A Data and Information System for JGOFS

Glenn R. Flierl, EAPS, 54-1426, MIT, Cambridge, MA 02139

James K.B. Bishop, LDEO, current address: Univ. Victoria, POB 1700, Victoria, BC

David M. Glover, Clark 4, WHOI, Woods Hole, MA 02543

Satish Paranjpe, LDEO, current address: NEC Systems, Princeton, NJ, 08540

Abstract

Large oceanographic programs such as the Joint Global Ocean Flux Study require data management systems that enable the exchange and synthesis of extremely diverse and widely spread data sets. We have developed a distributed, object-based data management system for multidisciplinary, multi-institutional programs. It provides the capability for all the scientists to work with the data without regard for the storage format or for the actual location where the data resides. The approach used yields a powerful and extensible system, in the sense that data manipulation operations are not predefined. We have implemented UNIX and DOS versions of the system, including data transfer procedures, a wide variety of data objects, applications programs, and a number of user interfaces (including Matlab and World Wide Web [WWW] browsers).

Introduction

The Joint Global Ocean Flux Study (JGOFS), established under the auspices of the Scientific Committee for Ocean Research (SCOR), is intended to be a decade long internationally coordinated program. The main goal of JGOFS is "to determine and understand on a global scale the processes controlling the time-varying fluxes of carbon and associated biogenic elements in the ocean, and to evaluate the related exchanges with the atmosphere, sea floor and continental boundaries." "A long-term goal of JGOFS will be to establish strategies for observing, on long time scales, changes in ocean biogeochemical cycles in relation to climate change." (SCOR, 1987) A large number of scientists are participating from both U.S. and foreign institutions. JGOFS investigators have begun a set of time–series measurements and global surveys of a wide variety of biological, chemical, and physical quantities, detailed process–oriented studies, satellite observations of ocean color and wind stress, and modeling of the biogeochemical processes. These experiments will generate data in amounts unprecedented in the biological and chemical communities; rapid and effortless exchange of these data will be important to the success of JGOFS. Our project has developed a distributed, object–based data system which permits PI's to swap data, both preliminary and final, and, as they begin to analyze and synthesize their findings, enables them to retrieve desired information from an on–line archive. Recently, the system has been adopted by the U.S. GLOBEC Georges Bank investigators, who are faced with similar problems of synthesizing highly divers data sets, gathered by PI's at many different institutions,

Microcomputers and workstations have dramatically altered the gathering and analysis of oceanic data. While the convenience and ease of use of these machines make them ideal for individuals working on their data, the process of exchanging data or collecting relevant information from archived data sets has been difficult and daunting. There are relatively few chemical and biological data sets available in the major archives, different groups use different formats with different procedures for manipulating data, often data must be ordered in batch and arrives on a magnetic tape. Our system overcomes these difficulties and makes it possible for the user of a small computer connected to a network to be able to locate and work with data anywhere in a distributed data base without regards to its location or format.

Figure 1 sketches the overall design of the system:

1) data servers hold data "objects" which accept requests and reply with the appropriate information
2) the central JGOFS or GLOBEC server holds a comprehensive directory of the objects and other information about the program. Other servers can be invoked to generate plots, manipulate data, etc.
3) WWW browsers and other programs (including executable commands, shell scripts, a MATLAB interface, etc.) present the information to the user. The data can be saved on the local storage in a useful format (or one of a set of useful formats), or imported directly into a program for processing.

Essentially, the distributed data archive, containing the most current versions of the field data, as well as large amounts of historical information, appears as an extension of the scientists' own data bases — as readily available and as familiar in structure. The users need not know where the data is physically located nor how it is actually stored.

Approach

Our approach to data management has been to construct a system in which the data (at least during the period of the experiment) is not gathered into a central archive but rather resides at the originators' sites and represents the PI's current best versions of the data set. We chose to eliminate the need for the PI's to create special data sets for sharing with others (either directly or by submitting it to a program office), along with all the attendant problems: considerable effort is involved in generating special data sets, information sent to program offices is rarely current and rarely updated, and the additional delays involved are often significant.

The storage format is likewise the PI's choice. Because the JGOFS data set itself, not to mention the archives of historical data, is complex and diverse, and the data sets are being prepared using a wide variety of programs (e.g., spreadsheets, MATLAB, word processors, as well as data acquisition software), we cannot

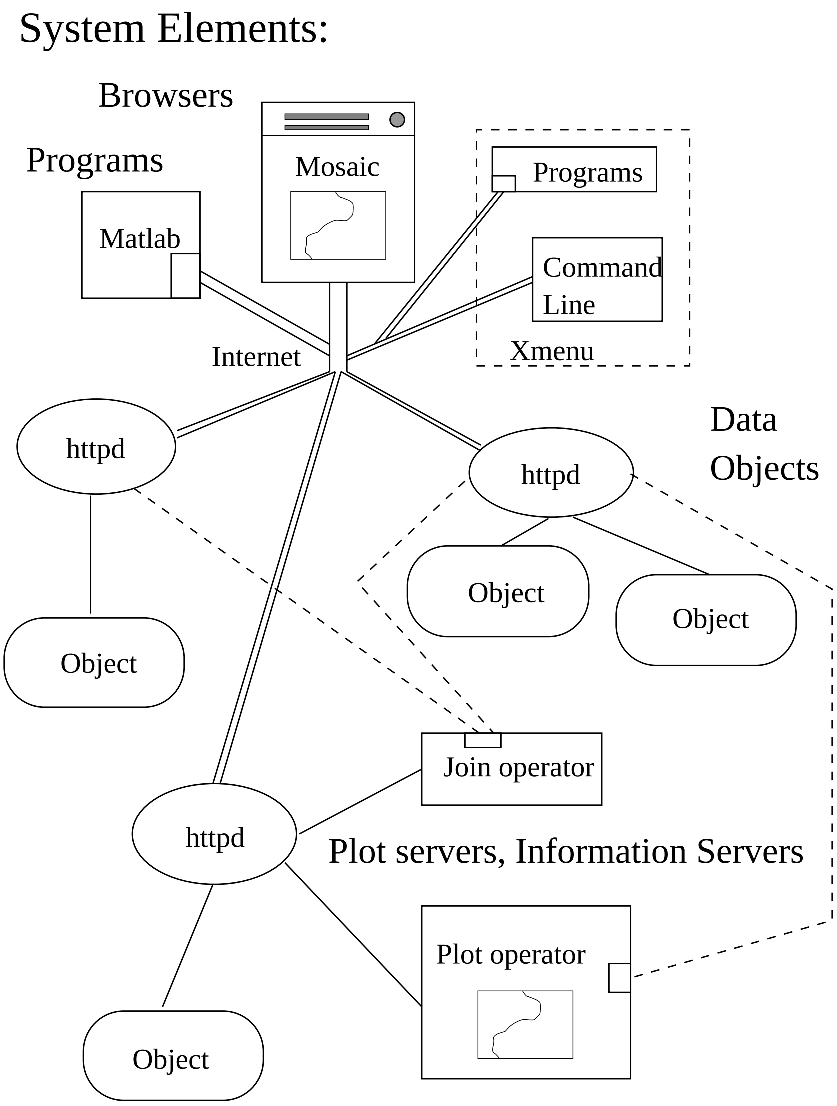


Fig. 1: Components of JGOFS data system

expect a process of forcing the users into a common mode to be very successful: it will not be possible to have all the functionality desired by every user available in any single program. The approach we have developed allows individuals to use freely the data storage and manipulation techniques they prefer, while still having straight–forward access to the entire JGOFS data base. Instead, our system provides a "window" through which others can read the data that the PI is using on the PI's machine (Figure 2).

