SCL: An Off-The-Shelf System For Spacecraft Control

By: Brian Buckley & James Van Gaasbeck
Interface & Control Systems, Inc.
1942 South Dairy Road
Melbourne, FL 32904

Abstract

In this age of shrinking military, civil and commercial space budgets, an off-the-shelf solution is needed to provide a multi-mission approach to spacecraft control. A standard operational interface which can be applied to multiple spacecraft allows a common approach to ground and space operations. A trend for many space programs has been to reduce operational staff by applying autonomy to the spacecraft and to the ground stations.

The Spacecraft Command Language (SCL) system developed by Interface and Control Systems, Inc. (ICS) provides an off-the-shelf solution for spacecraft operations. The SCL system is designed to provide a hyper-scripting interface which remains standard from program to program. The spacecraft and ground station hardware specifics are isolated to provide the maximum amount of portability from system to system. Uplink and downlink interfaces are also isolated to allow the system to perform independent of the communications protocols chosen. The SCL system can be used for both the ground stations and the spacecraft, or as a value added package for existing ground station environments.

The SCL system provides an expanded stored commanding capability as well as a rule-based expert system on-board. The expert system allows reactive control on-board the spacecraft for functions such as Electrical Power Systems (EPS), thermal control, etc. which have traditionally been performed on the ground. The SCL rule and scripting capability share a common syntax allowing control of scripts from rules and rules from scripts. Rather than telemeter over-sampled data to the ground, the SCL system maintains a database on-board which is available for interrogation by the scripts and rules. The SCL knowledge base is constructed on the ground and uploaded to the spacecraft.

The SCL system follows an open-systems approach allowing other tasks to communicate with SCL on the ground, and in space. The SCL system was used on the Clementine program (launched January 25, 1994) and is required to have bi-directional communications with the Guidance, Navigation and Control (GNC) algorithms which were written as another task. 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 Clementine SCL Flight Software was largely re-used from another Naval Center for Space Technology (NCST) satellite program.

This paper will detail the SCL architecture and how an off-the-shelf solution makes sense for multi-mission spacecraft programs. The Clementine mission will be used as a case study in the application of the SCL to a "fast track" program. The benefits of such a system in a "better, cheaper, faster" climate will be discussed.

Introduction

In 1988, the Naval Center for Space Technology (NCST) and Interface and Control Systems, Inc. began development of a spacecraft controller for a "black" program. Due to the political climate at the time, the requirement was levied for 180 days of autonomous operation for the satellite. Since then, the politics have changed, but the system which was designed and prototyped showed a great deal of promise and was funded for development even though the 180 day autonomy requirement was discarded. ICS has evolved the concept for Spacecraft Command Language (SCL) over the years and has developed a spacecraft flight control system which is

innovative in its approach to ground and space standardization.

The SCL system provides an embedded control system software package for the spacecraft which uses a rule-based expert system. This A.I. technique was prototyped and found to be awkward to use for the day to day operations of a satellite. We found that adding a high-level scripting capability integrated with forward chaining rules provides a powerful alternative to the traditional approach to spacecraft command and control. The SCL system is based on a Hyperscripting language which can be extended to meet the mission unique aspect of each spacecraft. The added benefit of this system is that it can be run on workstations to control the ground station mission operations. The system is designed to be portable to a wide variety of workstation-class computers and drives third party graphics products to provide a visualization interface. The SCL system is normally used as the integrating factor for ground stations. The SCL system is used to sequence operations and control other software packages, both custom and Commercial Off The Shelf (COTS).

The SCL concept is based on the unification of ground and space with the same control system. The SCL system provides a flight control system with an on-board database allowing scripts and rules to have visibility into on-board data samples. The workstation version of the SCL system can be applied from board level checkout up through mission operations. The SCL system not only allows re-use of the underlying control system throughout the phases of satellite development, but also allows re-use of the scripts and rules which are developed to configure the spacecraft. The SCL scripts and rules can be developed and tested in early phases of I&T and stored in a repository for use throughout the life of the spacecraft. This aids in the Configuration Management of "trusted" sequences for spacecraft configuration. These trusted sequences can be managed with a software configuration management tool and referenced throughout the development cycle of the spacecraft program. High-level mission tasking sequences can build upon underlying configuration scripts and rules.

