

Advances in Discrete-Event Simulation for MSL Command Validation

Alexander Patrikalakis and Taifun O'Reilly

Jet Propulsion Laboratory

California Institute of Technology

Pasadena, California 91109-8099

Email: amcp@jpl.nasa.gov and taifun@jpl.nasa.gov

Abstract—In the last five years, the discrete event simulator, SEQUENCE GENERATOR (SEQGEN), developed at the Jet Propulsion Laboratory to plan deep-space missions, has greatly increased uplink operations capacity to deal with increasingly complicated missions. In this paper, we describe how the Mars Science Laboratory (MSL) project makes full use of an interpreted environment to simulate change in more than fifty thousand flight software parameters and conditional command sequences to predict the result of executing a conditional branch in a command sequence, and enable the ability to warn users whenever one or more simulated spacecraft states change in an unexpected manner. Using these new SEQGEN features, operators plan more activities in one sol than ever before.

Keywords—Spacecraft; operations; sequencing; activity planning; discrete event simulation; resource management.

I. INTRODUCTION

Deep-space missions are becoming more and more complex. While the Mars Exploration Rovers mission (MER) had approximately 305k source lines of code (SLOC) in its Flight Software (FSW) [1], MSL has upwards of 3 million SLOC [2], an order of magnitude greater. FSW's complexity is reflected in the spacecraft parameter state space: MSL has 52128 FSW parameters, half of which are modeled in SEQGEN. The science payload also increased by an order of magnitude, from 11 pounds on MER to 165 pounds on MSL. SEQGEN models the command durations of all science observations as a function of spacecraft states, FSW parameters, and command arguments. The operational constraints of deep-space missions are also becoming equally more complicated; MER has 62 ground constraints (also known as flight rules) checked by SEQGEN, while MSL has 303 flight rules checked by SEQGEN. Activity planning and sequencing on MSL are done in an event-driven fashion, requiring changes to SEQGEN [3].

The discrete event simulator, SEQGEN, is used to simulate spacecraft and their interactions with the Deep Space Network, to verify spacecraft commands and arguments, to protect spacecraft from faults, and to ensure that spacecraft operations meet primary mission goals. To support the complexity of the MSL mission, we had to evolve SEQGEN. In this paper we discuss how MSL uplink sequencing and modeling needs drove innovation in the emerging field of activity-planning software for remote robots. These innovations eventually allowed the SEQGEN simulated command sequence durations to approach 95% accuracy with respect to as-run times of MSL Mars Hand Lens Imager (MAHLI) sequences. Before we discuss the innovations, we first discuss some of the components

of spacecraft models (adaptations) that run on SEQGEN to provide readers with some background regarding SEQGEN and its use.

A. Spacecraft Model File (SMF)

A Spacecraft Model File (SMF) contains attributes, commands, stimuli, ground events, and subroutines. Attributes represent modeled spacecraft state, and commands are FSW commands that can change the attributes within some simulated time frame. Stimuli are queued requests to change simulated spacecraft and future ground states. For example, SEQGEN uses specially named stimuli to let the simulation specify what happens on blocking and non-blocking sequence invocation and termination. Ground events perform the same roles as commands, but are not real FSW commands; they are used to model the physical behavior of DSN and spacecraft interactions, and as back-doors to update spacecraft state in the presence of anomalies. A mission's SEQGEN adaptation uses a combination of mission-specific and multi-mission SMFs. Finally, subroutines are reusable groups of simulation logic. Commands, stimuli, ground events, and subroutines alike all update simulated spacecraft and ground states with a number of STEPS in their corresponding RESULTS bodies.

Fig.1 lists an example of a contrived Martian laser model. The model tracks the power state of the laser, and the number of times operators fire the laser. This particular model contains two commands, POWER_LASER and FIRE_LASER, that can be included in a sequence. In the RESULTS sections of these commands, the model updates the laser's power state and the number of fired shots. The key difference between the SEQGEN simulation language and other procedural languages is that time is an integral part of the modeling language. One can explicitly model delays and command durations as steps of a command's RESULTS.

