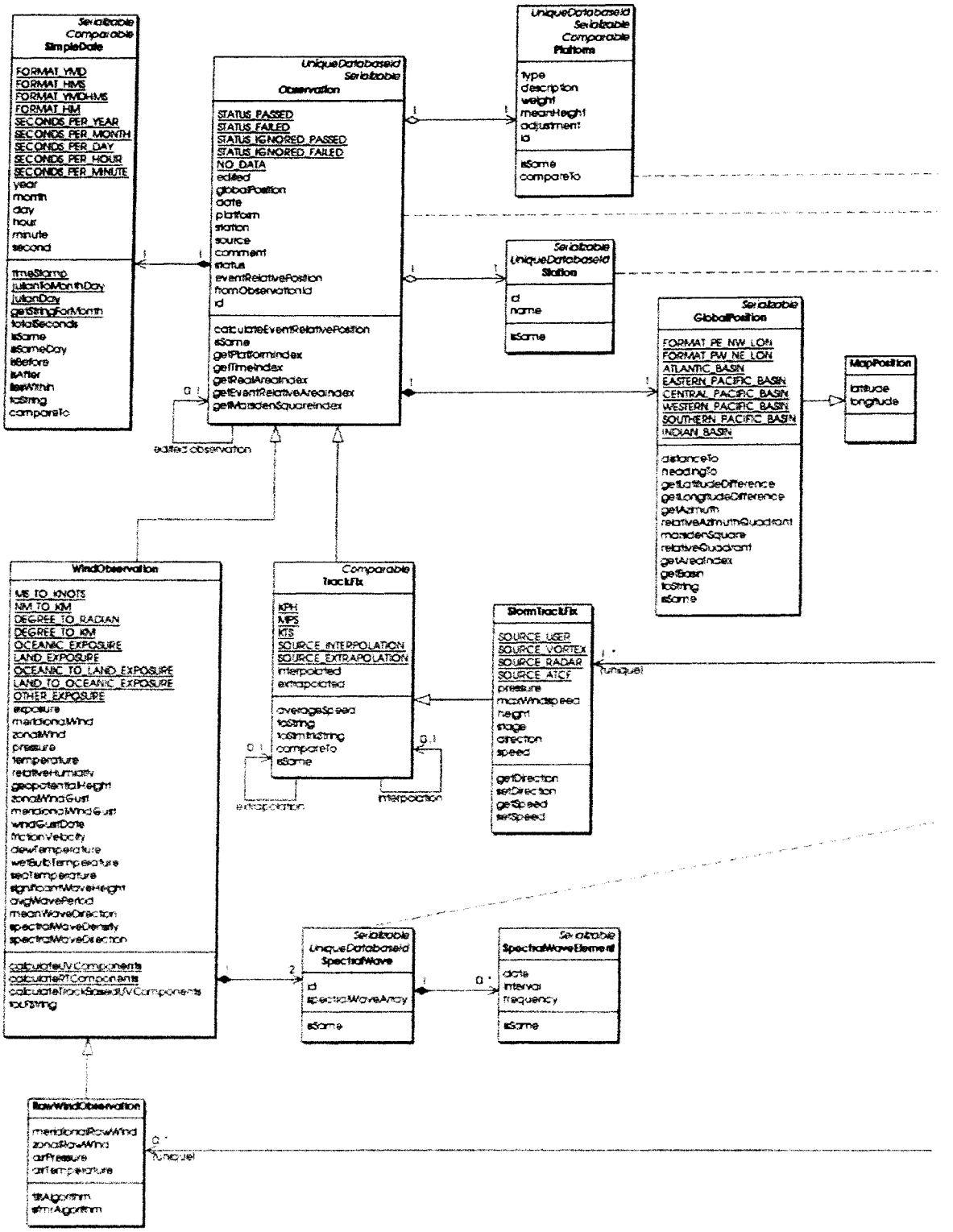
[20] Curtis, D., "Java, RMI and CORBA", http://www.omg.org/library/wpjava.html

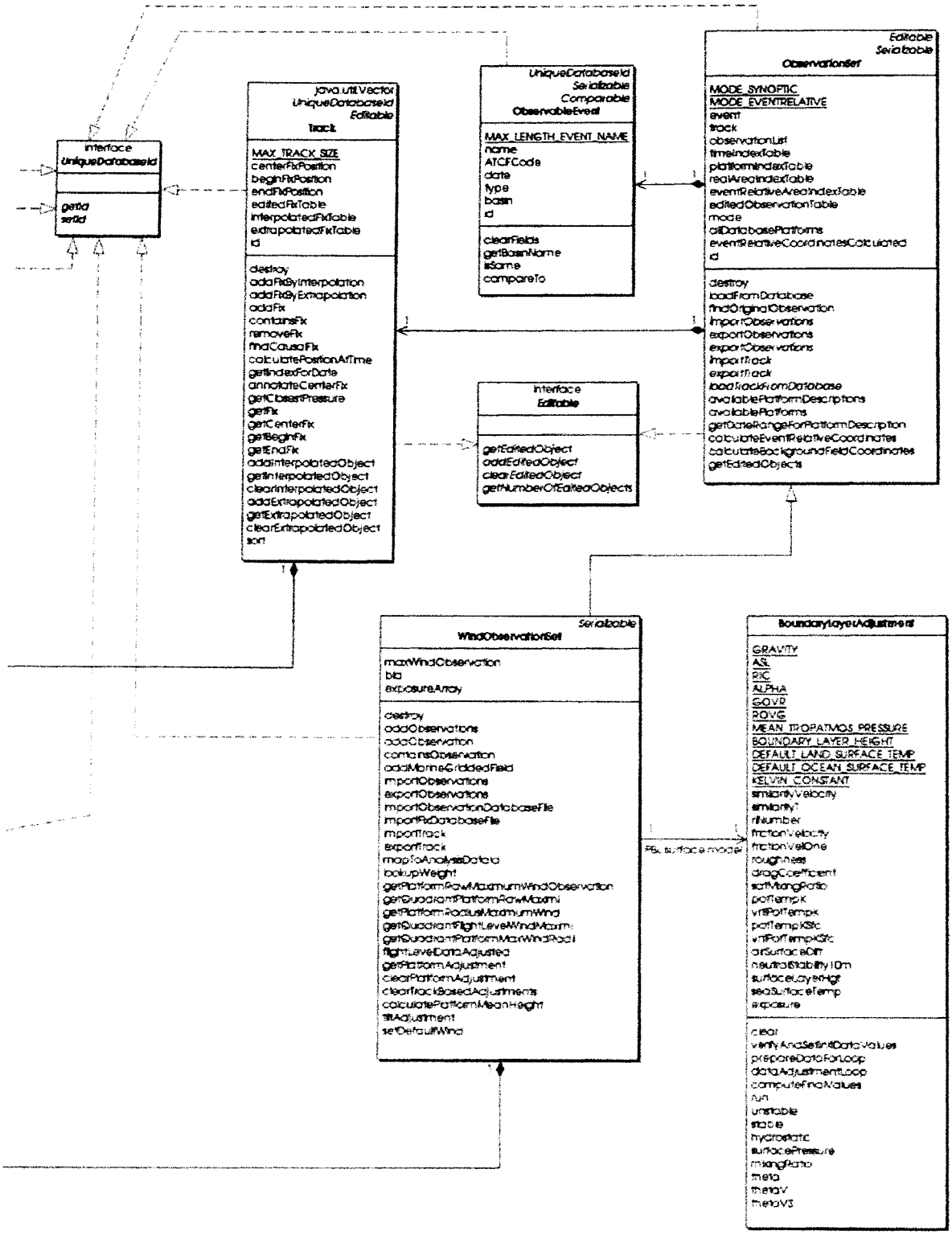
