An object-oriented system for telemetry data management

Viral V. Tolat

Department of Electrical Engineering
Stanford University
Stanford, Ca. 94305
tolat@nova.stanford.edu

1992 International Telemetering Conference

Abstract

In this paper we describe an object-oriented software system for realtime telemetry data management and display. The system has also been designed to be used as the primary means of data management during post-mission activities.

The software system consists of three parts: the data interface library, the data format specification and the display applications. The data interface library contains a set of object definitions and procedures to provide uniform access to heterogeneous data streams. The data format specification is used by the data interface library to extract data from the raw data stream. The display applications use the data interface library to access the data and present it to the user.

Currently, the interface between the data format specification and the data interface library is implemented procedurally and is modeled after a device driver. Each format is assigned a unique id and then accessed via that id. A data stream may be accessed by any number of different format specifications. A future implementation will separate the data format specification into a separate process with a message or RPC based interface. Therefore the data may be kept on remote systems and accessed in a transparent fashion. In addition, this model will support operation in distributed heterogeneous computing environments.

This system handles multiple simultaneous data streams and applications can access data from different streams relatively transparently. This is possible since data variables (objects) to be displayed are specified by a syntax that contains the specification of both the data streams and the format to use. In addition, the concept of a primary stream is introduced to allow the user to scroll through one data stream and have the other streams follow. Synchronization between streams is based on time information in the data streams.

Several applications have been written including various stripchart displays a tabular display and some other custom displays. A data analysis application similar to the UNIX

program “awk” is currently under development. It will provide the user with the ability to extract data, i.e., report generation, for display or further analysis in an object-oriented manner.

1 Introduction

As the complexity and processing capabilities of data acquisition and control computers increases so too must the capabilities of the computers that manage and display the data acquired. In particular, the increased capability of the acquisition computers has resulted in a substantial increase in the amount of raw data and processed data that is produced. It is absolutely necessary for the computers that accept this data to convert it into useful information.

The form of this useful information is application dependent, of course. In this paper we do not attempt to discuss the issues related to what useful information is or is not, instead we present an architecture or framework for a system that can be tailored to process data into useful information. Since we don’t a priori know what processing may be required for a data set, the system has been designed to allow modules to be easily attached to handle any future requirements. Of course a primitive set of processing procedures is provided to handle a broad range of simple requirements.

There are other systems on the market currently which provide facilities to manipulate and display data. The major drawbacks of all the systems available are their monolithic designs. Such monolithic designs are important and perhaps necessary from a marketing/business point of view, however, as the dinosaurs they are, they tend to become extinct rather quickly because it is difficult to add new utility to these programs without significant effort and possible degradation in overall system performance. And unless a user is willing to pay and most likely wait a long time until the next software release, the user must make do and invest effort in building this utility themselves.

With this in mind, the major advantage of the the system we have developed, is its modular nature and its use of standard system interfaces, e.g., UNIX and X11. In addition, as we will describe below, the system is designed in a way that supports scalability and distributed operation on heterogeneous systems. The interfaces are currently procedural, however, they have been specified in a manner that is well suited to a more loosely coupled interface, for example, one that may be distributed over multiple heterogeneous computers. For example, the interface between the data interface and data format specification is based on a device driver model and therefore allows the two components to interact either procedurally, via RPC, or messages and the two components could therefore also reside on two different computers attached by a network. And this would be invisible to the application component and therefore to the user too

2 Overview

The data handling system is made up of three primary components:

- Data Interface
- Data Format Specification
- Application Programs

The Data Interface (DI) component consists of data object definitions and procedures to extract the data from an archive or realtime data stream and to manipulate the data, e.g., perform computations, engineering conversions and type conversion. The Data Format Specification (DFS) is essentially a database in which is contained a description of the data as well as descriptions of the operations to be performed on the data. Also included as part of the DFS are special procedures not available as part of the DI that are required to extract and manipulate data. The third component, the application programs, utilize the DI and DFS to access, manipulate and display the data.