Domain experts can be interviewed and knowledge of system operation can be captured in the form of rules. These rules can be used to build up a simulation of the system and develop Fault Detection Isolation and Recovery (FDIR) scenarios. The SCL toolkit includes an Integrated Development Environment (IDE) which can be used on a desktop computer to prototype and test control algorithms. The event-driven nature of the SCL system makes it ideal for FDIR scenarios. Interviewing domain experts early in the project allows knowledge to be captured thereby reducing the effects of the brain-drain when key personnel leave the program.

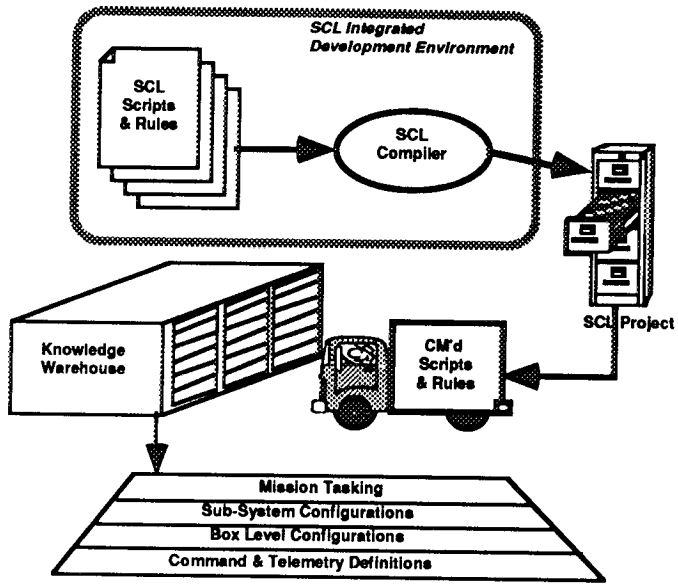

Figure 1. Approach to Knowledge Re-use

The Clementine spacecraft is a system which validates the SCL concept. The SCL software was originally developed for the NCST Advance System Controller (ASC) program. The SCL software and the flight controller and memory cards were re-used for the Clementine mission. Control loops for the attitude control system were analyzed and found to be appropriate for development in a native language rather than SCL.

The Clementine Guidance, Navigation and Control (GNC) software was developed in C and integrated as another task in the real-time operating system. The SCL system was tailored to allow bi-directional communication between the SCL and GNC software. This capability was eventually used to perform automated mapping of the moon. Information from the GNC system was used to sequence the commands required to configure the payload and collect image data.

Another program has recently benefited from the SCL software. The Environmental Research

Institute of Michigan (ERIM) has developed two systems using the SCL system. The Autonomous Rendezvous and Docking (ARD) mission was to explore the feasibility of spacecraft performing autonomous docking maneuvers and fluid transfer experiments. The ARD system was developed using SCL running on a single board computer interfacing to a series of I/O cards. The ARD payload passed acceptance tests and was integrated with the satellite bus, docking subsystem and fluid transfer systems. However, ERIM has removed the ARD payload from the Commercial Experiment Transporter (COMET) and Conestoga launch vehicle manifest. ERIM is currently searching for alternative launch vehicles.

ERIM reused the flight controller hardware and software design from ARD for the Robotic Material Processing in Space (ROMPS) experiment. ROMPS is a Space Shuttle Getaway Special (GAS) experiment which is manifested for flight on STS-64 on September 10, 1994. The ROMPS flight system was largely based on the ARD system with modifications to the low-level software. The ROMPS mission is to perform semiconductor annealing experiments in a microgravity environment. The system will control a robot and an annealing oven. This program has used SCL as part of a low-cost ground station. SCL is used in conjunction with National Instruments LabViews on Macintosh systems. LabViews provides a graphics engine used for visualization of data by the SCL system.

