Multi-Mission Sequencing Software

Laura Needels
Jet Propulsion Laboratory/California Institute of Technology
Pasadena, California

For Space Operations 2002 Conference, Houston, Texas
October 2002

Sequencing software for deep space missions has historically been one of the most critical parts of the ground software used to communicate with and control the spacecraft. At JPL, the sequencing software is responsible for planning and creation of science and engineering activities, checking command syntax, checking mission and flight rules, and translating the commands into packets which can be uplinked to the spacecraft. Significant effort has been spent by earlier missions to ensure the integrity of this software since errors in this area could cause the spacecraft to enter fault protection or cause the loss of the spacecraft.

Over the last several years, in an effort to reduce the costs associated with sequencing software, a multi-mission form of sequencing software has been developed. The sequencing software is now developed as two separate components. The multi-mission "core" software provides, in a generic sense, the capability to perform the functions needed in the sequencing software: planning and scheduling events, checking flight rules, and packetizing commands. The "core" software is then "adapted" to a project specific mission. The "adaptation" part of the software task involves providing the models for activities needed for planning and scheduling, converting the Command List for the project into models that can be used for sequence checking, coding project and mission flight rules and the modeling needed to support them, and developing project blocks for repetitive activities.

By partitioning the sequencing software into a "core" component and an "adaptation" component, JPL has been able to reduce costs. The core component of the software is verified once, by a central group, rather than by multiple projects. Projects can then focus on verification of just the adaptation part of the software. An additional cost saving with this methodology has been that it has allowed JPL to staff a multi-mission adaptation team where personnel can move from one project to another with minimal start-up and training time.

This paper describes the architecture of the JPL multi-mission sequencing system and demonstrates many of the benefits of the core/adaptation format of the sequencing software.

Introduction

At JPL, the system used to develop the sequences to develop the commands sent to the JPL interplanetary spacecraft consists of several different stages, sequence planning, sequence verification, and sequence translation into spacecraft understandable binary. The software used to support these activities is known as SEQ, and is shown in Figure 1.

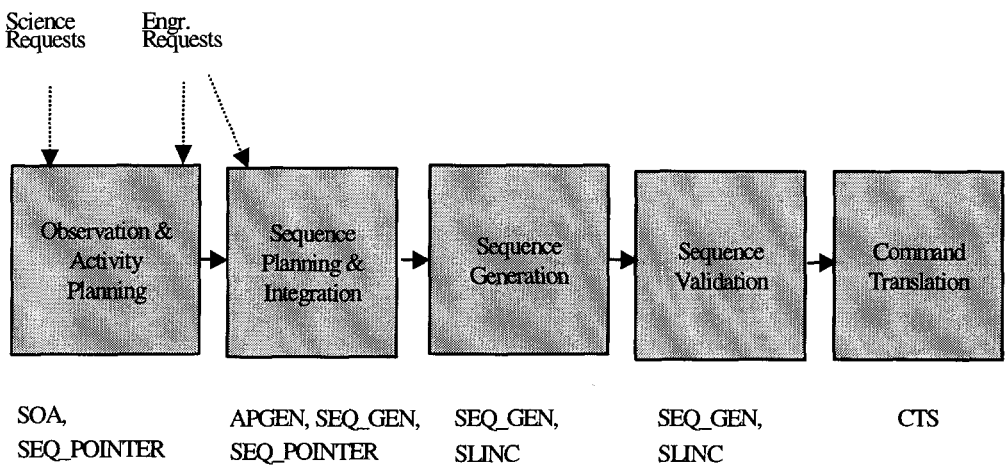

Figure 1: SEQ Components and Programs

The first part of this effort is the sequence planning. The initial stages of the sequence planning consist of planning engineering activities, planning science activities, checking constraints to make sure that science activities don't violate geometric constraints, and checking resource usage.

Two pieces of SEQ software that can be used for science activity planning are Science Opportunity Analyzer (SOA) and Planetary Observation Instrument Targeting and Encounter Reconnaissance (SEQ_POINTER). SOA has a very user-friendly interface that allows scientists and other planners to exercise trade studies and preliminary design for science observations. Search engines that will locate times of interest such as fly-bys, bow-shocks, occultations, angles, periapsis and apparent diameter are available. Once these opportunities have been located, the user can design a specific observation (eg. Continuous Scan, Roll Scan, Start Stop Mosaic, Stare). Critieria such as the primary target, secondary target, target offsets, duration, primary observer, and secondary observer are entered. Constraint checking for activity duration, distance, exclusion zones, and hardware limits is also completed by SOA.

Once the preliminary design has completed, the observations are passed to POINTER, where detailed calculations and constraint checking are completed. There are usually two differences between POINTER and SOA. The first is that SOA works at the activity level, but POINTER works at the command level. The second is that POINTER usually has a more complex turn model associated with its calculations than SOA.