The data handling system allows applications to access realtime data streams as well as archived data. The generic term "data stream" is used to denote both realtime data and archived data. A data stream is described by two or three different components: the DFS, a data description file (DDF) and a data file in the case of archived data. The DDF contains a information that describes the data, for example data rate, page size and page dimensions. The DDF has a field which indicates whether the data stream is realtime or archived data. For realtime data, the DDF also indicates the data source, e.g., network, standard input or some other device, the number of header bytes, the packet size and other information necessary to interpret a realtime data stream.

Data in archives is stored raw without any headers or additional record information. This was done in order to allow other programs to easily access the data archive files. The DDF and DFS contain the necessary information to interpret and process the data in the archive files. The association of a DFPS and DDF to a data archive file is performed at runtime. This was done in order to alleviate any management overhead necessary to statically associate a data file to a given DDF and DFS. The primary drawback and also the primary attraction of this approach is that any DFS and DDF can be used to access a data archive, correct or not. Therefore a user is not restricted to using a fixed DFS and DDF to access a data file and may interpret the data file in any way they desire.

An important feature of the data handling system is simple access to archived data and therefore the ability to "scroll" or "browse" through data. This is important both for realtime processing and static data analysis. In the realtime case it allows a user to view old data by by simply scrolling to it in a display application. The application just needs to make a request to the DI for data for a specific time and the DI performs the work to

retrieve the data. For static data analysis, the ability to browse a data stream is essential for examining data quickly and easily in an interactive manner without having to generate reports.

3 Data Interface

The functionality provided by the DI can be divided into five categories:

- Data acquisition.
- Data stream access and management.
- Data retrieval and manipulation
- Multiple stream coordination.
- Configuration management.
- Browsing.

3.1 Data Acquisition

A program named acquire is used to read realtime data and place it in an archive and also shared memory. The shared memory is used as a cache in order to optimize access to recently received data. Figure 1 illustrates the organization and flow of data. When a data stream is accessed, the DDF is accessed to determine if the data is archived or realtime. If the data is realtime, the DI determines if an acquire program is running for that data stream. An acquire program is started to obtain the data if one is not already running. If the data stream is static the specified data file is opened for read access and an acquire program is not started.

Because the format a realtime data stream may take is not known, a filter can be specified to interpret the realtime data stream and process it into a form suitable for the data handling system, e.g., header stripping, packet sequencing, etc. Such a filter can be specified in the DDF and used by the acquire program to process the input data.

The archive file accessed by the acquire program may be specified as read-only. This allows multiple machines to acquire a single realtime data stream and have only one machine with a large local disk actually store the data to disk (figure 2). The other machines access the archive file at startup and after a data page is received to determine the current size of the data stream. The archive file is accessible by the DI procedures onall machines. Since the DI is decoupled from the acquire process, it is not necessary for the DI to know which acquire process on which machine is actually writing the data to disk.

