N93-24935

Data Management for JGOFS: Theory and Design

Glenn R. Flierl, James K.B. Bishop, David M. Glover, Satish Paranjpe

Introduction

The Joint Global Ocean Flux Study (JGOFS), currently being organized 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." Participation from a large number of U.S. and foreign institutions is expected. 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 bio-geochemical 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.

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 an individual working on his or her data, the process of exchanging data or collecting relevant information from archived data sets is still difficult and daunting. Everyone uses different formats with different procedures for manipulating data; there is relatively little chemical and biological data available in the archives at NODC and it must be ordered in batch and arrives on a magnetic tape. We believe that this difficulty can be overcome—it should be possible for the user of a small computer connected to a network to be able to locate and work with data at NODC or indeed anywhere in a distributed data base without regards to its location or format.

Envision being able to sit at a microcomputer and ask what sets of phosphate data are available and then, based on the reply, ask for some suitable subset. The data could be imported to your local storage where it would arrive in the format you are accustomed to using for your own data or it could be used directly for creating a plot or as part of a calculation. Essentially, the JGOFS distributed data archive, as well as large amounts of historical data, would appear to be an extension of one's own data base—as readily available and as familiar in struc-

ture. The user would not need to know where the data is physically located nor how it is actually stored. We believe that the synthesis of the data from large experiments can only be accomplished when this kind of data exchange can occur (see Codd, 1990 for a discussion of the advantages and disadvantages of distributed database management).

Approach

Our approach to data management has been to construct a system in which the data (ideally) is not gathered into a central archive but rather resides at the originator s site and represents the PI's current best version of the data set. The storage format is likewise the PI's choice. Others can access the data without regard to storage method or location.

We are faced with requirements to manage and integrate extremely diverse sets of data. At the same time, many of the potential JGOFS PI's do not have extensive experience with large computers and data bases. With the rapidly growing capabilities of microcomputers and greatly reduced cost of workstations, however, we expect that most data gathering and preliminary analysis will be done on such machines. Based on these considerations we have built our JGOFS data system to satisfy two primary requirements: 1) A simple, usable, and flexible data base for micros/ workstations which can be used for data management of an individual PI's data sets and those which he or she collects from other investigators and archives. 2) Straightforward (or if possible even transparent) linkage to data sets on networked (SUN, VAX ... Cray) machines. Many data base programs exist for small machines and, in and of itself, there is little value in developing another one. Rather, we have begun building an "object-oriented" (to be defined in more detail below) data base which has many unique features making it especially suitable for experiments such as JGOFS and WOCE. For the new initiatives on Global Change, data management will likewise be of fundamental importance (Natl. Acad. Sci., 1991). Systems such as ours provide the flexibility to handle such widely diverse data sets from many different sites and to integrate the information into useful form.

Object-Oriented Data Bases

The basic element of the so-called "object oriented" programming languages and systems is an "object" which is a combination of data and programs capable of manipulating both the internal data and information passed from outside (Fig. 1). (We summarize the unfamiliar terms in Table 1 for reference.) The user does not deal directly with the internal information (and is therefore shielded from the complexities of its storage and format) but instead communicates with the program part of the object by passing it "messages" which cause it to calculate the appropriate response and return a message to the inquirer. Thus, the details of the manipulations are also generally hidden from the user; rather each object can receive a documented set of messages to which it responds with a documented set

of replies. We shall call the program part of the object the "method" (this terminology is a little simplified over that of object oriented languages such as SMALLTALK).

Object-oriented languages are gaining popularity because their modular structure allows building complex programs from simpler, individually tested components. Refer to Meyer (1988) and Date (1990) for discussions of this type of language. A more recent review, by Wegner (1990) describes detailed distinctions between conventional systems and those built using "object-based" or "object oriented" techniques for structuring. The inherent modularity of object-oriented programming allows for rapid testing of new ideas and easy changes, since usually only one object is involved and the other objects which have been combined to perform some operation can proceed unchanged (Waldrop, 1987).

Table 1 Definitions

object: a combination of data and program into an entity which interfaces with user programs.

message: information passed between user programs and objects or between various objects.

method: the program part of an object. Receives messages massages the relevant data sets and returns messages with the appropriate answers.

dictionary: a table connecting the name of an object with the data file and the method which make up the object.

constructed object: an entity which appears to be an object to user programs but which actually has no data set directly associated with it; instead the method for the constructed object requests information from other methods and manipulates the data in the replies to generate the data necessary to respond to the requests from the user.

server: the program responsible for channeling messages to the proper methods. It will also deal with the connections to networks and talk with servers on the other machines.

relational data base (RDB): a data base which deals with data sets organized as tables. Relations among various data sets are based on commonality of information in one or more columns of each of the data sets. E.g., two separate tables could represent the results of different investigators' processing of data from water samples; the data sets would be inter-related using the commonality of cast and bottle numbers.

inventory: a listing of data in a local data set

catalog: information about the existence contents, and procedures for acquiring various data sets; may also contain additional information to help users judge the relevance of the data base to their particular needs.

While a subroutine in FORTRAN, such as a matrix inversion routine, represents a simple form of an object, the concept gains considerably in power when applied to data objects (c.f., Dittrich and Duval, 1986). By packaging data with

programs, the user need not know the detailed methods and formats of data storage and can deal with any data in the system on an equal footing. Any 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 plot data from the user's own machine or from elsewhere with equal facility. Figure 2 illustrates this point, superimposing data from Stommel's atlas (stored on a PC and from the North Atlantic Bloom experiment (on a SUN). The commands producing the plot are also shown; note the commonality between the "function" style of referencing the data objects.

The basic set of queries and answers has been carefully defined to permit transfer of hierarchically structured data of all types including character, integers and real numbers, vectors and tensors. We have studied a number of general formats for guidance on the requirements for an interchange protocol; ours corresponds to netCDF, with extensions and reorganization to better represent geophysical data. Also comments and attributes of the data (e.g. units) can be passed along with the data itself. Queries include the option of subsetting the data in various ways. Some methods can handle messages beyond the basic set. For example, the method 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 an example of such a method for numerical model output and for objective maps. Each method will be able to handle a message asking it for help on the method's capabilities. There must be an association between the name of a data object (e.g. '#bot" for the bottle data from the Bloom study), the data files (bloom/bot*), and the proper method (bloom/jgbl) for handling messages working with the data in these files. This association is contained in a dictionary. ' Figure 3 shows part of the dictionary for the JGOFS North Atlantic Bloom 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 method. The relationship is therefore one-to many a single method 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 method and dictionary entry provided. In many cases, users can choose to put their information in a form already handled by an existing method so that a new one need not be written. Thus for most data sets the data can be made available simply by providing a dictionary entry. We envision the data manager for the JGOFS program as having the responsibility for maintaining the catalog and verifying that the data is accessible. For the bloom study, we have constructed a prototype information object which can be viewed with the same software as any other object and can tell the user, for example. which data sets have total CO_2 measurements (Figure 4). Programs working in a 'data independent" way with the JGOFS data base will communicate with a 'server" (Fig. 5) which consults the dictionary and passes information and requests to the proper method. The server then returns the data to the program. In this sense, the server acts like an input

subroutine which the main program calls to get data from files. However, the server also acts to gather data from across the network. If desired, the server may send a message to a server on a different machine, asking for particular data objects. Subroutines which interface with the server directly have been constructed.

