Distributed Expert Systems for Ground and Space Applications

By: Brian Buckley, Interface & Control Systems

Louis Wheatcraft, Barrios Technology, Inc.

Abstract

The workstation, minicomputer, and microcomputer marketplaces have been revolutionized in the past decade by systems that are both open and distributed. As a leader in this revolution, the Naval Research Laboratory's (NRL) Naval Center for Space Technology (NCST), has been employing reusable software components to build a series of test beds, test and checkout systems for satellite assembly line operations, and distributed control of satellite tracking stations. The Navy has taken this one step further by unifying ground and space operations with the development of the Spacecraft Command Language (SCL).

SCL is a hybrid software environment borrowing from expert system technology, fifth generation language development, and multitasking operating system environments. SCL was developed by the Navy to be the controlling software for their Advanced Systems Controller (ASC). The ASC is a MIL-STD-1750A based Telemetry, Tracking, and Control (TT&C) controller for a new generation of Navy spacecraft having the capability of autonomous operation for up to 180 days.

Today's spacecraft are becoming increasingly more complex, with added sensors, higher data rates, and more capable standalone and distributed processors. The SCL system allows on-board processing of data, which has traditionally been considered to be in the realm of the ground segment. The distribution of processing to the space segment allows the spacecraft controller to analyze data points on-board and make decisions based on knowledge stored in the SCL scripts and rules.

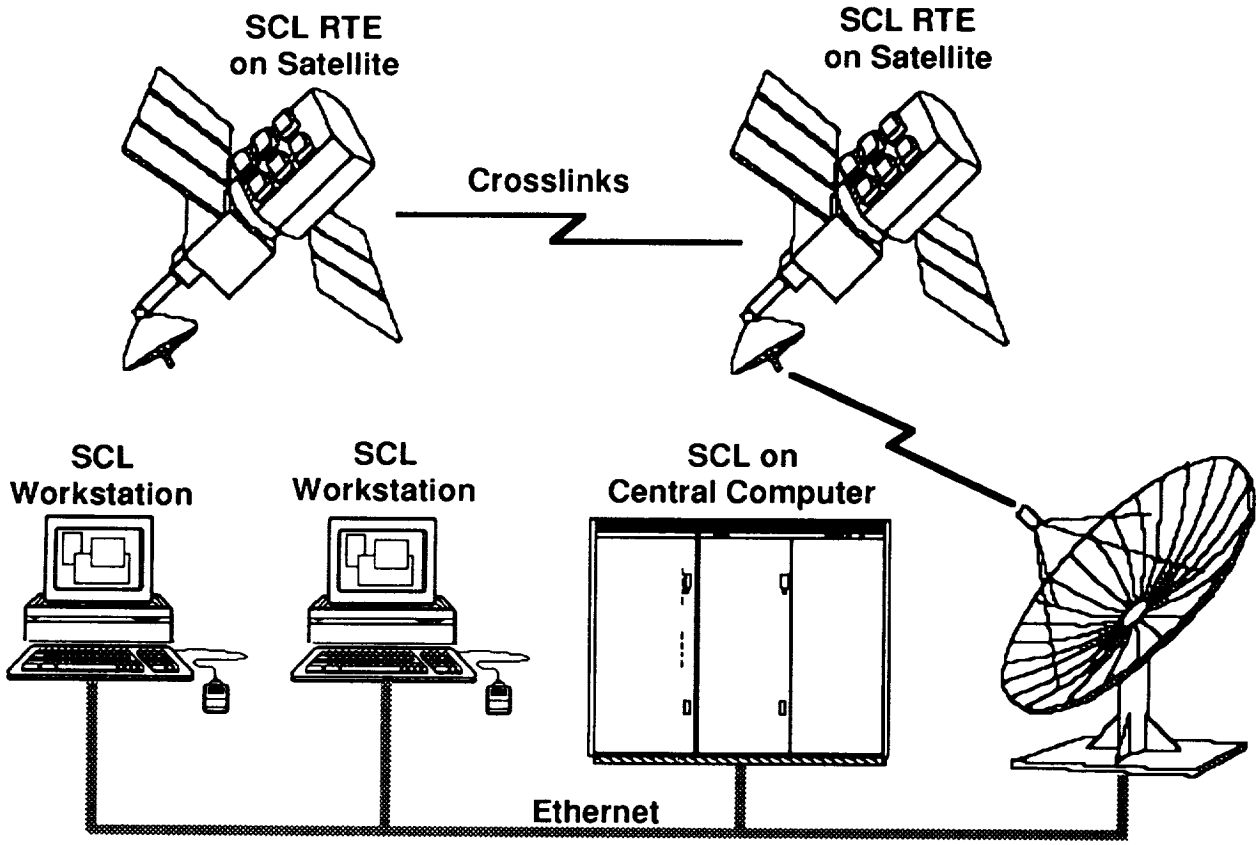
In addition, the spacecraft bus and payload systems are commonly developed independently, each having their own processors/controllers. Using a common distributed control language results in significant savings in total software development. The space-based SCL system can support distributed environments using a hierarchical scheme allowing subsystem controllers to communicate with a central controller.