B. Context Variable File (CVF)

A CVF contains constants, known as context variables, which can either be scalars or tables/matrices. Some of these variables have special meaning for SEQGEN; for example, specifying the minimum event duration or associating branching command sequence directives with if/else/end_if constructs. Some of the variables represent Deep Space Network (DSN) configuration tables that are called upon by the multi-mission DSN model files. The remaining variables are mission-specific and can be used to represent anything

```

SOFTWARE (LASER,
ATTRIBUTES,
SHOTS (TYPE, UNSIGNED_DECIMAL, DEFAULT, 0),
POWER_STATE (TYPE, STRING, RANGE,
  \ "ON", "OFF", "XN" \, DEFAULT,
  "OFF" ),
end,
COMMANDS,
FIRE_LASER (
  RESULTS,
  LASER::SHOTS = LASER::SHOTS + 1,
  COMPLETED, (00:00:01, 0),
  end
),
POWER_LASER (
  PARAMETERS,
  power_state (TYPE, STRING, RANGE,
    \ "ON", "OFF" \),
  end,
  RESULTS,
  IF, LASER::POWER_STATE != power_state,
  LASER::POWER_STATE = "XN",
  DELAY_BY, 00:00:01,
  LASER::POWER_STATE = power_state,
  COMPLETED, (00:00:00, 0),
  end,
  ELSE,
  E$ERROR="Laser already on!",
  COMPLETED, (00:00:01, 1),
  end,
  end
),
end)

```

Fig. 1. A contrived example of SEQGEN SMF file content.

```

/MAX_SHOTS
"const" 1000

```

Fig. 2. A contrived example of SEQGEN CVF file content.

from success/failure control codes to spacecraft operational mode restrictions on FSW commands. A mission's SEQGEN adaptation uses a combination of mission-specific and multi-mission CVF files. In a contrived example in Fig.2, we define a constant representing the maximum number of times the laser is allowed to be fired.

C. Flight Mission Rules File (FMRF)

A Flight Rules Mission File contains codified rules that are used to flag violations of constraints set by operators on the ground. FMRF flight rules can forbid illegal states (with `forbidden_synchronic`), disallow simulated state from being changed (with `lock`), and require a combination of simulated state to be true for a period of time before a state transition (with `required_antecedent`). SEQGEN versions prior to 32.4 only supported checking FMRF-based flight rules when affected model attributes changed and at the beginning of SEQGEN simulation runs. Generally, FMRFs are always mission-specific.

```

forbidden_synchronic (LASER-0001,
  SEVERITY, ERROR,
  MESSAGE, \ "The laser wears out." \,
  STATE, \ LASER::shots > MAX_SHOTS \
) ##end LASER-0001

required_antecedent (LASER-0002,
  SEVERITY, ERROR,
  MESSAGE, \ "Cool down before power on" \,
  DURATION, 02:00,
  NO_INTERIM_STATE,
  ANTECEDENTS, LASER::POWER_STATE == "OFF",
  RESULTANTS, LASER::POWER_STATE == "ON"
) ##end LASER-0002

```

Fig. 3. A contrived example of sample SEQGEN FMRF file content.

```

RT_on_board_block (lasr00001, \ lasr1 \,
STEPS,
  command_wait (1, POWER_LASER ("ON"),
    COMMENT,
    \ "Fire at will" \),
  command_wait (2, FIRE_LASER ()),
  command_wait (3, FIRE_LASER ()),
  command_wait (4, POWER_LASER ("OFF")),
end
) ##end ACTIVITY_TYPE lasr1

```

Fig. 4. A contrived example of SEQGEN SATF file content.