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).

(see Figures 3 and 4)

An object oriented system permits data operations in a very simple and flexible manner: in addition to objects directly related to a set of data. the server will be able to deal with what we might call constructed objects" (Fig. 6). While these appear functionally the same to the user's programs, the data retrieved from these objects is, created "on the fly." Since each of the methods for constructed objects is an independent program. the system is indefinitely extensible. While this discussion has been phrased in terms of a relational model the individual data sets may be organized hierarchically or in some other manner. We have currently implemented the operations of selecting data by various criteria, choosing which columns to examine, mathematical operations and a join facility. In addition we have prepared examples of more specialized oceanographic operations such as dynamic height computation and mapping onto many different map projections.

We comment briefly here on the distinction between our proposed system and "data independent" formats such as the common data format CDF (Treinish and Gough, 1987), GF-3. DIF the UNIDATA program. 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 "top-down," in the sense that all data must 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, we are adopting a "bottom-up" approach for which the goal is for each user to be able to work with data as if it were all stored in the fashion he or she prefers. There is no requirement to conform to a common standard (although if the

individual's preferences are quite different from anyone else's, it will be necessary to write a method). Given the proliferation and increasing power of microcomputers and the wide variety of systems for handling data (RS-1, STATPACK,... as well as those mentioned previously), we cannot 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 propose 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.

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 (though, of course, the details are largely hidden from most users by the method programs).

Using the JGOFS System

To illustrate ways of using the system we describe two different approaches (Fig. 7): 1) For users without other data bases or those who prefer to use one of our general formats for storing their data we provide a fairly extensive set of tools for handling the data. These are the programs and constructed objects that we will use in our own work. In addition, as more scientists use the system and develop their own software, we can provide an extensive set of "groupware ' so that one may be able to find a routine to execute the desired operation. The system will have much of the functionality of conventional data bases. Remotely-stored data can be either retrieved and saved on the local storage medium or may be used directly. 2) For users who are already handling data with a commercial data base system (e.g. LOTUS), the system would most likely be used with a program which talks to the server and writes results out in the form of a LOTUS data set (called "extr" in the PC version). It is then possible to retrieve remote data sets and have them arrive on the local system ready to be used directly—as far as the user is concerned the remote data sets are stored in LOTUS format. Secondly, by using a method" for converting LOTUS data. one could take advantage of data manipulation capabilities of the data base system which may not exist in LOTUS. Essentially the program would ask the server to use a constructed object which itself requests other data from the server. This request for other data is passed along to the "method" for LOTUS data which retrieves the information from the LOTUS files. The data flows back to the constructed object which transforms it and passes it back to the main program. If the main program writes the results from an object out in LOTUS format, the desired result is obtained: a new LOTUS file has been created by operating on one or more old LOTUS files. The flow of data in this kind of operation is sketched in Fig. 7d. Note that some of the information could actually come across the network from files in completely foreign formats. Remember that the use of LOTUS in the paragraph above is only for example and is by no

means restrictive: similar capabilities can be made available to DBASE, netCDF, GF-3 etc. users.

Our data base system thus has five important features which distinguish it from conventional and available systems: 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 believe, based on experiences with various data sets and large oceanographic experiments. that these features are very valuable for JGOFS, WOCE, and Global Change.

Implementation

User Programs, Methods, Constructed Objects,

In our implementations, methods are separate processes which transfer information via interprocess communications routines. When the data object is opened, the method is started up and parameters are passed to it. The calling process then begins to receive information from the method as outlined in the 'Interchange Protocol" section below. Because of the differences between a multiprocessing system such as UNIX and MS-DOS, we discuss each of these separately.

UNIX: In the UNIX implementation, the request to open a data object is passed via a queue to a resident server daemon. This forks a process to analyze the request, do the dictionary lookup, and begin the method. If the method program is local. it is started up and the parameters are passed to it using normal stdin/ stdout pipes. The responses from the method are returned to the user program on the queue (Fig. 8). On the other hand, if the method is on a remote machine, the request is transmitted via a socket to a resident server on the remote machine. which again handles dictionary lookup, and starts up the method process. This time, however the responses from the method are passed back via the socket to the user process (Fig. 9).

MS-DOS: Here we have implemented a small resident set of routines which handle starting a subprocess and passing information back and forth. The user program first must release unneeded memory (this is often done as compiler options); it then handles the dictionary lookup and executes an interrupt to the server. The server executes the method process. The method generates an interrupt to the server, which then flips control back to the user program. Message passing then occurs as a sequence of such tips; the message string is pointed to by the registers at the time of the interrupt, and the server copies the string from the sender's area to the receiver's area when control is exchanged. The communications routine uses the serial port. First the user connects to the networked server

with a standard login process. The desired object is accessed with a method which talks to the serial port (e.g., v24 for 2400 baud).

Interchange Protocol

At the core of the object oriented 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 methods and inverse methods is not too difficult. At the same time, it place serious limits on the system if not sufficiently flexible. We have based our interchange format on netCDF, expanded to include hierarchical structure and comments. We begin our description of the basic protocol by discussing the replies which come from the method:

1. ***Comments***: Plain text descriptive material which is transmitted with the data set. Here the scientist can describe details of the data acquisition, processing, and interpretation. References to relevant articles can be provided. 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 method).
   B. ***Size, Dimensions***: For vector/ tensor quantities. these give the total size and the shape of the information.
   C. ***Attributes***: Here is given ancillary information on the variable. These take the form of strings attribute=value, along with a count.
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 10. The first row of each level is transmitted; then the second and following rows at the lowest level. When data is one of the higher levels changes, then that row is sent along with the appropriate data from lower levels.
4. ***End***: Signals all the data has been returned. Note that the data model employed is effectively equivalent to a relational model (augmented by comments and attributes) if one defines an operation which ungroups or flattens the data set and a grouping operator. Thus, we expect all of the operations common with relational databases can be implemented with our expanded model, although some care is required in order to automatically ungroup and regroup data.

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

1. ***Projection***: Choice of which variables the method is to return.
2. ***Selection***: Restrictions, based on the usual logical operations <,=, >,<=,<>, >=. These selections are ANDed together.
3. ***Help***: (Not implemented in prototypes.) Methods should be able to inform the

calling programs about the kinds of arguments they can handle. As described previously, methods may handle more than just these parameters; for example, the mathematical method 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.

After the initial parameters are passed and checked, the method then simply receives requests for the next chunk of information and returns the next "row" of data. Or it may receive a request to terminate.

