Realtime Database Support for Environmental Visualization*

Craig M. Wittenbrink, Eric Rosen, Alex Pang,
Suresh K. Lodha, and Patrick Mantey

Baskin Center for Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

Abstract

Much research has been done on interfacing databases to visualization applications, and there are many varied approaches. We describe the visualization–database interface, lessons learned, design, and design trade-offs of our development on the REINAS project (Real-time Environmental Information Network and Analysis System). We have developed visualization applications that have been in public use for several years and that visualize data from relational database engines. The most popular of these tools are World Wide Web access tools that have documented popular success on the Web. In addition we have developed, in collaboration with environmental scientists, novel visualization tools that also derive their data from relational database engines–Spray and CSpray. In this paper we present the interface and API issues, as well as the development of the required middleware. The REINAS system is a sophisticated data management system whose requirements and construction were driven primarily by the visualization needs, and therefore presents a unique view of how to utilize commercial relational database technologies for environmental visualization. The REINAS system uses realtime data ingest and visualization from remote sensors; multiple heterogeneous and distributed databases; and fine grain queries within a spatio-temporal data organization. We also discuss applications in development, planned future enhancements, and future challenges.

1 Overview

The REINAS–Real–time Environmental Information Network and Analysis system– is an information management system, including a relational database and middleware, intended to marry data and visualization for environmental science. The project is an Office of Naval Research university research initiative for bringing environmental science into a more intimate relationship with state of the art visualization and information technology. Figure 1 shows a high-level view of the system. Lines indicate physical or wireless network connections. Lines are bi-directional unless indicated otherwise. Data flows generally from left to right, and feedback and control generally flow from right to left.

As the most visible part of our project we have numerous applications that interface to relational database technology. Much of the behind the scenes work has been in developing the system middleware and database schema [9], but the interface has also held numerous challenges. We have satisfied several of the challenges, and face in the near term some additional large challenges. To demonstrate our contribution, we first describe our project, solutions to date, and the trade-offs in different interface approaches.

This paper focuses on the interface between the REINAS database component and the visualization component. In particular, we discuss how the challenges of supporting real–time

*This project is supported by ONR grant N00014-92-J-1807

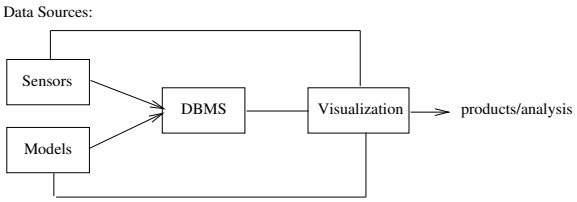

Figure 1: Major REINAS components.

data ingest and visualization have driven the design and the staged implementation of these components. The project goals are: our focus on regional real–time environmental science; our charter for a relocatable system; our requirement to support both real–time and retrospective data; and our requirement to support three classes of users with various needs: (a) scientists (meteorologists and oceanographers) who want the flexibility of accessing and combining various data sources for visualization and analyses; (b) operational users (e.g. forecasters) who want standard products; and (c) instrumentation and remote sensing users who need to validate and monitor an instrument centric view of the data.

We have three data sources to satisfy our users that work in parallel each with different tradeoffs, advantages, and disadvantages:

1. A long term data management relational database schema, designed for holding metadata, and guided by research in visualization.
2. A short term data management relational database schema, designed for minimal setup time but not attempting to hold metadata important for long term data management.
3. Environmental files for exchange, stand alone use, and legacy support. We currently support file formats specific to the project, and some more general formats such as HDF.

Related Work. There are similar systems to REINAS, and there is similar work in combining visualization to databases. We briefly review some of the relevant related work. A large multi-year development project, which focuses on global environmental science is the Sequoia 2000 project [16]. Stonebreaker et al. have also developed custom visualization tools, one which they call Tioga [17]. Another Sequoia 2000 prototype effort, developed interfaces to existing visualization programs [7]. Gershon et al. [4] overview the fundamental problems in large data managment and visualization. There are real-time systems that use less sophisticated database systems [3, 6, 15], and others that run at much less than the seconds timing of REINAS [5, 11]. The work most similar to ours, using a middleware and an API approach to create the interface, is in the GIS literature [1, 2]. In addition, see [8] for recent research articles in interfacing visualization to database. Our system is different than the mentioned literature, in that it does real-time to the second, provides middleware and apis for custom applications, works on a regional, not global, scale, and provides sophisticated 3D visualization.

This paper is an overview of the successes, approaches, and challenges we have faced in developing visualization from databases. Using our three data sources listed above we have developed several primary visualization applications: `Xmet`, Spray, CSpray, and `www-Met`. All of these applications interface to the second source mentioned above. Spray interacts with both relational database schema approaches and legacy files. `Xmet` and `www-Met` interact with only the simpler schema, but will be converted to use the full featured schema as it becomes more fully operational. The interfacing of the applications to the schema has been done primarily through custom API's supported by middleware crafted to handle environmental data.

2 Visualization Applications

2.1 Spray Rendering

We now present brief overviews of our visualization applications in an outside in or top to bottom description of REINAS. There are four major visualization applications: Spray, CSpray, Xmet, and World Wide Web Clients. In the following section, we describe the interfaces used to bring data into these applications.

The visualization component of Spray is organized into three different modes of operation targeted towards three different classes of users. These are the monitor mode for instrumentation engineers and recreational users to monitor the current state of instruments as well as the environment; the forecast mode for operational users who are interested in generating standard forecast products; and the analysis mode for scientists to perform retrospective analysis on their data. We now look at the each mode in more detail and how each interface with the database.

Monitor Mode

In monitor mode, users have a bird's eye view of the region of interest. See Figure 2. Environmental sensors are represented by simple icons. By selecting one or more of these sensors using a point and click interface or through a pulldown menu, users can view interpolated fields of a physical parameter (e.g. humidity, wind vector, etc.) or query individual sensors for time plots of the parameters they measure. The list of sensors currently supported include: fixed and portable meteorological stations, NOAA buoys, CODARS, wind profilers, seal tracks [12], and ADCP.