Fig.3 shows two contrived examples of FMRF flight rules. First, a `forbidden_synchronic` flight rule warns users about consumable depletion. In this example, the rule named LASER-0001 states that it is an ERROR for the laser to be shot more than MAX_SHOTS. Second, a `required_antecedent` rule enforces a cool-off period of two minutes between laser power cycles.

D. Sequence Activity Type File (SATF)

A SATF contains the on-board and ground command sequences called blocks, also known as activities. Ground blocks are used to model multi-mission DSN behavior such as Doppler ranging, View-Periods, and DSN ground station allocation. Ground blocks also model high-level communication windows in terms of lower-level spacecraft commands, and are mission-specific. On-board blocks are sequences that actually get executed on the spacecraft. SATF blocks are mission-specific, as spacecraft commands vary from mission to mission. The example listed in Fig.4 is an on-board block that turns on a contrived Martian laser, fires it twice, and turns it off.

E. SEQGEN adaptations and adapters

From the perspective of spacecraft operators, a SEQGEN adaptation refers to the sum of the SEQGEN program (the simulator); a user-defined function library that supplements SEQGEN's functionality (for example, with SPICE kernels [4]); a multi-mission model for the DSN, sequence engines, and orbital propagation timing and geometry; and a mission-specific model of spacecraft commands and state. The ex-

amples shown in Fig.1 - Fig.4 represent a subset of such a mission-specific model of spacecraft commands and state. Combined with other ancillary inputs such as light-time files, SCLKSCET files, DSN view-period and station allocation files, the above comprise the totality of a SEQGEN adaptation.

An adapter is a software developer that takes the SEQGEN program and multi-mission elements, adapts them to the modeling and verification needs of a specific mission, and creates the mission-specific models of spacecraft commands and state[5][6]. Adapters routinely create and modify SMF, FMRF, CVF, and SATF files. Less frequently, adapters will modify the user defined library.

II. SIMULATION IMPROVEMENTS

MSL, like its predecessor MER, uses a new version of Rover Markup Language (RML), instead of Virtual Machine Language (VML), to represent sequences in the MSEQ (MSLICE-Sequencing) database [7]. VML is an expressive language that permits all aspects of procedural programming: assignment, conditional branching, loops, local and global variables, and subroutines [8], allowing operators to sequence complex spacecraft activities. MSL chose to use RML over VML to leverage MER legacy and because RML is less complex. To support MSL uplink goals without VML, SEQGEN required updates to internal simulation features; these updates are described below.

A. Event-driven command sequencing

Until recently, SEQGEN did not support simulating command execution durations and the relative ordering of the commands in a sequence for determining the absolute start and end time of each command. As the status returned by a command's RESULTS is known only at the last simulated time step of the command, conditional sequencing is only possible when the results of a command have run to completion. As MSL requires the ability to conditionally terminate a sequence based on the last command's status (and other spacecraft states), we updated SEQGEN to support `command_wait` steps in sequences. `command_wait` steps instruct the sequence engine to wait until the command indicated has completed before dispatching the next command in a sequence. Event-driven sequences are exclusively composed of `command_wait` steps.

As the completion of command execution can take time, we made SEQGEN able to abort a command in the middle of its execution, terminating the command with a FAILURE status. Because commands are simulated to completion in event-driven sequencing, the return status of each command can be assigned to a block variable usually named `LAST_STATUS`. `LAST_STATUS` resembles one of more than two hundred DDIs (Defined Data Items, well-known spacecraft states) on MSL. Special commands in sequences refer to DDIs to conditionally execute more commands, depending on the value of the DDI. Now that SEQGEN can assign command return status to the `LAST_STATUS` block variable, it can be used to simulate the `LAST_STATUS` for each spacecraft sequence engine.

B. Conditional sequencing

In remote rover operations, operators need conditional branching in sequences because they cannot acquire knowledge of the rover's state in real time. Conditional branching allows operators to sequence tests for contingencies, and to kill sequences if an anomalous state exists. Thus, operators need to also be able to simulate multiple branches of conditional sequences.