Summary

We have constructed prototypes of the servers, methods, and constructed objects. We have much of the North Atlantic Bloom data entered into the system, along with various historical data sets and (separately) data from the SYNOP program. The process of documenting and training users will begin this year; assessment of the merits of the approach are still to come. However, we believe that, for on-going projects, on-line access to current data sets has many advantages. Likewise, the idea of building "extensible" data systems, analysis packages, and graphics packages should offer significant improvements in our abilities to share software.

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 pp.

Dittrich, K. and U. Daval, (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, N.Y., 534 pp.

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.

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: 484-186.

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

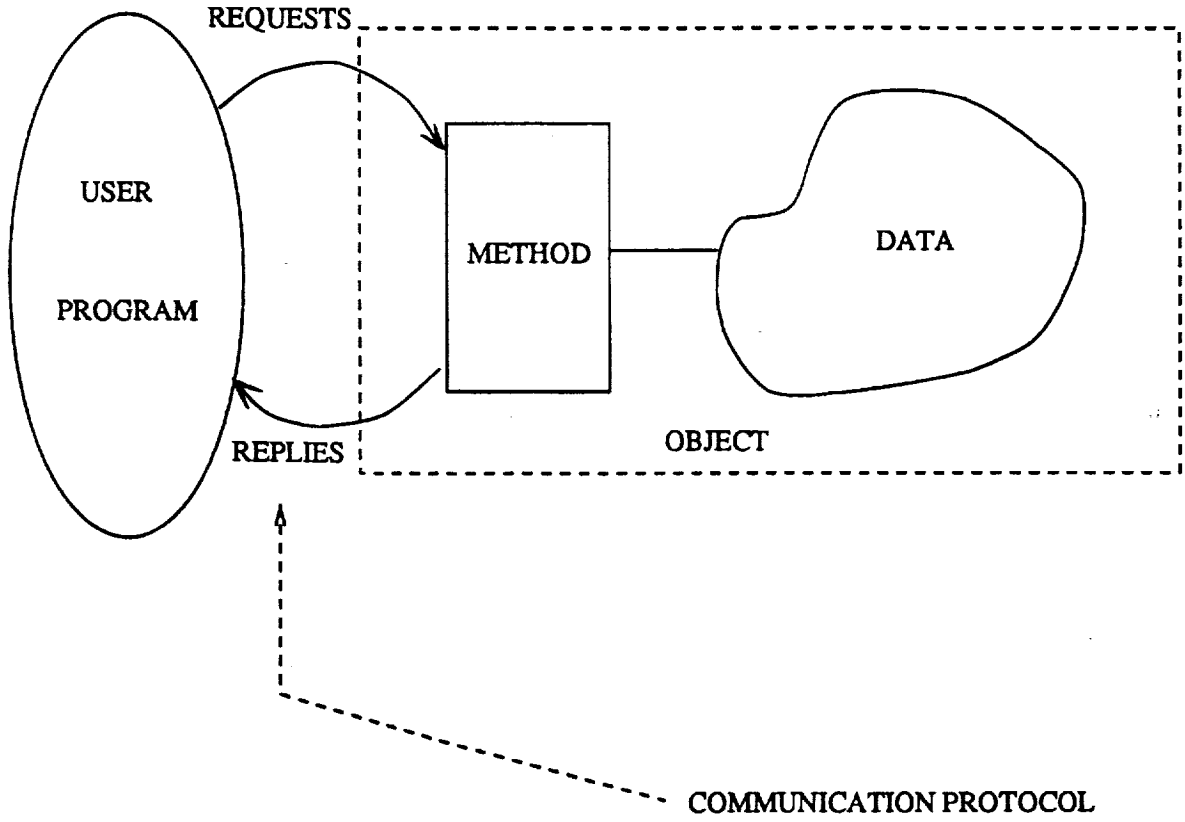

Figure 1. A data object packages together data and a program called a method. The data system accesses the information solely through requests and replies sent to the method. This communication protocol is common for all methods.

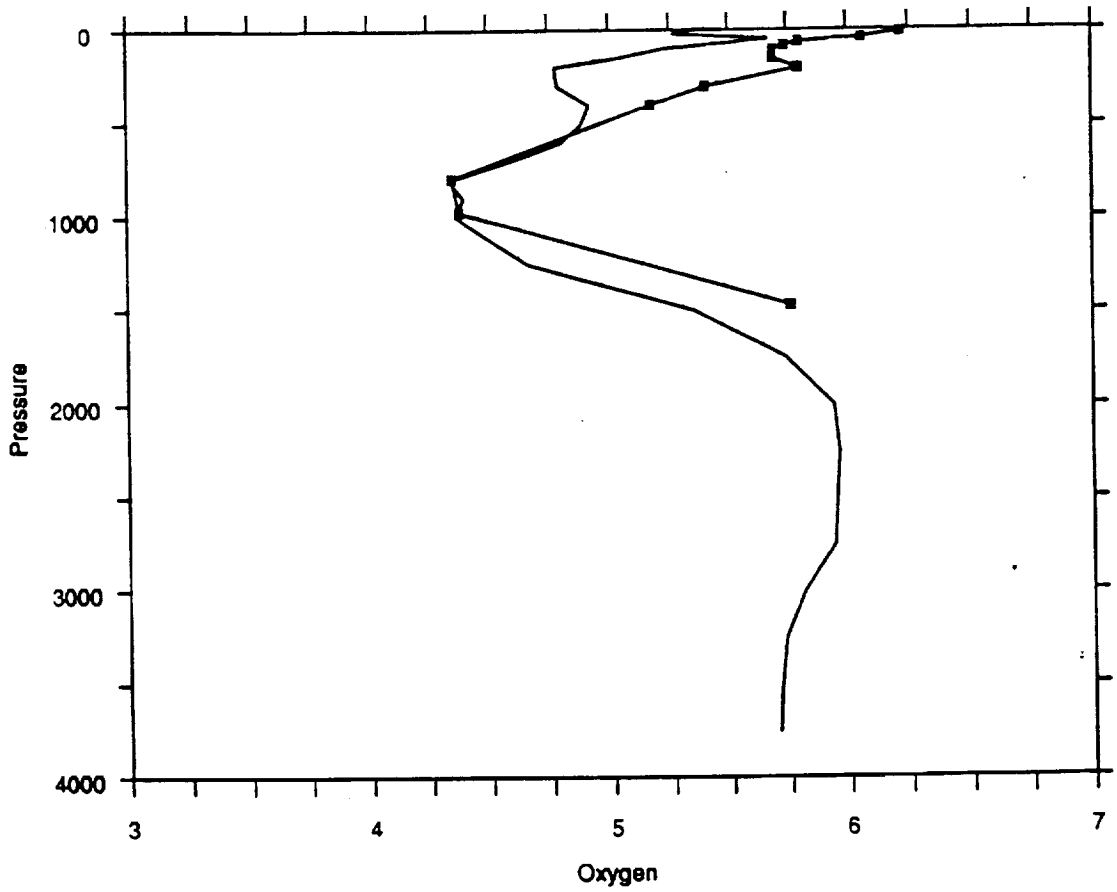

Figure 2. This figure was prepared with the following commands entered to MS-DOS (through the menu system):

window 3 4000 7 0
axis x .25 Oxygen 1 x
axis y 500 Pressure 1000 xxxx
plot ali (c:data wunsch,,station=75) o2 press*
plot v24(#bot(,station=24,cast=1) o2 press -3*