Currently, not all the data from these sensors are coming off the database. For example, some of them have to be retrieved from the sensor logs once every hour and stored in a local file. On the other hand, realtime data from Met stations are available through the Met server and can be used to interpolate the environmental field or generate time plots as the data arrives. The seal tracks and the mobile land and water based Met platforms are also of interest as GPS readings provide their current locations and can be used to aid the visual display.

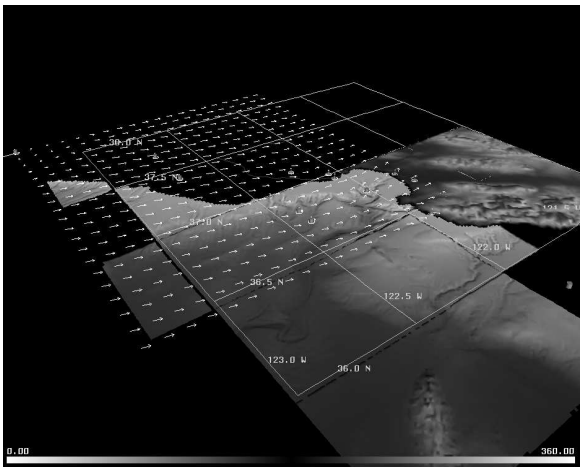

Figure 2: View of the Monterey Bay showing location of some sensors and an interpolated wind field from a subset of these sensors.

Forecast Mode

Operational forecasters are interested in generating standard products from forecast models and satellite observations. These products may include animated GOES satellite images, as well as maps that show contours of 500mb pressure field at 60m spacing of geopotential height against vorticity, 850mb pressure field at 30m spacing of geopotential height against relative

humidity (shaded above 90%), and others. Aside from standard products, users can also generate customized products e.g. different projections, different contour spacing, and heights. One can also register and overlay observation data with products e.g. wind barbs and animated GOES images. Figure 3 shows a typical forecast product.

Currently, satellite images are periodically and automatically retrieved via the Naval Postgraduate School. The images are stored locally in files and will be stored as blobs in the database.

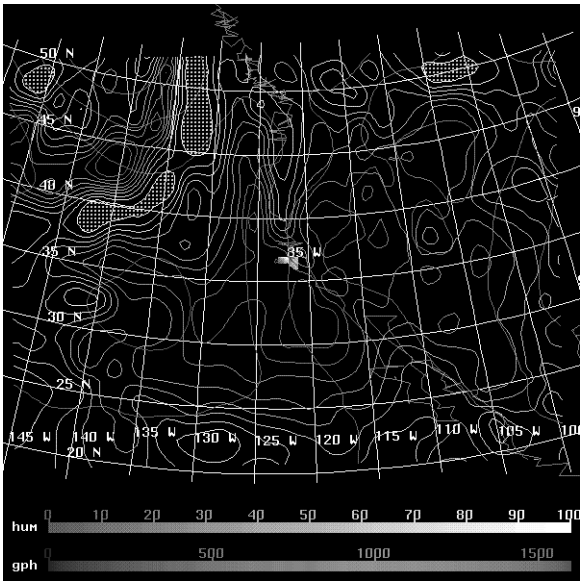

Figure 3: Sample standard forecast product where regions of relative humidity above 90% are cross-hatched.

Analysis Mode

This mode allows the scientific users to perform retrospective analysis on synoptic data. It allows users to explore large data sets interactively using different visualization techniques. It is also extensible and can easily grow with users' needs. The underlying mechanism that provides the visualization capability in analysis mode is based on spray rendering [13]. Spray rendering provides the users with the metaphor of spray painting their data sets as a means of visualizing them. In its simplest form, data are painted or rendered visible by the color of the paint particles. By using different types of paint particles, data can be visualized in different ways. The key component of spray rendering is how the paint particles are defined. They are essentially smart particles (or sparts) which are sent into the data space to seek out features of interest and highlight them. Among the advantages of this visualization framework are: grid independence (sparts operate in a local subset of the data space and do not care whether data is regularly or irregularly gridded), ability to handle large data sets (sparts can be "large" and provide a lower resolution view of the data set or they can be "small" and provide a detailed view of an area of interest), extensible (it is easy to design new sparts). Sparts can also travel through time-dependent data sets. Figure 4 shows the interfaces available in analysis mode as well as illustrates some of the possible visualization methods.

Spray is an evolving research system. Currently, it works with rectilinear grids only. Entire data sets are read in from files and do not take advantage of the particle nature of the sparts. We envision that once the data is ingested into the database, the sparts can make the equivalent of SQL calls to travel through the data space by requesting for the appropriate subset of the data from the database. Some caching and coherency measures need to be taken to ensure that the current database technology does not get bogged down with too many small requests. Finally,

one can also exploit the inherent parallelism in the independence among sparts to operate on local subsets of the data.

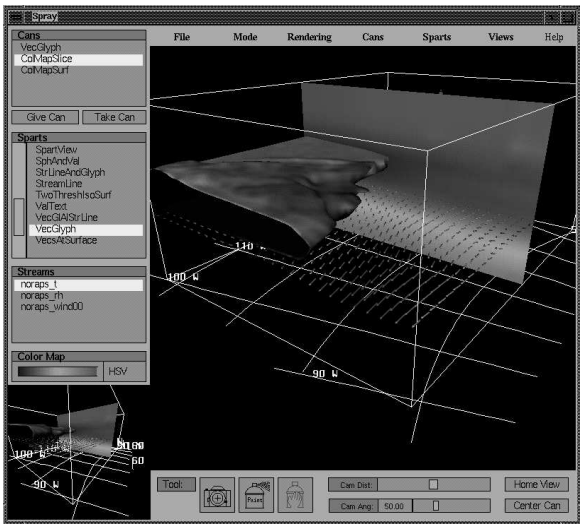

Figure 4: Sample visualization in analysis mode using spray rendering.