[21] Abie, H., "CORBA Firewall Security: Increasing the Security of CORBA Applications", http://www.ifi.uio.no/~abie/fw.pdf, 2000

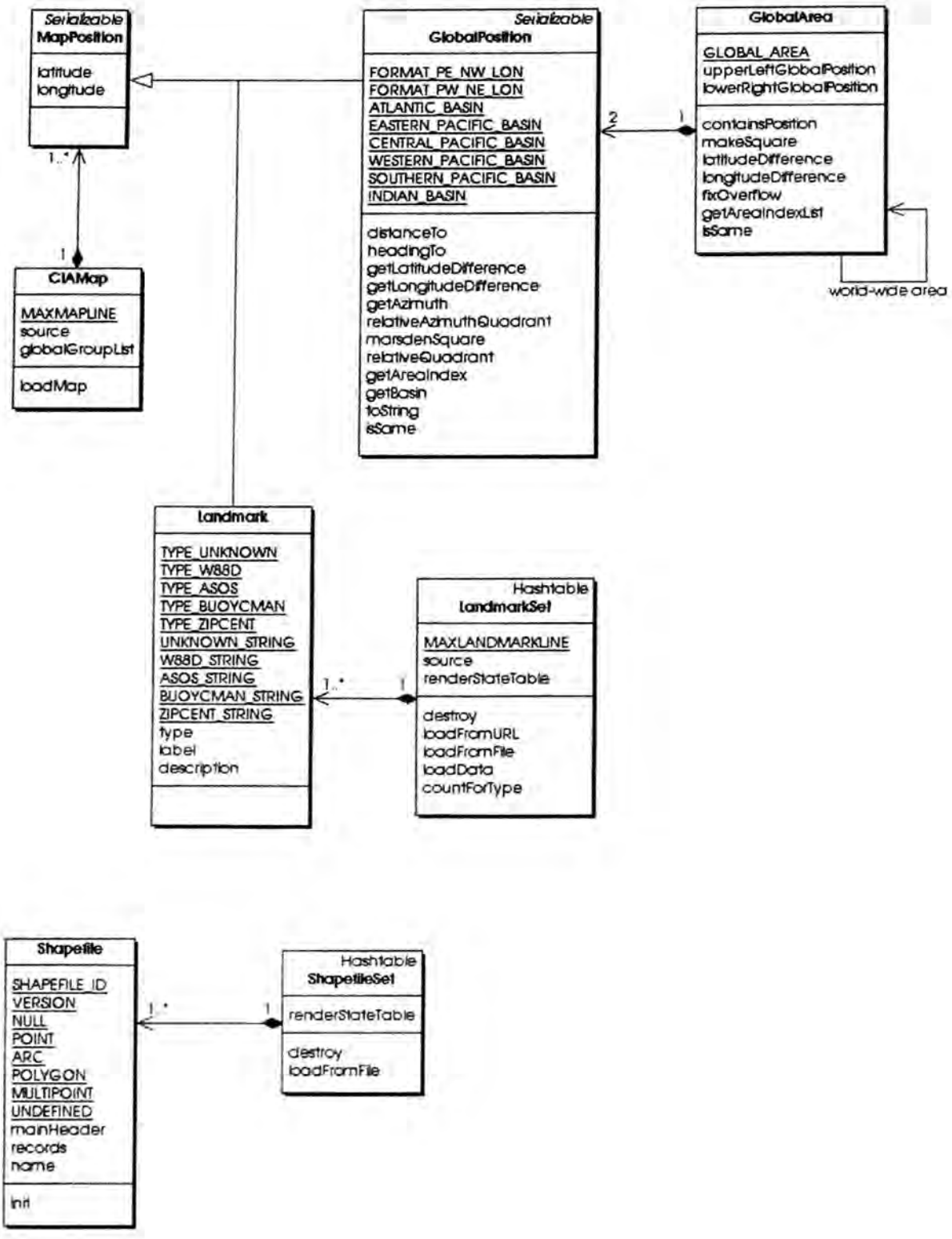
[22] "Java Web Start: