GENERIC TELEMETRY PROCESSING
Theory vs. Application

Richard L. Pettit, Jr.
Telos, Inc.
Jet Propulsion Laboratory/California Institute of Technology

ABSTRACT

The Space Flight Operations Center (SFOC) is a generic suite of ground data systems software. One main subsystem of SFOC is the Telemetry Input Subsystem (TIS). Utilizing techniques for the abstract representation of data, the TIS has provided a flexible software base that can be used as a baseline for multiple spacecraft missions.

INTRODUCTION

Throughout the implementation of multiple spacecraft missions that NASA has been responsible for, there have been many rewrites of ground data systems software. This created multiple software implementations, each one different, that must be written and maintained. The purpose of SFOC is to provide a generic suite of software that multiple *instances* could be developed from. This should not only need less support, but should also prevent the need for a complete rewrite of the ground data systems software for every mission.

The areas of TIS that will be treated separately in this paper are *Data Acquisition, Frame Sync*, *Extraction*, *Decommutation, and Data Distribution*. Each of these areas will be reviewed for it's merits in generality and it's ties into specific missions. The purpose of this paper is not to present a telemetry processing system in terms of it's capabilities as a telemetry system per se, but to analyze the implementation of a generic software suite.

Figure 1 shows the flow of data through the TIS in the Magellan High Rate (MHR) configuration. Modules labeled *MHR* are the mission specific sections of the system.

DESIGN CONSIDERATIONS

The design goal of the TIS was to localize mission specific code. The goal would be met by making a generic representation for data and providing means for converting this representation into code. Equally important would be the ability to represent *actions* in

generic terms and translating that into executable code. Unfortunately, only the representation goal has been implemented. The intended goal in this area was the ability to translate a more general language into a more specific one for compilation.

The first step in making the TIS generic was the use of the Standard Formatted Data Unit (SFDU). This standard definition provides for record headers that define the telemetry data being processed. Since the processing of this data is based on the information held inside these records, it is crucial that the definition remain fixed through multiple missions. If this were not the case, the need for rewrite would be more prevalent in the multi-mission scenario.

The use of tables is a widely used method within the TIS for the development of code that has mission specific features embedded into it. These tables manifest themselves both as pseudo languages that need compilation, and data files that are read in at run time. Utilities ranged from as simplistic as routing tables to as complex as programming languages using this method. By using tables instead of writing code, the task of implementing multiple missions not only becomes faster, but less tedious as well.

Using this building block method has lead to the development of tools for translating generic text input files into code for representation of data. The *SFDU Guru* is the most widely used tool and is described in the following section. Facilities that provide means for defining the routing of telemetry through the TIS have also been developed using techniques of translation from text tables to the implementation programming language. As more experience is gained in this area, more tools may be developed that generalize further sections of the TIS. Later implementations will be able to provide a visual interface that translates graphical representations into the input files of these tools.

THE SFDU GURU

This library was appropriately named the SFDU guru, since it contained knowledge of the structure and format of the SFDU headers. The SFDU guru provides a library of functions to access and modify fields in the SFDU. Part of the guru is a tool to create the library from a text representation of the SFDU structure. An example of this file is presented in the next section. By using this input text file, the need for rewriting the library was eliminated and alteration of the SFDU header structure became a simple task.

EXAMPLE OF THE SFDU GURU INPUT FORMAT FILES

```
STRUCTURE               # Structure definitions
sync-flags          1 b
```

This section of the input file describes the structures that will be generated. The three respective fields in the above line are the structure name, the size in bytes, and the conversion indication. In this example, the conversion specifies that a bit field structure should be generated.

```
sync_flags          # Field definitions
1 phase
1 soft_sync
1 scid_force
1 sclk_ref
1 sclk_cor
3
```

These are the bit fields that correspond to the structure in the first section. The conversion indication of *b* in the record definition causes the guru generator to check for bit field definitions in this input file. The *3* on the last line by itself signifies that the last three bits are not used.

```
struct sync_flags
    {
    unsigned phase: 1;
    unsigned soft_sync: 1;
    unsigned scid_force: 1;
    unsigned sclk_ref: 1;
    unsigned sclk_cor: 1;
    unsigned : 3;
    };
```

The resulting C structure reflects the definitions contained in the two previous sections.

DATA ACQUISITION

The function of data acquisition is to provide input to the TIS from one of several possible input sources. These sources are virtual circuits and disk files. After the data has been read, it is packaged into an internally defined structure for movement through the sections of the TIS.

The development of data acquisition required little specification of a generic base. Since the data sources are fixed and known beforehand, these sources can be hard-coded into the system without the worry of adding physical inputs sources. The possibility exists that this section of the TIS could read various byte stream files or a virtual circuit identified by a

unique name. These can be changed by entering a different name via the Control Directive Processor (CDP). CDP parses input commands and stores information regarding the state of TIS in a shared memory structure.

An important attribute of this section of the TIS is the use of the SFDU guru. This allows the abstraction of header data so that a change may be integrated by relinking the data acquisition module. This is also true for other sections of the TIS.

