

Distributed Space-Segment Control Using SCL

Jim Van Gaasbeck
jimv@sclrules.com
Interface & Control Systems, Inc
1938 Dairy Rd
Melbourne, FL 32904

Allan Posner
posner@sclrules.com
Interface & Control Systems, Inc.
8945 Guilford Rd, Suite 120
Columbia, MD 21046

Brian Buckley
buckley@sclrules.com
Interface & Control Systems, Inc
1938 Dairy Rd.
Melbourne, FL 32904

Abstract

Distributed, autonomous space-segment command and control is one of the main objectives of present-day space-vehicle design. This objective is oftentimes difficult to achieve, however, since the spacecraft bus and the payload are usually manufactured by two different organizations and have correspondingly different designs and architectures. As a result, bus-to-payload communication is usually restricted to payload command sequences; payload science and housekeeping data are often the only products that flow back to the spacecraft bus. SCL, a modular and distributed fifth-generation command-and-control environment, may be used to bridge the bus-payload interface, coordinating autonomous temporal and event-driven control between both environments. Although using the SCL run-time engine on both the payload and the bus represents the most straightforward and effective method for implementing this coordinated space-segment control, similar results may be achieved even if SCL is only used in one of the two environments. In this paper we will examine some of the operational efficiencies that are gained by developing and implementing distributed command-and-control flight-system architectures using SCL. Specific case studies were chosen representing the recent evolution of event-driven flight systems. Each successive system introduced improved event-monitoring techniques, yielding third- and forth-generation systems encompassing a solid foundation for autonomous operations.

Introduction

Historically, the designs of spacecraft and payload subsystems have been relegated to two separate organizations. Oftentimes the spacecraft-bus manufacturer is a different company from that which produces the payload. The resulting differences in systems architecture and processor design further separate the two subsystems, both functionally and electronically. Recent NASA discovery- and explorer-class missions, through their cost-effective designs involving using off-the-shelf buses have made the goal of distributing space-segment control between the bus and the payload even more challenging.

As a result, typical bus-payload interface designs are primarily targeted towards payload functionality and its associated data exchange. Bus-to-payload communication is usually restricted to payload command sequences,

parameter and table loads, etc. Conversely, payload science and housekeeping data are often the only products that flow back to the spacecraft bus. Even if maneuver- or attitude-related data are generated by the payload, those data are rarely utilized by the spacecraft bus, instead being packaged together with other telemetry and relayed to the ground for post-flight analysis.

For certain mission profiles, however, the luxury of controlling all bus-directed attitude and maneuver operations via direct ground control is simply not an option. Remote-sensing LEO missions involving single ground-station coverage, for example, must rely on on-board payload attitude feedback to control pointing during times when the spacecraft is outside of ground contact. In addition, closing the control loop between the spacecraft bus and the payload decreases the time between sensory observation and bus actuator response. Finally, implementing the bus-payload control loop within an operating environment that is both time- and event-driven in nature yields the highest degree of control-system fidelity. All of these features — critical to successful, autonomous space-vehicle operations — are inherent within the SCL system.

Background

In 1988 during the cold war era, the Naval Research Laboratory (NRL) commissioned Interface & Control Systems (ICS) to develop a system which would support autonomous operations for a period of 180 days. ICS researched the possibility of adapting existing Artificial Intelligence systems to an embedded environment. Non-deterministic nature, dynamic memory management, and sheer size precluded the use of many products that were available. After much research, it was determined that a custom solution was needed to perform AI functions in a processor-poor environment. At the time, R3000's running 20 Mhz were still on the drawing board. These radiation-hardened chips didn't appear for nearly a decade.

Although ICS prototyped and demonstrated a system that implemented a rule-based Expert System, procedural scripts were still needed. Adding procedural scripts and multi-scripting allowed the system to meet the autonomy requirements for the NRL's Advanced System Controller (ASC) Project.

