COMMERCIAL-OFF-THE-SHELF TELEMETRY FRONT-END PROTOTYPING

Keith Hogie
Jim Weekley
Jeremy Jacobsohn

ABSTRACT

The world of data communication and networking has grown rapidly over the last decade, and this growth has been accompanied by the development of standards that reflect and facilitate the need for commercial products that work together in a reliable, robust, and coherent fashion. To a great extent this commercialization, with its increasing performance and diminishing cost, has not been adapted to the data communication needs of satellites. As budgets and mission development and deployment timelines shrink, space exploration and science will require the development of standards and the use of increasing amounts of off-the-shelf hardware and software for integrated satellite ground systems.

The Renaissance project at NASA/Goddard Space Flight Center has engaged in rapid prototyping of ground systems using off-the-shelf hardware and software products to identify ways of implementing satellite ground systems "faster, better, cheaper". This paper presents various aspects of these activities, including issues related to the configuration and integration of current off-the-shelf products using telemetry databases for existing spacecraft, an analysis of issues related to the development of standard products for satellite communication, tradeoffs between hardware and software approaches to performing telemetry front-end processing functions, and proposals for future standards and development.

KEYWORDS

Satellite telemetry, communication front-end, telemetry processing, standards, CCSDS

INTRODUCTION

In spring 1995, the Mission Operations and Data Systems Directorate at NASA/GSFC initiated an effort to construct a satellite ground system in 90 days using commercial-off-the-shelf (COTS) hardware and software. The goal of this activity was to demonstrate that a small group of experienced engineers could integrate various pieces of commercial

hardware and software into a viable system that would meet the operational needs of an existing spacecraft at a fraction of the cost required to develop a more traditional (i.e., custom) solution. The system was completed within the 90-day time frame and processed live data from the SAMPEX spacecraft simultaneously with the operational systems. The real-time portion of the system supported the receipt of telemetry and tracking data and provided state modeling capabilities for real-time satellite status monitoring and commanding. The off-line portion of the system supported orbit and attitude determination and additional data analysis capabilities.

This prototype provided a good vehicle for investigating the configuration and integration of current COTS products to construct a ground system for a real spacecraft. For the initial configuration, the Loral Test and Instrumentation Systems (LTIS, now Lockheed Martin Telemetry and Instrumentation (LMTI)) model 550 communication front-end was used. Since then other front-end systems have been inspected, and results of those experiences are presented below.

FRONT-END PROCESSING

The fundamental capabilities of satellite front-end processing can be stated in the following three areas:

1) Receive data from the satellite data sources. Typically this data is presented over standard interfaces in some predefined format (RS-422, TTL, Ethernet, Nascom block, CCSDS frame).
2) Isolate particular data fields and place the data in an acceptable form (CCSDS packet extraction, telemetry frame extraction, bit/byte reversal, engineering unit conversion).
3) Make the data available to other applications (custom application programming interfaces, UDP/TCP sockets).

A transmission capability is also required in order to transmit commands and data to the spacecraft.

For our satellite interfaces the majority of our work was done with Nascom 4800 bit blocks. The blocks were received over RS-422 serial interfaces at rates of 224 Kbps and 1 Mbps. The Nascom 4800 bit blocks provided a basic mechanism for delivering a satellite bitstream from the ground antenna to our front-end. The satellite data inside the blocks was then extracted and passed through another synchronizer to locate the CCSDS data frames from the spacecraft. CCSDS packets were then extracted from the frames, and the packets were decomposed into their individual telemetry fields such as battery voltages, currents, temperatures, etc. With the LMTI 550 front-end, this processing was

accomplished in the front-end and current value tables were passed to workstations for processing. With the other products, the extraction of telemetry fields was performed in software along with the other ground system applications on the workstations.

We note that the interfaces between the telemetry processing components and the ground system applications varied widely. Each product had a different application interface, and some programming was required in each instance in order to integrate the front-end with our other applications. We are currently examining the use of the Common Object Request Broker Architecture (CORBA) as a mechanism for hiding these differences and providing a more consistent interface between the front-end functions and the application programs.