Fig. 2: Data providers can edit and work with data in their own format. The "viewport/ translator" program provides access to outside users.

Although commercial database vendors are also moving towards distributed systems (Codd, 1990), there are several ways in which a scientific database for a program such as JGOFS differ from commercial needs. First, as mentioned, the heterogeneous software and also hardware environments of both data providers and users must be handled. Second, the updating cycle is quite different: data should be altered only by the PI responsible, not by any of the other users. Finally, a long–lived scientific dataset requires much additional documentation, and it is important that this information remain linked with the data itself.

Data Objects

By using "object–based" technology, we can unify information in different data base systems and/or storage formats. Data in archives need not be reformatted, nor are PI's limited in their choices of how they wish to store and manage their data, though the system also provides the capabilities of a scientific DBMS.

If the data is to be stored in many different ways — from ASCII tables, to multiple file databases, to binary floating point, to full database systems — and yet dealt with in a uniform fashion, there must be a translation step which enforces the commonality. The key to the "object–based" programming languages and data systems is that this translator is bound tightly with the actual data and the *only* access to the information is via messages passed through the translator. [2]

Thus, our basic building–block is an "object" which is a combination of data and a translator program capable of manipulating both the internal data and information passed from outside (Figure 3). Since the user's application program does not deal directly with the internal information but instead communicates with the translator, it is shielded from the complexities of the data storage and format. Each object receives a documented set of messages to which it responds with a documented set of replies. Thus a program which can retrieve information from one object or data type can retrieve similar information from any other data set in the system (or on the network). Plotting routines, for example, can display data from the user's own machine or from elsewhere with equal facility; figure 4 illustrates this point, superimposing data from six different sites, stored on workstations from various manufacturers, and in a number of formats. Table 1 shows the Matlab commands producing the plot and the details of the different objects.

[2] In object–based languages, this executable routine would be called a "method." Meyer, 1988, Date, 1990, and Wegner, 1990 describe in detail the distinctions between conventional systems and those built using "object–based" or "object–oriented" techniques for structuring.

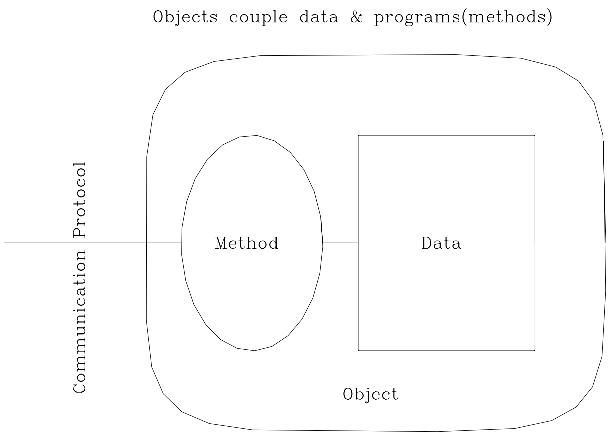


Fig. 3: The basic building block of object-oriented programming languages, the object. It is composed of a piece of code that knows how to read the data (the translator) and the data itself.

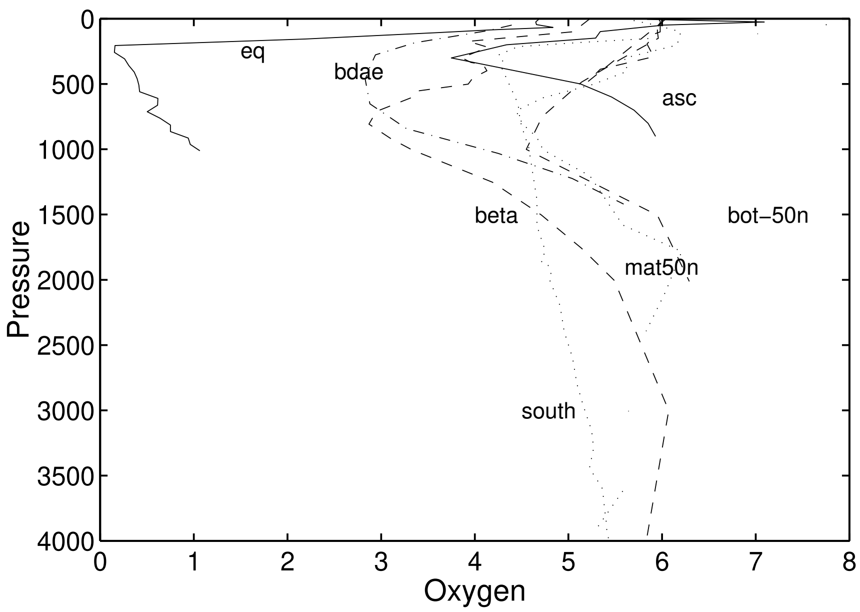


Fig. 4: A plot of oxygen vs. pressure from six different data objects with six different internal formats and stored in four different locations on the Internet. See Table 1 for details on location and storage format.

Table 1 Objects in Figure 4

object	location	machine type	storage method
asc	France	HP	flat ASCII file
beta	MIT	DEC Ultrix	scaled binary integers
bdae	Dartmouth	IBM	scaled binary integers
mat50n	MIT	Sun	MATLAB binary floating
eq	U. Victoria	Sun	multiple files/dirs
bot-50n	WHOI	Sun/	multiple files
south	U. Chicago	SGI	One file, mult. sta

The Matlab commands which created the plot are

```
loadjg('asc');plot(o2,press);hold on
loadjg('beta');plot(o2,press,'--');
loadjg('bdae');plot(o2,press,'-.');
loadjg('mat50n');plot(o2,press,':');
loadjg('eq');plot(o2,press);
loadjg('bot-50n');plot(o2,press,'--');
loadjg('south');plot(o2,press,':');
axis([0 8 0 4000]);axis('ij');hold off
-- additional commands for labeling, etc.
```

Requests and responses

We have based the "query–language" for subsetting upon standard Boolean algebraic expressions (for selecting based on the values of variables). We treat objects as functions — the queries are sent as arguments. Figure 5 illustrates this point, showing various subsets of the North Atlantic Bloom bottle cast data and the form of the object name used as the data source for the plotting command. In addition to constraining the information to be returned by a logical condition on the variable values, all objects permit specifying the subset of variables which the user wishes to see.