The days of large satellite programs were soon over, and in came “Better-Cheaper-Faster” programs. Clementine was one of the pioneering programs when it came to “Better-Cheaper-Faster”. Clementine leveraged off the ASC program and re-used the Spacecraft controller board, and the SCL flight software. Just a couple of years earlier, the Environmental Research Institute of Michigan (ERIM) was in need of an autonomous robot controller to operate an experiment in the shuttle bay. The ERIM Robot Operated Materials Processing System (ROMPS) project was to study the effect of semiconductor annealing in a micro-gravity environment. ERIM also leveraged from a previous program and used the SCL architecture to automate the robot tasks and control equipment used for the experiment.

These two programs helped start the evolution of the SCL system. Lessons learned from one program were incorporated into the next generation design.

Based on the publicity around the Clementine mission, the Far Ultraviolet Spectroscopic Explorer (FUSE) Program Management sought a solution to a shrinking budget by re-using SCL across the system architecture. The FUSE mission experienced significant budget cuts which resulted in the mission being changed from a geosynchronously-orbiting to a low-earth-orbiting satellite. Planning algorithms which had previously been planned for closed loop control using the ground system, needed to be incorporated on-board. This required a smart, data-driven architecture to control the Ultraviolet Telescope payload and the spacecraft bus. In the FUSE section below, we will examine the FUSE flight software architecture.

Although the FUSE mission offered many advances in the SCL architecture, the Navy Earth Map Observer (NEMO) takes the paradigm even further. The NEMO spacecraft is a hyper-spectral imagery satellite requiring management of high bandwidth image data, situational awareness based on an on-board GPS receiver, and on-board Fault Detection Isolation and Recovery (FDIR).

Another on-going program is the Space Station’s Interim Control Module or ICM. This is a quick turn-around mission requiring FDIR capabilities and Autonomous operations. The ICM system makes re-use of the SCL software System as the Flight Controller software and has re-packaged existing NRL booster technology. The SCL software system architecture has been exploited to meet the unique requirements of the mission.

The SCL System Architecture

The SCL system is an enabling core technology for command and control applications. The SCL system provides a seamless architecture between the ground and space segments. Commonality is the key to software and knowledge reuse. SCL is a hybrid system that employs a rule-based, event driven expert system as well as a procedural scripting capability. This portable architecture allows a symmetrical software architecture between ground and space and an inherent capability to reuse knowledge from one phase of the development life cycle to another.

The SCL development environment consists of a ground based Integrated Development Environment (IDE) used to develop SCL scripts and rules. The SCL Real-Time Engine (RTE) was designed to be portable and run in a real-time embedded systems environment. The SCL RTE represents the majority of the code necessary to implement an embedded spacecraft controller. Early on, ICS saw the need for the integration of ground and space operations with a common control system using a single control language. By using SCL in a distributed environment, the command language for ground and space segments share a common syntax. The SCL grammar is based on fifth generation languages and is very english-like, allowing non-programmers to write the scripts and rules that constitute the knowledge base. Because the ground-based SCL environment uses the same RTE as the spaceborne controller, scripts and rules can be developed and debugged on the ground-based RTE environment before requiring the spacecraft hardware for final checkout.

This common architecture provides a capability to have an on-board current value table. This allows the spacecraft or payload to have knowledge of the current state of all related sensors. On board software driven decommutation is used to keep the database populated with current values. Closed loop algorithms that previously had been executed on the ground can now be executed in space.

Space-Ground Architecture

The SCL system consists of five major components:

- The database describes digital and analog objects that represent spacecraft sensors and actuators. The latest data sample for each item is stored in the database. The database also contains derived items that are artificial telemetry items whose values are derived from physical sensors. Examples of derived items could be: average temperature, power based on current and voltage monitors, subsystem status variables, etc. Data structures required to support the Inference Engine are also stored in the database.
- The SCL development environment includes an integrated editor, the SCL compiler, decompiler, cross-reference system, explanation subsystem, and filing system. The development environment is also used as a front-end to control the SCL RTE. A command window is used to provide a command-line interface to the Real-Time Engine.
- The RTE is the portable multi-tasking command interpreter and inference engine. This segment represents the core of the flight software. This portion