2.2 Collaborative Spray

To facilitate the sharing of data and collaboration among science colleagues, we have also added a collaborative feature to the REINAS visualization system, enabling geographically distributed researchers to work within a shared virtual workspace and create visualization products [14]. There are several components that are needed to make this feasible: session manager, sharing data/cans, floor control, multiple window, audio/video support, and different collaboration/compression levels. Figure 5 shows some of these.

Users can collaborate at different levels. Sharing can occur at the image (visualization product), spray can (abstract visualization objects – AVOs), or data stream (raw data) level. At the image level, participants can see what the other participants see and may perhaps be able to change view points. At the can level, participants have access to a list of public spray cans put up by other participants. These public cans will generate AVOs from the remote hosts and distribute them to other participants. Users may also give permission to other participants to have direct access to data streams and replicate those on local machines for faster response times. The different levels of collaboration also imply different requirements for compression. Tradeoffs will have to be made between graphics workstation capabilities, network bandwidth and compression levels. Objects that need to be transmitted can either be images, AVOs (together with can parameters and other transformation matrices), or raw data.

This is an excellent opportunity where database technology can help the visualization. With a collaborative system, the data does not have to reside in one central database. In fact, the data could be distributed around several heterogeneous databases. The visualization application acts as a common front end for the users to query and visualize their data. Where the data actually resides can be made transparent to them. The current CSpray program performs this task by allowing you to visualize another participant's data. Because the requests are sent to the participant with the data, remote database queries can be made by all of the participants without sharing the data. The visualization products are shared, because each participant acts as a visualization server for everyone else.

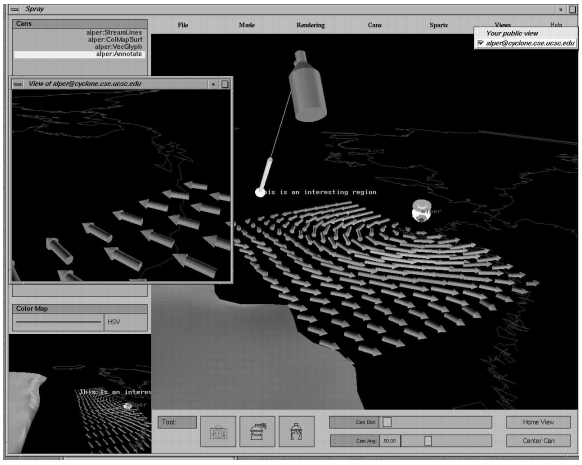

Figure 5: Collaborative visualization. The large graphics window shows the viewpoint of the local user. The smaller window shows what the collaborator is looking at. The "eye-con" shows the location of the other participant.

2.3 Xmet

`Xmet` was initially designed to be a system debugging tool that included some primitive visualization capability. However, along with the Xmet API server, and related software that populates a simple database, `Xmet` became the prototypical REINAS client application.

As a network application, `Xmet` was designed using the client-server model. Multiple instantiations of the `Xmet` client interact with a single "`XmetServer`". Although facilities exist for running multiple servers should performance degrade, this has rarely proven necessary. Communication between client and server is abstracted using the Xmet API (discussed later) and accomplished using Internet socket streams.

As a visualization application, `Xmet` was designed with very modest goals in mind, namely the display of concurrent time-series data. It was later expanded to also allow the display of two-dimensional CODAR ocean surface current radial and vector data, including CODAR animations. Figure 6 shows a screen shot from a typical `Xmet` session. The map window in the upper-left shows the Monterey Bay coastline and surrounding region, with glyphs indicating the position of real-time meteorological stations whose data is available through the server. The current site is emphasized, indicating that conditions from the Long Marine Lab meteorological station are being displayed. Below the map, current conditions for various sensors are given; to the right, graphs of sensor stream values over the previous twelve hours show how temperature, wind, and solar irradiance vary from mid-morning to early evening. Individual pop-up menus allow the strip-chart graphs to be user configured. Additional controls at the bottom allow the user to navigate through time, display wind-barb glyphs, fix the strip-chart display interval, and control the frequency at which the displayed current conditions and strip-charts are updated.

In alternate modes, the user may decide to configure `Xmet` with zero strip-charts and zoom in on the view of Monterey Bay as shown in Figure 7. In this case, CODAR ocean surface current radial data is under investigation, with meteorological conditions from the various stations overlayed using wind-barb glyphs.

From a visualization standpoint, `Xmet` lags far behind the modern, GL-based Spray and CSpray; however, its compactness, portability, and relative ease of use have made it fairly popular among REINAS users. Binaries for Sun/4, DEC ULTRIX, RS/6000, HP/UX, SGI, A/UX, and BSD/OS platforms have been built and continue to be widely distributed.