A distributed approach is also used with the ground segment. Data points downlinked from the spacecraft are routed to workstations that analyze and view spacecraft and artificial telemetry points in real time. The workstation's knowledge base is used to analyze the telemetry and adjust the spacecraft's high level tasking to maintain the mission profile.

To unify the space and ground segments, the NCST has chosen the SCL system as the standard for use on-board the spacecraft as well as in the ground stations. The SCL system will run on a central ground station computer as well as on individual workstations used for subsystem monitoring and control. The SCL Real-Time Executive (RTE) on-board the spacecraft will be monitoring health and welfare, processing telemetry, scheduling mission tasking, and managing payload configuration changes.

Connectivity between multiple SCL nodes is not limited to exchanges of database items. A workstation can directly connect to any remote version of the SCL RTE. This allows direct control, interactive commanding, and real-time query of the remote SCL RTEs. This direct connect capability includes the version of the SCL RTE on-board the spacecraft.

This paper presents the SCL concept of the unification of ground and space operations using a distributed approach, describes the SCL system, offers examples of potential uses for the system,



Ground and Space Network using SCL

and details current distributed applications of SCL.

Introduction

The Naval Center for Space Technology (NCST) is in the process of developing the Advanced Systems Controller (ASC), which is a major upgrade to its current microprocessor based spacecraft controllers. The ASC hardware is based around the Honeywell GVSC MIL-STD-1750A processor and has been designed to be general purpose to allow tailoring of the system to meet the requirements of other spacecraft programs. The ASC software is based on the Spacecraft Command Language (SCL) Real-Time Executive (RTE). SCL is a hybrid system that employs a rule based event driven expert system as well as a procedural scripting capability.

The SCL development environment consists of a ground based windowed system used to develop SCL scripts and rules. The integrated environment consists of an editor, a compiler, decompiler, tracing subsystem, explanation subsystem, and the RTE. The SCL 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. The NCST 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 before requiring the spacecraft hardware for final checkout.

The NCST has incorporated a control system in its ground stations for the past decade. This control system has proliferated to integration and test environments, and to many of the supporting ground stations around the world. The SCL system has been integrated with this control system to provide additional, or "value-added" capabilities to the existing systems. Besides the goal of early deployment and checkout of the SCL software, the NCST felt that the existing ground stations could benefit from the expert systems capabilities provided by the SCL system.

The existing NCST satellites are well characterized and are managed by several software components and Orbital Operations Handbooks (OOH), which define configurations, constraints, and contingency plans. This knowledge can easily be translated into SCL's scripting language. The resulting knowledge base consists of SCL scripts, rules, and functions. The knowledge base is used in real time to monitor and detect changes in the vehicle configuration, maintain the configuration, move efficiently from one configuration to another, monitor system health, and perform command verification. At a ground station, copies of the SCL system are used on workstations to analyze telemetry and drive third party graphics products. The SCL workstations are able to advise an operator of anomalous conditions and suggest corrective measures, compare the current configuration against the desired mission tasking profile, and provide a capability to autonomously maintain vehicle configuration.

A New Way of Doing Business

In the past, only a ground-based command and telemetry database needed to be managed. With the advent of the ASC concept, the on-orbit SCL database must also be considered. The field sites throughout the world must also have knowledge of the database items that are on board, as well as the scripts and rules that are loaded on the ASC. All ground stations must have knowledge of the orbiting satellite's database.