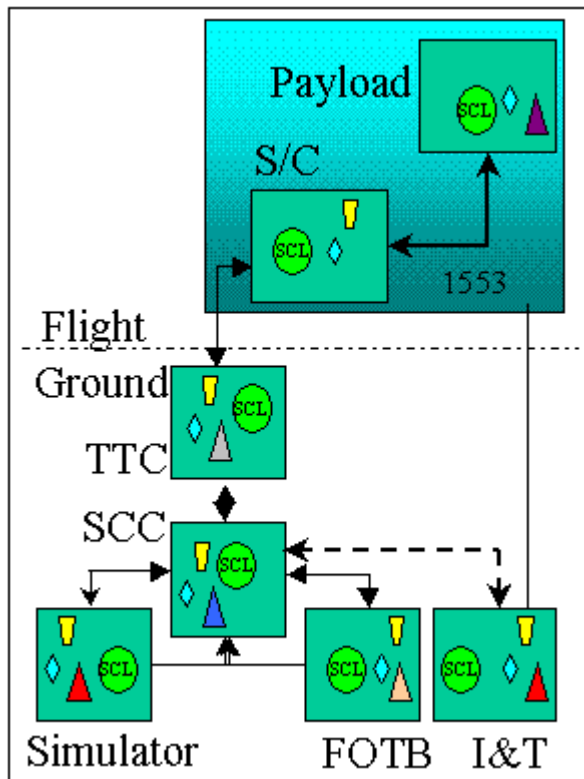


Figure 1 - SCL Space Ground Architecture

of the software is written in C to allow ease of porting to a specific hardware platform (ground or space).

- The Telemetry Reduction program is responsible for filtering acquired data, decommutation (if necessary), storing significant changes in the database, and presenting the changing data to the Inference Engine.
- The project is the collection of SCL scripts and rules that make up the knowledge base. On the ground-based systems, the project contains an integrated filing system to manage the knowledge base. In the space environment, the binary knowledge base is uploaded to the spacecraft and stored in memory.

Depending on the needs of the user, all components of SCL can be run on a single system, or may be distributed among systems; see Figure 1. The development environment can be used to directly connect to a local or remote version of the SCL RTE. This direct connect capability is also supported for the space segment to allow interactive commanding and query of the system.

Space-Space Architecture

Catalog shopping has invaded every facet of our lives. NASA can now shop for satellite busses by catalog also. This fixed-price turn-key approach has led to cost savings for NASA, but has required that vendors offer an off-the-

shelf solution that is proven, can be quantified, and can be replicated at will. Cost containment has mandated a standard bus controller, standard interfaces to payloads, and a conservative approach to newer technologies. For this reason, the Payloads are sometimes more sophisticated than the bus controllers. In these scenarios, systems like SCL can be employed to offer a supervisory function for the bus controller itself.

High-level interfaces between a payload and the bus controller allow the payload to make bus-tasking requests. In the case of FUSE, the payload is an Infrared Telescope that will steer the bus as the satellite is in low earth orbit. This architecture allows the more modern payload controller to employ more cutting edge technology to effect autonomous operations. High-level goals can be used to task the bus; ephemeris data, slew commands, downlink functions, and uplink functions can be placed under the control of an intelligent payload controller. As you will see in the next several sections, this capability has evolved from its earliest beginnings, and still evolves today.

SCL for the ROMPS Program

The Environmental Research Institute of Michigan (ERIM) was awarded the Robot Operated Material Processing in Space (ROMPS) program. ERIM needed an experiment controller that would fly on the shuttle for 11 days and operate while the astronauts slept. The ROMPS mission used a derivative of an industrial robot that utilized a 3 degree of freedom arm. The ROMPS experiment was used to investigate the effects of high temperature annealing in a low-G environment on semiconductor materials. The robot was tasked to move more than 200 pallets of semiconductor material into an annealing oven for a specified period of time. The experiment was required to be fully autonomous and be performed on an aggressive budget.