FRONT-END ARCHITECTURES

Our activities included work with the following four front-end products:

1) Lockheed Martin Telemetry and Instrumentation (LMTI) model 550
2) Software Technology, Inc. OS/COMET
3) Veda Omega
4) National Instruments LabView

The most complete prototype was constructed with the LMTI 550 front-end which was configured to receive live data from the SAMPEX spacecraft and fully process over 3000 telemetry parameters. The other three products were used in later prototypes which were not as fully configured but were performed to compare and contrast the various products.

LMTI 550

The LMTI 550 hardware front-end utilizes a dual-bus architecture built on a VME chassis using VXworks. It passed data to other ground system applications executing on HP and Sun workstations using its data gather application program interface (API). The front-end chassis contained the following cards:

- Serial interfaces for RS-422 and TTL signals, frames synchronization, CRC checks
- LMTI field programmable processors (FPP), basically SPARC 10 processors, which were configured to perform packet extraction, field extraction and engineering unit conversion
- Controller board that provided an Ethernet interface between the front-end and the workstations
- SCSI interface and hard disk to support data logging and retrieval on the front-end

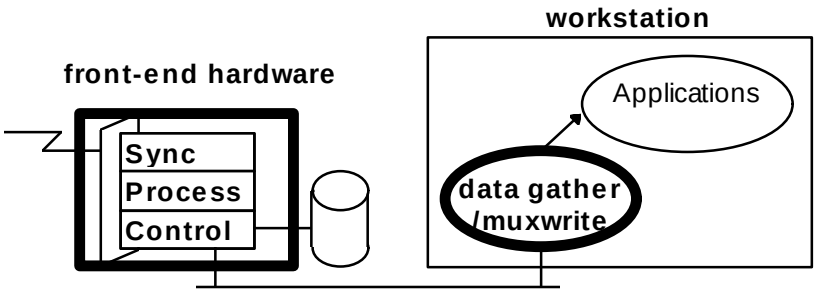

Figure 1 - LMTI 550 hardware and software components

The LMTI processes data by passing tag and data values on its high speed multiplex bus. Algorithms resident on the LMTI 550 FPPs are triggered as particular tagged data values appear on the bus. They receive the data value, perform various operations, and place the results back on the bus under a new tag value. These algorithms can be selected from standard libraries, or custom algorithms can be written in C. We configured the tagged data flows and algorithm processing on the LMTI 550 by developing PERL scripts to read spacecraft telemetry definitions from a satellite project database (PDB) and to generate an ASCII configuration file that could be imported by the LMTI 550. We also modified the source code for some LMTI 550 standard algorithms to produce custom algorithms.

A current value table of telemetry values was generated on the LMTI 550, gathered on the workstations, and fed to a data server to distribute to other applications for displays and spacecraft state modeling. Some displays were created with the LMTI display capability. Both LMTI and our state modeling product used Dataviews as their display engine which supported similar displays with either package.

The LMTI 550 provided a mature, flexible system with good documentation. Other configurations are available; for example, a frame synchronizer and controller can be provided in the chassis and all further processing can be done in software on a separate workstation

OS/COMET

The Open Systems COMET (OS/COMET) package from Software Technology, Inc. is a software product that takes data once it has been captured and performs telemetry decommutation, analysis, distribution and display. It does not have any specific hardware components. It requires the development of device drivers that interface with hardware interfaces and pass data to and from its “software bus”. Once data is on the OS/COMET software bus in the proper format, other processes such as the telemetry decommutation (TLM) process can be configured to process the data.

In OS/COMET, data to be shared among processes is stored in memory files (MFILEs) which are accessible across distributed platforms. MFILE definitions identify characteristics of the data fields such as type (e.g., character, integer, float), limit check ranges, units and descriptive information. A common mechanism for getting data into MFILEs is via the TLM process taking raw data off the software bus, performing field extraction and engineering unit conversion as specified in ASCII configuration files, and placing the resulting values into the indicated MFILEs. Other standard processes supplied with OS/COMET include character based display windows, graphical plotting tools, and the Sherrill-Lubinski Graphical Modeling System (SL-GMS) user interface. SL-GMS can be used to generate complex graphical displays based on data from the MFILEs. These applications are all integrated with the OS/COMET MFILE mechanism to provide display and modification of the values in the MFILEs.