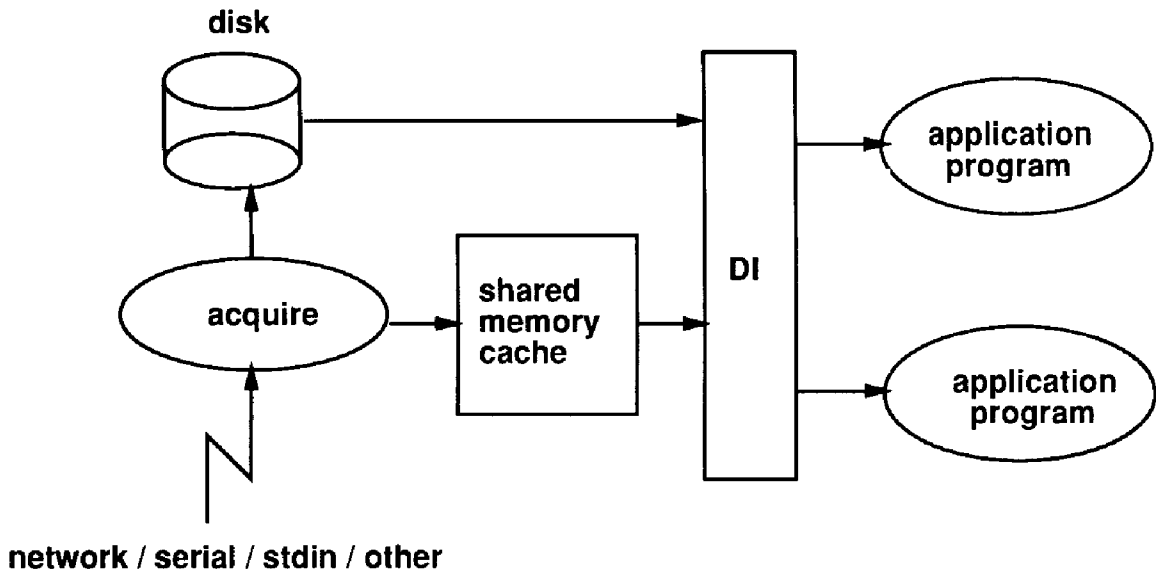

Figure 1: Acquire to DI to Application data flow

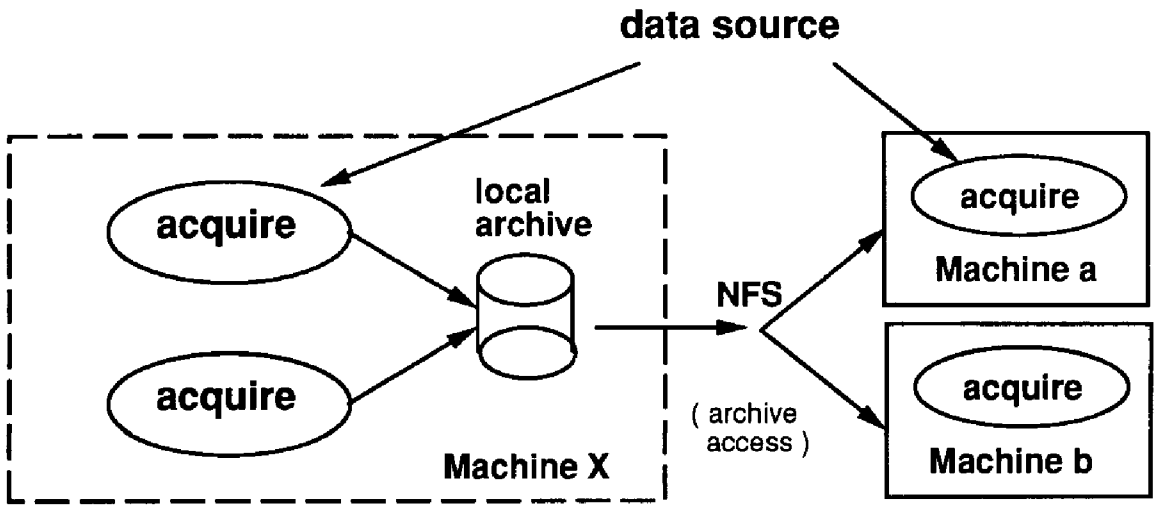

Figure 2: Multiple host access to realtime archive file

A useful feature of the acquire program is its ability to accept data from standard input (figure 3). In particular this allows an application program to be written that processes data from a realtime stream and constructs a second realtime stream that can be input to a second acquire process. This second realtime stream could be prediction of future data, for example. A display application then has access to both data streams through the DI and can simultaneously display data from both streams thus displaying actual and predicted data in the same display.