Architecture", http://www.java.sun.com/products/javawebstart/architecture.html, 2002

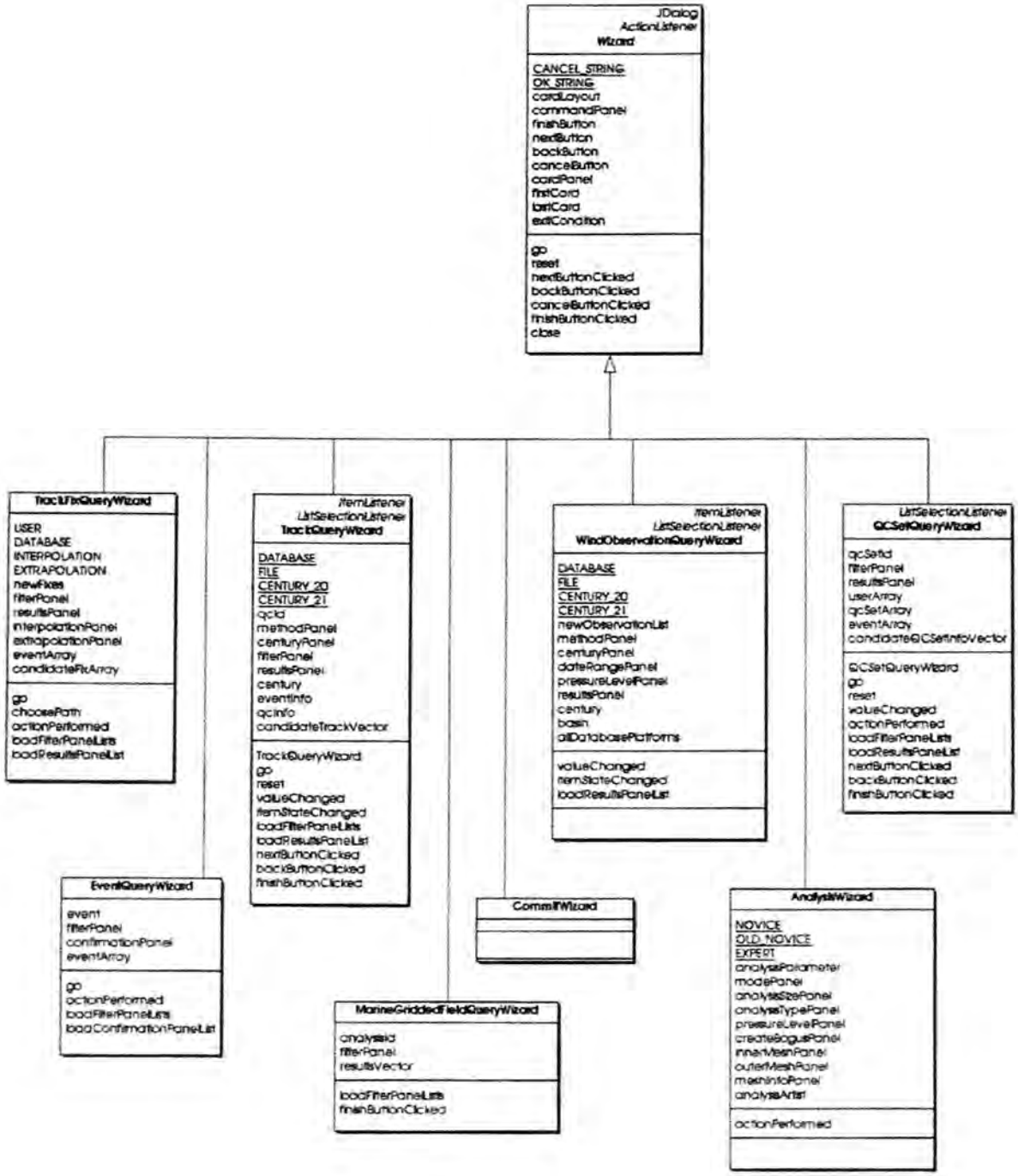
APPENDIX A

Class diagrams fundamental to the application.