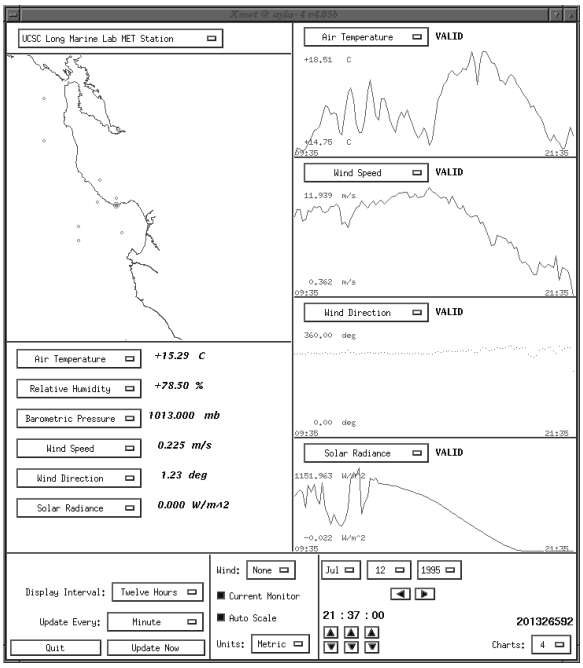

Figure 6: Xmet: The Prototypical REINAS Client Application

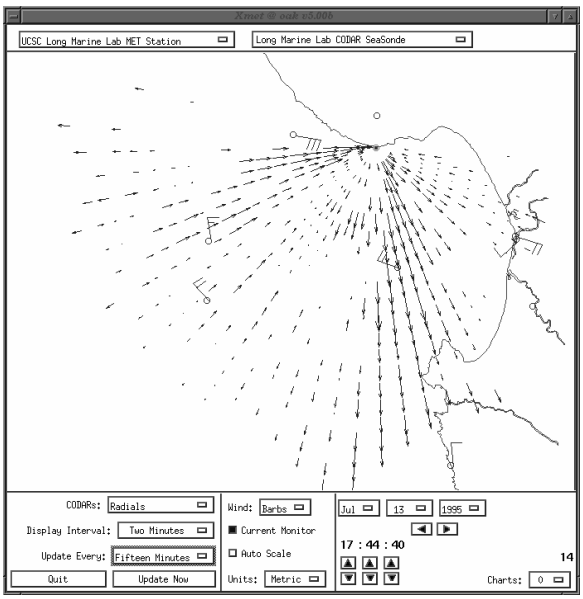

Figure 7: Xmet: Configured for CODAR Radial Data Display

2.4 World Wide Web Clients

The growing popularity of the World Wide Web (WWW), and our desire to provide real-time demonstrations to users who were not previously aware of REINAS, led to the development of HTML aware applications, which when used in conjunction with the National Center for Supercomputing Application's `httpd` server, provide a WWW interface to the Xmet API.

The `www-Met` application is integrated into a virtual tour of the Monterey Bay region, highlighting sites where the project has deployed or connected instrumentation (interested readers are invited to visit `http://sapphire.cse.ucsc.edu/reinas-instrument-tour/`. Web visitors are first presented with a map of the region (Figure 8) allowing them to select a site to visit.

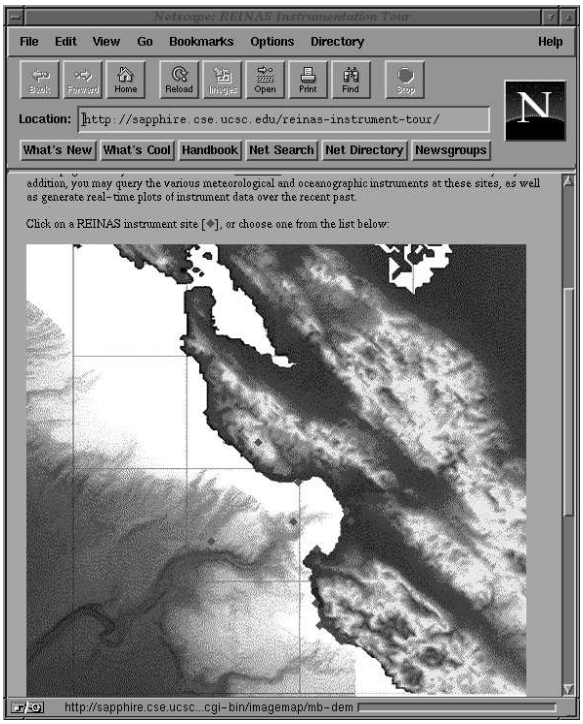

Figure 8: www-Met: Opening Screen

Pages describing each site, and the instrumentation accessible through REINAS at that site, can be reached in this way. Where meteorological instrumentation is present, users can choose to query for current and recent weather conditions; Figure 10 shows a sample query with its results.

Although far more primitive than either `Xmet` or the Spray and CSpray applications detailed previously, the instrument site web tour currently receives approximately 4800 interactions generating over 1500 separate requests for data from the database in a given month. Despite its simple interface and relatively primitive visualization options, `www-Met`'s usage statistics dwarf that of `Xmet` and Spray.

A similar web oriented application is `www-CODAR`, which displays a real-time ocean surface current vector map, via the Xmet API (see Figure 11). `www-CODAR` received over 570 visitors in June, 1995, the first full month it was publicly accessible. Each vector map typically requires about 400 database queries to produce under the Xmet API; hence, these 570 visitors symbolize over a quarter-million Xmet API transactions alone.