MSL uses DDIs to enable sequences to query actual spacecraft state and to prematurely abort such sequences in case the queried spacecraft state was not nominal. MSL exposes spacecraft state at the sequence and command levels using DDIs. There are sequence directives, a special kind of command, that will conditionally execute the commands in the "if/else" branches of a conditional statement depending on whether the equality relation involving the DDI is true or false.

Using the MER Rover Sequencing and Visualization Program (RSVP), it was possible to use if/else_cond block steps in SATFs to explore the future [9]. Because RML used in the uplink pipeline on MSL does not translate to `if_cond` or `else_cond` steps in the SATF files produced by MSEQ Server, the operator preference had to be communicated to the SEQGEN simulation another way. Currently, MSL uses "magic comments" attached to the relevant IF/ELSE sequence directives in SATF to assume true, false, or nothing about the DDI (in)equality test. Magic comments are specially formatted COMMENT entries attached to a `command_wait` step in block in an SATF file. Magic comments are magical because they allow operators to make assumptions about spacecraft states accessible via DDIs.

Fig.5 lists the sequence block `lasr00002`. Command `IF_COND(DDI, relation, value)` will enter the conditional branch if the relation between DDI and value is true. Command `ENDIF_COND` marks the end of a conditional branch, and command `TERMINATE` immediately kills the current instance of a sequence. Magic comments are present on the steps with numbers 2 and 5. The first magic comment in step 2 forbids the simulations sequence engine from dispatching step 3. The magic comment in step 5 allows the simulation's sequence engine to dispatch the command in step 6. The outcome of actually executing `lasr00002` on the spacecraft will depend on whether the spacecraft successfully powered the laser ON in step 1, and on the local mean solar time at step 5. This "magical" function of magic comments requires adapters to parse the command's comment in the mission-specific part of the adaptation.

As contingency planning continues to be a part of rover sequence planning, we realized that conveying true/false branch preference is better dealt with by the SEQGEN simulator. We took the parsing burden off the adapters in the mission-specific model and added such a feature to SEQGEN, by introducing the `ASSUMED_MODEL_VALUES` tag to event steps in SATF. Then, all that adapters need to do is query the presence and value of properties in this tag using the new built-in function `GET_VALUE_FOR_KEY`.

C. Model attribute history persistence

Implementing flight rules that specify how long heaters are not allowed to be turned on before and after the rover

```

RT_on_board_block(lasr00002,\lasr2\,
VARIABLES,
  LAST_STATUS (TYPE, INTEGER),
end,
STEPS,
  command_wait(1, RETURN_ASSIGN_TO,
    LAST_STATUS,
    POWER_LASER("ON")),
  command_wait(2, COMMENT,
    \"[assume false]\",
    IF_COND ("LAST_STATUS"
      "NOT_EQUAL",
      0)),
  command_wait(3, RETURN_ASSIGN_TO,
    LAST_STATUS,
    TERMINATE (SUCCESS)),
  command_wait(4, ENDIF_COND()),
  command_wait(5, COMMENT,
    \"[assume true]\",
    IF_COND ("LOCAL_TIME",
      "GREATER_THAN"
      10:00:00)),
  command_wait(6, RETURN_ASSIGN_TO,
    LAST_STATUS,
    FIRE_LASER()),
  command_wait(7, ENDIF_COND()),
end
) ##end ACTIVITY_TYPE lasr2

```

Fig. 5. A contrived example of SATF content with conditional branching driven by magic comments.

goes to sleep is possible by WAITing on rover computer and heater power states in stimuli spawned by the commands that govern heaters and computer power state. However, proactively checking this rule is cumbersome due to the large number of heater zones (100+), the three types of heating modes (low level zone control, the immediate warm-up requests, and scheduled warm-up requests), and the at least six different ways to cause a computer power cycle. Furthermore, it is complicated by the fact that heaters can be scheduled to turn on even when both computers are off [10].