In recent years, SCL has gained a great deal of attention due to the desire for standardization of spacecraft control systems. The SCL system is portable to most embedded microprocessor platforms and operating systems. The underlying messaging system used for the uplink and downlink protocol is isolated from the SCL system. The Clementine system used the CCSDS communications standard although most missions have unique protocols. The AIAA, JPL, the Air Force and NASA have been looking closely at SCL as a basis for a standard on-board system architecture. The fact that SCL can be used on the ground also has added benefits in a "better, cheaper, faster" environment.

The answer to better, cheaper, faster lies beyond the Clementine mission. The Clementine mission was a high-risk, fast-track mission which went from vapor-ware to hardware in roughly two years. A new management approach and many innovative steps were taken along the way. Traditional or "old guard" methods were sidestepped to meet the aggressive schedule and budget.

Its a money thing

Support for a standard operational approach to spacecraft control is spawned by shrinking budgets. Today's tight budget situations don't allow for fresh starts; millions of dollars can be spend replicating existing technology. Systems such as SCL allow for multi-mission application of the same control system. This standardization reduces software development, training and maintenance costs. Ground stations can be retrofitted to support existing satellites with a higher-level system which supports advanced automation features. Operator workload can be reduced, and advisory systems can be developed using the Expert System which is incorporated in the SCL system.

The key to developing a new software approach is to invest in technology which embraces an Open System architecture, industry standards, and allows room for growth. New technologies can be merged in as appropriate and existing code can be replaced with COTS products. COTS solutions will reduce the development and maintenance costs since they can be spread across a customer base. New technology can be phased-in by using a value-added approach. Older, high-maintenance code can be retired as confidence grows in the newer system.

Training is important. Getting the day-to-day users of the system up to date on the nuances of the system will improve productivity and allow exploitation of the new capabilities of the system. Experts can provide help in choosing the best alternative for implementation of requirements. The experts need to be brought in at the beginning when the Systems Engineering is being done. Too often, a system is force fit into an existing design when it could have been engineered into a more elegant solution.

Below you will find a description of a system which has taken this approach. The Clementine spacecraft as well as the ROMPS GAS experiment have developed operational concepts around the SCL system.

SCL System 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. These database objects include command actuators for commanding the spacecraft systems.
- The development environment is a window based application that 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 Executive (RTE). Extensive use of pull down menus and dialogs are used to control the system.
- The RTE is the portable multi-tasking command interpreter and inference engine. This segment represents the core of the flight software. This portion of the software is available in both C and Ada to allow ease of porting to a specific hardware platform (ground or space).
- The Telemetry Reduction program is responsible for filtering acquired data, storing significant changes in the database, and presenting the changing data to the Inference Engine. Limit checking and engineering unit conversion can be enabled on a point by point basis.
- The project is the collection of SCL scripts and rules that make up the knowledge base. On 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. 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 of the system.

The Clementine Experience

The Clementine management approach was to have a team of engineers to work on the project from cradle to grave. There would be no transition from one team to another. The Clementine team was a talented group of young, motivated engineers. The team had experience on other satellite programs, but was young enough not to be jaded by many of the large DoD and NASA programs. The team made numerous personal sacrifices for an opportunity to shake up the satellite community.

Clementine Command And Telemetry Software

The Clementine system software introduced several new concepts to spacecraft command and telemetry processing. These concepts supported the rapid development of the Clementine flight software. Most of these innovations are generic in nature and can be applied to other spacecraft. The following paragraphs will briefly describe Clementine's command and telemetry software and will highlight some of the innovative aspects of the software.