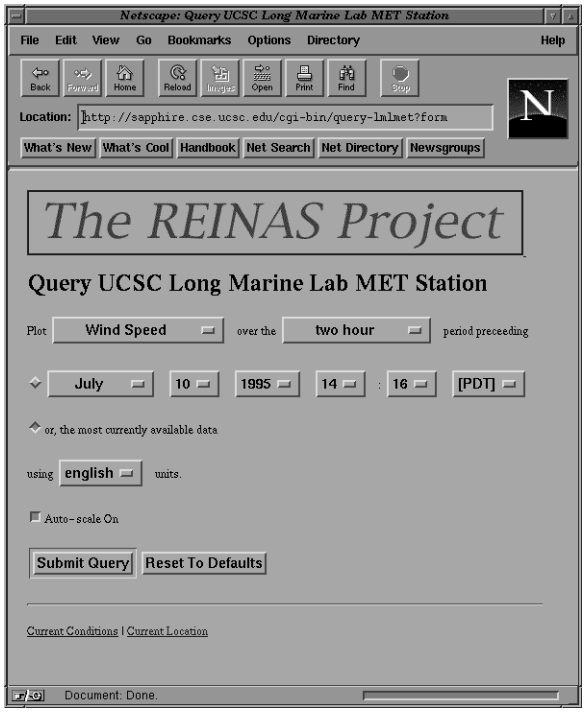

Figure 9: www-Met: A Sample Query Form

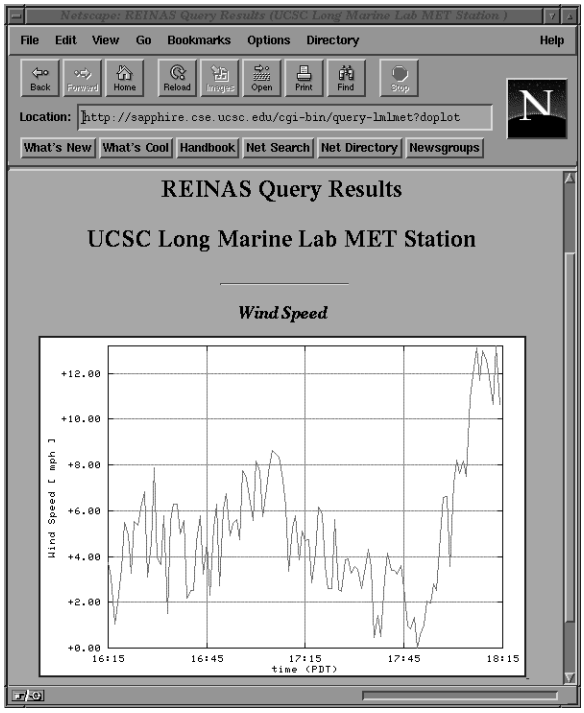

Figure 10: A Sample Query Result

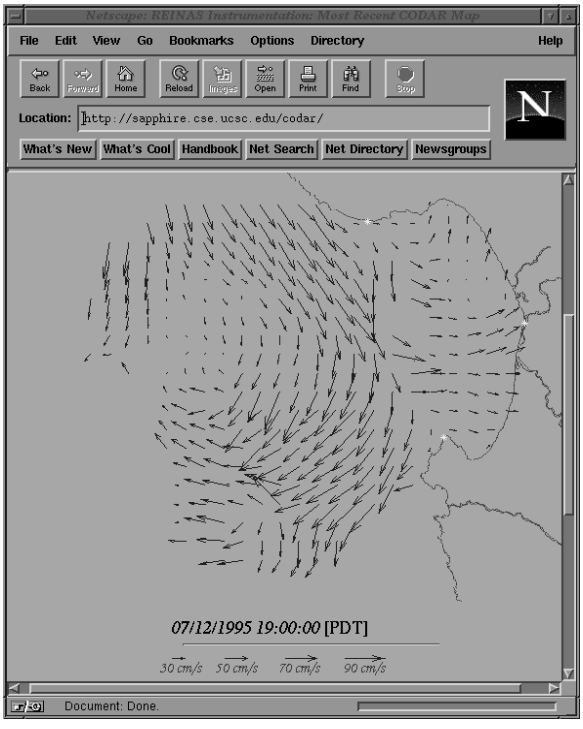

Figure 11: Ocean Surface Currents on the World Wide Web

3 Application Programming Interfaces

The primary interface of our distributed applications to the database engines and to files is through several application programming interfaces (APIs). There are three primary APIs now in use. The `Xmet` API currently provides a generic interface to the simpler schema; the RSObject (for R(EINA)S) API provides an interface to the full featured schema at a higher level; and the REINAS low level API provides a mechanism to simply passes SQL queries directly to the database engine, as well as handling ferrying data back and forth. The RSObject API is written on top of the REINAS low level API, and also has facilities to access data from the Xmet server using the Xmet API, and to files accessing them directly. In this section we quickly describe them, and the issues and design trade-offs that were used.

3.1 Xmet API

The Xmet API is opcode based, with most opcodes occurring in request/reply pairs. A list of Xmet opcodes is given in Figure 12. All Xmet API requests conform to a standard type,

request	reply	function
mo-getinit	mo-retinit	initialization (meteorological)
mo-getinfo	mo-retinfo	return site metadata
mo-getcurrent	mo-retcurrent	most current meteorological data
mo-getall	mo-retall	arbitrary meteorlogical data
mo-getavg-all	mo-retavg-all	arbitrary meteorlogical means
mo-getmax-all	mo-retmax-all	arbitrary meteorlogical maximums
mo-getmin-all	mo-retmin-all	arbitrary meteorlogical minimums
mo-getseries	mo-retseries	arbitrary meteorlogical timeseries
co-getinit	co-retinit	initialization (CODAR)
co-getinfo	co-retinfo	return CODAR site metadata
co-getradials	co-retradials	return arbitrary CODAR radials
co-getvector	co-retvector	return arbitrary CODAR vector
so-getrecent	so-retrecent	return server usage statistics
none	so-retdie	inform client of server's demise

Figure 12: Xmet API Opcodes

detailed in Figure 13; the opcode and parameter values determine the precise meaning of the request. Replies are similarly formatted in a standard type, detailed in Figure 14. For example,

```
typedef struct {
    long        opcode;   /* request opcode */
    long        iop1,     /* integer operands (32-bit) */
                iop2,
                iop3;
    TimeType    dop1,     /* time operands */
                dop2;
} XmetRequestType;
```

Figure 13: Xmet API Request Structure

to request the most current weather conditions from meteorological site #2, a request structure with the `opcode` field set to *mo-getcurrent* and `iop1` set to *2* is forwarded to the server through a library routine. The server replies with a reply structure identified containing a *mo-retcurrent* opcode and `iop1` set to *2*. The weather conditions are encoded in the `data` field, with extracted

```
typedef struct {
    long        opcode;   /* reply opcode */
    long        iop1,     /* integer operands */
                iop2;
    long        mask1,    /* logical operands */
                mask2;
    TimeType    tstamp;   /* time operand */
    long       *data;     /* variable length opcode specific data */
} XmetReplyType;
```

Figure 14: Xmet API Reply Structure

with other library calls. The `tstamp` field identifies the time for which the data is valid. The remaining fields are unused in this transaction.

Although admittedly primitive, this simple API has demonstrated remarkable utility, and has been used by such diverse applications as high-end visualization systems such as Spray and CSpray, various X/Motif toolkit applications, world-wide-web server glue code, and simple text-oriented client programs, on a variety of architectures. Functionality can and has been extended simply by defining new opcodes and updating the server (most of the queries and all CODAR support was added in this way). Inline compression, handled by the library routines which send and receive requests and replies, addresses basic concerns about network transport efficiency.