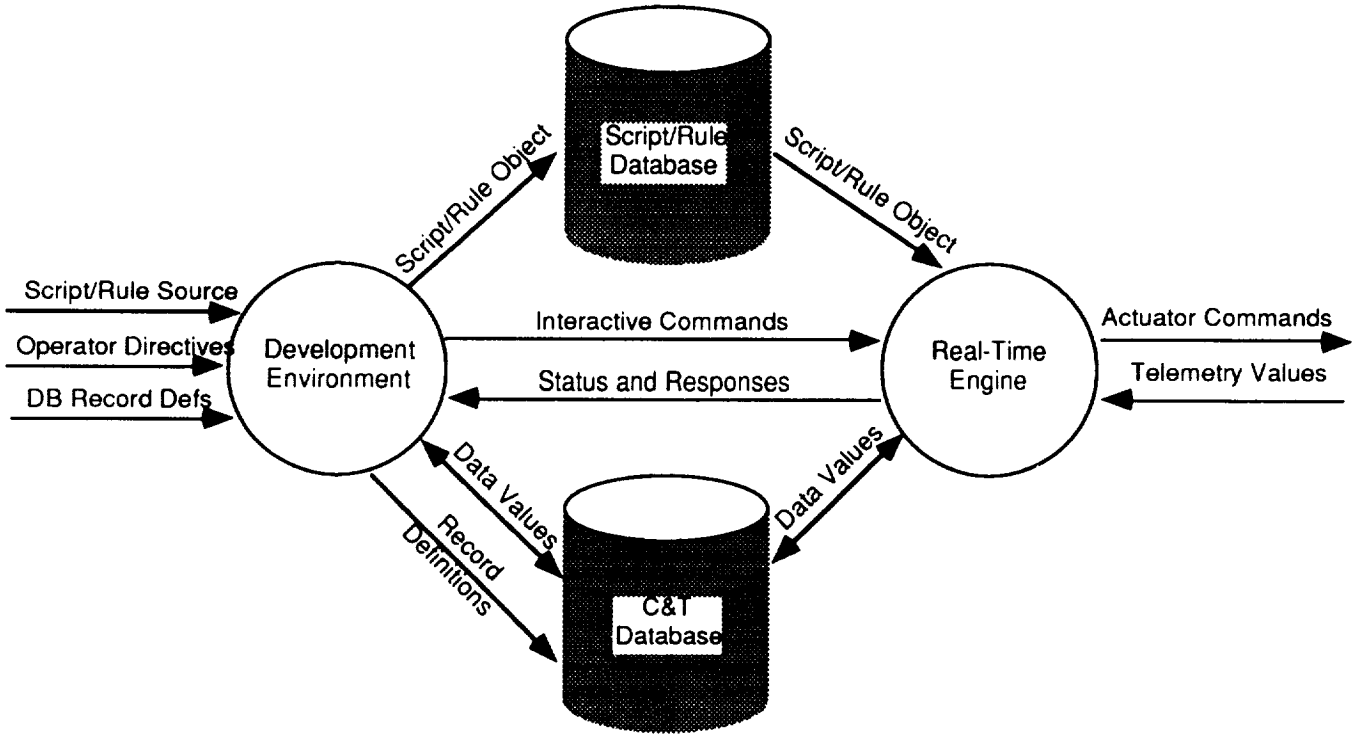
The current generation of spacecraft has a control system used for ground operations, an embedded control system for the spacecraft controller, and hard-coded algorithms for specialized hardware. Rather than use several different sets of software, the NCST approach is to use the same SCL software for spacecraft control functions as well as for ground station control. The SCL system is portable and has been designed to be used in embedded systems as well as workstations and minicomputers. This approach allows a common SCL grammar to be used for the ground station, the spacecraft controller, and payload controllers.

A major departure from the past and present spacecraft control systems, is the concept of using an existing, validated software "shell" for control system development. In the past, it was felt that each new spacecraft system needed a unique on-board software controller. Thus a special team of specialized programmers would develop a new control system from scratch. This proved to be a high risk as well as a very expensive approach both in terms of cost and schedule. With the SCL concept, the only unique software is the low level hardware interface code, the database, and the knowledge base. This approach has several advantages:

- The development cycle can be shortened; much of the code is "off the shelf".
- Risk is reduced. The SCL system has been proven both on ground and on space processors.
- The knowledge of a system is embedded in the controller.
- The learning curve for ground station operations is reduced since the knowledge is captured on the system.
- Consistent operations. Tasking of the vehicle is performed the same for both the ground and space segment.

The scripting contained in the knowledge base is written using a high level language that can be easily learned and understood by the subsystem engineers, thus not requiring a team of specialized programmers.

Mission tasking has traditionally been based on a time-line. At a given point in time the spacecraft is commanded to perform a function. Commanding can be carried out via stored commands, and by interactive commanding from the ground station. The time-line approach has proven to be cumbersome and difficult to administer. Previously, commanding was done "blind"; no database was available on the spacecraft for interrogation. With the availability



SCL System Dataflow

of SCL's scripting capabilities and flight GPS receivers and other equivalent devices, an on-board expert system that is performing real-time monitoring of the current spacecraft position can allow field of view tasking to be implemented. The spacecraft can collect data when an area of interest is in the field of view, and it can dump stored data when a field site is within its field of view. This approach can greatly simplify the mission tasking definition and the reduce the need for as many men in the loop.

The NCST envisioned using the SCL system on-board the spacecraft to share the testing burden, since it would be able to detect and isolate faults and report them to the ground. Time saved during the integration and test phase of the program can result in significant monetary savings. The capability to do self-diagnosis is desirable since a low earth orbiting spacecraft (LEOS) is in view of a ground station for only a small percentage of its orbit.