The first line sets the data units for the lower left and upper right corners of the viewport. The next two draw the axes. The third plotted the solid line from data stored in an indexed version of the Stommel and Luyten form (one integer per variable, scaled suitably). Station 75 was at latitude 36.25 and longitude -22.77 and was taken in July 1981. The last line plotted the marked points from the North Atlantic Bloom bottle data (Williams, priv. comm.) on 5/10/89 at 41.097, -23.030. The "v24" method communicated over the serial line to the server.

Figure 3

This is part of the dictionary gofs.dct for the North Atlantic Bloom study data. The data sets were compiled by George Heimerdinger for this archive. The first part represents the form filled out by the PI.

```
&form
  pi=nd
  ship=nd,cruise=nd
  stations=nd
  depthmin=nd,depthmax=nd
  latmin=nd,latmax=nd,lonmin=nd,lonmax=nd
  datemin=nd,datemax=nd
  instrument=nd
&repeat
parameter=nd,description=nd,units=nd
************************************
#tco2=18.83.0.11::/d2/guest/bloom/jgbl(/d2/guest/bloom/bre )
  pi=P.Brewer
  ship=All,cruise=119.4-119.5
  stations=17-13
  depthmin=2,depthmax=3503
  latmin=41.097,latmax=59.763,lonmin=-23.030,lonmax=-17.647
  datemin=89/04/22,datemax=89/06/06
  instrument=bottle
  parameter=press,description=nd,units=decibars
  parameter=alk,description=nd,units=uEq/kg
  parameter=tco2,description=nd,units=uMol/kg
  parameter=doc,description=nd,units=uMol/l
  parameter=doc_sd,description=nd,units=uMol/l
  ************************************
#poc=18.83.0.11::/d2/guest/bloom/jgbl(/d2/guest/bloom/duc)
  pi=H.Ducklow
  ship=All,cruise=119.4-119.5
  stations=15-13
  depthmin=2,depthmax=3468
  latmin=46.25,latmax=59.763,lonmin=-20.808,lonmax=-17.647
  datemin=89/04/22,datemax=89/06/06
  instrument=bottle
  parameter=press,description=nd,units=decibars
  parameter=poc,description=nd,units=uMol/l
  parameter=pon,description=nd,units=uMol/l
  parameter=thyincorp,description=nd,units=pMol/l/hr
  parameter=leuincorp,description=nd,units=pMol/l/hr
  parameter=bactabund,description=nd,units=cell/l
  ****************************************
#bot=18.83.0.11::/d2/guest/bloom/jgbl(/d2/guest/bloom/bot)
```

```
  pi=R.Williams
  ship=AII,cruise=119.4-119.5
  stations=17-16
  depthmin=0,depthmax=3503
  latmin=41.097,latmax=59.763,1Onmin=-23.030,1Onmax=-17.647
  datemin=89/04/22,datemax=89/06/06
  instrument=bottle
  parameter=press,description=nd,units=decibar
  parameter=depth,description=nd,units=m
  parameter=temp,description=nd,units=degC
  parameter=theta,description=nd,units=degC
  parameter=sal,description=nd,units=ppt(psu)
  parameter=o2,description=nd,units=ml/l
  parameter=o2sat,description=nd,units=ml/l
  parameter=aou,description=nd,units=percent
  parameter=no3,description=nd,units=uMol/l
  parameter=no2,description=nd,units=uMol/l
  parameter=po4,description=nd,units=uMol/l
  parameter=sio3,description=nd,units=uMol/l
  ****************************$**********
#ctd=18.83.0.11::/d2/guest/bloom/jgbl(/d2/guest/bloom/ctd)
  pi=R.Williams
  ship=AII,cruise=119.4-119.5
  stations=17-16
  depthmin=0,depthmax=3518
  latmin=41.097,latmax=59.763,lonmin=-23.030,lonmax=-17.647
  datemin=89/04/22,datemax=89/06/06
  instrument=ctd
  parameter=temp,description=nd,units=degC
  parameter=sal,description=nd,units=ppt(psu)
  parameter=o2,description=nd,units=ml/l
  parameter=theta,description=nd,units=degC
  parameter=sigmat,description=nd,units=kg/m
.  .  .
&end
#info=/d2/guest/bloom/infos(gofs.dct)
```

Figure 4

A dialog inquiring about data objects in the North Atlantic Bloom Study containing total CO2 information.

```
/d2/guesttbloom> table
Input object(with projection/selection)?
#info(*,parameter=tco2)
1 object        2 objdef        3 pi            4 ship
5 cruise        6 stations      7 depthmin      8 depthmax
9 latmin        10 latmax       11 lonmin       12 lonmax
13 datemin      14 datemax      15 instrument   16 parameter
17 units
variable number or range xx,xx? (0 to finish,-1 for list)
1,3
variable number or -anse xx,xx? (0 to finish,-1 for list)
9,12
variable number or range xx,xx? (0 to finish,-1 for list)
O
Convert to real numbers? (0=no, 1=yes)
O
Stop at beginning of group? (0=no, 1=yes)
O
object  objdef                          pi          latmin   latmax   lonmin   lonmax
#tco2   jgbl(/d2/guest/bloom/bre) P.Breyer    41.097   59.763   -23.030  -17.647
#pco2   jgbl(/d2/guest/bloom/tak) Takahashi   41.097   59.763   -23.030  -17.647
```