Although the application programmer is limited in the types of requests that can be made to the server when using this API, the ease with which common requests can be made has helped make this interface the most popular (in terms of number of applications developed and transactions handled) among REINAS project client applications to date.

3.2 RSObject API

The RSObject API was designed for the handling of environmental data. In the design of this API, there were goals in object oriented design of the entire API and to satisfy several goals fundamental to language design. The design issues are numerous and involve tradeoffs in 1) type checking, 2) security principle 3) efficiency 3) maintainability 4) readability 5) flexibility [10]. Others have delved into object oriented API designs such as Bernath [1], Berril [2], and those in [8].

In designing the RSObject API we considered numerous alternatives, and settled upon one which was best in terms of the metrics given. The RSObject API uses separate types, but passes objects around using pointers for efficiency and generic calls. We do dynamic type checking on the object type. What follows is a list of the metrics that we used to evaluate various API choices:

1. Type checking: compile(static) No
2. Type checking: runtime (dynamic) Yes
3. developing supporting: easy, fewer routines to support
4. efficiency: must do run-time checks
5. API use: simple to understand, simple to use
6. multiple get's use multiple typed variables, and checking will be done for you RSGet(. . . temp,) RSGet(. . uncertainty . .)
7. Abstraction: Yes. abstraction in definition and implementation
8. Automation: Yes
9. Defense in Depth: Yes (mostly)
10. Information Hiding: Yes. Only the known scientific types are visible. enumeration constants are hidden.
11. Labeling: NA
12. Localized Cost: No, dynamic object checking is cost for all accesses

13. Manifest Interface: Yes
14. Orthogonality: Yes
15. Portability: Yes
16. Preservation of Information: Yes
17. Security: Yes
18. Simplicity: Yes (and No harder to implement)
19. Structure: yes
20. Syntactic Consistency: Yes
21. Zero-One-Infinity: NA

The API we designed yields a score of 15 versus 8 and 13 for two other likely alternatives, where each feature is given one point. This proposed solution uses separate types, passes objects using pointers to amortize method cost, allows dynamic type checking more in line with an object oriented programming style, and allows extension to use X resource type constants. Here is an example query of the API:

```
status = RSGet(object, objectType, &returnObject)
```

The object which is to be queried has multiple settings, such as locality, time, and other qualifiers, which can be set on the object such as RSSet() calls. This provides for arbitrary manipulation of the object's characteristics as well as the ability to enhance objects, not use visible constants (magic numbers), and do dynamic default behavior of objects upon creation. The type of data to be queried is a variable found in the return type itself. The objects may be queried often with queries that don't change the object itself, simply the return values.

We also use a multiple class hierarchy using C type definition design. Some objects can be creat'ed while other base class objects are static declarable, providing a reasonable mix for efficiency ease of use and type checking. As examples, there are types, elements, and aggregations. Environmental parameter units are specified as part of the container in which they are retrieved from. The field descriptor gives the actual units. Units are described within the RSField descriptor. If a user wishes to change the units, an RSSet operation is performed, and the corresponding field will reflect the change. For example, for a fully generic temperature object, the filled in values may be the result of the query, or may have been set by the user, so they are more like commands.

The return values from the data base engine are in character format. The REINAS low level API passes these and the internals of the RSObjects convert these to binary (int, float, double), and then populate the structures that are visible to the user program. The major class type is statically defined, such as RSParameter, and then the minor class is dynamically by the result of the query. The programmer must have knowledge of the basic scientific types. They must be aware of how to manipulate and operate on those types. An example struct follows:

```
typedef struct rsparameter{
int type;
float latitude;
float longitude;
float z;
float time;
float fval;
}RSParameter;
```

The application programmer uses the parameter, description lists, series, and localities, to manipulate and interact with both the real time vacuum, accessing data before they are loaded, and the data base. Localities define the bouding box in space and/or time that allows culling of the environmental queries. The descriptor list is an array of RSField's that are used to encode the data types. Numerous methods are supported and objects, such as getting one measurement, getting an entire time series, and getting a large field of scattered data. We have also developed some convenience functions for object duplication, and querying the size of data that can come back.

4 Underlying Issues

The applications and APIs simply show our results to date. We now discuss several underlying issues that are important in the development of the REINAS system.

4.1 Granularity of Data Access

The underlying issues in interfacing the visualization programs really lie with their intended usage. With Spray visualization we have been able, and were required, to experiment with the granularity of requests made to the database. Because of the particle nature of Spray, each spart operates independently on its local data subset. Two immediate consequences are that the size of the local space may be varied, and that each local subset, whether overlapping with another region or not, can be operated on independently. These two are important considerations if sparts are to separately make queries to the database. If the granularity is set too fine, the overhead of setting up the calls will overwhelm the database server. On the other extreme, if the granularity is too coarse, the advantages of working on small blocks of data are lost. At any rate, some caching and coherency measures need to be taken to ensure that the current database technology does not get bogged down with too many small requests. In general, sparts travel in more or less the same direction, hence spatial and temporal coherency is high. However, optimistic caching of anticipated trajectories may not always be practical. Take for instance the case of sparts that trace out ribbons in flow fields. If the flow is highly divergent or turbulent, the sparts may request widely different regions of the data space to integrate its next position. Similarly, some preprocessing of the data may be necessary to help optimize database queries. For instance, when dealing with or querying scattered data sets, it may make sense to bin the data into different sub-regions or arrange them in a hierarchical structure.

Even when optimizing for simplicity, as was done with the simple Xmet API, questions about the appropriate level of granularity arose. In particular, the best way to formulate the result of a request for ocean surface current vectors was not immediately obvious. The difficulty stems from the nature of the data, and not the characteristics of the visualization; in this case, a snapshot of the ocean surface current includes an arbitrary and varying number of vectors. A reply might group all possible vectors together, a convenient but inefficient approach in this case. Or, a formulation requiring each vector to be returned independently adopted, a more flexible but also inefficient approach (in terms of number of transactions). The later approach was adopted, and appears to be working well.