ERIM re-used a spacecraft controller from a previous project without modification. New I/O devices were added to control the robot, the oven and a capaciflector experiment. The System Controller software was based around SCL and remained largely unchanged. The SCL software allowed ERIM to re-load the on-board database to describe the hardware devices. The control algorithms were written in the form of SCL scripts and rules. Scripts and rules are analogous to the Command Store Memory (CSM) on most satellites. Scripts and rules are written in the SCL high-level scripting language and are designed to be reloaded on a daily basis, or they can remain resident for the life of the program.

The SCL software uses a data driven approach to define the unique requirements of each mission. The SCL database allows the end-user to define the characteristics of all sensors and effectors. The sensors can also be sampled on-board and decommutated to allow visibility into the current value for each device. This allows closed-loop control algorithms to be executed on-board. The low-level acquisition software and the I/O driver-level interfaces are normally customized for each program. The uplink and downlink protocols are also normally customized for the

mission. Air Force, Navy, and NASA missions often use different uplink and downlink protocols.

The ROMPS program was able to get a jump-start by having the bulk of the on-board controller software done before the project began. ROMPS was a second-generation SCL system for ERIM.

Since the SCL system is symmetrical and uses the same architecture for the ground systems, SCL was also used for desktop simulation of spacecraft subsystems, the testset control system, and as the operational ground control system.

Early prototyping of spacecraft systems with a software simulation allowed for the development of the software control system in parallel to the hardware development. The engineers were able to develop control system scenarios and stimulate them in desktop environment. As hardware became available, testing could be performed with hardware in the loop. This configuration allows a step-wise integration of hardware with a software simulation of missing components; see again Figure 1.

The ROMPS testset consisted of off-the-shelf hardware and software. The National Instruments LabViews product and the SCL system were used to control the testset. SCL contains bridge code to the LabViews interface. LabViews was used as the visualization system and for much of the board level hardware control. The SCL system was used to script test procedures. A detailed set of LabViews screens was developed by the customer to show the robot arm angles and the "bins" containing pallets of semiconductor material. Screens were also defined to monitor and control the annealing oven.

SCL includes a module called DataIO, which is used to decommutate and store telemetry. The log files are in standard formats, which can be translated and analyzed, using Microsoft Excel spreadsheets and graphs. Again, inexpensive off-the-shelf tools were used for data analysis.

Since the ground control system was largely the same as the testset, many of the SCL scripts and rules which were tested during I&T could be used for operational control. The LabViews screens were also re-used for real-time operations. The scripts and rules were designed in a manner which allowed them to be used in the test environment as well as migrated onto the spacecraft.

The ROMPS mission was launched on September 9, 1994 on STS-64. The 11-day mission was 100% successful. The SCL system processed more than 200 semiconductor material samples while the astronauts slept. The ROMPS hardware was located in a "Get-Away Special" (GAS) canister located in the shuttle's payload bay. The SCL system was running on a 12 MHz 80286 "look-alike" with ceramic parts. The ground system was a distributed network of personal computers at Goddard Space Flight Center connected to the NASA Hitchhiker interface.

SCL for the Clementine Program

The Clementine (AKA Deep Space Probe Science Experiment) spacecraft was a BMDO/NRL sponsored

program that tested lightweight BMDO sensor technology while gathering science data. The Clementine program was the first of the "Better, Cheaper, Faster" missions. It went from "vaporware" to hardware in 18 months. The mission cost ~\$75M with \$50M of the cost for the Titan II rocket. The Clementine vehicle was launched in February of 1994. The primary mission was to map the surface of the moon for the first time since the Apollo days, and to perform an asteroid intercept in deep space.

After orbiting the Earth for a brief period, the Clementine vehicle was sent on a trajectory that took it to the moon, where it entered a lunar orbit. The probe was in lunar orbit for several months and returned more than 1.5 million images detailing lunar soil mineral composition. The Clementine mission achieved all mission objectives with the exception of the asteroid intercept.