Clementine Command Processing

The command processing software performs four functions: (1) synchronize and reassemble incoming command data words into command packets; (2) verify and authenticate the command packets; (3) dispatch complete command packets to destination tasks; (4) execute command processing control functions. Clementine commands and data are formatted as packets with a header that includes a word count, a routing code, and a secondary identifier. The command processing software receives these packets as a stream of 16 bit words. The command software reassembles a packet from the incoming data words and after verifying and authenticating the packet, passes it to the operating system through a function call. The operating system software delivers command packets to queues that are assigned to software tasks. This arrangement

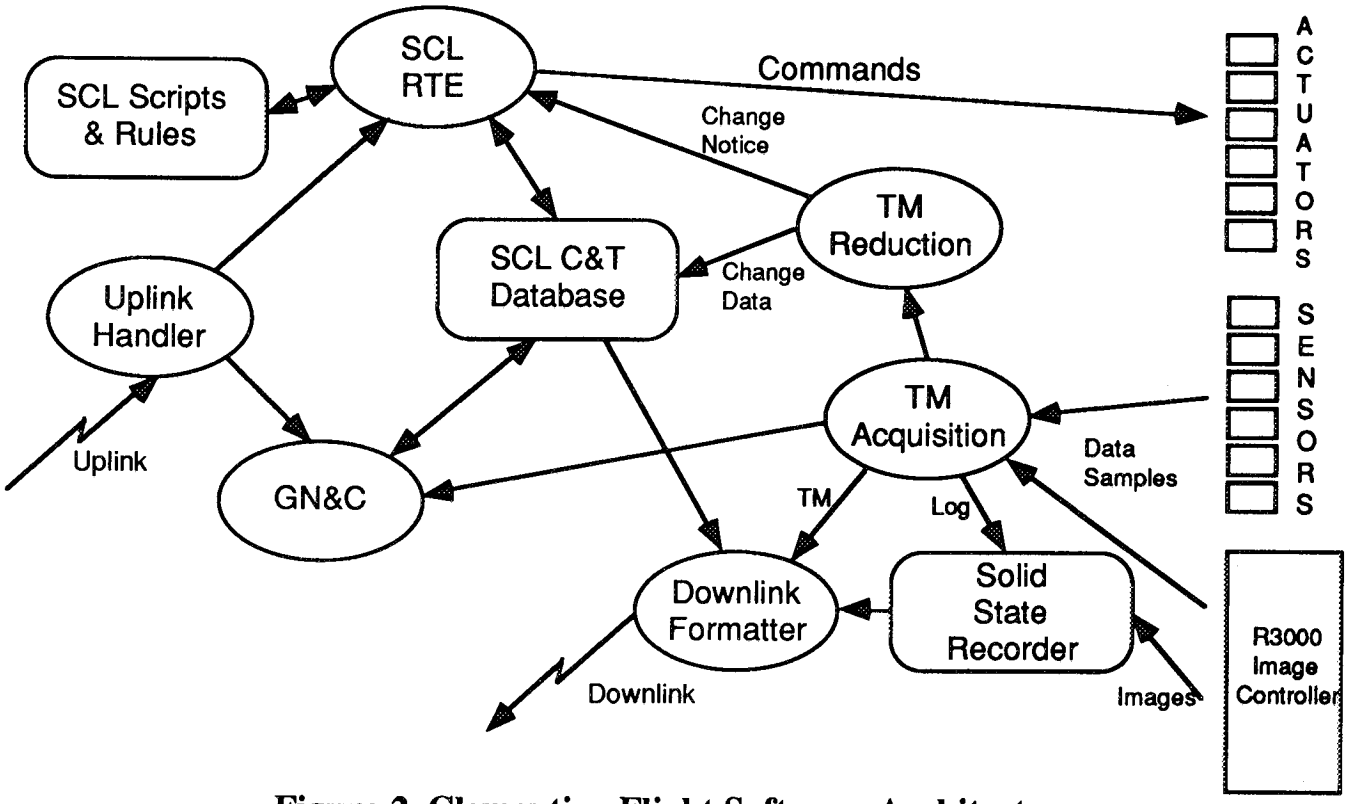

Figure 2. Clementine Flight Software Architecture

distributes the responsibility for command execution among the various Clementine software tasks. Because command execution is distributed to other tasks the Clementine command processing software is simply an input task which forwards messages to other tasks.