4.2 Support for Multicasting

The current implementation of CSpray uses point to point TCP/IP connections to support collaboration between users. While providing reliable communication links, it severely limits the number of active participants because of the $O(N^2)$ connections necessary to provide a shared common workspace for N participants. An alternative that we are looking at is reliable multicast where there are only $O(N)$ connections in a session. This significantly reduces network traffic especially when participants are making liberal use of their spray dosage (i.e. transferring large amounts of graphics primitives). This is also a more efficient setup in situations where one participant is briefing (broadcasting) to the entire group as data is routed among the participants as opposed to individually sending to each one.

Multicasting would also be useful in the simpler world of the Xmet API. Typically, several clients are active simultaneously, and many are simply configured to monitor current meteorological conditions at a common site. Currently, each client must independently request such data from the server, even though the server might receive several identical requests almost simultaneously. Multicasting the identical reply would both reduce network traffic and reduce the number of database interactions as well.

4.3 Simple, Safe Code

Nowhere are the advantages of a simple, compact, well-understood API versus a sophisticated, more powerful, and better tuned approach more obvious than comparing the ease of extensibility, in practice, of the Xmet API with the other interfaces developed in REINAS. Although the Xmet API was developed first, with a fairly primitive understanding of the types of queries that would be useful, the interface quickly outgrew its prototype classification and took on a large share of the responsibility of transferring data from the database to a variety of REINAS visualization client applications. This occurred because of the ease of use and speed at which the API could be extended when new environmental data or new types of queries were introduced. The Xmet API is simple enough for fast maintenance, and many changes may be made in hours or days. The Xmet API is also simple enough that it can be quickly understood, and application transactions require only a few lines of code. As a result, it is used by a variety of REINAS client programs.

In addition, the Xmet API could be and was extended without affecting (usually) the functionality of previously written clients. Functionality was extended by adding opcodes; the request and reply structures were never changed. As a result, clients only needed to be updated if they specifically needed to take advantage of the queries available through the new opcodes. This has often not been possible with the RSObjects API, as it was written on top of Xmet, the REINAS low level API, and as an interface directly to files. Although the Xmet API limits the type of queries that can be made to the enumerated commands, it is extensible enough that, in practice, this is not a significant drawback. The command approach does break clients, when the enumeration of command types, or the result flags must be or are inadvertantly modified, but errors were quickly tracked down.

The REINAS low level API, and RSObjects API, are more powerful, and require more effort to understand as they include sophisticated features like SQL support, blocking and nonblocking reads, multiple handles, and must support the full featured

schema that has hundreds more data types. There of course, remains much work for improving query performance that is not completely the responsibility of the database. For example, some Xmet queries result in a large number of transactions between client and server. In this phase of the project, these performance issues have not become too important, even with hundreds of users a week. Often, disk space has become a more considerable problem for project management, and application support. Currently, older data are archived off-line in anticipation of using a tertiary storage solution. These API trade-offs show how different small and large software projects are in terms of their extensibility and development cycles. The difficulties in developing a large software system have not dissappeared, but in attempting to solve a large part of the data management problem we have at least used an incremental approach and completed the prototype of a much more sophisticated system. We hope to soon complete newer visualization applications which fully exploit the RSObject API and the features of the REINAS System.

4.4 User Interface

The separation of of functionality into three modes to meet the needs of the three user classes seem to place an obstacle after extended use. Users seem to switch from one mode to another mode because the functionality was not directly available under that mode. Another drawback to that organization was that not all the data were available in each mode. For example, monitor mode had access to sensor measured data but not to model data as in analysis mode. This proved to be weakness for users who are interested in comparing the numerical forecast versus the actual measurements. We have since redesigned the interface to take these limitations into account. The current, ongoing visualization development effort is now tool based instead of mode based. That is, users will activate one or more tools to do data transformations and/or visualizations. Each tool have a set of input ports that can be associated with different data streams either from files or from the database. Output from each tool can be routed to a graphics window for rendering, or back to the database or file for saving. Tools accomplish the functionality provided previously in the different modes e.g. field interpolation, contour lines, isosurfaces, etc. They are actually made up of simpler elements that work on a chunk of data at a time, the results of which are passed to other elements within the tool. That is, requests are continually made to retrieve a subset of the data for the tool to work on. Within the tool, elements process the data in a dataflow fashion, and may also send feedback to request neighboring data. In this fashion, we have married the dataflow approach with the active agent approach. The underlying execution model is transparent to the user as they are simply interacting with different tools.

5 Challenges and Lessons Learned

In the environmental sciences it has been traditional to place environmental data into proprietary file formats. The visualizations made from these formats have been custom developed, but the difficulty in using other researcher's data has been not only the conversion of the formats, but the understanding of the lineage of the data. The REINAS project has been focused on trying to aid in the development and experimentation of systems that would solve the long term data management problem. While this is a

fundamental difficulty, it is not the only aspect of our research. We have also looked in depth into the marriage of our visualization applications to this experimental information system. In so doing, we have developed a range of solutions that have interesting trade-offs.

By simply using a relational database, and putting the environmental data into tables, we were able to provide a large jump in functionality over using files alone. A centralized server accessing the database, and handling distributed network requests through a primitive but extensible API has been very successful in serving a wide range of visualization applications. The `Xmet`, `www-Met`, `www-codar`, and Spray applications demonstrate the succesful integration of a relatively simple schema (less than 500 lines of SQL), with a server, and a network client/server-based API. In addition, new sources and queries can be added to the system fairly quickly because of the overall simplicity of the system. But, despite its success, this system does not solve the long term data management problem.