The SCL system was used on-board the Clementine spacecraft to perform TT&C functions, system health and welfare monitoring and anomaly resolution. The mission required a high degree of autonomy and made extensive use of the SCL command and control capabilities. The Clementine mission took a conservative approach and only used the scripting capabilities of SCL for the main mapping phase of the mission. Once the entire moon had been mapped using on-board sensors, autonomous operations were tested. Autonomous mapping was achieved by using SCL rules to trigger data collection based on spacecraft position.

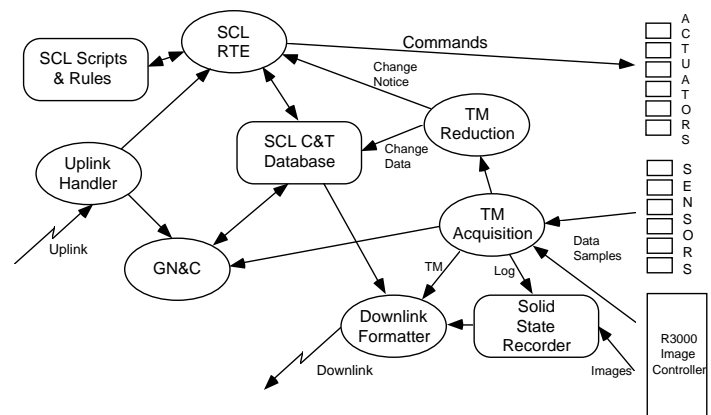


Figure 2: Clementine Flight Software Architecture

The SCL system follows an open-systems approach allowing other tasks to communicate with SCL on the ground, and in space. For Clementine, the SCL system was required to have bi-directional communications with the Guidance, Navigation and Control (GNC) algorithms which were written as another task; see Figure 2. Sequencing of the spacecraft maneuvers are handled by SCL, but the low-level thruster pulse commands are handled by the GNC software. Attitude information is reported back as telemetry, allowing the SCL expert system to inference on the changing data. The SCL Flight

Software used on Clementine was largely re-used from another Naval Center for Space Technology (NCST) satellite program.

The Clementine flight software design closely mirrored that of the ROMPS mission. The Clementine flight processor is the MIL-STD-1750A and uses the Ada version of the SCL Real-Time Engine. SCL was also used on the ground system to develop mission tasking loads and interface with the existing ground control system. The Clementine program achieved a significant science return and for an unheard of budget. The Clementine mission has been held as a model of the "Better, Cheaper, Faster" philosophy to be adopted by NASA space programs.

SCL for the FUSE Program

The Far Ultraviolet Spectroscopic Explorer (FUSE), a NASA-supported, Earth-orbiting mission scheduled for launch in the summer of 1999, is designed to investigate basic physical processes in the universe. Through NASA-directed restructuring, the mission architecture has evolved to include a low-earth-orbiting satellite, utilizing significant on-board and related ground-segment autonomy, and an autonomously-controlled ground station. As a result, the Johns Hopkins University (JHU), who leads the development of this first in the series of Mid-Explorer (MIDEX) missions, investigated options for reducing the costs of development and operations, while minimizing project risk. SCL, whose artificial intelligence capabilities and Expert Systems Technology were successfully used on the Clementine mission, was specifically selected as the basis for the combined, autonomous space- and ground-segment development and operation.

FUSE Spacecraft operations are controlled by the Central Electronics Unit in the Spacecraft bus. A separate subsystem, the Instrument Data System (IDS), controls the operation of the FUSE far-ultraviolet (FUV) telescope. The IDS contains a level of sophistication that exceeds that of many spacecraft controllers. IDS features include a moderately-powerful CPU (a 17-MHz Motorola 68020), a large stored-command memory, and the SCL expert system used for stored-command processing. Virtually all science operations, as well as Spacecraft pointing functions, are controlled via the IDS.