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. Data structures required to support the Inference Engine are also stored in the database. These items 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. 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.
- 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. 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.

Fielding the System

Due to the power and the advantages provided by SCL, the NCST decided to put the SCL system into the field immediately for several reasons:

- Risk Reduction - prove the system is viable through a series of proof of concept efforts.
- Capture knowledge of key personnel to allow the system to aid integration and test efforts.
- Allow parallel development of knowledge bases on workstations.
- Develop a concept for adaptive mission tasking, and field of view commanding.
- Allow development of simulations for Air Force and Navy Projects.
- Integrate SCL with existing systems to allow value-added features.

The SCL system was put through its paces early in its development cycle in a series of proof of concept efforts. The Lab Test Bed proof of concept used the SCL RTE on a UNIX platform to control a prototype satellite. To demonstrate the enhanced capabilities of SCL, the following demonstrations were performed:

- SCL Satellite Configuration and Auto reconfiguration. The goal of this demonstration was to show SCL is capable of:
  - Transitioning from one control language to another. This was demonstrated by translating existing control system command procedures into SCL scripts.
  - Commanding the spacecraft and receiving telemetry responses.
  - Detecting that the spacecraft is not in a desired configuration and notifying the operator.
  - Automatically reconfiguring the spacecraft to a safe state in event of an error.
- SCL Commanding. The goal of this demonstration was to show several SCL capabilities:
  - Simple commanding and verification by monitoring telemetry points to verify that the command was successful.
  - Mission constraint checking by verifying telemetry prior to a command being issued to prohibit a potentially damaging command from being sent.
  - Abstract commanding by using SCL's high level commanding capabilities. Scripts are used to check telemetry and manage primary and redundant sides of boxes, and allow a default side to be active.
- Fault Tolerant Configuration. The goal of this demonstration was to show the SCL capability to react in real time to telemetry changes and implement an alternate course of action if conditions warrant. This demonstration used redundant sides when the primary side did not respond.

```
constraint    HOT_SWITCH
subsystem     TRANSMITTER
category      SWITCHING
priority      15
activation    YES

    if
        (BIU_CROSS or BIU_NORM)
    and
        XMIT_POWER = ON
    then
        reject
        execute fault_log with constraint_err
    end if
end    HOT_SWITCH
```

SCL Constraint Example

- Field of View Operation. The spacecraft position coordinates are telemetry database items and may have rules associated with them. When a significant change in the position data occurs, the rules associated with them are executed. The rate that the position is updated is determined by the desired ground track accuracy. The field of view can be calculated from the position coordinates and compared to the area of interest. If the area of interest is within the field of view, the rule may execute scripts or sequences of commands to change the spacecraft configuration.
- Mission Tasking. The goal of this demonstration was to highlight three tasking aspects of SCL. First, SCL is required, at a minimum, to reproduce the current capabilities to schedule and activate various configurations. Second, SCL is capable of a variety of common cyclic functions that can be scheduled within the SCL kernel. Third, SCL is capable of multitasking. This multitasking capability will reduce the time, effort, and complexity of resolving the varied resource needs between program entities. By supporting multitasking, SCL can satisfy requirements from many different sources.
- Self-Testing. The NCST has designed the ASC with a goal of improving testability. By using the ASC processor as an asset for testing, parallel testing can be accomplished. Having an intelligent controller allows the system to perform self-diagnosis, trouble shooting and reporting of problems. SCL enhances testing by providing a flexible and re-usable means of implementing on-board testing. At the subsystem level, scripts and rules can be written to provide a test environment for a specific subsystem. SCL can send commands to the unit, and react to the telemetry responses from the unit. This would provide a common test method for many layers of the system, allowing consistent testing throughout the phases of system integration. At this level, SCL supports command and telemetry verification for each box.

```
-- This script prepares the Reaction Control
-- Subsystem for a thruster firing and calls a
-- subroutine script to perform the actual
-- firing. It accepts two parameters: one
-- indicating which thruster to use and
-- another specifying the duration of the firing

const  fore 1     -- define a constant for the

                  -- forward thruster

script Maneuver1     thruster, duration

    -- command to enable thrusting
    set RCS_ENABLE to ON
    -- allow propellant flow
    set TANK_ISO_VALVE to OPEN

    if
        thruster = fore
    then
        -- call Fore thruster subroutine

        execute ManvFore with duration

    else
        -- call Aft thruster subroutine
        execute ManvAft with duration

    end if

    -- command to disable thrusting
    set RCS_ENABLE to OFF
    -- command to isolate tank
    set TANK_ISO_VALVE to Closed

end    Maneuver1
```