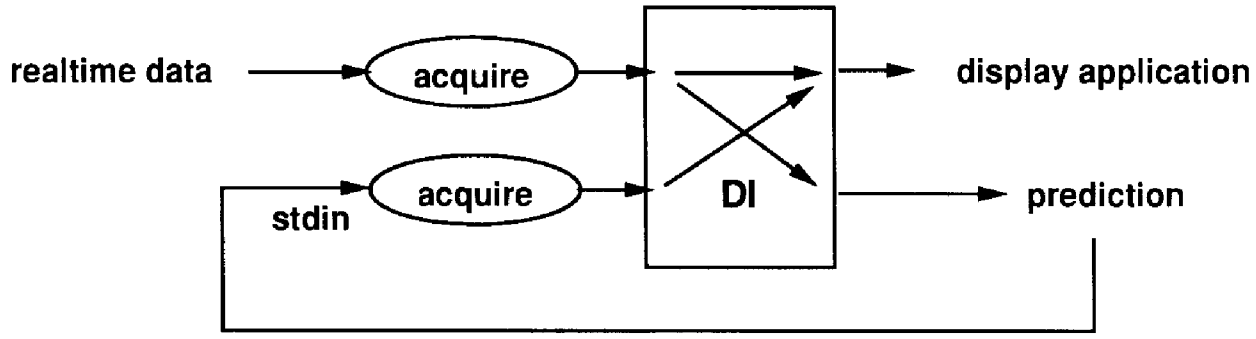

Figure 3: Using acquire with standard input (stdin)

3.2 Data stream access and management

Access to data streams is provided by the DI in two ways. For realtime data, the DI coordinates with a data acquisition program to retrieve data from a shared memory cache or an archive file. For static data streams, the DI manages access to the data file. To an application the only difference between access to a realtime stream or static stream is that the total size of a realtime stream is constantly increasing as new data arrives. The DI provides a callback facility to notify applications upon arrival of new data for realtime streams. This facility provides the necessary information to be used with callback procedures in X11 so there is no conflict between processing user input and processing the arrival of new data.

The DI caches data to optimize access to the stream by the applications, for example, the last data page accessed is always cached in local memory since many applications access the same data page many times before accessing another data page. We are considering adding a caching object to allow applications to specify caches in order to optimize data access. Procedures to optimize updating and loading of the cache would be provided to assist applications.

Given the ability to access static data, figure 4 illustrates how model data and realtime data can be viewed simultaneously. This allows the comparison of actual results against data produced by a model. This configuration is necessary if the model data cannot be generated in realtime.

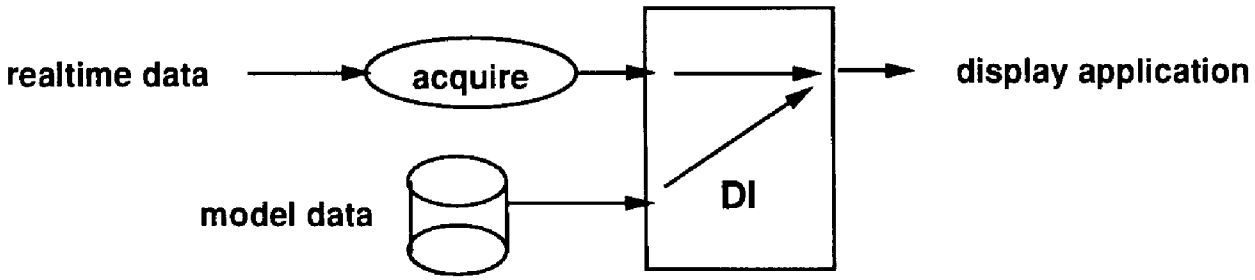

Figure 4: Using acquire with realtime data and static model data

3.3 Data retrieval and manipulation

The DI provides applications with an object oriented interface to data in a stream. To access a data object, an application must first specify the object to be accessed. A data object is specified by a name and a corresponding data stream. A nomenclature has been developed to specify a stream and to specify data objects. The notation is

```
[DATAFILE*]DDF*DFS*OBJ
```

where DATAFILE is the name of an archived data file, DDF is the name of a data description file, DFS is the name of a file that contains the data object definitions and OBJ is a nomenclature which describes the data object to retrieve from the data stream. In order to simplify the specification of streams, the DI supports the definition of logical streams. A logical stream is specified by a string name and is associated to a DFS, a DDF, and possibly a data archive file if the stream is static. Data objects can be accessed using logical streams with the notation

```
$STREAM*OBJ
```

Logical streams are defined in a configuration file which is discussed later. Default values may be defined for the DFS, DDF and DATAFILE, therefore allowing data objects to be referenced without using the entire nomenclature when the defaults are desired. The nomenclature allows the specification of operations to be performed on data retrieved from a stream. The format is

```
{(-%)}VAR{[N{+}]}
{%}VAR
```

VAR is the name of the data object. The symbols enclosed in curly braces are optional and are used to specify operations on the raw data. A % symbol indicates that the data should be "scaled" as specified by the DFS. A - symbol indicates that the value returned should be the difference of the current value and the previous value. The previous value is cached by the DI in the data object.