It is easier to model heater activity separately from rover computer power cycles and check heater usage against computer power cycles at the end of a simulation run. Now, SEQGEN can persist historical model attribute values throughout and across multiple simulation runs. Setting PERSIST_COUNT to values greater than one for an attribute will maintain just as many historical values of that model attribute. Functions GET_ATTRIBUTE_AT_TIME and GET_TIME_OF_ATTRIBUTE_TRANSITION allow adapters to query the past value of a model attribute, and to query transitions, allowing flight rules like the one above to be checked at the end of a simulation run, instead of during the simulation. Persisting histories across multiple SEQGEN simulation runs allows us to check this kind of rule during multi-sol plans, which require multiple runs.

```

SOFTWARE (utility,
SUBROUTINES,
pre_command(
  RESULTS,
  LOCAL, STRING [], matches,
  matches = regex_match(C$CMDSTEM,
    "HW.*", ""),
  IF, LISTLEN(matches) > 0,
  E$ERROR="Take care with HW commands"
  end,
  end
),
end)

```

Fig. 6. A contrived SMF that flags all commands that begin with HW with an error.

D. Regular expression pattern matching

Previously, the SEQGEN program had limited string-processing capability; it was not possible to compare a string to a regular expression and look for matches. We added regular expression pattern matching functions to SEQGEN, allowing the MSL adapters to use succinct regular expressions to specify a valid file path argument of data management commands, instead of using a convoluted loop that examines file path strings, character by character.

Regular expressions can also be used to implement flight rules that can be triggered by hundreds of commands. For example, lack of testing might forbid hardware commands from ever being used during nominal operations. So, instead of specifying each of the 100 or so fictitious hardware commands as being forbidden during operations, one can use a regular expression-based rule specifying all hardware commands with `".*HW.*"`.

Fig.6 demonstrates one way to use regular expressions to check flight rules. `pre_command` is a reserved subroutine that SEQGEN executes right before executing the RESULTS of each command. `C$CMDSTEM` is the SEQGEN symbol name containing the name of the command being processed. When the count of the matches of the regular expression `HW.*` in `C$CMDSTEM` is greater than zero, the operator sees an ERROR.

E. Run-time interpretation using the evaluate built-in function

The MSL SEQGEN adaptation models many of the fifty-two thousand FSW parameters. FSW parameters are organized into groups, and a set of commands corresponds to each group of parameters. Furthermore, just as there are more than 30 mechanisms that have the same kinds of tolerance parameters, many parameter groups have many copies, for example, one copy for each mechanism. Finally, all the parameter groups belong in one of approximately thirty operational categories. Operational categories with many parameters allow individual parameters in a group to be set, without resetting the remaining parameters in the group, using special parameter setting commands. These parameter setting commands take an array of parameter `<name, value>` pairs. Operators use the array variant when the number of parameters in a group is too large to justify resetting all of the parameter values.

```

FIRE_LASER (
  PARAMETERS,
    nshots (TYPE, UNSIGNED_DECIMAL),
  end,
  RESULTS,
    LASER::SHOTS = LASER::SHOTS + 1,
    COMPLETED, (1 + nshots * 00:00:02, 0),
  end
),

```

Fig. 7. A contrived command with a duration that depends on its arguments.

The MSL adaptation simulates FSW parameter-setting commands that take an array of parameter-value pairs by using a new run-time interpreter. Given the parameter name, SEQGEN first looks up the name of the SMF model and the model attribute name that is tracking that parameter in a CVF table. Next, SEQGEN builds a string that assigns the new parameter value to the model attribute it just looked up. Finally, SEQGEN interprets this string and performs the assignment using the `evaluate()` step. Using lookup tables and the interpreter allows us to avoid implementing a subroutine that tests for 23,038 different FSW parameter names and updates the corresponding model attribute with the given new parameter value. Furthermore, as all SEQGEN RESULTS are evaluated, either implicitly or explicitly with `evaluate`, SEQGEN performs the same in time for both methods.