Although it is still a goal to independently represent algorithms as well as data, this section of the TIS has met it's design goals insofar as little change would need to be performed to create an instance for a new mission.

FRAME SYNC

Frame synchronization operates by searching the incoming telemetry data for a bit pattern that identifies the beginning of a frame. This bit pattern is called the pseudo noise code (PN code). Once the PN code has been found, the format identification (FID) is located at a fixed offset from that point. The FID identifies the frame type. Frame sync is easily generalized since the PN code, FID, and offsets are values that can be put into structures that are modified on a per mission basis. These values are currently kept in a header file which is included during compilation. Eventually they could be placed in a text file that is read in at run time.

The portion of frame sync that is not easily modified is the processing of anomalies, error reporting, and data correction. This problem is insidious throughout the TIS and steps are being taken to reduce the amount of alteration that would be required to effect such a change when new missions are developed.

EXTRACTION

The extraction section of the TIS requires great knowledge of the data portions of the incoming telemetry. Of all the sections of the TIS, this one is the least general. This is because of the mission specific nature of the data that extraction is responsible for processing.

A prototype version of extraction began with the development of a programming language that provided for the definition of data at a low level. Although C provides definition of bit fields, the size of these bit fields is restricted to the size of the processor register; in this case, 32 bits. C also lacks a mechanism to provide named access to data fields whose size is not an even multiple of 8 bits. The decision to attempt the use of a general purpose bit language, was to determine if it was possible to remove some of the complexity from

writing such code in the C language. Although the prototype implementation succeeded in achieving that goal, it was decided that writing extraction in C would make it easier to maintain.

Phasing through a prototype of extraction allowed for a review of decisions made about software design along the way. It became clear that the mission specific nature of this section of the TIS made it difficult to develop a mechanism that allowed for simplistic representation of incoming data and the form, that it must take as an output product.

DECOMMUTATION

Most of the processing of the telemetry data in the TIS up to this point has been involved with alignment, correction, and reorganization. Since the data was now ready for decommutation, the TIS had the potential for creating an elegant mechanism for data generalization, and succeeded in doing so.

The main tool in the decom process is the decom map definition language. This language provides for definitions of channels in terms of their channel name, width, and offset. Contiguous channels are organized in data hierarchies called blocks. Recognizing different types of frames is done by analyzing data that is copied into variables. This analysis is performed using a switch statement much like the one used in the C programming language. Code segments inside the case portion of the switch may be definitions of channels or calls to other blocks that may define channels or switch on even more variables. This flexibility of data representation allows for a user tailorable polymorphism. These map sources are compiled into C code and the resulting object is linked with a run time library to produce an executable decom map.

Decom is not without mission specific code however. Data placed inside output channels will most likely change from mission to mission, but this would only require a change to the run time support libraries.

EXAMPLE OF A DECOM MAP SOURCE FILE

This is an input source code file that is compiled by the decom map compiler. The map compiler generates C code that is compiled by the C compiler and linked with a run time support library resulting in an executable decom map. Note the use of *figurative constants* to define the offset resulting from the existence of a header of a specific type. A large portion of this particular map was deleted for sake of brevity.

```
BLOCK MAIN
   DEFINE WIDTH = 8                        # Defines the width of channels
   DEFINE OFFSET = SECONDARY_HDR           # Defines an offset to skip

   skip 2*8                                # Skip two bytes
   H-0002                                  # These are header channels
   H-0001
   H-0023(1)                               # Default width can be overridden

   CALL HLM_1A                             # This block is defined below

   RMF_REF ( 3 )                           # RMF_REF is a variable
   SWITCH RMF_REF % 2                      # Switch on the variable
     CASE 0
        skip 8
        CALL RADAR
     ENDCASE
     CASE 1
        skip 120
     ENDCASE
   ENDSWITCH
ENDBLOCK

BLOCK HLM_1A
   # definition of the map continues just as above
   .
   .
   .
ENDBLOCK
```

DATA DISTRIBUTION

This section of the TIS is responsible for routing data to multiple receivers. These receivers are other SFOC subsystems that process the output data of the TIS. Data distribution is implemented as many tasks, each one providing the TIS data to a separate receiver. The reason for this design is to prevent the entire TIS from blocking should just one output channel block for some reason. It would be unreasonable for all processing to stop because one output channel blocked on a write.

Data distribution resembles data acquisition insofar as the output paths are known entities that are not likely to change in form or function in the foreseeable future. Therefore, it is reasonable to assume that this function is written in as general a way as possible.

The individual processes in the data distribution section of the TIS are called *metafile servers*. A metafile is a handle to an I/O facility that serves various sources and destinations. With a call to the metafile open module, an endpoint to an I/O facility can be provided regardless of its type. Currently supported endpoints for I/O redirection are plain byte files, spooler files, and broadcast circuits. Each of the metafile servers provides output to a unique endpoint. There may be more that one server to provide output to broadcast endpoints, for instance, if they are unique endpoints. The “parent” process controls routing of data to the appropriate endpoints and is responsible for spawning servers as the need becomes evident.