The packetized command uplink simplified the design of the command processing task and supported the rapid development and integration of Clementine's entire flight software system because:

1. Only the packet header is fixed, the remainder of the packet is defined by the receiving task. This allowed software designers the freedom to specify command formats that were suited to their requirements.
2. It simplifies the integration of software modules, such as the SCL Real Time Engine, that were reused from other programs. SCL software relied on command formats and interfaces that were defined long before the Clementine program was initiated.
3. New software tasks are added to the system without impacting the command processing software. A new task only has to create a command queue and then register to receive command packets through the queue. This simplified the incremental build-up of the flight software.
4. Command packets can be rerouted by changing the routing code or by altering the operating system's routing tables. This capability was used operationally to support some of the processor failure modes.
5. New commands can be defined without impacting the command processing software. This supported the incremental build-up of the flight software.

Clementine Telemetry Processing

The Clementine telemetry processing software performs four major functions: telemetry acquisition, telemetry reduction, telemetry distribution, and telemetry logging. All four functions are implemented by a single software task. The telemetry task operates in one of two modes: bypass or DHU. When in the bypass mode, the telemetry task is responsible for formatting telemetry data into telemetry frames in addition to the four functions listed above. When

it is in the DHU mode the telemetry task transmits its telemetry data to the Data Handling Unit (DHU) which then is responsible for formatting the data into telemetry frames. The process that is responsible for formatting the telemetry frames is responsible for transferring the frame data to the downlink hardware interface.

All of Clementine's telemetry, except for images dumped from the Solid State Data Recorder (SSDR), is organized into packets. Clementine telemetry frames are not filled with commutated data, as is the general case for spacecraft. Instead, Clementine's telemetry frames serve as a transport mechanism for the telemetry packets. The telemetry packets are used to transport the spacecraft's housekeeping, status, and memory data.

Commutated telemetry frames do provide a consistent source of data where housekeeping measurements and status items such as temperatures, voltages and relay indicators are concerned. To fill this need for consistent engineering data, the Clementine telemetry system provides a packet that contains synchronous, commutated data. The content of the Housekeeping telemetry packet, or HK packet, is defined by an uplinkable commutation format. The commutation format specifies the order in which the various spacecraft engineering sensors are sampled. As many as four commutation formats can stored in the spacecraft memory and any one of the four formats can be active. The Clementine commutation format supports up to 16 variable length minor frames per master frame and subcommutated telemetry items up to 16 deep.

Telemetry Acquisition

The telemetry acquisition function is responsible for processing the commutation formats. The acquisition function processes the format information, building up a minor frame in a temporary buffer. When the minor frame is complete, the acquisition function formats the frame into a single HK packet and then transfers the HK packet to the distribution function. The acquisition function can be commanded to switch to another of the four possible formats at any time and it will begin processing the new format after the current frame is complete.

The acquisition function is also responsible for acquiring packetized telemetry from other software tasks. An example of such a packet is the Attitude packet produced by the Attitude Determination and Control (ADAC) task. This packet contains information that defines the spacecraft's current attitude along with rate and status information. The acquisition function acquires these packets through message queues which it creates and manages. Two queues are created: the critical queue and the normal queue. The critical queue is for status information that is vital to the operation of the spacecraft such as the current vehicle command count, telemetry processing state and the spacecraft time. The normal queue is for all other telemetry packets.

Telemetry Reduction

The telemetry reduction function is responsible for maintaining the current telemetry value database that is provided to support the Spacecraft Command Language interpreter and inference engine. The reduction function receives identified telemetry values from the acquisition function and processes the values before updating the current value database with the telemetry values. Two of the processing options performed by the reduction function are change detection and engineering unit conversion.