The square braces on the end are used to denote subsampling or averaging. The number N denotes the number of values to be returned. The + symbol indicates that the data should be subsampled by averaging. The notation %VAR is shorthand for (%)VAR.

If the name of the object is found in the DFS, the data object is constructed using information from the DFS and the based on the operations specified in the nomenclature. The returned data object contains an internal reference to a specific data stream. Actual data is then retrieved from a stream by referencing the data object and specifying a time or page for which a value is desired (figure 5).

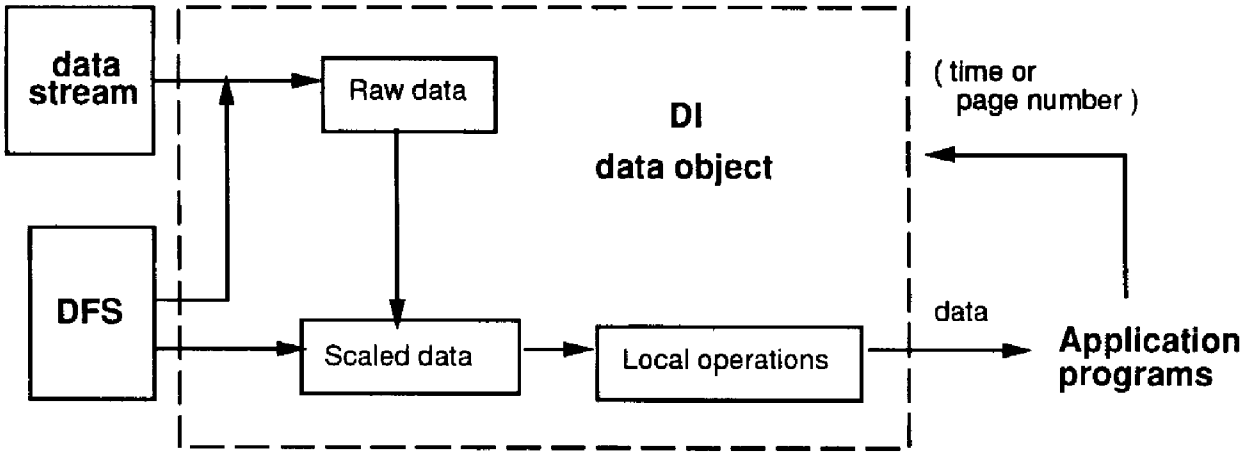

Figure 5: Data retrieval flow

The DI provides a broad range of extraction procedures to retrieve data from a data stream. These extraction procedures include those to handle standard data types such as bit fields, bytes, shorts, longs, strings and BCD. In addition, extraction procedures are provided to handle arrays of the above data types. Extraction information is specified in the DFS for a data object. For special data types, the extractions procedures must be provided in the DFS.

A number of scaling procedures are also provided by the DI. These include procedures to perform polynomial scaling using gains, use of lookup tables and the ability to specify an arbitrary computation on a value. The computation can use values from the other data objects and supports the use of constants, trig functions, logarithms, exponentials, bitwise operations (AND,OR,XOR,NOT) and logical operations (EQ,NE,GT,....). The computation is specified as part of the DFS. Special scaling procedures can be provided by the DFS.

Procedures ire provided in the DI to search an archive or realtime stream for data at a given time. When the application requests data for an object at a specified time, the DI performs an intelligent search of the data archive for the desired time. First, the most recent data retrieved is checked. Next, a calculation is made to determine where the desired data may be located in the archive and that data is examined. If no match is found the search is initiated again from the beginning and/or end of the archive and a calculation is made once more. This method is applied since data may not be time sequential in an archive. If no match is found, then finally an exhaustive binary search is attempted.