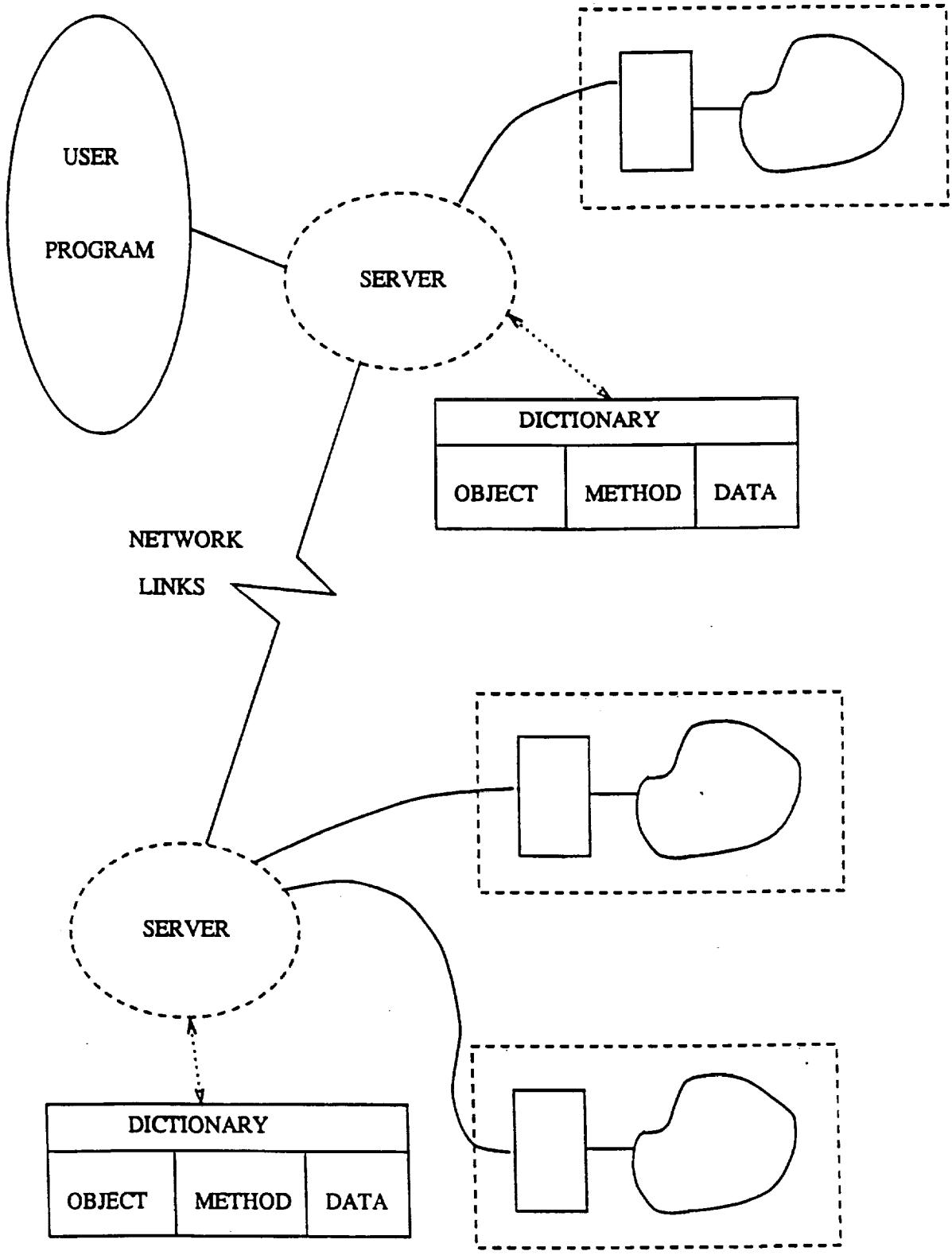

Figure 5. A server connects the user program to the proper method and data. A dictionary contains the mapping between object names, method and data. Note that one method can be used with multiple data sets. The server can talk over communications lines to servers on other machines as well.

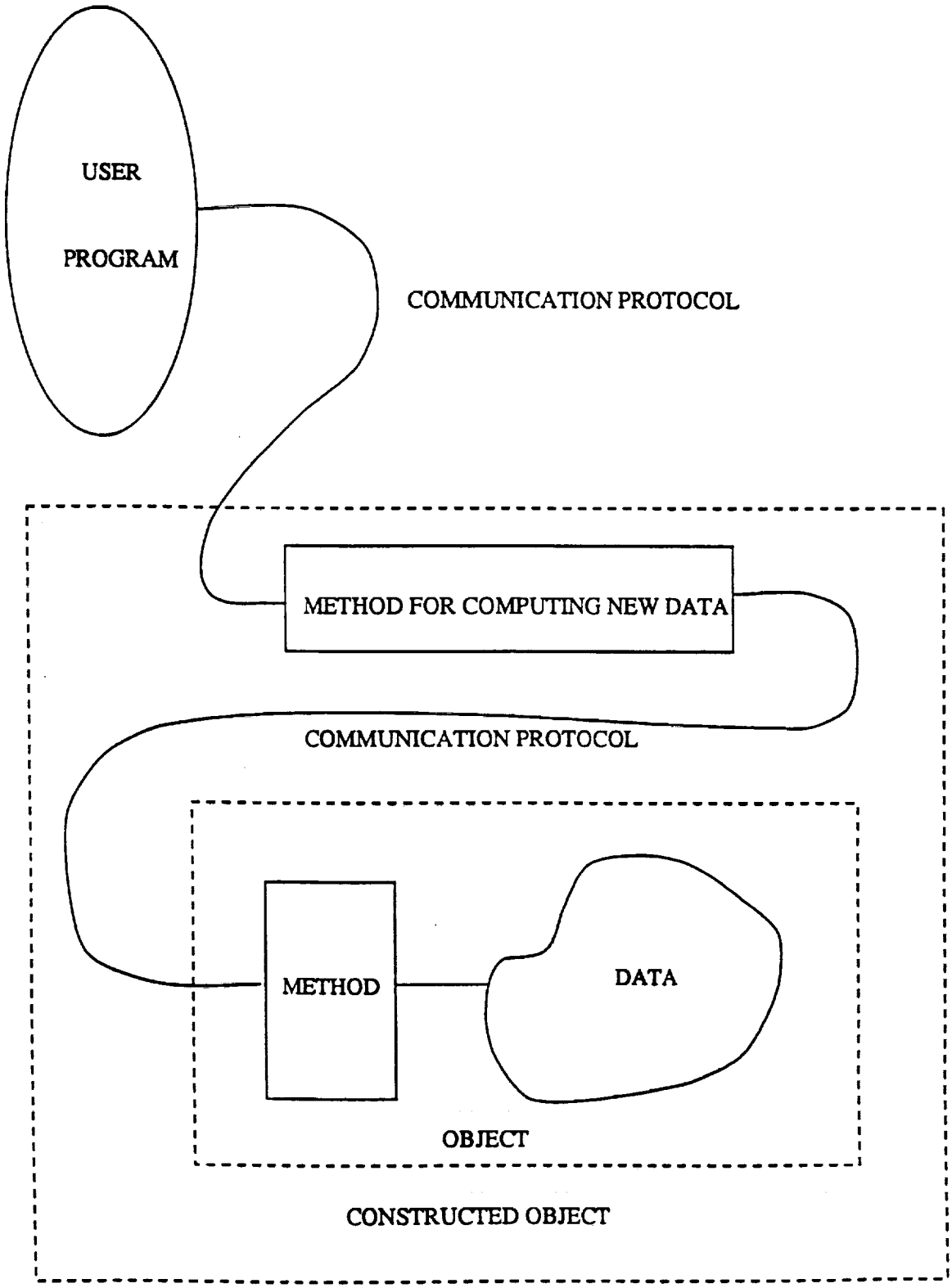

Figure 6. A constructed object consists of a method which builds a new data object from one or more data objects. Because it uses the communication protocol for both its input and output, it can be accessed by user programs just as another object, and it can work with inputs from any data objects in the system.