Some translators can handle messages beyond the basic set. For example, the translator for the CTD data in the North Atlantic Bloom archive deals with data at 2 decibar intervals, but permits the user to select different increments (e.g. 100 decibar) to reduce the volume of data. Likewise, it would be desirable to retrieve satellite data in discrete data form by asking for the value averaged over specified latitude and/or longitude bands. We are working on examples of such translators for numerical model output and for objective maps.

The responses from the translators have been carefully defined to permit transfer of hierarchically structured data of all types, including character, integers and real numbers, vectors and tensors. We have examined a number of general formats for guidance on the requirements for an interchange protocol; ours has many similarities to netCDF, with extensions and reorganization to better represent geophysical data such as sets of stations from a cruise. Also comments and attributes of the data (e.g. units) can be passed along with the data itself. The responses have three distinct formats (although this again could be expanded):

HyperText Markup Language (HTML) when the output is being displayed by a browser, simple flat–file tabular listings, and a format tailored for our data access subroutines.

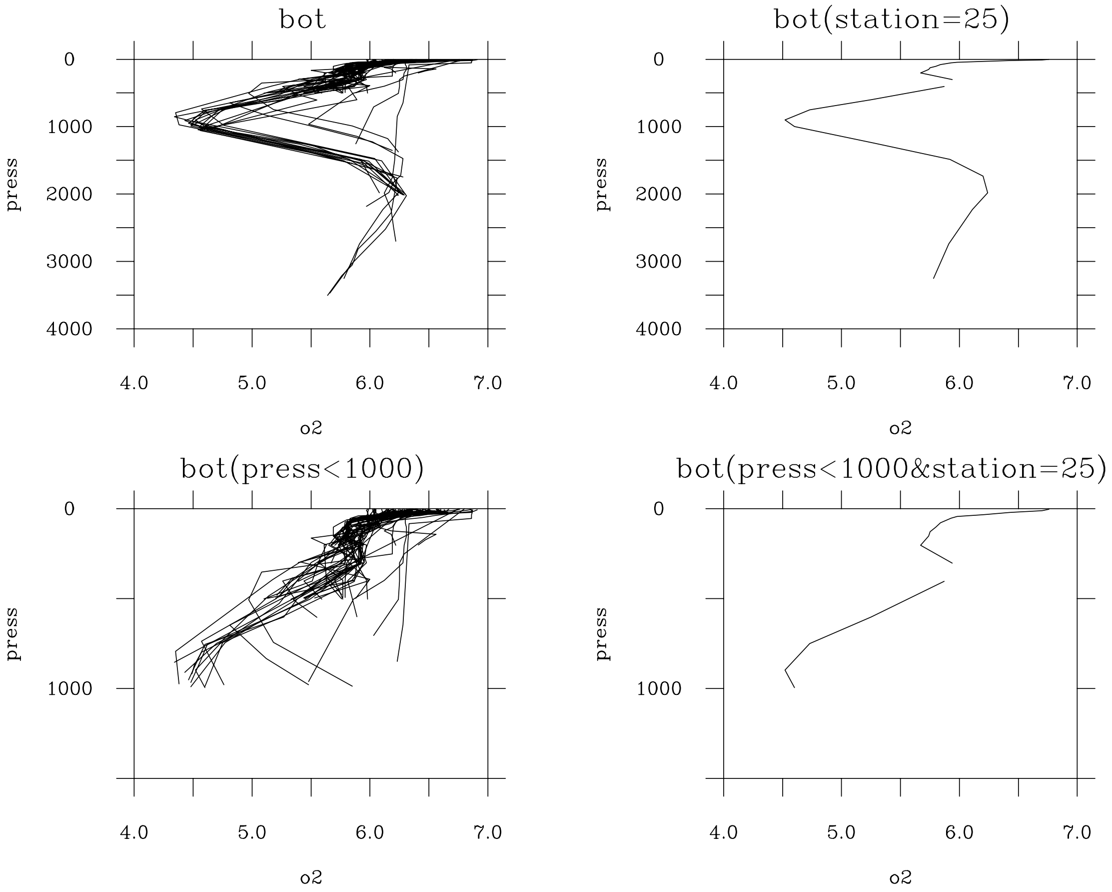


Fig. 5: Four different views of the same data object "bot". The different views were created by subsetting the data from the data object with four variations on the same command: in the upper left; `pl "bot" o2 press`, the upper right: `pl "bot(station=25)" o2 press`, the lower left: `pl "bot(press<1000)" o2 press`, and the lower right: `pl "bot(press<1000&station=25)" o2 press`.

The interchange format is certainly similar to self–describing formats such as the common data format CDF (Treinish and Gough, 1987), GF-3, DIF, netCDF, etc. These efforts provide a flexible approach to storing all kinds of data and sets of subroutines for retrieval and, in some cases, manipulation and plotting of the data. In the case of CDF and GF-3, arbitrary data types are accommodated and the organization of the data can be specified. However, these efforts are different from ours in that they expect all data to be entered into the specified (general) format, the procedures for accessing the data are generally oriented towards large machines (e.g., FORTRAN and magnetic tape-based for GF-3), and most of the software comes down from one group. In contrast, our translation takes place dynamically and invisibly; the goal is for each provider to be able to choose a storage format suited to the particular data set. At the same time, it is important to take advantage of these other efforts, both from the point of view of the data already available therefrom and because of the expertise and experience others have had. Our low level interchange protocol is essentially a general data format, and we have designed it after careful consideration of the previous efforts in this area.

In addition, these objects have an aspect of the “class” and “inheritance” characteristics of objects in programming languages: they can point to servers which handle other operations upon their data (e.g., graphical display, additional subselection, mathematical transformations, ...)

Directories

There must be an association between the name of a data object (such as `/jgofs/bloom/bot` for the bottle data from the North Atlantic Bloom study), the data files (`/d5/glenn/bloom/bot`) [or other required information such as the top–level directory for a multi–file, multi–directory data set], and the proper translator (`jgbl1`) for handling messages and retrieving the data. This association is contained in a directory structure, each node of which can contain a dictionary of locally held objects and a dictionary of remotely held objects. Table 2 shows part of the directory structure and the local/ remote dictionaries for the JGOFS study. Note that there could be, for example, several distinct CTD–formatted data bases which would be known to the system as different objects, but which share the same translator. The relationship is therefore one–to–many: a single translator may apply to many different data sets.

New data sets are added to the system in a simple way. The data must be placed on a machine which is connected to the network (this will often be the case already) and a translator and local/ remote dictionary entries provided. In many cases, users can choose to put their information in a form already handled by an existing translator so that a new one need not be written. Thus for most data sets the data can be made available simply by providing a directory entries. The data managers for the JGOFS or GLOBEC programs will have the responsibility for maintaining the master directories and verifying that the data is accessible.