The DI supports subcommutated data, i.e., data that does not always appear in a data page or record. If the actual data for a requested time or page is not present in the data, a flag in the data object which is accessible by an application is not set to denote that data for the last request was not found. In this case, the data object retains the data from the last request. The subcomm information is stored in the variable data area of the DFS.

3.4 Multiple stream coordination

Data in a stream can be accessed either by a page number or by time. Synchronization between streams is performed via time. The DI supports the concept of a primary stream which is used to coordinate access to the other streams. This is primarily a concept for the user and application to manage accessing multiple streams. The user sets the primary stream and then views data from all streams based on the current time of the primary stream.

When the primary stream is realtime, updates are made to a display as new data becomes available. The application sets the current display time to the time of the most current data for the primary stream. The data for all streams for that time is then retrieved by the application. This allows the other streams to "follow" the primary stream.

When the primary stream is a static file, the user or program determines the times to retrieve. If other streams are being viewed and they are realtime, displays are updated if necessary as new data arrives.

3.5 Browsing

There are several modes of browsing supported by the DI:

- Local browsing
- Realtime browsing
- Global browsing

Local browsing refers to a mode in which an application is controlled independently of other applications and only input from the user in the case of interactive applications. Local browsing is supported by the standard data access mechanisms described above. Realtime browsing is a mode in which applications change their display window to display the most current data available for the primary stream. These displays are updated automatically as new realtime data arrives

Global browsing is similar to local browsing except that a single application can control multiple other applications. The DI provides support for coordinating browsing between separate application programs through a common file. Since stream synchronization is performed via time, only the current browse time needs to be communicated to all applications being used for global browsing. Any application can be used to control global browsing or a special browsing application may be used. The DI provides procedures for applications to access and set global browse information. The global browsing is implemented in such a way that many users may browse on a single system without

interfering with each other. The browse information is stored in a browse file the name of which is set as a configuration parameter.

3.6 Configuration management

The DI supports the use of a configuration file which can be tailored by each user. The information defined in the configuration file includes the definition of logical streams, the names of temporary files, the browse file name, the time of MET zero and the default and primary stream names. Many of the options specified in the configuration file can be overidden by command line arguments. The DI provides a procedure to parse the command line and process arguments to set the default DFS and data file, the primary stream and the configuration file to use. By default the DI uses an environment variable or the file `.gserc` in the user's home directory. When an application receives a HANGUP signal, the DI automatically reloads the configuration file.

4 Data format specification

There are two distinct parts to the DFS. The first is a table or database of data object definitions called an mtable. The second is a set of procedures that implements stream specific operations for data extraction, type conversion, etc. The DFS mtable and procedure definitions are linked by an id that is stored in the mtable. This id is used by the DI to access the DFS procedure definitions.

The DFS mtable essentially defines the format of the data in data stream (realtime or archive). The structure of the DFS has evolved from the NASA M-Table format used to describe telemetry. For a given data stream, the DFS contains the logical page size and page dimensions, the data rate and definitions for data objects in the stream.

The following fields are contained in the DFS for each data object:

- varcode: indicates format of data in stream, e.g., byte, short, byte order, signed, etc.
- scalecode: indicates how to convert raw data value into an engineering value or other useful form, e.g., string identifier.
- vartype: indicates the data type, e.g., integer, float, string, etc.
- varsize: size of data items, e.g., string size or array size.
- frequency: frequency of the data with respect to the page rate.
- nitems: number of data items for a data object.
- offset: byte offset of data in stream.

- scale limits: high and low scale information for displays.

- yellow and red limits: warning and critical limits for data values.

- gains (a0-a5) : used for engineering conversion (scaling).

In addition to the fields above, the mtable also contains a variable data section in which special information for each variable can be stored:

- Scaling information for engineering conversions. This is usually information that is more complex than gains. For example, if the gain for a data variable is dependent on the value of another data variable, the various sets of gains and the name of the dependent data variable is stored.