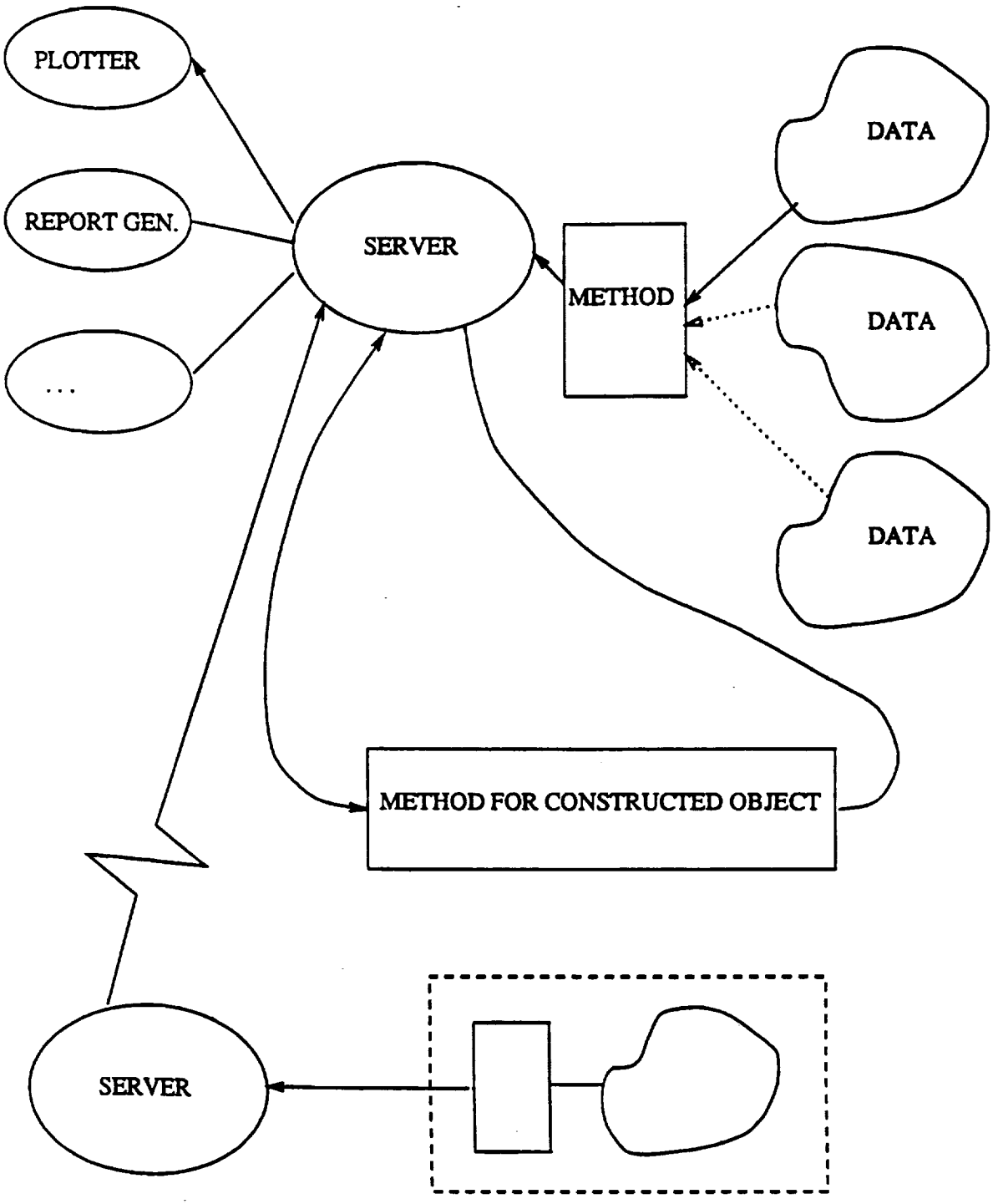

Figure 7a. Using the data system as a primary database: The upper part of the figure depicts local operations, such as reading data from various tables, merging them, and plotting the results. In this case, the user would often choose a single storage technique and use only a default method. In the middle is sketched a constructed object, used for data transformations. The lower part shows gathering related data from the network for local display.