Table 2. Segment of a dictionary

This segment shows the kind of directory information about a data set that can be incorporated in the dictionary as well as the form taken by a local data object [translator(parameters)] and a remote object

Directory Structure:
/jgofs
    /bloom
    /eqpac
        /tt007
        /tt008
  ...
Local Object Definitions:
tco2=jgbl2(/d5/glenn/bloom/bre)
ctd=jgbl(/d5/jgofs/bloom/ctd)
Remote Object Definitions:
snow=//dataone.whoi.edu/jgofs/bloom/snow
- V. Asper
- Abundance of aggregates *gt* 0.5 mm
ctd=//lake.mit.edu/jgofs/bloom/ctd
- R. Williams
- Hydrocast CTD Dataset

As Table 2 shows, the directories contain a brief summary of important information about the data object so that a scientist browsing the system can decide whether to examine the object further. This information appears in listings as shown in Figure 6. The data manager will also gather a fuller description of the information which can then be obtained from the data descriptor hyperlink (Figure 6b).

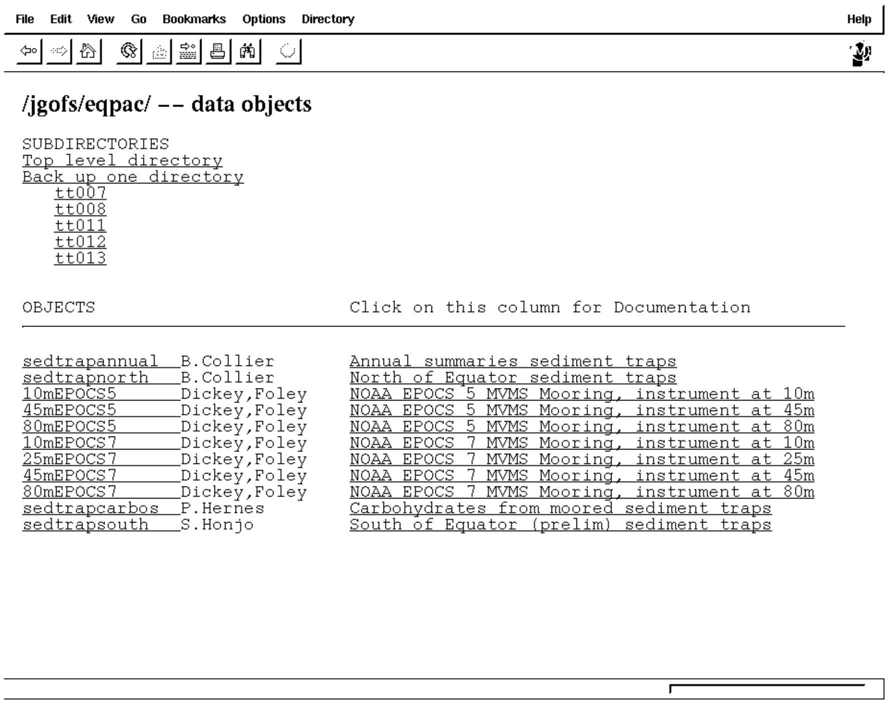


Fig. 6: Directory listings, showing the overall organization of the data and a detailed listing of the objects in one subdirectory.

Servers

In our implementations, translators are independent processes which transfer information via interprocess communications routines. Our system evolved from our own socket–based server and communications routines (resident on each machine) to one building on the now common HyperText Transport Protocol Daemon (httpd) developed at the National Center for Supercomputer Applications (NCSA). [and CERN?] In this case, we use the Common Gateway Interface (CGI) which is the way in which httpd permits executable programs to transmit information to the requesting machine easily. Since many scientists and labs are installing httpd on their systems to serve various documents and figures, we can take advantage of its widespread use to simplify our own software.[2] A JGOFS data server has two CGI's: one for transmitting directories of data known to this machine, and one for serving data objects actually contained thereon.

[2] An important software design principle is, of course, to let other people build and maintain the hard parts.

In the case of browsers, the directory function creates a hyperlink to the machine and directory name by which the object in known on that machine (figure 6). When invoked, the httpd process on the relevant machine is called with the object name and the set of parameters requested (figure 7). The JGOFS CGI program is started on the remote machine, and it looks the object up in the local directories, which resolve the object name into the name of the executable translator and additional parameters such as the file names it needs. The server is overlaid with the translator and the combined parameters are passed to it. The translator extracts the information and returns HTML text to the browser.

For access via programs, the equivalent of an "open" call, which works with a data object name, first looks up the desired object in a set of local system or user directories if the machine name is not specified explicitly. The look–up resolves the object to either a local translator and a set of parameters derived by combining those in the directory (for example, file names) with the parameters supplied by the user. In the case of a local object the appropriate translator is started as a subprocess, with the parameters being passed, and it is connected to the application. It then returns the data to the program. In this sense, the translator acts like an input subroutine which the main program calls to get data from files. However, when the object is resolved to a remote address, the httpd on the remote machine is called with the object and parameters. It then behaves as described above.

User Interfaces and Programs

User interfaces are often an integral part of a database system, with program language support an afterthought. We have taken the opposite approach, defining the application program interface (currently consisting of 7 functions/ subroutines). The application program interface subroutines are linked with user programs, allowing them to extract data from local and remote objects transparently. Most programs such as plotting or analysis routines will need to use only the "open," "read," and "close" function calls. Thus, the application side of our system consists of a set of independent programs which can be invoked from the command line or shell script. In addition, we provide an X–based graphical interface (figure 8) which is responsible for initiating these and passing them the required parameters. Again, this allows the system to be extended with the extensions being incorporated into the user interface without requiring modifications to existing code. In addition, we have a text–based menu system, an `loadjg` function for Matlab which mimics the `load` function but gathers data over the network from data sets, including, of course, transparent reformatting.