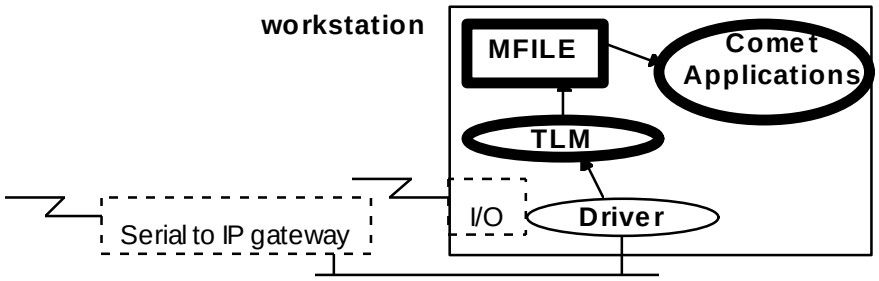

Figure 2 - OS/Comet software components

The system was configured by modifying the PERL scripts used to configure the LMTI 550 from a satellite project database. The main change was to generate output in OS/COMET formats to define MFILEs and configure the TLM process instead of the LMTI formats. Telemetry packets in Ethernet/IP were the data source and a device driver was developed to get our data into the OS/COMET environment. Other data sources could be I/O cards installed in the workstation with appropriate drivers to integrate them into the OS/COMET environment.

OS/COMET provided a convenient environment for displaying and changing data values in the MFILEs. However, some problems were encountered with OS/COMET while creating the configuration files for the telemetry parameters in the MFILE and the telemetry decommutation configuration file. There were some inconsistencies in the documentation and the version of software in use. Also, when the entire set of OS/COMET processes was activated for processing all 3000 parameters for a satellite, some processes requested up to 60 MB of operating system swap space. This was more than the other packages required.

The OS/COMET product requires C programming to integrate telemetry and command interfaces into the overall OS/COMET software environment. It also does not provide any standard support for the CCSDS protocols so other systems were used to get data into CCSDS packet formats for input.

Veda ITAS/OMEGA

The Veda front-end products investigated were their ITAS hardware front-end units and their Omega software package. The ITAS hardware product was not used but testing was done with the Omega software package which can execute on both the ITAS front-end hardware as well on our workstations. An interesting feature of the Omega software is that it could perform frame synchronization on a stream of data without any front-end hardware synchronizer. It supported a software frame synchronization capable of executing at up to 2 Mbps.

The Omega configuration options were mostly similar to the LMTI 550 except that the Omega software did not provide any CCSDS processing capabilities such as packet assembly. Also, if the software did not provide a required capability, our only option was to go back to the vendor for changes. Display generation was similar to the LMTI since Omega also uses the Dataviews package.

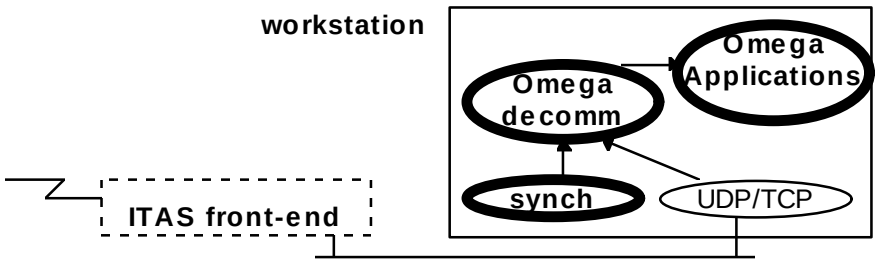

Figure 3 - ITAS hardware and software components

No real-time data was processed through the Omega software because at that time we did not have their hardware front-end and Omega did not support any data sources other that the ITAS front-end and data files on disk. Since then the product has been modified to support TCP socket interfaces. Also, Omega's interface to other applications did not provide as many options as the LMTI 550 and OS/COMET. It basically provides parameter tag and data information in UDP packets and lets a user develop software to select desired information.

LabView

The LabView product from National Instruments is a software product that provides extensive graphical display capabilities combined with a graphical programming system. The programming environment is oriented toward the development and operation of programs called “virtual instruments” (VI) which control hardware interfaces and integrate into the overall LabView processing and display environment.

A feature of this package is that it is supported on a wide range of platforms and operating systems such as MacOS, Windows 95, Windows NT, Solaris, HPUX. Components can be developed in one environment and then used on the other environments.