All IDS software tasks are "expert system aware". Each task implements a set of event triggers that can be used, at the discretion of the FUSE planners and operators, to trigger SCL rules. Although the IDS also provides the capability to decommutate telemetry from IDS telemetry packets, routine events are sent directly to the SCL expert system so that any rules associated with an event can be evaluated with a minimum of overhead. SCL rules can be associated with event triggers. When an event occurs, any rules associated with the event are executed by the SCL expert system. Rules in turn can execute SCL scripts. A rule is a simple If-Then type of statement that is compiled as an individual SCL code block. A script is a block of IDS commands and SCL statements that are similar to a typical stored-command list. Any combination of rules and

scripts can be stored in the IDS on-board memory.

Because the IDS operates in an event-driven manner, many activities that would traditionally be initiated via time-based commands may instead be initiated using rules that respond to event triggers. The event-driven nature of the IDS promotes increased operational efficiency by allowing observation steps to proceed as quickly as possible and supports modular stored-command loads that are largely reusable.

Ultimately one of the most important lessons learned from the FUSE development effort lies with the flight SCL scripts. Those scripts, combined with the capabilities of the IDS-based SCL RTE, possess the unique ability to interact with the *spacecraft* attitude control system (ACS) in an event-driven environment; despite the fact that the ACS is *not* based on the SCL architecture.

As Figure 3 indicates, FUSE coarse attitude is maintained to an accuracy of 2 degrees using only coarse sun sensors, magnetometers and torquer bars. Once fine-attitude control has been achieved, the ACS IRU and the reaction wheels can maintain an attitude accuracy of 10 arcminutes. The unique, closed-loop, control-concept

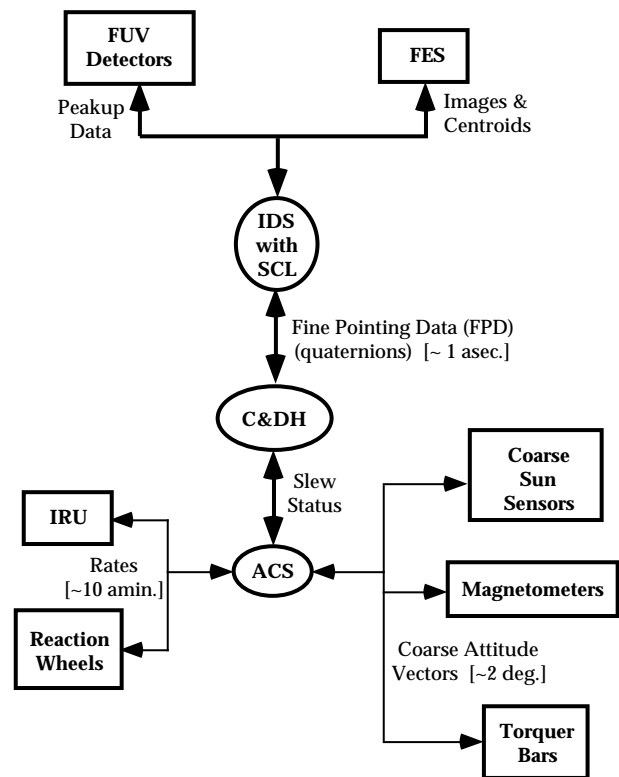


Figure 3: FUSE Attitude Control Concept

employed on FUSE involves the sending of Fine Pointing Data (FPD) from the IDS to the ACS. The FPD, obtained through the processing of visible star data from the Fine Error Sensor (FES) and FUV-detector count-rate data, are sent by the IDS to the ACS and are processed to maintain a pointing accuracy of less than 1 arcsecond. As part of the

closed-loop feedback process, the ACS sends “slew-complete” messages back to the IDS.

As such, not only will the IDS be able to direct ACS-based slews via the fine-pointing directives that it sends, but it will also be able to operate closed-loop with the ACS, starting its next activity upon receiving the slew-complete flag from the ACS. It is this ability to operate between disparate flight-hardware environments that is one of the true strengths of the FUSE IDS-based, SCL implementation.