- Lookup tables. Given a raw data value, a lookup table can be accessed to return a string or numerical value.

- Information for subcommutation of data. If a data value only appears in at certain times dictated by some other value in the data, e.g., major frame number or dump address. If the data for an object is not available, the DI returns the last retrieved value and an indication that no data was found. This allows applications to selectively ignore the condition or respond to it.

- Information for constructing a data object from other data objects, e.g. concatenation of bytes from other data objects.

- Formulas to be used in special computations. The formula specification can consist of constants as well as other data objects and supports all arithmetic and trig functions. In addition the formula specification supports logical comparisons and bitwise operations. The formula is stored in RPN as a data structure.

Multiple levels of scaling conversion can be handled either with formulas or by creating a new data variable whose raw value is the scaled value of a data variable. This allows scalings to be cascaded ad infinitum.

The variable data section of the mtable is constructed using procedures to build a database. Entries into the database are specified as arguments to a procedure. This approach is useful for building variable sized objects and for hiding the structure of the data from the user and application. The data object entries in the mtable are specified in an include file that is compiled into a program, which when executed, writes the data objects to a file. This program also executes the variable data procedures which in turn write their information to the file too.

In each DFS, a data variable named `TIME` must be defined. It is used to determine the time of a given data page. The data variable `DATACHK` can be defined too. The quality of a page is taken to be good if the value returned for this data variable is non-zero. It is used to determine the quality of the data page. Its definition is optional.

The DFS procedure definitions are used by the DI and DFS mtable to operate on data objects, either to extract data from a stream or to operate on already extracted data. For example, if data in the stream has been compressed, the decompression algorithm would be contained in the DFS procedure definitions. The DFS procedures are kept separately from the DFS mtable since there is no standard method for storing code segments with data in UNIX. The DFS procedures are accessed by the DI through a device-driver type of interface. A library of DFS procedures must statically be assigned a unique id. This id is stored in the mtable file. The id is used by the DI to access a DFS procedure table which contains the address of a dispatch procedure for the specific DFS. When a data object is accessed that has an operation code (varcode or scalecode) not supported by the DI, the DFS library identified by the id is accessed to support the operation code.

Currently, the DFS libraries are optionally linked into applications that may require them. Stub procedures are provided for all possible ids so that if procedures are not provided for a given id, a stub routine that returns an error is used. At link time, the specification of a DFS library overrides the stub procedure. This model is easily extendable to a system in which the DFS procedures are accessed as services via remote procedure calls (RPC) or some other interprocess communication mechanism. This would allow the DFS service to reside on other machines.

5 Applications

Several applications have already been written as part of the data handling system. The kinds of applications can be divided into three classes: support, display and data processing.

Support applications includes a transmit program for simulating realtime data on the network and a program for dumping a DFS mtable in a readable form.

Display applications currently implemented include two stripchart programs, a tabular display program with limit checking, a textual stripchart program (striptab), and a hex display program. Future applications include a spectral display. One stripchart program displays up to five data variables in a single plot window and the other displays multiple data variables each in their own plot window. The first stripchart program is very useful for comparing different but similar data variables. All the programs are designed to display both static and realtime data and they all support local and global browsing. This makes these applications especially useful for non-realtime (post-mission) analysis of data.

Currently under development is an “awk” like program. The awk program in UNIX is a very effective program for processing text files. It uses a “C” like notation to specify

actions to taken after reading each line of text from a file. Our awk program is similar, however, it processes data records from a stream. Our awk also accepts “C” syntax which is then converted into actual C and then compiled and linked to the DI. We expect this awk to be the primary data processing program used. Our awk can be used to extract data for other applications such as MATLAB or it can be used to process the data itself. Since it supports full C functionality, any arbitrary program can be used.

The striptab program is a simple report generator, however, it is very useful for extracting displaying telemetry values in an ascii terminal. In addition, the output of striptab can be piped into other programs for further processing, e.g., we pipe the output of striptab into a plot generation program to generate hardcopy plots of data.