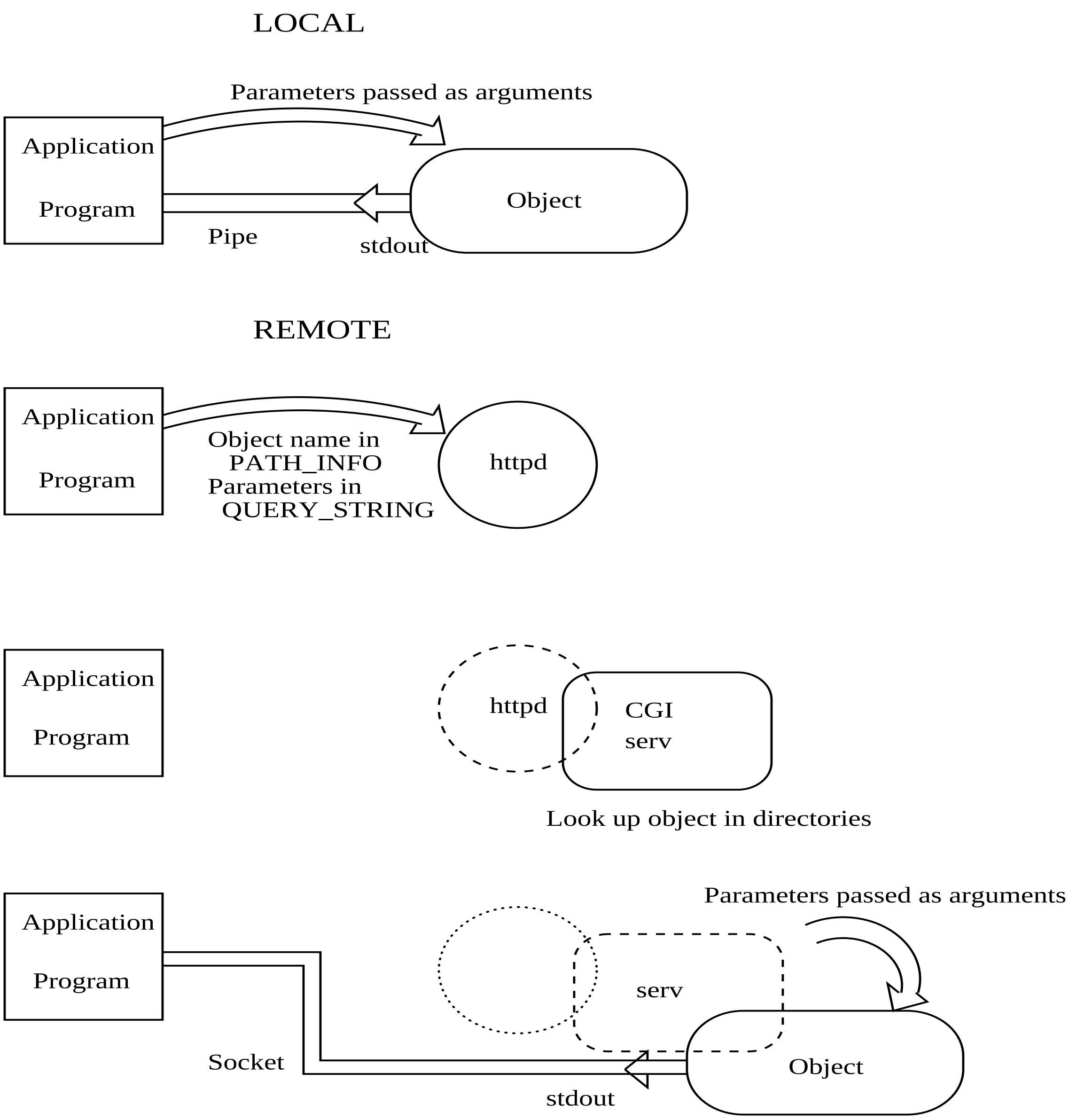


Fig. 7: A diagram showing how the software connects applications making requests to the objects that can fulfill those requests. When the object is not local, the request is routed to an httpd server on a remote machine.

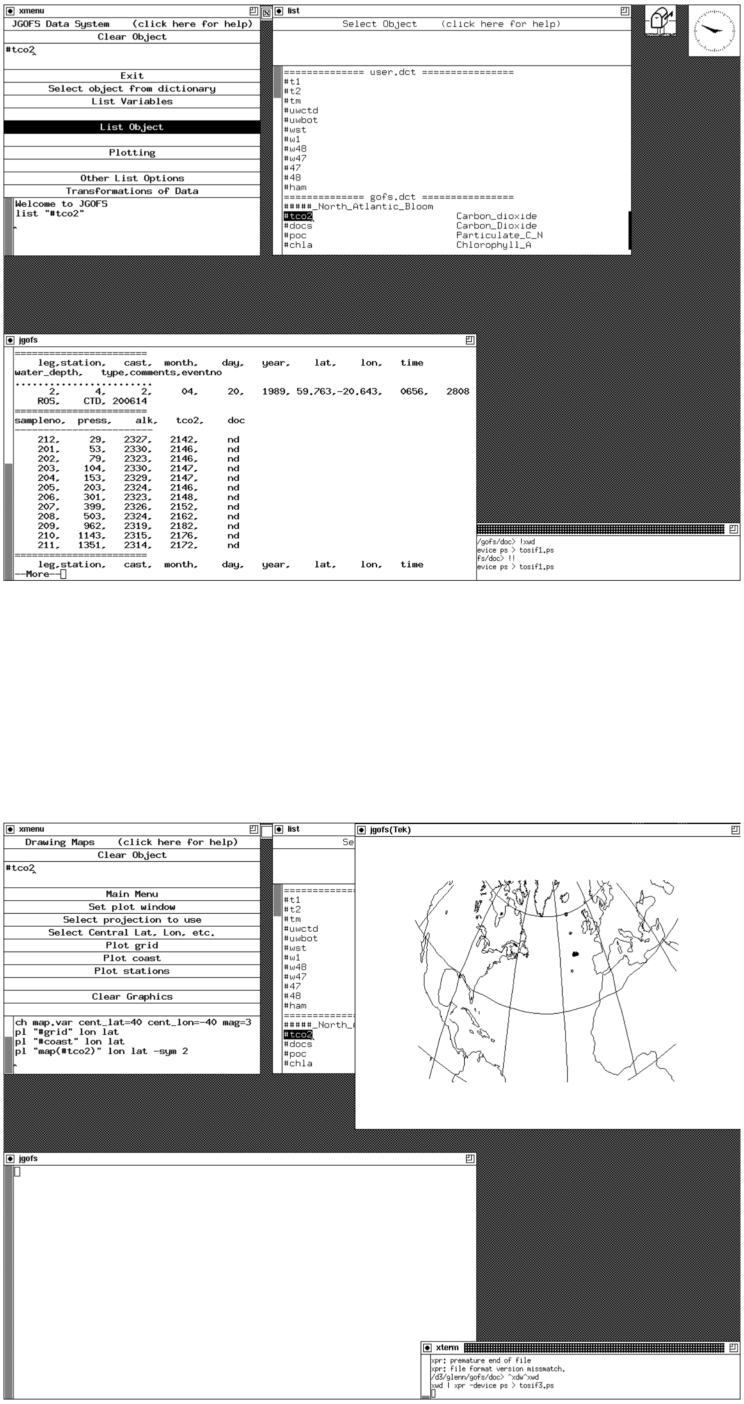
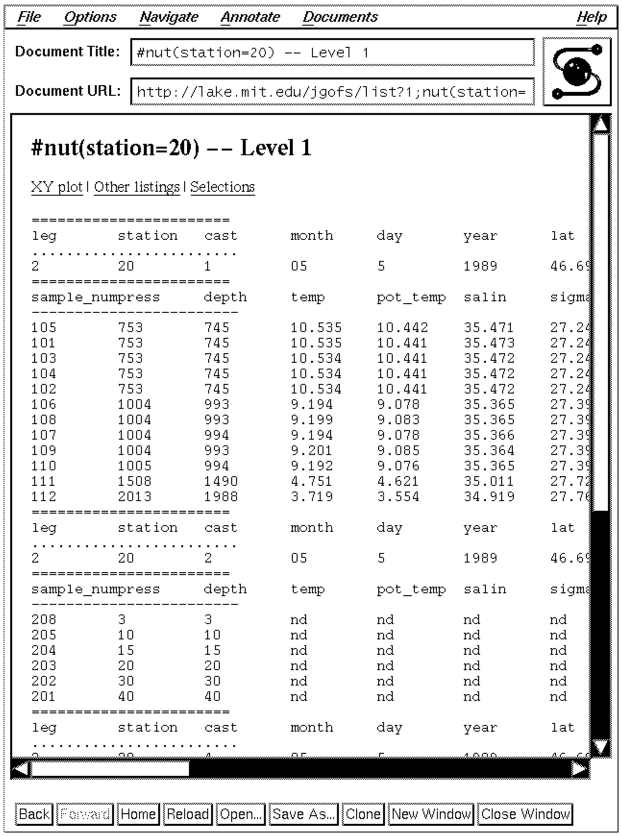