The telemetry reduction function also allowed packets of data to be decommutated on-board to allow the SCL script and rules to have visibility into on-board data samples. This was a leap forward from traditional spacecraft software designs.

Telemetry Distribution

The distribution function is responsible for prioritizing and distributing the telemetry packets to the downlink. Packets from the Critical queue are assigned the highest priority and are distributed ahead of all other TM data. The HK telemetry packets are next in priority and packets from the Normal queue are assigned the lowest priority. If the distribution function is operating in the DHU mode, the function transfers the packets in priority order to the DHU through a dual port RAM buffer. The DHU is responsible for inserting the individual packets into the telemetry frame when operating in the DHU

```
-------------------------------------------------------------------------
-- SEP_DETECT -- Rule to detect separation from Titan II second stage.
--    Schedules separation operations script...this is the
--    initial sequence of events for Clementine
-------------------------------------------------------------------------
rule          sep_detect
subsystem     dspse
category      launch
priority      30
activation    yes
continuous    yes
    if LVSEPIN1 = SEPARATD and LVSEPIN2 = SEPARATD then
        deactivate sep_detect     --make this rule dormant
        -- establish attitude, take Star Tracker cal. shots
        execute LEO_Sep_OPS in 1 second
    end if
end    sep_detect
-------------------------------------------------------------------------
-- LOWVDET -- detect low voltage & schedule the safing script
-------------------------------------------------------------------------
rule          lowVdet
subsystem     eps
category      batteries
priority      4
activation    yes
continuous    yes
    if rawvalue of BATTPMON <= 360 then
        deactivate lowVdef     --make this rule dormant
        execute LEOsafing in 1 tick
    end if
end lowVdet
-------------------------------------------------------------------------
-- LEOsafing -- script which safes the spacecraft
-------------------------------------------------------------------------
script LEOsafing
    set DHUSELNO                  -- Take no pics
    execute ReactWheelsOff        -- RWs off
    set GNC11_ALLSTOP             -- stop all S/C rotations
    set SWCRITE2                  -- Image Processor Off
    execute IMUstop               -- IMUs off
    execute TrackersOff           -- STs off
    execute ACSDisable            -- Turn off ACS
    execute CamsOff               -- Cameras off
    -- check if star tracker doors are open...if so close
    if STARAOPN = 1 and STARBOPN = 1 then   -- Close both doors together
        execute ActBothDoors
        execute ACSDisable in 180 seconds
    else
        if STARAOPN = 1 then      -- Close A Only
            execute ACTSTA
        end if
        if STARBOPN = 1 then      -- Close B Only
            execute ACTSTB
        end if
        execute ACSDisable in 180 seconds
    end if
    wait 1 second
    set SWCRITEE              -- Transmitter Off
end LEOsafing
```

Figure 3. Example of Clementine Scripts & Rules

mode. If the distribution function is operating in the Bypass mode it is responsible for inserting the individual packets into the telemetry frame.

Telemetry Logging

The telemetry logging function is responsible for storing a time history of selected telemetry items in a log file on board the spacecraft. The purpose of the log file is to provide a means of capturing and storing telemetry data on board the spacecraft during periods when the spacecraft is unable to communicate with its ground stations. The log can be dumped by ground command or stored command. When the log is dumped, the log records are formatted into telemetry packets and transferred to the telemetry distribution function.

The log file can reside in either the HKP processor's RAM or on the SSDR. The log can be maintained in either stop on full format or in a circular format where new telemetry values overwrite the oldest values once the log becomes full. Telemetry items are selected for logging by ground command or by stored command.

The log file is maintained in a change only format, that is, the telemetry items that are selected for logging are first processed by the telemetry reduction function to determine whether the value of the item has changed since the last time it was acquired. If the item did not change it is not stored in the log. If the item did change a record containing a time stamp, the item's identifier, and the item's new value is written to the log.