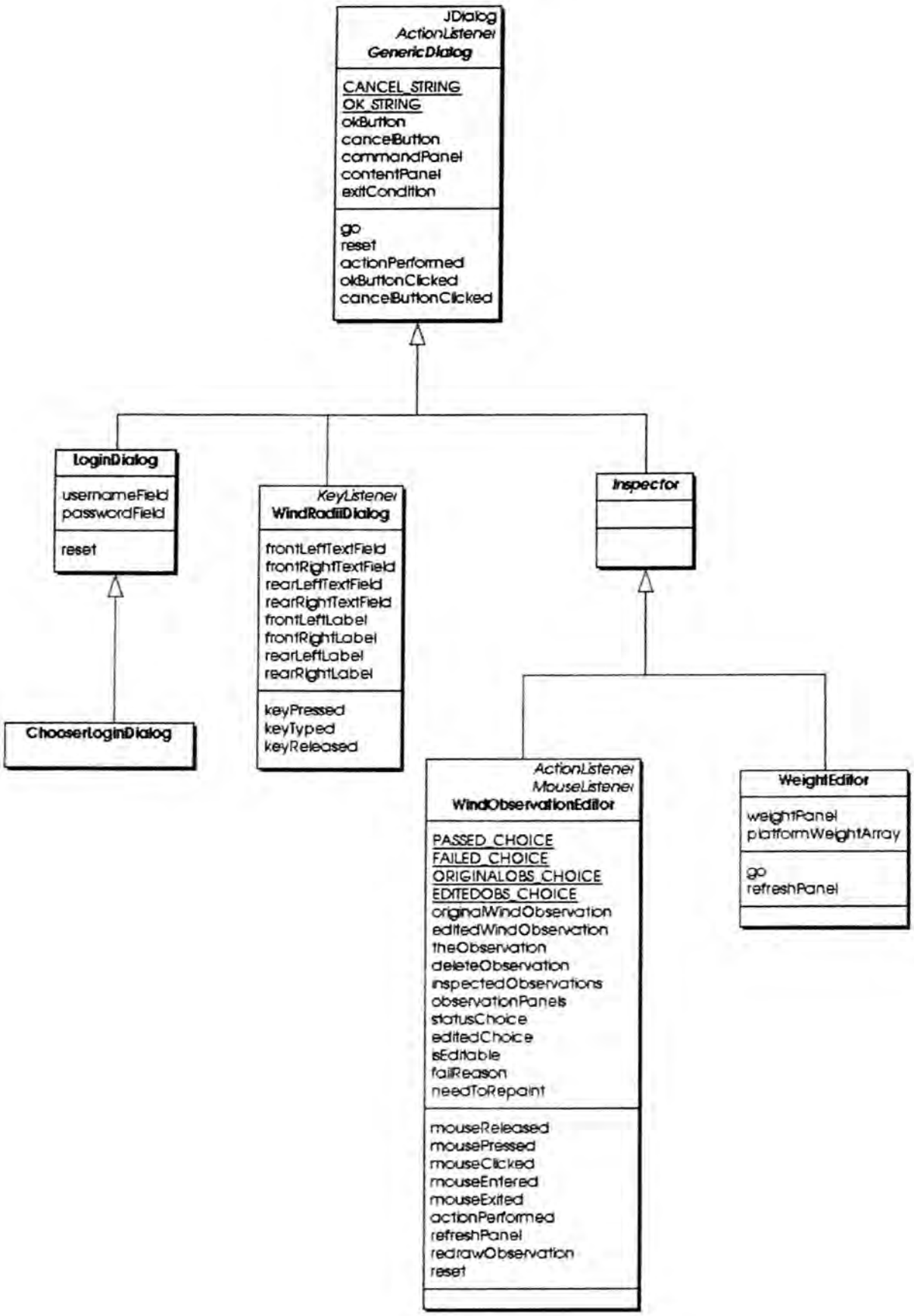


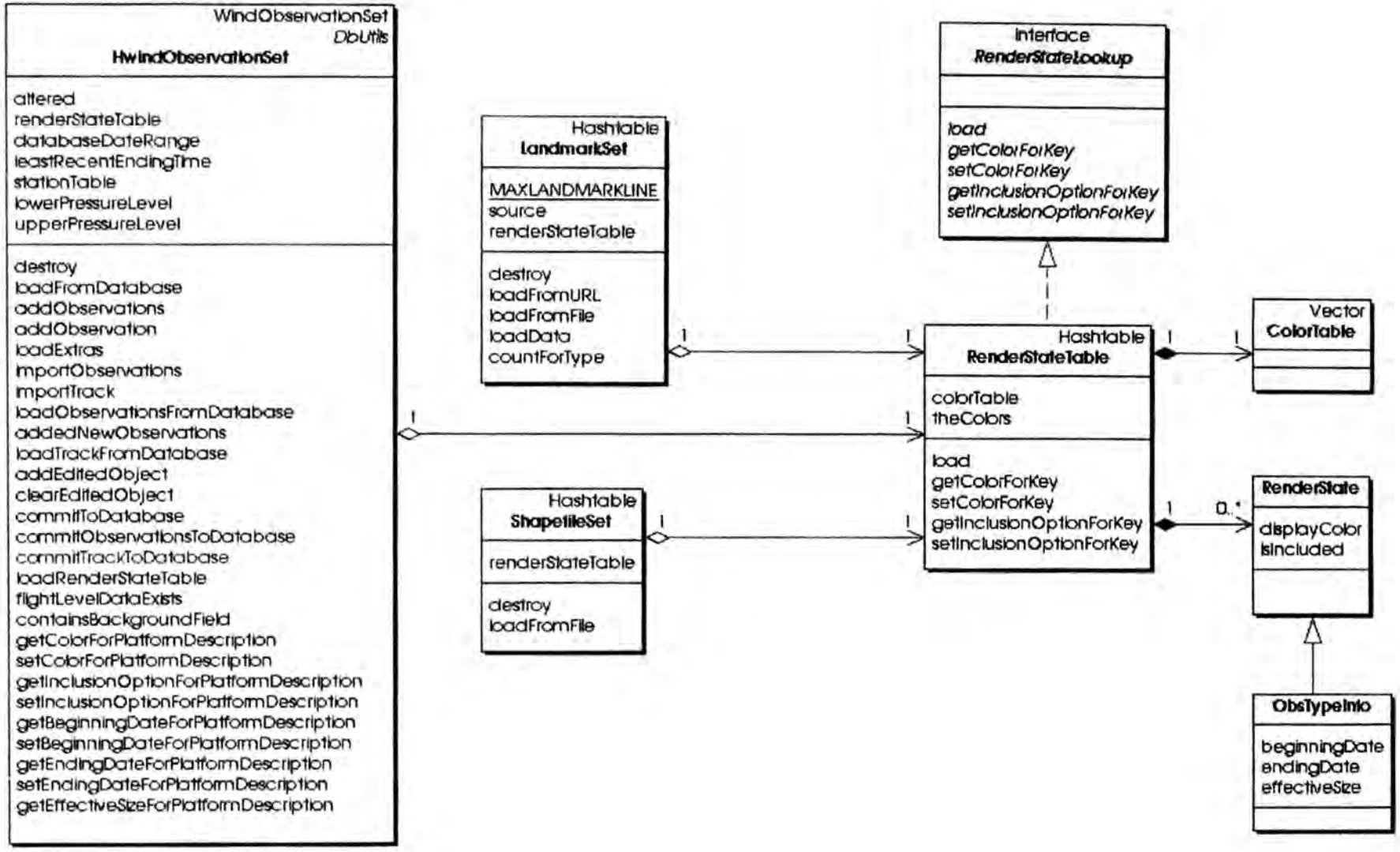

Interface
UniqueDatabaseId
getId
setId
java.util.Vector
UniqueDatabaseId
Editable
Track
MAX_TRACK_SIZE
centerFixPosition
beginFixPosition
endFixPosition
editedFixTable
interpolatedFixTable
extrapolatedFixTable
id
destroy
addFixByInterpolation
addFixByExtrapolation
addFix
containsFix
removeFix
findCausalFix
calculatePositionAtTime
getIndexForDate
annotateCenterFix
getClosestPressure
getFix
getCenterFix
getBeginFix
getEndFix
addInterpolatedObject
getInterpolatedObject
clearInterpolatedObject
addExtrapolatedObject
getExtrapolatedObject
clearExtrapolatedObject
sort
UniqueDatabaseId
Serializable
Comparable
ObservableEvent
MAX_LENGTH_EVENT_NAME
name
ATCFCode
date
type
basin
id
clearFields
getBasinName
isSame
compareTo
Editable
Serializable
ObservationSet
MODE_SYNOPTIC
MODE_EVENTRELATIVE
event
track
observationList
timeIndexTable
platformIndexTable
realAreaIndexTable
eventRelativeAreaIndexTable
editedObservationTable
mode
allDatabasePlatforms
eventRelativeCoordinatesCalculated
id
destroy
loadFromDatabase
findOriginalObservation
importObservations
exportObservations
exportObservations
importTrack
exportTrack
loadTrackFromDatabase
availablePlatformDescriptions
availablePlatforms
getDateRangeForPlatformDescription
calculateEventRelativeCoordinates
calculateBackgroundFieldCoordinates
getEditedObjects
Interface
Editable
getEditedObject
addEditedObject
clearEditedObject
getNumberOfEditedObjects
Serializable
WindObservationSet
maxWindObservation
bia
exposureArray
destroy
addObservations
addObservation
containsObservation
addMarineGriddedField
importObservations
exportObservations
importObservationDatabaseFile
importFixDatabaseFile
importTrack
exportTrack
mapToAnalysisGrid
lookupWeight
getPlatformRawMaximumWindObservation
getQuadrantPlatformRawMaxim
getPlatformRadialMaximumWind
getQuadrantFlightLevelWindMaxim
getQuadrantPlatformMaxWindRadi