Fig. 8: Screens generated using the X–based menu system.

We also have a GUI which uses a WWW browser to access local and remote data sets. Finally, the full HTML–based (HyperText Markup Language) system allows one to "walk" through the dictionary and data sets to find the information of interest. One can then make tables, files, or plots of the information. In this case, the WWW browser is communicating directly with the translator (via the CGI facility in httpd), asking for particular subsets of the data to be constructed and returned as HTML documents.



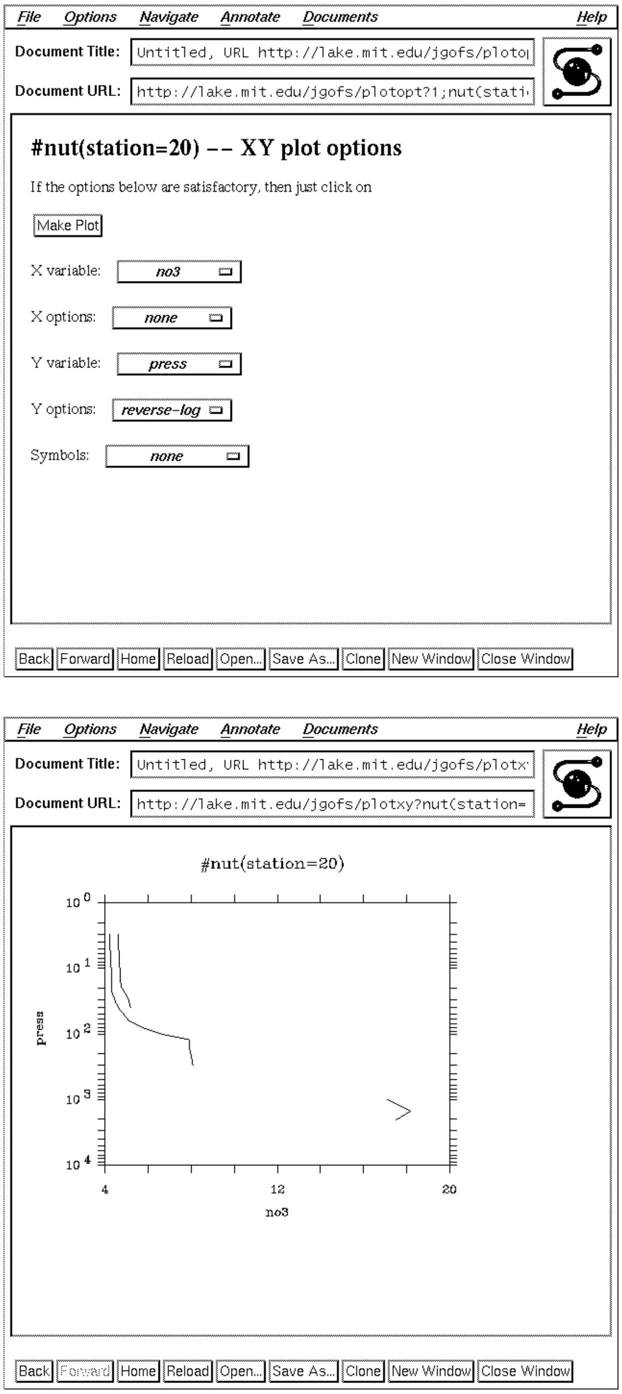


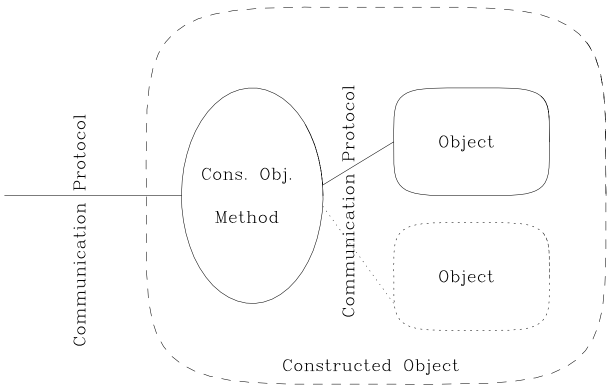
Fig. 9: Screens from the WWW interface

Data Base Operations

Up to this point, we have described essentially a "data independent" method for exchanging data and working with a distributed data base. But data base systems also provide routines for manipulating data. For example, a relational data base, which basically works with tabular data in columns with column headings, usually permits selecting by row or column and joining two tables together based on common columns (e.g. tables of data from two PI's working with water samples from the same Niskin bottles could be joined by matching the cast number/bottle number).

An object–based system permits data operations in a very simple and flexible manner: in addition to objects directly related to a set of data, the system permits what we call "constructed objects" (Figure 10a). The examples show (10b) an object created by calculations on the data in another object and (10c) an object which incorporates and melds information from two other objects `bloom/tco2` and `bloom/poc`. Constructed objects consist of translators which take their information not from data files but from other objects; the information is transformed in some way by the program and then passed to the application. While these appear functionally the same to the user's programs, the data retrieved from these objects is created or modified "on the fly." Since each of the translators for constructed objects is an independent program, the system is indefinitely extensible.