The logging function is designed to initialize the log with the current value of all items that are selected for logging when the log is created or whenever it is reinitialized. This feature establishes a baseline for the change only values that will subsequently be written to the log.

Telemetry Processing Summary

The Clementine telemetry software introduced several new ideas to spacecraft telemetry systems. These innovations made significant contributions to the rapid development and integration of the Clementine flight software and contributed to the efficient operation of the spacecraft. The innovations include:

1. A packetized telemetry downlink which provides for synchronous, commutated data acquisition.
2. The capability to store multiple telemetry commutation formats on board.
3. The ability to load new HK packet commutation formats from the uplink.
4. An on-board telemetry storage log that is filled with change only telemetry.
5. An on-board telemetry reduction process and current telemetry value database.

Lessons Learned

The SCL system started life as a prototype system which supported only rule-based processing. It became obvious that it would be cumbersome to apply a strictly rule-based system to spacecraft command and control. ICS added the scripting capability to SCL to support procedural, time-based commanding scenarios. The scripting capability was integrated with the rule-based capability so that the system shared a common syntax and command interpreter. The SCL scripting capability is analogous to the Command Storage Memory (CSM) on earlier spacecraft. The SCL scripts and rules share a common Hyperscripting grammar. The system was developed in a manner to allow a core set of directives to be supported, and allow the user to extend the grammar with a mission unique set of directives. The SCL compiler used in the ground development environment allows addition of keywords, and the Real-Time Engine (RTE) can be extended to support the new features at run-time.

The Clementine flight software team was made up of several companies which worked together (around the clock at times) to develop an integrated system. The companies that developed the flight software also developed the ground station software together. This allowed interfaces to be defined more easily and consistently. The relatively small team worked to our advantage since all the players knew each other by name and could interact and make decisions quickly. There were very few managers to interfere with the decision making process. The NRL management "rode herd" over the engineers and coordinated the efforts. The team was able to work together without corporate or political fences.

The engineers that performed the systems engineering also developed the ground and flight systems and flew the spacecraft during mission operations. The cradle to grave philosophy allowed for a consistent interface between engineers. Day to day interaction between companies maximizes progress in the fast-paced development environment. Not having to transition the program from one group to another resulted in a substantial time and cost savings. The engineers who were intimate with the subsystem designs were responsible for the day-to-day tasking of the satellite. This allowed for experts to be available virtually anytime a problem arose.

The development and integration of the software was compressed into a short period of time. If a development testbed for the flight software had been available months sooner, a greater level of testing could have been accomplished. As it was, we had to schedule time at two sites: the testbed, and the flight article. It wasn't unusual to have around the clock and weekend testing, especially towards the end of the schedule. Competition for the testbed was at the point that one company would jump on while another pulled back to correct a software bug. Hardware bugs which took the system down were devastating. Software simulators for testing the Attitude Determination and Control system were refined throughout the life of the program. These simulations, along with the Guidance Navigation and Control (GNC) flight software evolved throughout the mission. New code was uploaded to the spacecraft to handle the current phase of the mission. Code which handled earth orbit was obsolete for handling lunar orbits, etc. This made for an incremental development, test, and operate cycle for the GNC code. This cycle worked out quite well because of the high fidelity of the simulators which were produced.

With the fury of software development activity on a day to day basis, configuration management was a monumental task. At times, 3 shifts a day were modifying and testing code. The ground software was evolving as quickly as the flight software. Two testbeds needed to be kept in vnc. Two accounts were maintained on the tbed minicomputers. One was the operational ɔunt, and the other was the development ɔunt. Each shift was informed as to which ɔunt was the current operational account and what modifications had been made. Coordination and attention to detail was mandatory.