SCL Script Example - Mission Tasking

- Simulation. Frequently, in the production of the spacecraft, there are subsystem components that have not been integrated or are missing due to troubleshooting or modification. In their absence, test procedures have to be modified or must be postponed until the component is available. Having a simulator for a missing component is desirable so test procedures can be run without modification and testing can proceed without the complete system in place. In the event a particular box is absent from the spacecraft, its presence could be simulated by the SCL inference engine, either operating in its embedded form on-board the spacecraft, or in its ground system form operating on the ground station processor. To simulate a missing component, a knowledge base must be developed to respond to commands defined for the component. When a command is sent to the component, the associated rule is executed and a corresponding telemetry response is generated.

```
rule        BATT3_TEMP
subsystem   EPS
category    BATT3
priority    15
activation  YES

   if
      BATT3IT > 50
   then
      set alarmlevel of BATT3IT to RED
      execute battsafing with 3, priority = 30
   end if
end   BATT3_TEMP
```

SCL Rule Example

Further Proving of the System

In another proof of concept, the NCST wanted to test the expert system technology in a "real-world" scenario. This proof of concept required that the SCL system be compared to a commercial off the shelf (COTS) expert system. Both SCL and the COTS system were required to be capable of being used in embedded systems (i.e., blown in PROM). The expert systems were to be used to implement the flight algorithms for NRL's upper stage used for orbital insertion of satellites. The upper stage is spin stabilized until it reaches the insertion orbit. Once in the desired orbit, the upper stage is spun down and stabilized using momentum wheels and reaction control thrusters. The upper stage then jettisons the spacecraft allowing it to move into its parking orbit.

All aspects of the orbital transfer maneuver are controlled by the Attitude Control Electronics (ACE). The ACE subsystem is semi-autonomous and can issue thruster commands to maintain the desired attitude. The ACE control loops were developed in the flight processor's native assembly language. The development of the algorithms required years of design, testing and elaborate simulation. Within 3 months, two prototypes were generated using SCL and the COTS expert system. The two prototypes were exercised using the same flight qualification test used for acceptance testing of the original ACE flight software.

The results of this effort proved that the COTS expert system was NOT able to keep pace with the flight control loops, resulting in additional thruster burns to stabilize the spacecraft. The knowledge base for the COTS expert system was re-designed, but was still unable to keep up with the control loops. The SCL system however, performed the ACE algorithms as efficiently as the flight software.

Based upon the successful demonstrations of the SCL system, the NCST baselined the SCL software as the control system for the Advanced Satellite Controller. The system has also been chosen as the control system for two NASA projects, one of which will launch in September of 1993.

Satellite Simulations

In the summer of 1990, the NCST was chosen to provide spacecraft simulations for the Space Defense Initiative Office (SDIO) Standard Mobile Segment program. The NCST chose to use the SCL system to provide command response capabilities and electrical and thermal modeling for the FLEETSATCOM and GPS

satellites. An SCL rulebase was developed to decode binary commands and insert an appropriate response in the telemetry bit stream. Telemetry from this bit stream was distributed over Ethernet to Air Force contractor workstations. The SCL system was integrated with the NCST's TT&C system to allow real-time simulation of command responses. The electrical and thermal models were also developed as part of the SCL knowledge base. These models provided a detailed emulation of the spacecraft in real-time or up to 600 times real-time.

The SCL simulations were developed on workstations and delivered on the host computer as a stand-alone entity. The system was activated from the TT&C system, and runs with little or no operator intervention. The only intervention required is when an operator wishes to generate anomalies in a scenario. These simulations have been delivered to Air Force contractors, to the National Test Bed, and to the NAVSOC facility at Pt. Mugu, California. The NAVSOC personnel currently use the simulators for FLEETSATCOM training. In the near future, the SCL software will be embedded in a high-fidelity hardware simulation for a NCST program.

Integration with Existing Systems

Currently, the SCL system is completing its integration with the NCST's ground station software where it will become part of a client-server model. The SCL system will reside on several workstations as well as the ground station's central computer. All communications will be through a packetized message passing protocol over Ethernet. The ground station TT&C software is responsible for telemetry decommutation and distribution. The SCL workstations will monitor appropriate telemetry to generate operator advisories and drive graphics interfaces, which are used to indicate the spacecraft configuration, health, and welfare.

The NCST tracking stations are taking advantage of the distributed aspects of SCL using a network of minicomputers and workstations. Once the NCST's ASC is launched, the full potential of SCL can be exploited. In addition to the ground-based network, the spaceborne ASC platforms will be capable of communicating with each other. Ground stations will be able to perform the central site load to one ASC. The ASC that is loaded will be capable of forwarding the applicable script, rule, and database loads to the other ASC's. Since the ASC's can be in constant communication with each other, the mission tasking load can be balanced among the cluster of ASC's. This concept of adaptive tasking will be managed by the on-board SCL expert systems. Each SCL knowledge base will know the configuration of the on-board ASC, and can query the other ASC's to obtain their current configuration and tasking profile. Having this capability will create a network of ground and spaceborne SCL platforms. With the ability to upload databases and knowledge bases, the possibilities for this network are tremendous.