Constructed Objects build new objects from old ones



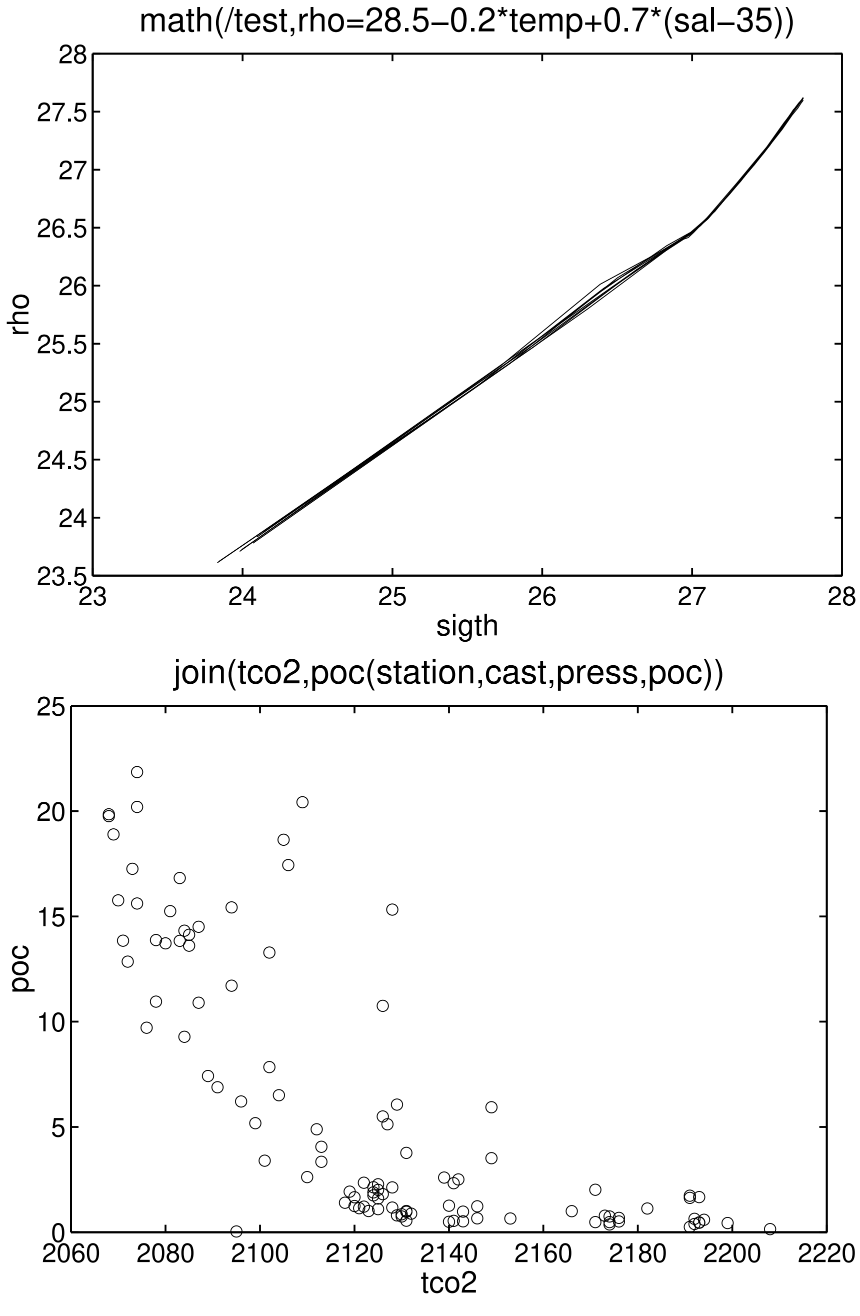


Fig. 10: a) A block diagram of a constructed object showing the translator which manipulates information and one or more objects which provide information from other data sets. b) A plot using "`math`" which adds a new computed column to the data set. c) A plot created by using the constructed object translator "`join`". The command was:

```
loadjg('join(/jgofs/bloom/tco2,/jgofs/bloom/poc(station,cast,press,poc))') plot(tco2,poc,'o')
```

To recap, we have implemented the database operations of (a) selecting data by various criteria, (b) choosing which variables to include, (c) mathematical operations, and (d) combining information from two objects based on common information — a kind of join operation. In addition, we have prepared examples of more specialized oceanographic operations such as dynamic height computation and mapping onto many different map projections. The advantage of the object–oriented approach is that other operations — e.g., interpolation of one data set against another — can be readily implemented and added to the system at any time and by anyone. The constructed object translators need not be on the user's machine: the request can be addressed to another machine just as in the case of a regular object.

Summary

Our data base system thus is distinguished from conventional and available systems by five important features:

1) the ability to handle data in arbitrary formats
2) data transfer from remote, networked data sets
3) extensible — data manipulation routines or relational functions can be added at any time
4) new data can be added to the system in a simple way without a lengthy conversion
5) this system can be used either interactively or with user–written programs.

We have constructed servers, a number of different translators, and constructed objects. We have much of the North Atlantic Bloom data, the EqPac data, and the HOTS data sets on–line, along with various historical data sets and some data sets from the SYNOP (Synoptic Ocean Prediction) program and the Polymode Local Dynamics Experiment. These verify the system's ability to handle a wide range of data types and internal formats. For on–going projects, on–line access to current data sets will be essential, and the providers must have considerable freedom in how they build their files and systems. Likewise, the capability building "extensible" data systems, analysis packages, and graphics packages offers significant improvements in our abilities to share software. Our system provides a solid basis for information management in multi–institutional, multi–disciplinary programs.

Acknowledgements

This work was supported by NSF grants OCE–16502 to MIT, OCE 91–17013 to LDEO, and OCE 91–16502 to WHOI. This is WHOI contribution number 9099. We'd also like to thank Christine Hammond and Robert Groman for their suggestions and advice.

Appendix: Interchange Protocol

At the core of our system is the interchange protocol. It must be sufficiently flexible to transmit data and information of many different kinds, yet also simple enough that writing translators and inverse translators is not too difficult. At the same time, it would place serious limits on the system if not sufficiently flexible. For our interchange protocol, we have drawn ideas from netCDF (Rew, *et al.*, 1993), but added hierarchical structure and more straightforward comment capability. The application sends the translator its arguments and requests information. The replies from the translator consist of:

1) **Comments:** Plain text descriptive material which is transmitted with the data set. Here the data provider can describe details of the data acquisition, processing, and interpretation. References to relevant articles can be included. Such information is vital for a data set which has long–term value.
2) **Variables:** The data is identified by named variables. These are grouped into different levels (e.g., cruise header, station header, station data). Within each group is a sequence of variable declarations. These consist of
   A) **Variable name:** These must be unique within a data object. There is also a considerable advantage to using common names and units throughout a program (some of this work can be done by the translator).
   B) **Size, Dimensions:** For vector/ tensor quantities, these give the total size and the shape of the information.
   C) **Attributes:** Ancillary information on the variable, in the form of strings `attribute=value`, can be included. Attributes include units, reference to methodology/ instrumentation, etc.
3) **Data:** The data values are all transmitted as ASCII strings so that there is no intrinsic data typing. To understand the sequence of data transmission, consider a 3 level hierarchy as sketched in Figure 11. The first row of each level is transmitted; next comes the second and then following rows at the lowest level. When data in one of the higher levels changes, then that row is sent along with the data from lower levels. This process minimizes repeat transmission of data which has not changed.
4) **End:** Signals that all the data has been returned.