The second type of planning tool that is used to assist with activity planning is Activity Plan Generator (ApGen). ApGen allows the user, more often a spacecraft engineer, rather than a scientist, to plan activities while monitoring resource constraints. Desired activities might be Deep Space Network (DSN) contacts, science activities, and general engineering activities. The resources monitored might be solid state recorder space, propellant, and battery state of charge. The model fidelity of both the activities and the resources increases during mission development. For example, in the beginning of mission development, science activities might be assumed to use an assumed level of power and output an assumed level of data on the solid state recorder. As the mission design becomes more exact, different science activities may be modeled with different power and data volume needs. Information about science activities might even come from SOA or POINTER. ApGen also contains a scheduler which will allow a user to define activities which are fixed in time, additional desired activities, and the scheduler while place activities on the timeline so that resources are not violated. The scheduler will also show which of the desired activities could not scheduled. The user can then reprioritize activities (only two

Type A activities and three Type B activities) or add additional activities which would alleviate resource constraints (add an additional DSN contact so that space on the data recorder becomes available).

Once a sequence has been generated and integrated, the checking is done by SeqGen. SeqGen checks command syntax, mission rules, and flight rules. The sequence that SeqGen checks might have been developed by ApGen or any number of other tools. Command syntax, such as the number of input parameters, parameter types, etc. is done by SeqGen. SeqGen's primary use is for mission and flight rule checking. Historically, the mission and flight rules checked by SeqGen were more timing related, such as "The catbed heaters must be turned on 5 minutes before the thrusters are fired," or "Once the spacecraft has been launched, the mission phase may never be set to pre-launch,". SeqGen has also been used for checking commands against spacecraft states, such as "The command may be used only while the spacecraft is nadir pointed," or "The command may not be issued while the spacecraft is in eclipse,". However, more recently, SeqGen has also been used to check for resource constraints at the command level. SeqGen creates a spacecraft sequence file that contains the commands that will be sent to the spacecraft.

The last step in the process is translating the commands into spacecraft readable binaries and converting the sequences into packets. Spacecraft Language Interpreter and Collector (SLINC) is used for these two steps. The only type of error checking that is done here is if a parameter is out of range, it will not be translated, and SLINC will stop processing.

Multi-mission Architecture

In an effort to reduce project costs and development time, one area that began to develop was the idea of reuse of sequencing software. The time and money involved in generating this software was significant, and so software architectures that allowed for software reuse began to develop. SEQ was partitioned into two parts, Core and Adaptation. SEQ Core software has the ability to perform a specific function. SEQ Adaptation provides project specific components that rely on core functions.

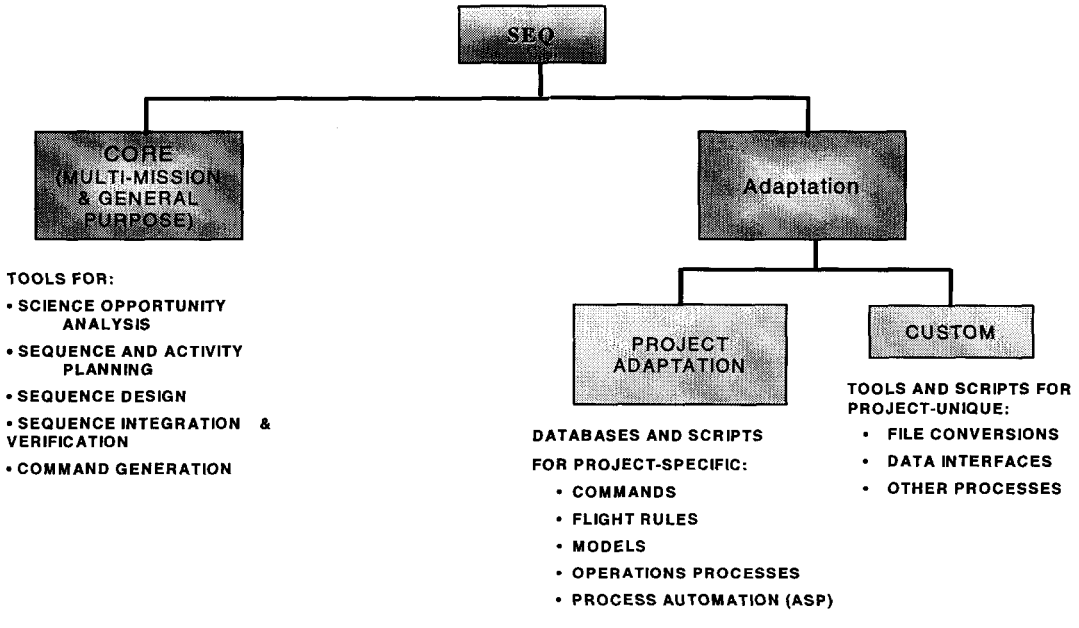

Figure 2: Architectural View of SEQ

An analogy for this sort of division can be seen in tool used for word processing. In SEQ terminology, the word processing software is piece of Core Software. The word processing software allows the user the ability to pick various font types (Times, Ariel, Courier, and Symbol) and font representations (bold, italic, and underlined) and paragraph alignments (left justified, right justified, and center justified). And it is the user (the Adapter) who writes the text, chooses the styles and decides the purpose of the document (a letter or a memo).