SCL for the NEMO Program

The Naval EarthMap Observer (NEMO) is a space-based remote sensing system for collecting broad-area, synoptic, and unclassified Hyperspectral Imagery (HSI) for Naval Forces and the Civil Sector. NEMO meets unique requirements for imaging the littoral (shallow water) regions on a global basis, and also meets civil needs for imagery supporting land use management, agriculture, environmental studies, and mineral exploration.



NEMO provides:

- HSI to characterize the littoral battlespace environment and to support littoral model development;
- Automated, on-board processing, analysis, and feature extraction using the Naval Research Laboratory's (NRL's) Optical Real-Time Adaptive Signature Identification System (ORASIS)
- Demonstration of real-time tactical downlink of hyperspectral end products directly to the field to support the warfighter.
- HSI for Department of Defense (DoD) usage and for civil applications such as environmental monitoring, agriculture, land use, geology/mineralogy, and hydrology; and
- A cooperative industry and government program to share costs and leverage system utility under the Defense Advanced Research Project Agency's (DARPA) Joint Dual Use Application Program (JDUAP)

Spacecraft and Payload Characteristics:

- Hyperspectral Coastal Ocean Imaging Spectrometer (COIS) provides moderate spatial resolution with a 30/60 meter ground sample distance (GSD) and a 30 km swath width
- High spectral resolution of 10 nanometers with a spectral range of 400 to 2500 nanometers
- Panchromatic Imaging Camera (5 meter GSD) co-registered with the Coastal Ocean Imaging Spectrometer
- Real-time feature extraction and classification with greater than 10x data reduction using NRL's ORASIS algorithm
- High-performance Imagery On-Board Processor (IOBP) provides greater than 2.5 gigaFLOPS of sustained

computational power

- On-board data storage (48 gigabit)
- High data rate X-Band Downlink (150 Mbps)
- Low data rate S-Band Tactical Downlink (1 Mbps)
- Commercial satellite bus (Space Systems Loral LS-400)
- Preconfigured Interface (PCI) for secondary payloads/experiments

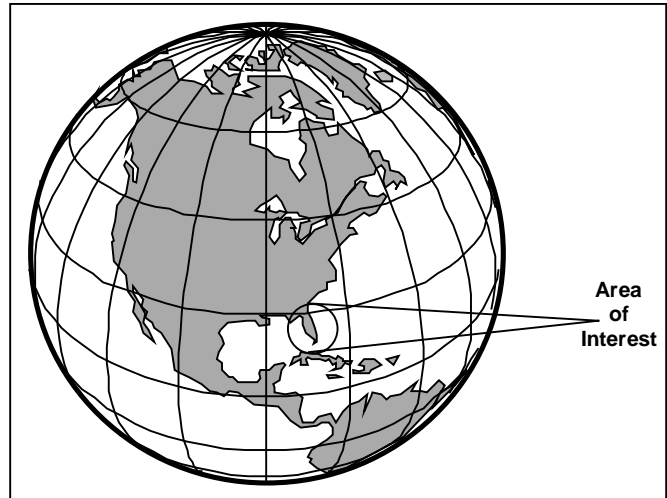


Figure 4: NEMO Mapping Scene Generation

FUSE observation planning and scheduling is performed at the ground system and then an observation plan, in the form of SCL scripts and rules, is sent from the satellite control center (SCC) to the FUSE payload controller. Once the plan is executed on board, much of the tasking proceeds in an event-driven manner. A fairly significant amount of detailed planning at the SCC is required, however, to build each uploaded plan. NEMO's designers seek to reduce some of the routine-image planning activities by moving them on-board the payload controller. The goal is to allow NEMO's controllers to concentrate on the images rather than the mechanics and details of capturing each image.

The NEMO payload-controller software takes a modest step towards implementing on-board image planning. Although significant capabilities could have been exploited on the NEMO program, the system designers were careful to set achievable goals for on-board autonomy and image planning. On NEMO, SCL will be used to automate operations and support FDIR in much the same way it will be used on FUSE. This reuse of software and operations concepts will help reduce the risk to the NEMO program.