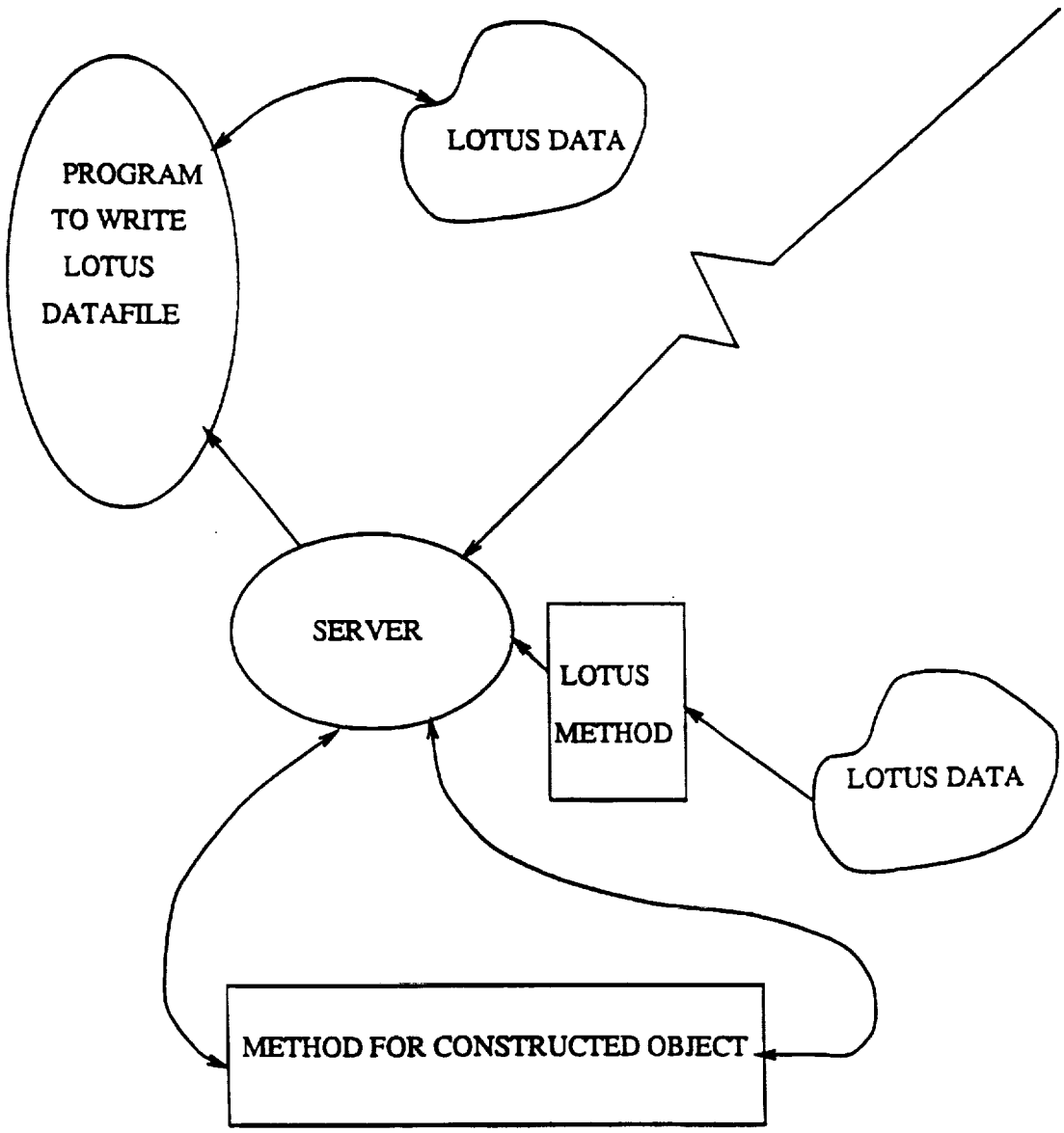

Figure 7b: Using the data system to gather data for another database (LOTUS is used as an example). In this case, the primary interaction is through a program which translates data objects into files readable by LOTUS. The program can be used over the network to import data for local use. Database operations beyond those supported by LOTUS can also be accomplished by running the local data set through a LOTUS method, then through a constructed object method, and then through the program to write a new LOTUS file. Such a procedure could be used, for example, to add a dynamic height column.

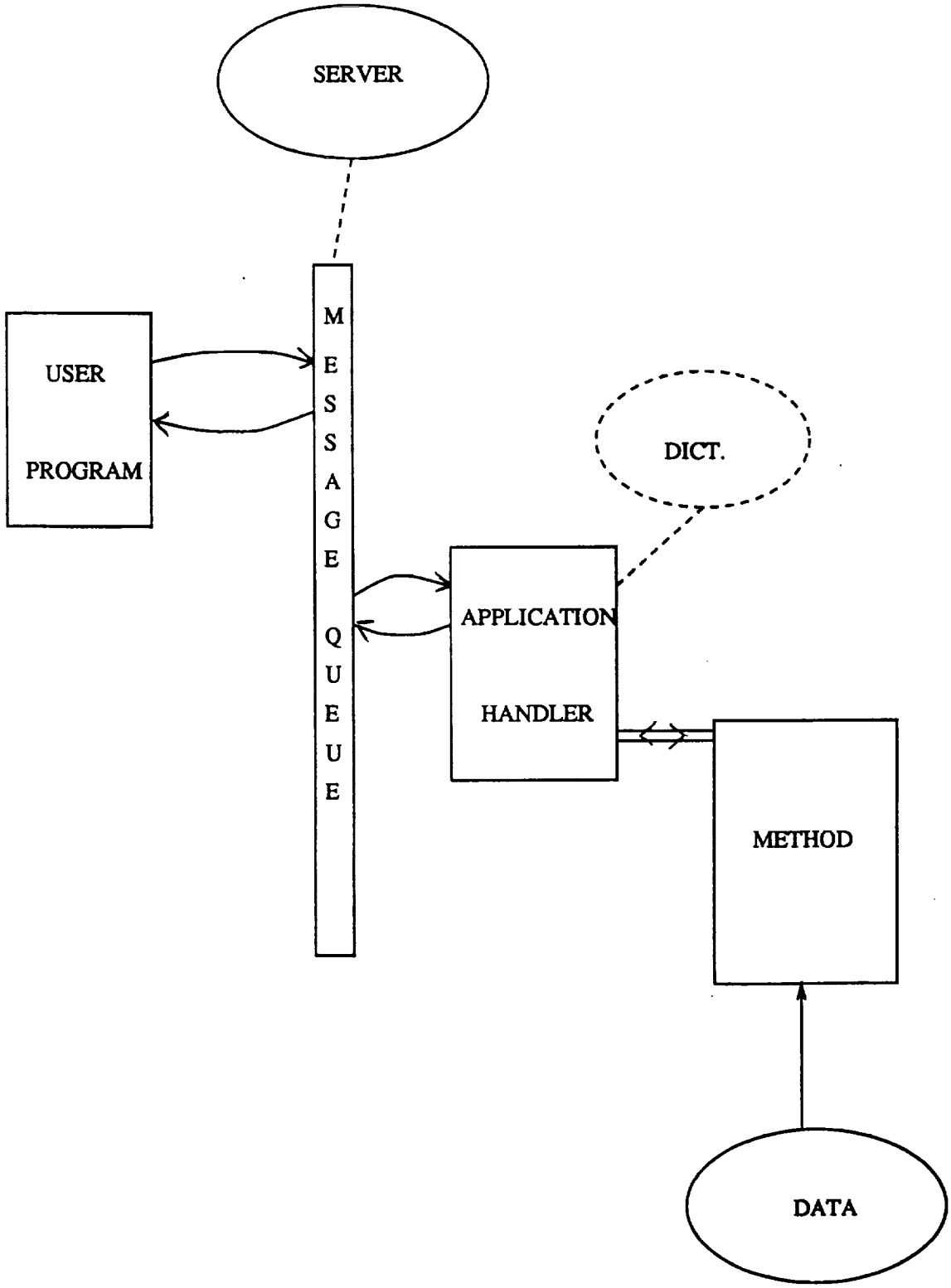

Figure 8. Flow of information in UNIX implementation for a local method. The server watches the queue and starts the application handler which looks up the object in the dictionary and starts the method. The protocol is passed from the server via a queue to the handler and then via a pipe to the method.