Note that the data model we employ is effectively equivalent to a relational table (augmented by comments and attributes), if one defines an operation which ungroups or flattens the data set and a grouping operator. Thus, all of the access operations common with relational databases can be implemented with our expanded model, although some care would be required in order to automatically ungroup and regroup data.

Next consider the queries which the translator receives. When it first begins, the translator is passed a set of parameters — in the examples, these appear as arguments with the translator or object name appearing as the function. Among these parameter strings, each program is expected to handle

1) **Projection:** Choice of which variables the translator is to return.
2) **Selection:** Restrictions, based on the usual logical operations $<$, $=$, $>$, $<=$, $<>$, $>=$. These restrictions can be combined with AND, OR, and NOT operators, as well as parentheses to form a full logical expression.

As described previously, tranlators may handle more than just these parameters; for example, the mathematical translator deals with strings of the form `variable=expression`, with the `variable` perhaps being a new name and the `expression` being a standard FORTRAN style mathematical formula.

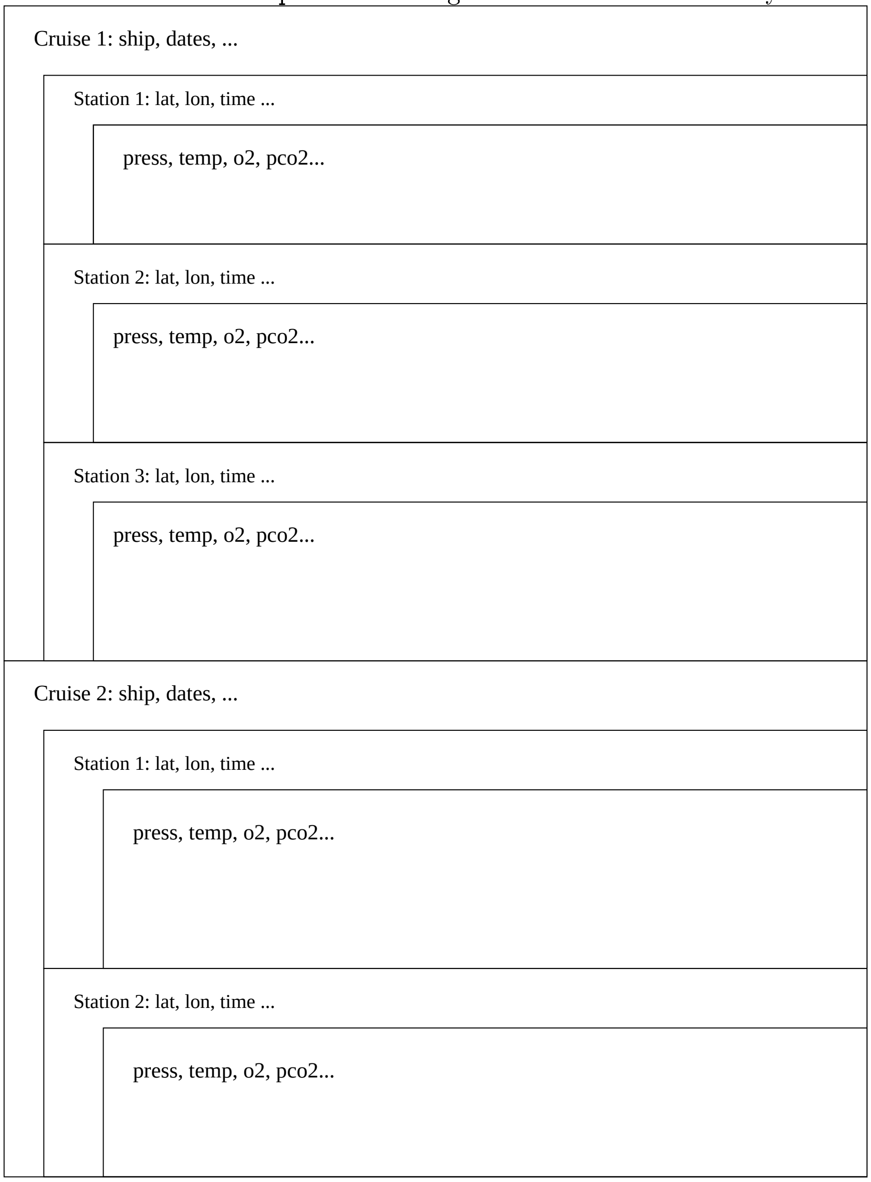


Fig. 11: A diagram of a hierarchical structured set of data tables.

References

Codd, E.F., (1990), *The Relational Model for Database Management, version 2*, Addison Wesley Publishing Co., Reading, MA, 538 pg.

Date, C.J., (1990), *An Introduction to Database Systems, vol. 1*, Addison-Wesley Publishing Co., Reading, MA, 5th edition, 854 pg.

Dittrich, K. and U. Duval, (ed.), (1986), *Proceedings of the 1986 International Workshop on Object–Oriented Database Systems.* IEEE Computer Society Press, Washington, D.C., 237 pp.

Meyer, B., (1988), *Object–Oriented Software Construction*, Prentice Hall, New York, NY, 534 pg.

National Academy of Sciences – National Research Council, (1991), *A U.S. Strategy for Global Change Data and Information Management.* (in prep.) Committee on Geophysical Data, Board on Earth Sciences and Resources. Natl. Acad. Press, Wash., DC, ca 33 pp.

Rew, R., G. Davis, and S. Emmerson (1993) *NetCDF User's Guide, Version 2.3* available from the Unidata Program Center, P.O. Box 3000, Boulder, CO., 80307.

SCOR (1987), *The Joint Global Ocean Flux Study: Background, Goals, Organization and Next Steps, Report of the International Scientific Planning and Coordination Meeting for Global Ocean Flux Studies, Paris, 2/17–19/87.* Available for SCOR Secretariat, Dept. of Oceanography, Dalhousie Univ., Halifax, Nova Scotia, Canada, B3H 4J1, 42 pp. See also *The Joint Global Ocean Flux Study: North Atlantic Planning Workshop, Paris, 9/7–11/87.*

Treinish, L.A. and M.L. Gough, (1987), A software package for the data-independent management of multidimensional data. *Trans. Am Geophys. Union, EOS*, **.** 68(22): 633–635.

Waldrop, M.M., (1987), Artificial intelligence moves into the mainstream. *Science*, **237**: 484–486.

Wegner, P., (1990), Concepts and Paradigms of Objected Programs. *OOPS Messenger (ACM)*, **1**: 7–87.