flightLevelDataAdjusted
getPlatformAdjustment
clearPlatformAdjustment
clearTrackBasedAdjustments
calculatePlatformMeanHeight
sfcAdjustment
setDefaultWind
PBL surface model
BoundaryLayerAdjustment
GRAVITY
ASL
PIC
ALPHA
GOVR
ROVG
MEAN_TROPATMOS_PRESSURE
BOUNDARY_LAYER_HEIGHT
DEFAULT_LAND_SURFACE_TEMP
DEFAULT_OCEAN_SURFACE_TEMP
KELVIN_CONSTANT
similarityVelocity
similarityT
rINumber
frictionVelocity
frictionVelOne
roughness
dragCoefficient
satMixingRatio
parcelmpK
virtParcelmpK
parcelmpKSfc
virtParcelmpKSfc
airSurfaceDiff
neutralStabilityL0m
surfaceLayerHgt
seaSurfaceTemp
exposure
clear
verifyAndSetInitDataValues
prepareDataForLoop
dataAdjustmentLoop
computeFinalValues
run
unstable
stable
hydrostatic
surfacePressure
mixingRatio
theta
thetaV
thetaVS

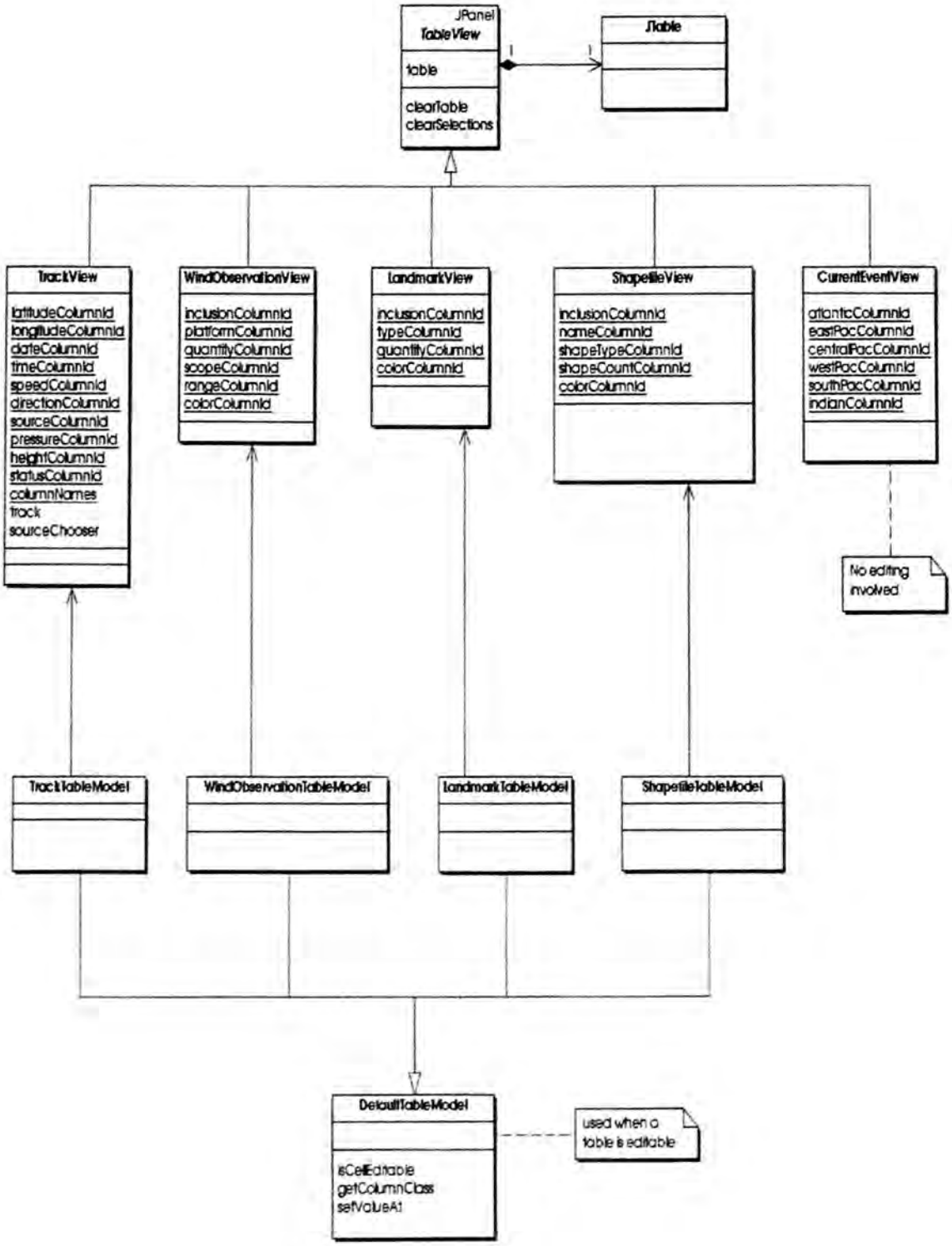

Serializable
MapPosition
latitude
longitude
1..*
1
CIAMap
MAXMAPLINE
source
globalGroupList
loadMap
Serializable
GlobalPosition
FORMAT_PE_NW_LON
FORMAT_PW_NE_LON
ATLANTIC_BASIN
EASTERN_PACIFIC_BASIN
CENTRAL_PACIFIC_BASIN
WESTERN_PACIFIC_BASIN
SOUTHERN_PACIFIC_BASIN
INDIAN_BASIN
distanceTo
headingTo
getLatitudeDifference
getLongitudeDifference
getAzimuth
relativeAzimuthQuadrant
marsdenSquare
relativeQuadrant
getAreaIndex
getBasin
toString
isSame
2
1
GlobalArea
GLOBAL_AREA
upperLeftGlobalPosition
lowerRightGlobalPosition
containsPosition
makeSquare
latitudeDifference
longitudeDifference
fixOverflow
getAreaIndexList
isSame
world-wide area
Landmark
TYPE_UNKNOWN
TYPE_W88D
TYPE_ASOS
TYPE_BUOYCMAN
TYPE_ZIPCENT
UNKNOWN_STRING
W88D_STRING
ASOS_STRING
BUOYCMAN_STRING
ZIPCENT_STRING
type
label
description
1..*
1
Hashtable
LandmarkSet
MAXLANDMARKLINE
source
renderStateTable
destroy
loadFromURL
loadFromFile
loadData
countForType
Shapefile
SHAPEFILE_ID
VERSION
NULL
POINT
ARC
POLYGON
MULTIPOINT
UNDEFINED
mainHeader
records
name
init
1..*
1
Hashtable
ShapefileSet
renderStateTable
destroy
loadFromFile


JDialog
ActionListener
Wizard
CANCEL_STRING