The tight schedule and late availability of the flight unit and testbed caused a compression of the test schedule. Because the Clementine sponsors were willing to accept some risk, hardware and software testing was limited. The orbital mechanics of the asteroid encounter required that the spacecraft be launched at a given date otherwise the opportunity would be missed. The level of fault tolerant design for both hardware and software are some of the tradeoffs which had to be made. A single string system was acceptable. The software was designed to make up for some of the hardware shortcomings.

Once operational, the team faced new problems. The one glaring problem was burnout of the players. During the first week of operation, many of the team slept on conference room tables, in chairs, and on the floor (in the winter in Alexandria, Virginia). Several people were told to go home or to their hotels to get some sleep. The dedication of the team members was exemplary. Many hadn't seen their families in many weeks, even then it was only for few days.

On-board operations proved to be tricky at times. Many members of the team had developed satellites in the past, but had not dealt with the fact that they must monitor and command the vehicle around the clock. The team tried to keep a two day cushion on the mission tasking. This would allow for light duty on weekends and allow for problems to be investigated without the threat of a missed pass looming over their head.

The set of software tools were limited. No mission planning system existed. Many of the passes followed a standard script: configure the payload, slew the spacecraft, collect images to the solid state recorder, turn off the payload, wait until the earth is in view, slew the spacecraft and dump the solid state recorder. The commands for these scenarios were entered into Microsoft Excel with formulas for the times. The orbit analysts would determine the start time of the scenario and a time could be filled in on the spreadsheet and all other times calculated as an offset from the start time. The command sequences were moved over to the microVAX computers and translated into an upload sequence by the SCL software.

The configuration management of the spacecraft flight software and mission tasking sequences

was limited to a CM tool and a log book. The spacecraft would occasionally have a system reset which would require that all code patches be uploaded and a new set of tasking sequences be uploaded. This had to be managed by hand and was subject to human errors. It would have made a great deal of sense to have a software tool to automate the management of the flight software and mission tasking.

The lesson that Clementine points out is that code re-use is a key factor in the success of a fast track program. Having a flexible architecture which can be applied from one program to another allows for substantial time and cost savings. The Clementine flight software and ground system software relied heavily on software which had been developed for other Naval Center for Space Technology programs. These systems along with other software originally developed for NASA were merged along with some new concepts to develop a flexible command and telemetry system which itself can be used on other programs.

Conclusions

The SCL system has proven that a re-usable system can be successfully used for spacecraft command and control. All of Clementine's requirements were met with the exception of the asteroid flyby. The SCL system also helps promote a standard interface for the many facets of ground and space. The ROMPS Space Shuttle GAS experiment is due to fly in September of 1994. The ROMPS flight will lend further evidence that a multi-mission control system can be deployed. Although ARD and ROMPS are dissimilar missions, they are using the same hardware and software design. There is already talk of another flight for the ROMPS experiment.

The Clementine flight control software is general purpose in nature and can easily be adapted to other programs. The SCL system itself is being commercially marketed for workstations and embedded systems. The SCL system also has applicability to industrial control systems, Intelligent Vehicle Highway System (IVHS), medical, petrochemical and power system industries.

JPL, the AIAA and the Air Force have used the SCL system as a basis for standardization efforts. ICS is presently on several technology steering committees for DoD and Civil space. The SCL system sets a benchmark against which other systems will be compared. The flexibility of the architecture and the open systems aspect of the SCL software gives the system broad appeal.

The system does introduce information management problems that are overcome by software tools and a disciplined approach to configuration management. This approach must also extend to the distribution of flight and ground databases and knowledge bases. The system is several years into its development, has been the subject of numerous proofs-of-concept, and is in use at several sites. The SCL system provides a low-cost, low-risk solution for many of today's command and control environments.

The success of the Clementine mission lies with the dedication of the team members and their talent in the development of a spacecraft in a surprisingly short period of time. They were able to draw on experience and re-use software and hardware designs from other programs to develop a system which is flexible and pushes the state of the art in software technology for spacecraft.