F. New modeling features improve duration estimate accuracy for MAHLI imager

As shown in Fig.1, the duration of each command in an adaptation is defined explicitly in one or more places. These durations do not need to be constant; in fact, command durations can vary depending on any number of factors, such as the simulated state or command arguments. Fig.7 shows a contrived `FIRE_LASER` command that takes an `nshots` argument and uses it to dynamically compute the command's duration during simulation.

As a testament to how our modeling refinements improved SEQGEN's accuracy when simulating the duration of MAHLI science activities, great improvement is seen after sol 70, when FSW parameters were introduced to the MAHLI SEQGEN model (Fig.8). Like the command arguments in Fig.7, we use model attributes that correspond to the MAHLI FSW parameters to simulate command durations. On average, SEQGEN simulates MAHLI sequence durations to within 5% of actual run times. Accurate sequence durations are important because the MAHLI is used during sample acquisition and processing, complex activities that have small time margins.

III. VERSATILE GROUND CONSTRAINT CHECKS

Fig.1 shows an example of checking a flight rule in the RESULTS section of the `POWER_LASER` command. Fig.3 shows an example of a flight rule in an FMRF file on the number of LASER shots fired. Previously, these were the only two ways to check flight rules. MSL required us to show the root cause of flight rule violations, add new RESULTS sections, and user-defined rules, to the FMRF file.

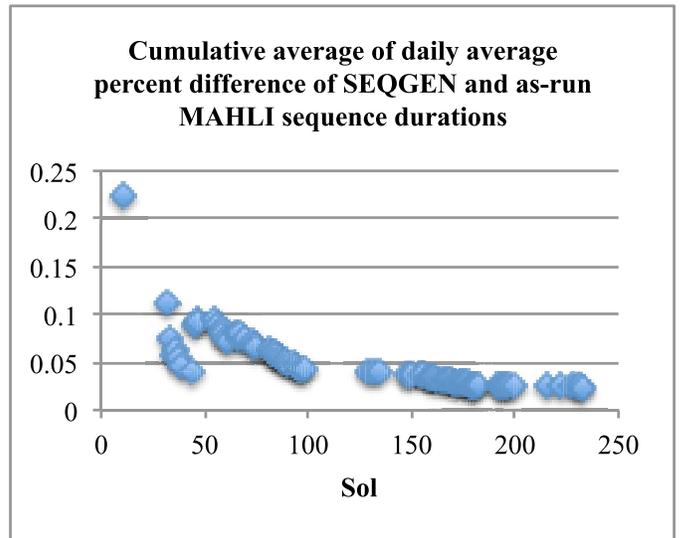


Fig. 8. MAHLI SEQGEN model accuracy [11]. The vertical axis represents the cumulative average of the percent difference of as-run and simulated MAHLI sequence durations. Gaps in data indicate periods of time when the MAHLI was not used for science.

A. Constraint violations always show cause

The genealogy of a command is the chain of sequences and command numbers that map directly to a specific instance of a command. Previously, if a command violated a flight rule in an FMRF file, SEQGEN only reported the time at which a flight rule violation occurred due to the command execution. To expedite root cause analysis, we updated SEQGEN to always report the genealogy of the command whose execution violates a flight rule in an FMRF file in addition to the violation text. As the violation text is a string constant, it was not possible until now to relate the exact genealogy to the operator. This capability allows operators to pinpoint the source of flight rule violations quickly and precisely, without running many simulations with sequences removed one at a time. For example, if the third command in the sequence listed in Fig.4 was the 1001st laser shot fired in the example model, the FMRF rule in Fig.3 would be violated. When the third command of `lasr00001` violates the rule in the FMRF file, the genealogy of the violation would end in `$lasr00001_3CMD`. Knowing the command number, 3, allows the operator to modify or remove the third command, so that the flight rule is no longer violated.