A schema designed primarily by our collaborator, Bruce Gritton, of MBARI holds promise for helping to solve the long term data management problem. The added cost for capturing and using the additional meta-data, and relations among the data, is the complexity. The current schema is over 7000 lines of SQL. To maintain, insert, and verify instrument data that is loaded requires a full time database administrator. There has been a considerable amount of work put into simplifying the process of adding new instruments, but it still requires about the complexity of developing and debugging a Unix device driver. To capitalize on the features available in this full featured schema, we have developed a pass through low level API, and a higher level object oriented API for environmental data. This allows the in memory representation to match the database, while not forcing the application programmer to use SQL directly. Our research goals were to have Spray rendering use smart particles to drive the queries, but the level of support by the database and the Unix OS for networked programming are such that this was impractical. We have therefore implemented a blocking query approach that may be improved to use a caching scheme for effective performance. The visualizations that may be done are similar to those done with files, but there is larger flexibility in developing time series visualizations, and in selecting and deselecting sources.

Our near-future research involves the enhancement of the information system to hold more data, such as AVHRR, Ocean Models, etc. and to do a closer integration of measurement and model data. This is where our science users are driving us as they need to compare the effectiveness of their models in capturing real world phenomena. They may also use the visualization to experiment with different ways to interpolate the measurement data, which is necessary to kick-start the models. Further into the future are the video applications on which experimentation is just beginning, both from the technology side and from the science side. It may be that relational database technology will not support the multimedia performance necessary for a video database, and the numerous vendor extensions do not clearly solve these either. In the end we can only hope to develop the appropriate interface so that our visualization applications become wired to the latest valid scientific data sources, while still supporting the legacy sources.

Acknowledgments

We would like to acknowledge the efforts of some of the students who have been active in REINAS, including: Bruce Montague, Carles Pi-Sunyer, Bryan Mealy, David Kulp, Skip Macy, Tom Goodman, and Jim Spring. We would also like to thank the many colleagues involved in this collaborative effort, especially our science colleagues Professor Wendell Nuss, Bruce Gritton, Kang Tao, Dr. Francesco Chavez, Dr. Dan Fernandez, and Professor Jeff Paduan. The REINAS systems group has been instrumental in providing the foundation from which to develop a real application test bed – led by Professor Darrell Long and Chair Pat Mantey and supported by Andrew Muir. The REINAS visualization team, Tom Goodman, Naim Alper, Jonathan Gibbs, Jeff Furman, Elijah Saxon, and Michael Clifton, have provided much of the coding, and development for the Spray and CSpray applications. In addition, John Wiederhold, Catherine Tornabene, and Ted Dunn helped develop the network of Monterey Bay area instrumentation that attracts most of our users.

References

[1] T. Bernath. Distributed GIS visualization system. In *GIS/LIS Proceedings, Vol. 1*, pages 51–58. American Society for Photogrammetry and Remote Sensing, Nov. 1992.

[2] A. Berrill and G. Moon. An object oriented approach to an integrated GIS system. In *Proceedings of GIS/LIS*, pages 59–63, San Jose, CA, Nov. 1992. American Society for Photogrammetry and Remote Sensing.

[3] R. S. Cerveny et al. Development of a real-time interactive storm-monitoring program in Phonenix, Arizona. *Bulletin of the American Meteorological Society*, 73(6):773–779, June 1992.

[4] N. D. Gershon and C. G. Miller. Dealing with the data deluge. Special report: Environment, part 2. *IEEE Spectrum*, 30(7):28–32, July 1993.

[5] S. Howes. Use of satellite and radar images in operational precipitation nowcasting. *Journal of the British Interplanetary Society*, 41(10):455–460, Oct. 1988.

[6] J. Intriery, C. Little, W. Shaw, R. Banta, P. Durkee, and R. Hardesty. The land/sea breeze experiment (LASBEX). *Bulletin of the American Meteorological Society*, 71(5):656, May 1990.

[7] P. Kochevar et al. Bridging the gap between visualization and data management: A simple visualization management system. In *Proceedings of Visualization 93*, pages 94–101, San Jose, CA, Oct. 1993. IEEE.

[8] J. P. Lee and G. G. Grinstein, editors. *Database Issues for Data Visualization, IEEE Visualization '93 Workshop*. Springer–Verlag, 1994.

[9] D. Long, P. Mantey, C. M. Wittenbrink, T. Haining, and B. Montague. REINAS the real-time environmental information network and analysis system. In *Proceedings of COMPCON*, pages 482–487, San Francisco, CA, Mar. 1995. IEEE.

[10] B. J. MacLennan. *Principles of Programming Languages: Design, Evaluation, and Implementation*. Holt, Rinehart, and Winston, New York, second edition, 1987.

[11] M. Milnes. Interpretation of remotely sensed images using historic data. *Journal of the British Interplanetary Society*, 41(10):451–454, Oct. 1988.

[12] G. W. Oliver. Visualizing the tracking and diving behavior of marine mammals. In *Proc. Visualization*, page in press, Atlanta, GA, Oct. 1995. IEEE.

[13] A. Pang. Spray rendering. *IEEE Computer Graphics and Applications*, 14(5):57 – 63, 1994.

[14] A. Pang, C. M. Wittenbrink, and T. Goodman. CSpray: A collaborative scientific visualization application. In *Proceedings SPIE IS &T's Conference Proceedings on Electronic Imaging: Multimedia Computing and Networking*, volume 2417, pages 317–326, Feb. 1995.

[15] D. Schwab and K. Bedford. Initial implementation of the great lakes forecasting system: A real-time system for predicting lake circulation and thermal structure. *Water Poll. Res. J.*, 29(2/3):203–220, 1994.

[16] M. Stonebraker. Sequoia 2000: A reflection on the first three years. *IEEE Computational Science and Engineering*, 1(4):63–72, Winter 1994.

[17] M. Stonebraker et al. Tioga: A database-oriented visualization tool. In *Proceedings of Visualization 93*, pages 86–93, San Jose, CA, Oct. 1993. IEEE.