Reusable Controllers

Recently, SCL was chosen for a commercialization of space contract funded by NASA Goddard Space Flight Center. The Autonomous Rendezvous and Docking (ARD) satellites will use SCL to control docking and fluid transfer experiments. The ARD satellites will use off-the-shelf spacecraft computers based on the 80186 chipset. The ARD satellites will be low earth orbit satellites. The ground stations will use SCL to monitor telemetry and send commands. The two satellites will both be controlled by an embedded version of SCL and will communicate with each other during the docking procedure through RF modems. When the satellites are within a kilometer of each other, one satellite will act as master, and the other as slave. The SCL knowledge base on one platform will be sending commands to control the maneuvers of the other.

The same off-the-self spacecraft controller will be used for a material processing experiment to be launched as a NASA Get Away Special (GAS) on-board the Space Shuttle. This captive experiment will use SCL to control an oven and a robot that will be used to place material samples in an oven to test the effects of annealing in a weightless environment.

Lessons Learned

Development of a distributed expert system for ground and space did not come without its share of technical and psychological obstacles. The

following paragraphs give an overview of some of our challenges.

Portability: The SCL system was originally developed on a Macintosh II platform. The Macintosh proved to be a highly productive environment because of its integrated toolkit for windowing, the operating system, and the filing system. The software development tools available on the Macintosh were the most affordable and most sophisticated at the time. We made great strides in the development of the systems, but we were faced with the chore of porting the system to other platforms. The NCST tracking station uses the Digital Equipment Corporation (DEC) VAX family of computers and workstations running the VMS operating system. The SCL code was originally written in C, and we had to convert the real-time engine and database loader to ANSI compatible C. We also needed to support UNIX platforms and IBM PC platforms. This required adding conditional compilation statements for some of the include files since paths are different.

To keep the core software identical on all platforms, the operating system specifics and the I/O have been abstracted to a very small number of routines, which are replaced on each system. These routines interface directly to the host operating system to schedule execution, map memory sections, and obtain systems time. The low level I/O and network I/O is also handled in this group of routines. This abstraction of I/O has allowed the system to be easily ported to multiple hardware platforms, operating systems, and real-time executives for embedded systems.

Another major obstacle was communication between local and remote versions of SCL on a non-homogeneous network. Different machines were either big-endian or little-endian (high byte then low byte in memory, or vice-versa); they also use different floating point formats. The low-level I/O modules were modified to determine the "sex" of the local and remote SCL systems and perform any data transformations necessary. The network I/O has been sufficiently abstracted to allow communication via TCP/IP and DECnet protocols over Ethernet, Appletalk, serial communications such as RS-232, and custom protocols.

We felt it was prudent to allow the spaceborne, embedded version of SCL to perform native access to all data structures including the database, and the knowledge base. To allow the embedded SCL to have native access, the ground based development environment had to support a cross-compilation of all data destined for a remote version of SCL. By setting a software switch, the data streams, and files produced, are formatted in the target processor's native data structures. The development environment is also capable of decompiling the data streams from the target platform.

Necessity vs. "Feature-itis": As the system evolved, new features were added as needs and requirements dictated. At one point we found ourselves adding features because we thought it would make the system "slick". As we found out, new features and new code created side effects that were not discovered without extensive testing. We also found that many of the "slick" features were difficult to duplicate on other platforms, since they did not have an integrated toolkit like the Macintosh. We were striving to maintain a common look and feel for the product across platforms. Because of the porting of the code to multiple platforms, the system was baselined (frozen) and only changed for maintenance and bug fixes.

The SCL proofs-of-concept resulted in another company providing an objective analysis of our product. Our system was compared with commercial products to test functionality as well as real-time performance. As a result of the comparisons, we added several extensions to the grammar, and an additional user-selectable, inferencing strategy. Other behavioral quirks were also corrected. We did not however, add more object oriented features due to the real-time considerations. The SCL system is designed for real-time embedded environments and pre-allocates all data structures prior to startup. The SCL RTE does not perform any dynamic memory allocation due to memory fragmentation issues. Traversing the data structures necessary to implement additional object-oriented features would degrade the real-time performance and increase the memory requirements for the system.

Information Management: For past programs, the spacecraft controller used a low-level command-language and did not support an on-orbit database. The ground station and test systems were the users of a database. With the introduction of the ASC (with SCL on-board), the ground sites as well as the spacecraft must

contain copies of the database. Additionally, a core set of scripts and rules must also be managed. In the past, it has been difficult to ensure that all mission sites contain a database which describes the same command and telemetry points as the central site. The prime contractor is now responsible for delivering and configuration managing databases for each site and platform. Distribution of the databases are from a master database at a central site. All other sites' databases are derived as a subset of the central site's database. The spacecraft data points are the same from site to site, but the databases at each site can be extended to include ground specific data points.