Examples of Core functionality and Adaptation tasks for the SEQ software tools are listed below.

SOA Core Functionality

1. SOA shall read a configuration file containing user specified file names and default values for SOA initialization.
2. SOA shall have the Parameter and Model data from the User Interface Component for interprocess communications.

SOA Adaptation Tasks

1. Define the configuration file to contain the correct kernel files, rule files, previously loaded queries, search engine information, model files, default times, default bodies, etc.
2. Define the mapping between the parameter and model variables used in SOA and other programs SOA communicates with.

POINTER Core Functionality

1. Provide the capability to display error and warning messages directly to the uses and/or the event listing hardcopy.
2. The time formats written to a sequence file shall be Greenwich Mean Time (GMT), epoch plus GMT, absolute clock time, epoch plus relative clock time.
3. During adaptation, read the Spacecraft Activity Type File for definitions of the SEQ_POINTER-applicable activities.
4. The operator shall be able to override the default workstation desktop color assignments and fill patterns by editing the Application Resource File.
5. Perform range checking on each parameter value.

POINTER Adaptation Tasks

1. Implement the error and warning messages, and the models needed to trigger them.
2. Define a set of useful epoch times.
3. Provide a Spacecraft Activity Type File that contains definitions for the activities that will be used during modeling.
4. Provide a project adapted Application Resource File if the default configuration is not acceptable.
5. Provide ranges for parameter values.

ApGen Core Functionality

1. ApGen shall be delivered complete with an installation program.
2. Activity types shall have a mechanism to specify activity resource usage.

ApGen Adaptation Tasks

1. Provide an architecture which supports ApGen usage and the environment variables and files needed by the program.
2. Provide the resource usage for each of the steps of a given activity. Define the resources the will be monitored.

SeqGen Core Functionality

1. SeqGen shall display an indication of each rule violation, either in the window of the subsystem involved in the rule, or in the timeline near a request causing a the rule violation.
2. Any textual display of time shall be either in UTC, or JPL local time or user local time including automatic conversion to daylight savings time, or the flight projects Spacecraft Clock units or epoch relative at the option of the user.
3. There shall be a command to allow the user to add a phrase of up to 1000 characters into the runlog. SeqGen shall append the time of day to the phrase.
4. SeqGen shall convert "activities" into "steps".

SeqGen Adaptation Tasks

1. Coding of the rules that are checked by SeqGen.
2. Define a useful set of epoch times.
3. Generate useful messages that will be included in the runlog. These messages will often include information about rule violations, comments about the beginning or ending of a block, or messages about parameters.
4. Define the steps needed to complete an activity

In the most widely used of these Core tools, SeqGen, the Adaptation effort has evolved so that it even maintains a version of multi-mission aspects of the Adaptation that are used by all projects. These multi-mission aspects are not functionality, as with a Core tool, but rather common pieces of Adaptation that every project can use. This is shown in Figure 3 below.