TELEMETRY ROUTER

Different types of telemetry data that pass through the TIS have different paths. For instance, not all telemetry data needs to pass through the extraction phase. To make the definition of routing general, a routing table was created that defines what data goes where and in what sequence. This table is processed by the router generator and the result is a C structure defining the paths through the TIS that the data must take. This type of abstraction differs from the previous examples. Instead of defining data, *data flow* is defined. Although this is not a concrete process abstraction, it is an example of the type of higher level process abstraction that will make the TIS easier to define. A portion of the routing table for the current implementation of the TIS follows.

```
# route.tab - router definition table
RED         DIST EXTR
SED         DIST DECOM EXTR
RCD         DIST EXTR
MRO         DIST
TD_INIT     DIST FSYNC EXTR DECOM QQC-SUM
```

CONCLUSIONS

The TIS has made extensive use of data abstraction. The further generalization of the TIS could be done by process abstraction. Work being done in this area includes visual programming languages and modular Computer Aided Software Engineering (CASE). These tools allow for the abstraction of process and control in terms of computer graphics and produce source code as a product. It is this type of high level development that could

lead to a generic system that is completely configurable using computer graphics. To make that transition, the first step must be taken. SFOC is providing crucial momentum.

ACKNOWLEDGEMENTS

I would like to extend my fondest gratitude to these people for sacrificing their time to enlighten me further about the quest for TIS and for broadening my outlook with their indomitable insistence that it could be done.

John Diehl
Al Johnson
Teih Ku
Jack Morrison
Robin O'Brien
Norm Pevyhouse
Deborah Shaft
Warren Taylor
Chingyow Wang
Bob Wendlandt
Betsy Wilson
Jim Wilson

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

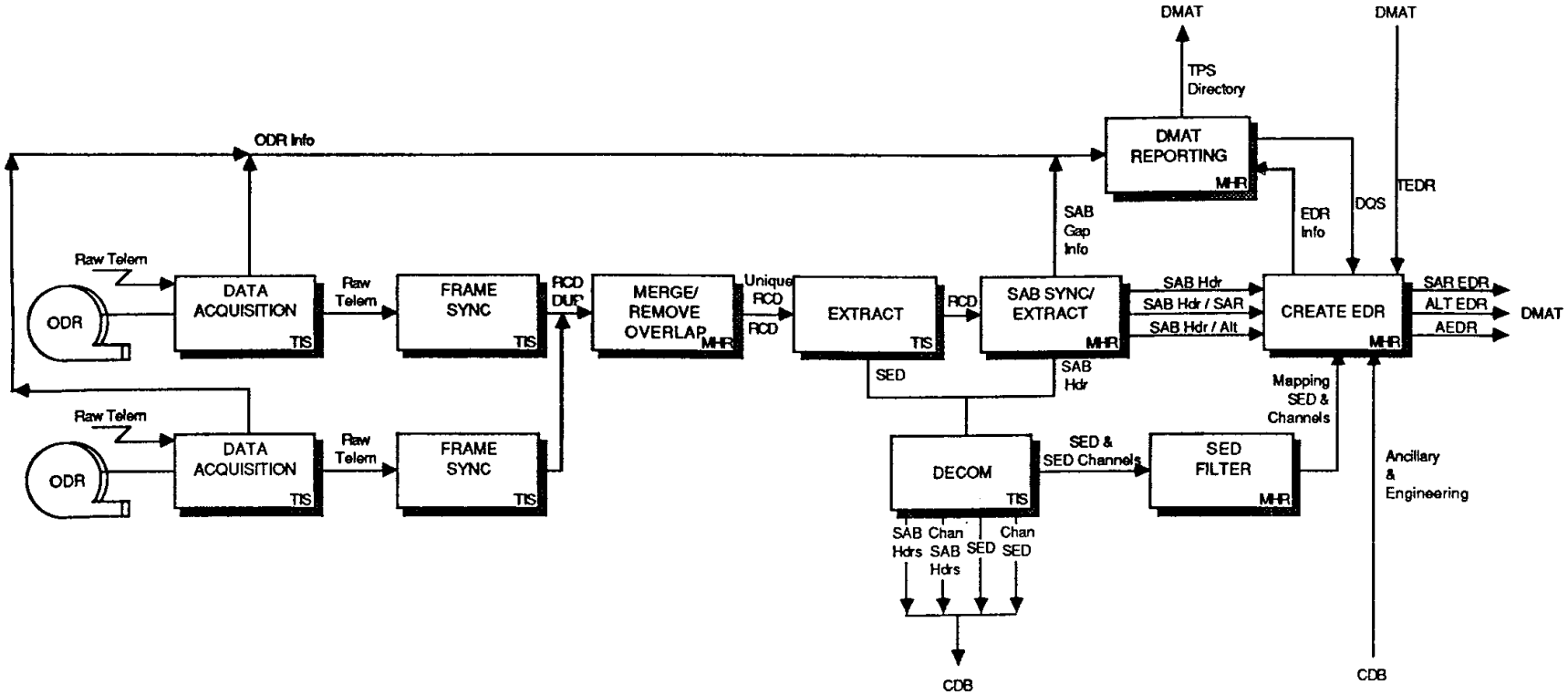

Figure 1 TIS Data Flow