The addition of on-board imaging planning is what will set NEMO apart from FUSE. NEMO's on-board image-planning capability will build on the event-driven commanding foundation established by the FUSE software. A separate image-planning task will be added to the

payload controller software to transform high-level imaging commands into imaging events and into the lower-level commands needed to actually collect and process an image. Imaging events are the messages that are sent to the SCL software that cause it to evaluate SCL Rules and, consequently, to execute the image tasking. The low-level commands are the actual commands sent to the camera controllers, attitude control system and the solid state data recorder among others.

A NEMO imaging command contains many information fields describing the location of the imaging target and the desired sensor configuration. Target location information includes the target's starting latitude and longitude, its ending latitude and longitude and the sensor "look" angle to an arbitrary point in the target field; see Figure 4. Sensor configuration information includes which cameras are to be used for an image and the image-processing configuration.

Up to 9 Image commands are batch processed by the NEMO image planning software. Current operations concepts have this occurring about one orbit ahead of the actual imaging orbit. The batch processing will yield a sensor event table that feeds the imaging events to the SCL software and an attitude control system slew table that specifies the spacecraft slews needed to capture the entire sequence of up to 9 image swaths.

When the actual imaging time arrives, the events in the table are distributed to the SCL software and the spacecraft slews to each image target starting point. The events are used to trigger SCL rules that in turn execute on-board software that extracts the camera configuration data from the currently-executing imaging command and then decomposes the information into the required low-level commands. This continues until each of the imaging commands has been executed for the current orbit. When a new orbit begins, a new set of tables will be available to control the next imaging sequence.

At all times, SCL rules will be monitoring the progress of the imaging sequence along with the payload subsystem health in general. If need be, the rules can abort an imaging sequence and place the payload into a safe, idle configuration.

It is hoped that this enhanced planning capability will allow NEMO's mission planners to concentrate on the images that are to be acquired rather than the details of how to acquire each image. This is important because, as a dual government and commercial spacecraft, the demand on NEMO's imaging resources will be significant.

Conclusion

In order for future low-cost missions to achieve the scientific throughput required by the increasingly demanding public and private sectors, in-flight autonomy must be implemented. Despite the fact that proposed

decreases in ground-station coverage and operations personnel have provided the catalyst for implementing on-board autonomy, progress has nonetheless been slow and stepwise. All too often, the desire to implement a technological quantum leap has been all but thwarted by the need to guarantee mission success. In addition, despite the existence of proven, COTS-based systems for implementing autonomy, companies still shy away from technologies that are "Not Invented Here (NIH)".

Another factor in slowing the inclusion of automation has been the NASA desire to do "mail order shopping" for satellites. Purchasing a satellite bus from a catalog has forced satellite manufacturers to use conservative (i.e., outdated) design approaches to the Satellite Control Computer. These "off the shelf" spacecraft offer a turn-key bus which provides a payload plate, a power harness, and data harness, such as a 1553 bus. This philosophy offers no incentive for automation of the Bus Controller; consequently, the new payload controllers are more sophisticated than the bus controller itself.

Without a doubt, experiences from projects such as FUSE and NEMO serve well to demonstrate the tremendous benefit of using an extensible, modular environment, such as SCL, to close the control loop between the spacecraft bus and the payload. SCL's event-driven architecture simplifies what used to be time-line-based operations, so that space systems can now be designed around discrete events and goals. Accordingly, through using SCL, the bus-payload interface becomes somewhat more transparent, facilitating autonomous temporal and event-driven control between the two otherwise disparate environments and resulting in tremendous operational-cost savings throughout mission life cycles.

References

Interface & Control Systems Inc. Home Page
<http://www.sclrules.com>

FUSE Home Page
<http://fuse.pha.jhu.edu>

NRL NEMO Public Home Page:
<http://nemo.nrl.navy.mil/public/index.html>

NRL ICM Public Home Page:
<http://issbus.nrl.navy.mil>