OK_STRING
cardLayout
commandPanel
finishButton
nextButton
backButton
cancelButton
cardPanel
firstCard
lastCard
exitCondition
go
reset
nextButtonClicked
backButtonClicked
cancelButtonClicked
finishButtonClicked
close
TrackFixQueryWizard
USER
DATABASE
INTERPOLATION
EXTRAPOLATION
newFixes
filterPanel
resultsPanel
interpolationPanel
extrapolationPanel
eventArray
candidateFixArray
go
choosePath
actionPerformed
loadFilterPanelLists
loadResultsPanelList
ItemListener
ListSelectionListener
TrackQueryWizard
DATABASE
FILE
CENTURY_20
CENTURY_21
qcId
methodPanel
centuryPanel
filterPanel
resultsPanel
century
eventInfo
qcInfo
candidateTrackVector
TrackQueryWizard
go
reset
valueChanged
itemStateChanged
loadFilterPanelLists
loadResultsPanelList
nextButtonClicked
backButtonClicked
finishButtonClicked
ItemListener
ListSelectionListener
WindObservationQueryWizard
DATABASE
FILE
CENTURY_20
CENTURY_21
newObservationList
methodPanel
centuryPanel
dateRangePanel
pressureLevelPanel
resultsPanel
century
basin
allDatabasePlatforms
valueChanged
itemStateChanged
loadResultsPanelList
ListSelectionListener
QCSetQueryWizard
qcSetId
filterPanel
resultsPanel
userArray
qcSetArray
eventArray
candidateQCSetInfoVector
QCSetQueryWizard
go
reset
valueChanged
actionPerformed
loadFilterPanelLists
loadResultsPanelList
nextButtonClicked
backButtonClicked
finishButtonClicked
EventQueryWizard
event
filterPanel
confirmationPanel
eventArray
go
actionPerformed
loadFilterPanelLists
loadConfirmationPanelList
CommitWizard
AnalysisWizard
NOVICE
OLD_NOVICE
EXPERT
analysisParameter
modePanel
analysisSizePanel
analysisTypePanel
pressureLevelPanel
createBogusPanel
innerMeshPanel
outerMeshPanel
meshInfoPanel
analysisArtist
actionPerformed
MarineGriddedFieldQueryWizard
analysisId
filterPanel
resultsVector
loadFilterPanelLists
finishButtonClicked

Artist
DEFAULT_SCALE_FACTOR
widestStroke
wideStroke
regularStroke
artistGlobalArea
graphics
color
font
fontColor
scaleFactor