One of our main interests in this product was to see how it worked as a tool for use during integration and test of satellite components and then during operation. The goal here is to provide a consistent tool and user interface during the entire spacecraft life cycle.

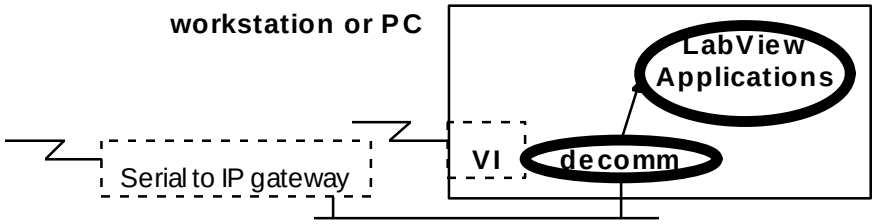

Figure 4 - LabView software components

The LabView system is oriented more toward development of graphical interfaces for laboratory and instrumentation systems. It is well suited for integration and test of devices during spacecraft construction. We are currently investigating its use as a system for building a full ground system for spacecraft operation. Like most of the other packages, the LabView software does not provide any specific support for CCSDS packet extraction and assembly. We are using other systems to break a data stream down to the CCSDS packet level and then passing the packets to LabView.

LESSONS LEARNED

The main lesson learned was that configuring telemetry front-ends is still a very time consuming, error prone, and custom process. All of the products we examined provide point-and-click options for various parts of the configuration process. However, none of the products could be quickly and easily configured to support all of our requirements. We feel that this in not so much the fault of the products as a reflection of the very wide range of telemetry options in use and the lack of sufficient commonality among them to allow the vendors to develop standard solutions. The following sections discuss some key problem areas along with recommendations for future solutions.

Wide variety of Nascom block types

The first problem encountered involved the format of the satellite bitstream data packed inside Nascom 4800 bit blocks. Data was received from both the NASA Wallops Flight Facility and Deep Space Network earth stations. Even though both systems use 4800 bit blocks, different formats are used for packing data into the blocks. Since our system also performed satellite orbit determination it needed to process tracking data which was also in different formats from the different networks.

Since Nascom blocks are not self-identifying, the solution was a crude one of identifying each of the possible combinations of source/destination codes, spacecraft ID, and data type fields and specifying separate data processing paths for tracking data and bitstream data for CCSDS frame processing. This was not difficult to accomplish, but it is not an elegant solution since it is difficult to extend to other missions and is quite prone to error. It reflects the fact that the legacy systems in use are based on a wide range of format options that exist across missions.

The primary solution here must come from more consistent and easily identifiable data formats from all agencies' earth stations. When that consistency is available, COTS products can be developed with standard options to handle this task with less development. Some future NASA missions are moving toward synchronizing CCSDS frames at the earth station and forwarding the frames via TCP/UDP based technology. This should be very helpful in eliminating the need for custom serial interfaces and special frame synchronizers at all of the users systems.

Limited CCSDS processing capabilities

The LMTI front-end had the most extensive CCSDS processing capabilities but even it did not support CCSDS packet telemetry formats when we began using it. The CCSDS packet telemetry capability was in testing at the time and was received in the summer of 1995 and used in our first prototype. None of the other products provided any native CCSDS processing options in their system. However, the Veda products can be configured to synchronize CCSDS frames just like any other frames. Veda also indicated they had some prototype code for CCSDS packet extraction but that was not tested. Also, even though the LabView product does not provide any native CCSDS capabilities, vendors have developed interface cards along with LabView virtual instrument software to support CCSDS processing. LabView in combination with these other products can provide a full range of CCSDS capabilities.

However, vendors are not entirely at fault for lack of CCSDS support. One problem is that the CCSDS recommendations allow a wide range of options for items such as packet

sizes, packet types and data formats and it is very difficult for any vendor to develop products that can support all of the possible CCSDS options. A standard definition of how to use a subset CCSDS options would be very useful.

No standard data types and bit/byte ordering.

Each spacecraft investigated defines telemetry field types and bit and byte ordering with the byte orderings such as those defined in Table 1. Also, bit fields were defined which require first reordering bytes according to the data type and then finding the proper bits.

The basic problem is that there are different understandings of bit/byte reversal options, bit ordering (MSB or LSB), and byte orderings, so that each product needs further programming and configuration to produce correct results. Also, fields can be up to 8 bytes long and that was not supported on all products. None of the front-ends provided standard field definitions for all field types for all missions examined.