Currently the SCL development environment compiles scripts, rules and database records and assigns ID's to each. Scripts, rules, and database records are referenced by these ID's. To keep all sites synchronized, a given ID must correspond to the same script, rule or data point at each site. The situation is further complicated when the user references data points or scripts on another platform or network node. Several options are available to guarantee that an ID is unique among all nodes and platforms in the network. The strongest contender as a solution is to use a hash algorithm to define the node and script/rule/database item combination. This scheme results in a 64-bit ID for each object and doubles the current size of the ID in the SCL intermediate code. In addition to the extra data word requirement, several table lookups are required to efficiently look up the address for the data structures.

Another area of concern was how to distribute the calculation of the artificial data points (derived items). It was felt that the prudent approach was to divide the derived item calculations between ground and space. Rules are used to calculate derived items and are defined on the appropriate platform. Spaceborne derived items might be used in calculations for attitude control, where derived items would be used on the ground to drive graphics displays. The ground derived items are further distributed to workstations that analyze telemetry for specific subsystems.

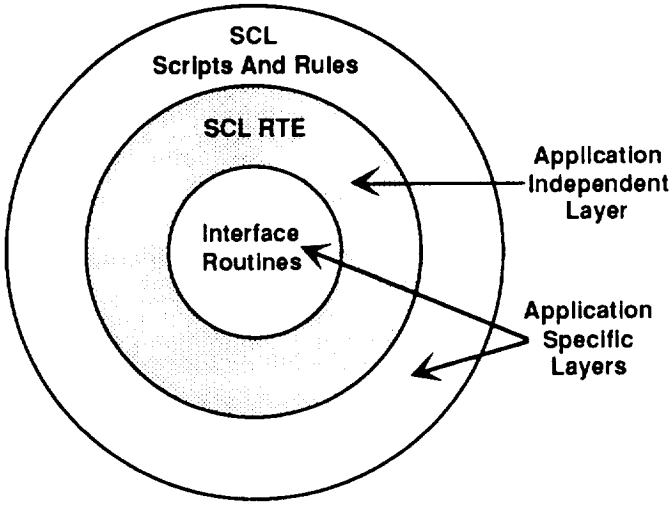
Windowing & Reusable Code: As prototypes of the SCL system were ported to other platforms, the amount of code was increasing rapidly. This problem was complicated by the fact that the windowing systems on each platform (Macintosh, OSF/Motif, Microsoft Windows) behaved quite differently from a programming viewpoint. We estimated that a man-year would be required to bring a programmer up to speed on each windowing system to generate a production quality application. We also saw that many software functions were being replicated even within the same programming team. It was at this point we decided to port the SCL development environment over to C++ to promote code reusability and abstraction from the windowing system specifics.

We saw four layers of class libraries that needed to be defined. The foundation of the system is the portable filing system layer and the database management layer. The filing system layer implements a class library modeled after the Macintosh resource files. The filing system is based upon a four character ASCII key that is used as an index. Each key (or resource) can contain variable length data structures identified by a unique name and index. This filing system is used to maintain a consistent interface across all platforms.

The data management layer is based largely upon the public domain National Institutes of Health (NIH) class library. This class library manages commonly used data structures: lists, sorted lists, data blocks, strings, collections of objects, etc. This layer is a simplified subset of the functions contained in the NIH class library. Data persistence is implemented to allow data structures to be maintained in memory once they have been read from disk. This has significant performance advantages since the disk is only accessed again when the file is closed, or the programmer explicitly requests that the data be written back to disk.

Abstraction: The SCL system was envisioned from its inception to be applicable to other types of controllers and other satellite programs. To accomplish this, the system had to abstract the specifics of the application from the knowledge engineer. The grammar for SCL is a hyper-scripting language that supports object-oriented features. The object-oriented approach allows the Real-Time Executive to treat Actuators, Sensors, and Derived Items essentially the same way. All decisions as to how to perform the I/O is deferred to the lowest level interface routines. These routines are the "glue" between the logical and physical interfaces. This approach allows a few

hundred lines of code to perform any transformation required by the hardware interface. This approach allows identical scripts to run both on workstations and the flight processor. Both implement the same functionality, but one could communicate over Ethernet, while the other communicated with a TT&C bus, or an I/O card in the same chassis.



SCL Software Architecture

Abstraction is also the key to keeping the vast majority of the SCL code portable. All operating system specifics are isolated from the code in just a few routines. These routines are replaced on the target machine with calls to systems services specific to that operating system.