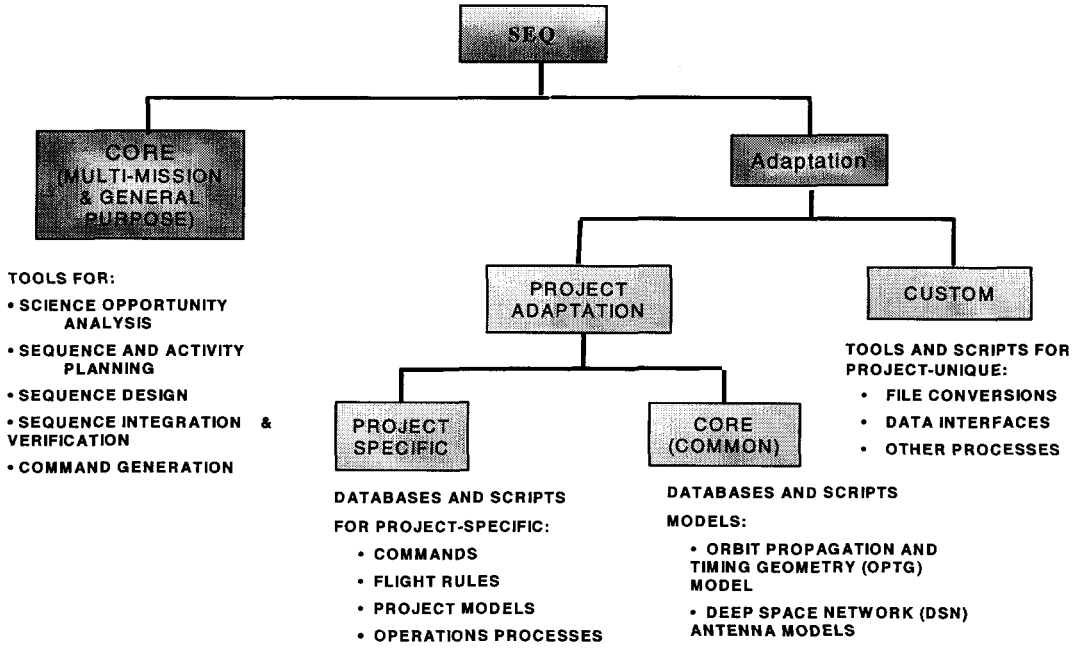

Figure 3: Expanded View of SEQ Architecture

Project Specific Components of Adaptation are described below:

- The model response to commands. This includes checking rules, changing model states, etc.
- Designation of commands that are hardware commands versus commands that are commands to be handled by the flight software. These are safety checks that make sure that certain commands are run only on the correct processor.

- Flight and mission rules implementation. These are included in the modeling to automate the prevention of undesirable spacecraft states.
- Implementation of project specific modeling for states or resources. For example, modeling of the Pointing and Control Subsystem may be needed to model timing of commands. Or, resources such as battery state of charge may need to be modeled at the command level rather than the activity level.
- Implementation of blocks (sometimes called activities) which are a canned series of commands that will be expanded for the user.

Common (Multi-mission) Components of Adaptation:

- Modeling needed to read configuration files which control parameter values on the spacecraft.
- Interpretation of Orbit Propagation and Timing Geometry (OPTG) files and models. These models take the keywords and data in an OPTG file and use it to model when eclipse, occultation, and other orbit and timing events occur.
- Descriptors for DSN view period files that contain information needed to model the range of antenna visibility.
- Modeling needed to support the use of DSN antennas and equipment, including generation of messages needed to generate keyword files used at the DSN.

Advantages and Disadvantages of this Architecture

There are several advantages and disadvantages of this architecture. The advantages of using the multi-mission Core/Adapted software are time and cost savings. The disadvantage of this architecture is the need to balance the needs of multiple customers.

Since the basis for the sequencing system already exists (the Core), simple commands (eg. a NOOP) can be passed through the sequencing system very quickly after the start of the work. Binary commands that will be sent to the spacecraft during Assembly, Launch, and Test Operations (ATLO) can be generated in less than one week. This implies that testing using the real sequencing system can begin almost immediately.

By having a Core piece of software, the correctness of the functionality of the software has already been verified. Projects need to verify only the implementation of the adaptation. In cases where projects have requested new capability, projects may be involved in validating the new capability.

Since the Core is used by many projects, there is a pool of available adapters who are familiar with the sequencing software. Adapters can readily transition from one project to another. It is also much easier for a project to get extra assistance during critical times because adapters are familiar with the sequencing software and its use.

However, since the Core supports many customers, considerable effort must be spent to balance differing project needs. Some customers may wish to have capabilities that conflict with other project needs. One example that occurred recently involved a new project that wanted to enter decimal values in a hex field and have them automatically translated to hex ("10" would be translated to "A"). However, existing projects were entering the values as hex values (a user entering "10" meant the decimal value "16"). This change could not be accommodated for the new user. Similarly, as sequencing strategies have evolved, limitations in previous capability or software design strategies have been exposed. Some of these problems must be preserved to accommodate legacy users. Finally, there have been times recently when it has been difficult to balance the delivery schedules of new users. When several projects have critical

events (launch, orbit insertion, etc.) in the same year, it has been difficult to meet the needs of the different ground data system integration schedules.

Conclusions

Over the last several years, the entire sequencing system has been converted to an architecture using Core and Project Adaptation software. Each part of the sequencing package (the science planning and scheduling programs, the resource modeling programs, and the constraint checking programs) has a Core program that provides the functionality needed and an accompanying Adaptation that customizes the Core to reflect project specific rules, models, and activities.

The change to a multi-mission architecture has resulted in cost and time savings to the project because the real sequencing system can be used almost immediately to generate commands for the spacecraft and because the functional aspects of the programs (the ability to expand activities into individual steps, the ability to display warning and error messages, and the ability to convert spacecraft clock time into UTC or JPL time) are tested once, rather than by each project. Use of a common set of sequencing software tools also benefits projects because a pool of Adapters who are familiar with the software exists. The initial ramp up time on a new project is very low and individuals available to alleviate critical schedules. The difficulties involved with this architecture have been balancing the needs of several customers.

Acknowledgements

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