Data Type	Description	Length	Byte order
UI	Unsigned Integer	2	N, N+1
UI386	Unsigned swapped Integer	2	N+1, N
ULI	Unsigned longword	4	N+2, N+3, N, N+1
ULI320	Unsigned unswapped longword	4	N, N+1, N+2, N+3
CHAR	Character string	1 to 50	N+1, N, N+3, N+2, N+5, N+4, etc.
TIME	UTC time format	8	N, N+1, N+2, N+3, N+4, N+5, N+6, N+7

Table 1 - Satellite data field types

Time fields are another special area with many different formats (i.e., 4, 5, 6 and 8 bytes in various orders) that are not supported by standard configuration options on COTS products. Further, once the time field bytes are in order, many formats exist along with different start times.

This can be addressed by identifying each possible field format and associating a standard name to the field format. Then spacecraft and ground system developers, vendors, and integrators will have a better chance of developing and using products that are quicker and easier to configure.

CCSDS commanding protocol and formats not supported in current vendors products

For sending data to the spacecraft, our missions all used the CCSDS COP protocol. This involves breaking data to be transmitted into small groups of bytes, adding checksums, exclusive-ORing, and other special formatting. During transmission there are also special protocols to follow for flow control and acknowledgment. If this data is to be delivered to the earth station using Nascom blocks it must also be packed into the blocks in the proper format. The basic problem is that none of the front-ends utilized had options for supporting this entire process. Our solution was to write C code to perform all the data formatting and build Nascom blocks. From this point, the standard front-end capabilities for transmitting Nascom blocks could be utilized. Nascom blocks need to be removed from this process and a protocol more like TCP should be developed.

No standard for interfaces between front-ends and user applications

Finally, all of the front-ends provided different mechanisms for exchanging data with application processes. These interfaces can be summarized as follows:

- LMTI 550 data/gather and muxwrite - proprietary LMTI API that provides flexible capabilities for transferring data between applications and any stage of front-end processing
- OS/Comet MFILEs and API - shared memory files and proprietary STI API that provide a full data server environment
- Veda Omega API - TCP/UDP socket API providing blocked tag/data pairs
- LabView VI - support for TCP/UDP socket capabilities

Standard APIs would simplify integration and allow easier upgrade or replacement of front-ends. We are currently investigating the application of CORBA to this problem.

CONCLUSION

COTS products are available and can be configured to meet satellite front-end requirements. However, the configuration and integration process still requires extensive customization due to the wide range of different spacecraft data formats, front-end configurations and interfaces to applications. If satellites are to be deployed “faster, better, cheaper” more work is needed to develop widely used standards for:

- capabilities for synchronizing data as it is received from the antenna and passing it to users with standard network technology, such as TCP/UDP/IP, to greatly reduce the need for costly and complex custom front-ends

- common definitions for telemetry field types (e.g., UB, UI) and byte orders so vendors can provide corresponding configuration options
- more precise definitions for the usage of CCSDS protocols so vendors can implement these as standard options
- common application interfaces to front-ends using technologies such as CORBA to hide the differences of each front-end and allow replacing front-ends without impacting applications

The only way to achieve NASA's goals of "faster, better, cheaper" is to get more well-defined standards in place to allow system integration to be done with plug-and-play components configured from standard, vendor provided options. This has been done with LAN and WAN data links, network equipment and distributed applications, and similar concepts can be applied to space.

Achieving maximum ease of integration requires much more work on end-to-end designs to ensure better integration of spacecraft, communication links and ground systems. Without this, integrating satellite ground systems will remain a time consuming, custom and expensive process.

ACKNOWLEDGMENTS

Support by NASA under contract NAS5-31500 is gratefully acknowledged. The encouragement and support of Gary Meyers and Mike Bracken of the Renaissance team at GSFC was particularly valuable.

REFERENCES

1. Data Format Control Document for the SAMPEX Project Data Base, 511-4DFC-0290, Mission Operations and Data Systems Directorate, NASA/GSFC, January 1993

2. Data Format Control Document for the Project Data Base Supporting the XTE Mission Operations Center, 510-4DFC-0193, Mission Operations and Data Systems Directorate, NASA/GSFC, August 1995