xFactor
yFactor
draw
WindObservationArtist
failedColor
rawColor
platformMaxWindTable
mostRecentDate
maximumWind
mastLength
sprigLength
realWidth
qcSet
draw
drawPlatform
redrawObservation
redrawMaxWindObservationSymbol
drawMaxWindObservationSymbol
drawMostRecentObservationSymbol
drawQuadrantMaxWindObservationSymbol
drawObservation
TrackArtist
beginFixColor
centerFixColor
endFixColor
realWidth
draw
drawBeginFix
drawCenterFix
drawEndFix
drawFix
CIAMapArtist
latLonColor
latSpacing
lonSpacing
draw
AnalysisArtist
previousEastX
previousWestX
previousNorthY
previousSouthY
currentNameList
draw
eraseMeshes
drawMesh
ShapefileSetArtist
pointWidth
draw
drawShapefile
drawShape
areaContainsShape
LandmarkSetArtist
draw
drawLandmark


JDialog
ActionListener
GenericDialog
CANCEL_STRING
OK_STRING
okButton
cancelButton
commandPanel
contentPanel
exitCondition
go
reset
actionPerformed
okButtonClicked
cancelButtonClicked
LoginDialog
usernameField
passwordField
reset
ChooserLoginDialog
KeyListener
WindRadiiDialog
frontLeftTextField
frontRightTextField
rearLeftTextField
rearRightTextField
frontLeftLabel
frontRightLabel
rearLeftLabel
rearRightLabel
keyPressed
keyTyped
keyReleased
Inspector
ActionListener
MouseListener
WindObservationEditor
PASSED_CHOICE
FAILED_CHOICE
ORIGINALOBS_CHOICE
EDITEDOBS_CHOICE
originalWindObservation
editedWindObservation
theObservation
deleteObservation
inspectedObservations
observationPanels
statusChoice
editedChoice
isEditable
failReason
needToRepaint
mouseReleased
mousePressed
mouseClicked
mouseEntered
mouseExited
actionPerformed
refreshPanel
redrawObservation
reset
WeightEditor
weightPanel
platformWeightArray
go
refreshPanel


WindObservationSet
DbUtils
HwindObservationSet
altered
renderStateTable
databaseDateRange
leastRecentEndingTime
stationTable
lowerPressureLevel
upperPressureLevel
destroy
loadFromDatabase
addObservations
addObservation
loadExtras
importObservations
importTrack
loadObservationsFromDatabase