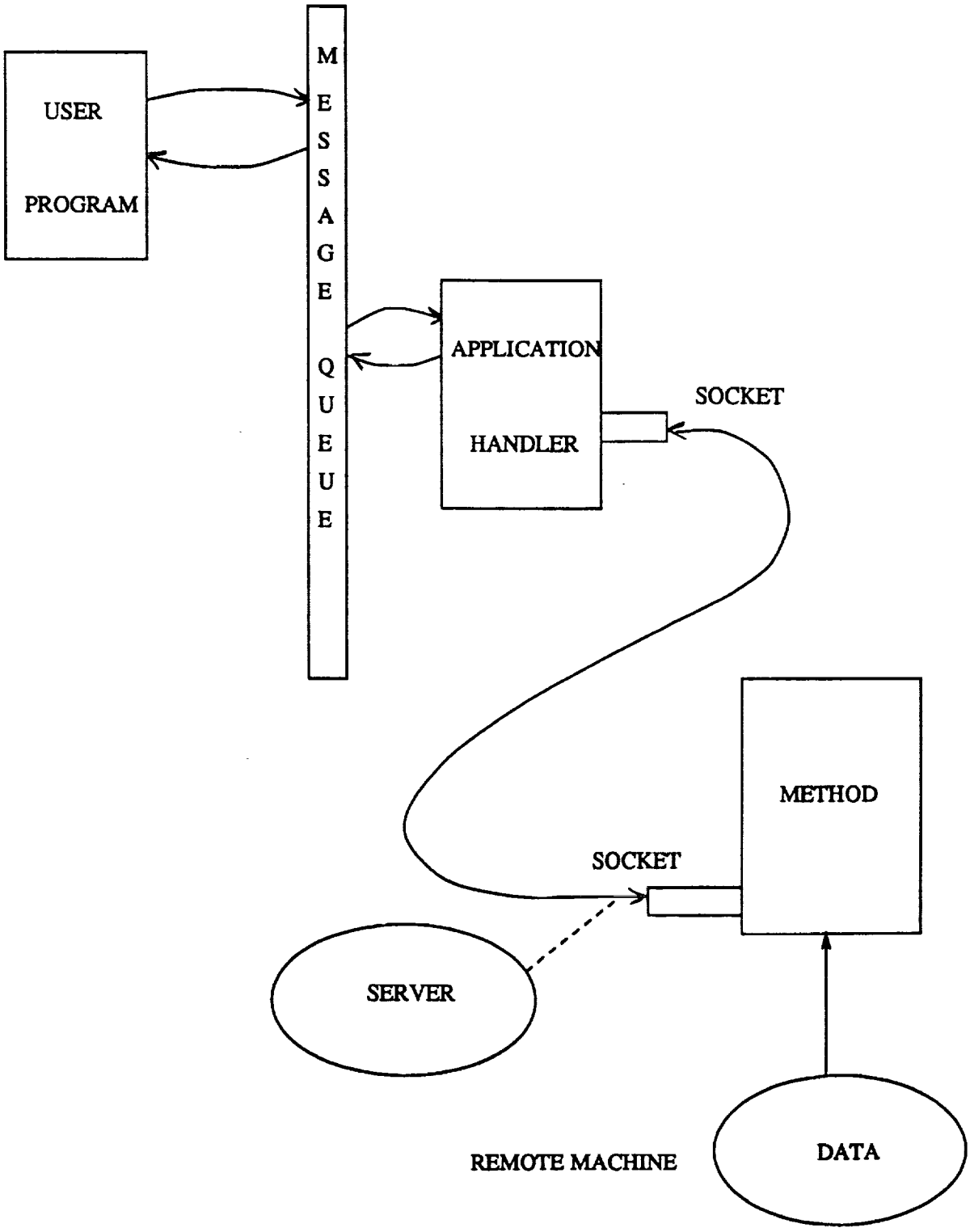

Figure 9. Flow for a remote method. The handler now communicates via a socket to the remote machine's server, which starts the method and connects it to the socket.

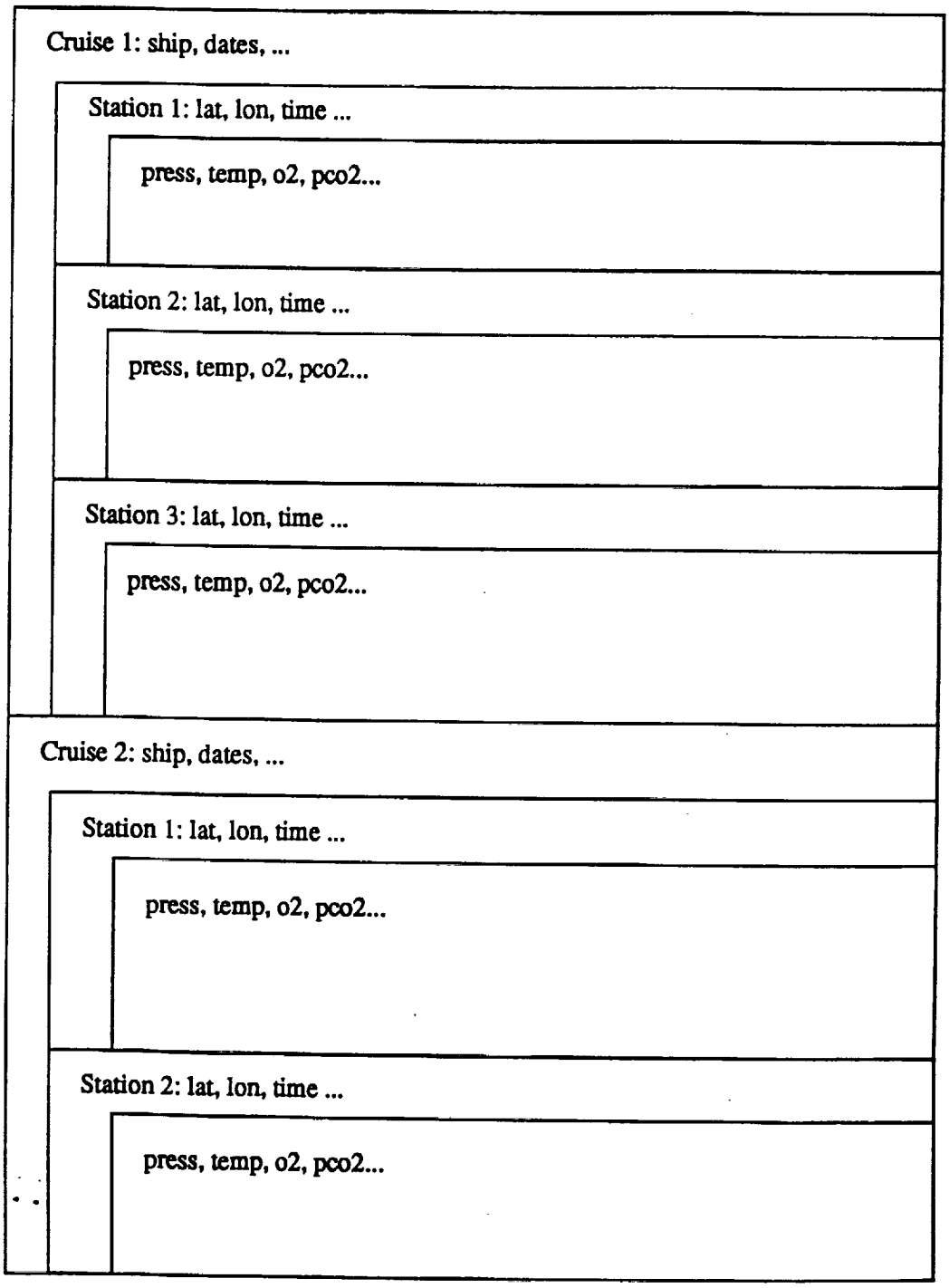

Figure 10. Hierarchical structure for a multi-cruise data object.