Fear Of Artificial Intelligence/Expert Systems: Perhaps the largest hurdle to overcome is an inherent fear or apprehension of managers that they do not want to lose control of their spacecraft to a computer. They simply don't want to accept the perceived risk, and are more comfortable with the old or existing methods. To overcome some of the initial negative reactions, we have had to avoid the terms artificial intelligence and expert systems. Instead we use the term "Smart Control System". There is probably a better term. The main points that must be made are:

- Existing methods of ground up development are just re-inventing the wheel and are more risky because of the use of "new" and unproven code. They are also more costly because of the time to develop a controller from scratch and because of the increased schedule time.
- Rules are already embedded (hard coded) into existing software. But because they are coded by specialized programmers, it is difficult for the subsystem engineers to review and understand how their systems are being monitored and controlled.
- As stated previously, the SCL system employes the concept of using an existing validated software "shell" for control system development. With the SCL concept, the only unique software is the low level hardware interface code, the database, and the knowledge base. The scripting contained in the knowledge base is written using a high level language that can be easily learned and understood by the subsystem engineers, thus not requiring a team of specialized programmers. The rules that exist in the traditional controllers are now structured as individual items that are evaluated by the inference engine. This structure makes it easy for the subsystem engineers to review and understand the how their subsystem is being monitored and controlled.
- The amount of control given to the SCL system can vary depending on the needs and the configuration of the system. SCL can be used to duplicate an existing system's capabilities, perform data pre-processing, self-test, or autonomous control. The amount of control given to the system can be determined by the project office.

Expert systems have been recognized as being applicable to ground and space applications. However, association with Artificial Intelligence continues to project negative connotations for some people; the relationship must often be avoided.

Other Uses for SCL

The use of abstraction and object-oriented techniques allows the SCL Real-Time Executive to be applied in a variety of areas:

Spacecraft controllers: The RTE is available in both C and Ada making it applicable to a wide variety of processors.

Subsystems Controllers: The SCL system is designed for distributed environments and as such allows for a hierarchal bus structure for subsystem controllers to report to a system controller.

Centralized ground station computers: The system can be used in conjunction with X-Terminals to manage ground station resources such as antennas, frame syncs, command encoders, etc. The system can also be used for scheduling ground station resources.

Workstations: The SCL system is ideally suited for use on workstations to allow distributed processing and parallel analysis. The system is also useful for driving graphics and visualization tools. Results from local workstations can be reported back to a central computer, or commands may be uplinked to change or correct the mission profile. SCL has been demonstrated to be capable of analyzing the spacecraft configuration and providing advisories for operators and spacecraft engineers.

Integration and Test: The SCL system is used at Integration and Test (I&T) facilities to perform automated command procedures. The procedures developed at the I&T facility can then easily migrate to the tracking facility. If SCL is also used for the spacecraft, the scripts and rules can also migrate to the spacecraft. The on-board processing capabilities of SCL allow a spacecraft to perform self-health diagnostics and report its status to the ground.

Test Equipment: Quite often, mission unique hardware must be developed for test equipment. The equipment often requires a control system and a language to allow it to be commanded to perform its functions. SCL can easily be embedded in the target hardware to provide a standard interface for all phases of testing.

Simulation: SCL has been proven to be capable of providing command response simulations as well as detailed modeling of systems. The simulations can be used to augment missing subsystems during spacecraft integration as well as provide a common platform for system training of operators and other personnel.

Conclusions

The SCL system is quite feasible for use on distributed systems for ground and space. The SCL system also helps promote a standard interface for the many facets of ground and space. The system does introduce information management problems that are overcome by a disciplined approach to configuration management. This disciplined approach must also extend to the distribution of databases and knowledge bases. The system is several years into its development, has had 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.

Acknowledgments

We would like to thank the following people for their contributions to this paper: Dave Schriftman, and Patrick Pinchera.

References:

1. Buckley, Brian and Wheatcraft, Louis: *Spacecraft Attitude Control using a Smart Control System*, SOAR Symposium, Houston TX., July 1991

2. Buckley, Brian and Wheatcraft, Louis: *Spacecraft Command Language - A Smart Control System*, Interface and Control Systems, Melbourne, Fl., Barrios Technology, Houston, TX., March 1991

3. Van Gaasbeck, James: *Technical Overview of the Spacecraft Command Language* Naval Research Laboratory, Washington D.C., 1991

4. Interface and Control Systems: *SCL User's Guide*, Naval Research Laboratory, Washington D.C., 1990

5. Buckley, Brian and Wheatcraft, Louis: *Rapid Prototyping of a Spacecraft Controller*, JAIPCC Symposium, Houston TX., March 1991

6. Buckley, Brian and Wheatcraft, Louis: *Spacecraft Simulations with a re-usable Smart Control System*, JAIPCC Symposium, Houston TX., March 1991