addedNewObservations
loadTrackFromDatabase
addEditedObject
clearEditedObject
commitToDatabase
commitObservationsToDatabase
commitTrackToDatabase
loadRenderStateTable
flightLevelDataExists
containsBackgroundField
getColorForPlatformDescription
setColorForPlatformDescription
getInclusionOptionForPlatformDescription
setInclusionOptionForPlatformDescription
getBeginningDateForPlatformDescription
setBeginningDateForPlatformDescription
getEndingDateForPlatformDescription
setEndingDateForPlatformDescription
getEffectiveSizeForPlatformDescription
Hashtable
LandmarkSet
MAXLANDMARKLINE
source
renderStateTable
destroy
loadFromURL
loadFromFile
loadData
countForType
Hashtable
ShapefileSet
renderStateTable
destroy
loadFromFile
Interface
RenderStateLookup
load
getColorForKey
setColorForKey
getInclusionOptionForKey
setInclusionOptionForKey
Hashtable
RenderStateTable
colorTable
theColors
load
getColorForKey
setColorForKey
getInclusionOptionForKey
setInclusionOptionForKey
Vector
ColorTable
RenderState
displayColor
isIncluded
ObsTypeInfo
beginningDate
endingDate
effectiveSize
1
0..*


JPanel
TableView
table
clearTable
clearSelections
JTable
1
1
TrackView
latitudeColumnId
longitudeColumnId
dateColumnId
timeColumnId
speedColumnId
directionColumnId
sourceColumnId
pressureColumnId
heightColumnId
statusColumnId
columnNames
track
sourceChooser
WindObservationView
inclusionColumnId
platformColumnId
quantityColumnId
scopeColumnId
rangeColumnId
colorColumnId
LandmarkView
inclusionColumnId
typeColumnId
quantityColumnId
colorColumnId
ShapefileView
inclusionColumnId
nameColumnId
shapeTypeColumnId
shapeCountColumnId
colorColumnId
CurrentEventView
atlanticColumnId
eastPacColumnId
centralPacColumnId
westPacColumnId
southPacColumnId
indianColumnId
No editing involved
TrackTableModel
WindObservationTableModel
LandmarkTableModel
ShapefileTableModel
DefaultTableModel
isCellEditable
getColumnClass
setValueAt
used when a table is editable