As shown above, genealogies reveal the ordinal number of the command that spawns a sequence, so it is possible to identify a specific instance of a sequence that is executed multiple times. On MSL, sequences that maintain the file system and remove data products are reused. Showing the genealogy of rule violations allows operators to identify violations in reused sequences with the master sequence, and therefore, sol, they occurred in.

B. Constraints can output multiple errors

Previously, when a flight rule violation is triggered, SEQGEN could only output one message, and while the message was customizable using attributes in scope, it was not possible to build the error message incrementally using the assignment

```

user_rule(LASER-0002,
TRIGGERS,
  check_power_cmd(
    TRIGGER,
    COMMAND_START, POWER_LASER,
  end,
RESULTS,
  IF, power_state == LASER::POWER_STATE,
  E$ERROR = "Laser already on!",
  end,
end), end) end,

```

Fig. 9. A contrived user-defined flight rule that triggers of a command in the SMF listing in Fig.1.

operator. This made it difficult to identify the names of three or more concurrently executing sequences that violate the MSL concurrency manager flight rules. Thus, we added support for rule RESULTS sections to SEQGEN, allowing adapters to iteratively build error messages and include information about all offending concurrent sequences. These RESULTS sections follow the same format as the RESULTS sections of commands.

C. User-defined constraint types

Previously, flight rules were triggered based on changes in attribute values and optionally based on duration constraints. Also, there were only eight predefined types of rules. Adding RESULTS sections to flight rules in FMRF files was the first step towards implementing user-defined rules. User-defined rules allow adapters to trigger violation messages when simulated state changes and when events including commands, stimuli, ground events, and blocks (sequences) occur. Regular expressions can be used to limit the scope of any trigger. Furthermore, rules were moved from one common `__RULE` model element to individual model elements, so each rule can now keep track of its own `LAST_VIOLATION_TIME` attribute. Previously, this would result in declaring attributes with the same name multiple times, resulting in a syntax error. Using all the trigger types available, we confirmed that it is possible to migrate all flight rules still present in MSL SMF to user-defined types, finally separating spacecraft state and behavioral logic from ancillary rule-checking state and logic. For example, the flight rule implemented in the SMF sample in Fig.1 that forbids turning the "laser" on when it is already on, could be implemented using a command-based trigger. Fig.9 shows an example of the user-defined rule type variant of this rule.

IV. UPDATED EXTERNAL INTERFACES

The SEQGEN simulator, along with related Mission Planning and Sequencing tools like MPS Editor and SLINCii have been updated to support new features mandated by MSL uplink needs. MPSEditor is a mission planning and sequencing tool, that also serves as the integrated development environment of choice for SEQGEN adapters. SLINCii translates spacecraft commands to binary op-codes. We describe some of these updates below. Fig.10 below shows SEQGEN's upstream and downstream interfaces.

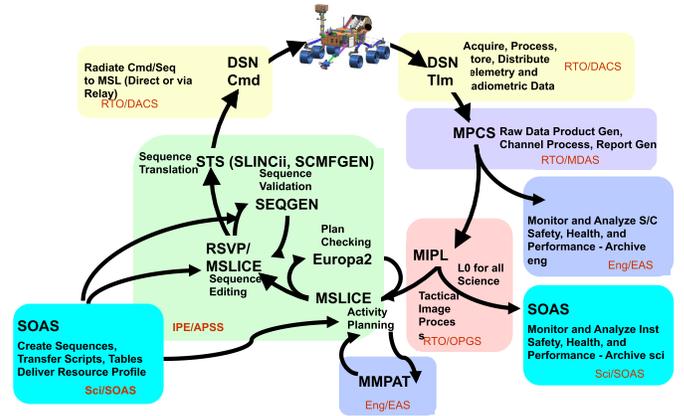


Fig. 10. MSL GDS UL/DL Context Diagram [12].

A. Relatively-timed commands in SSF

SEQGEN produces a Spacecraft Sequence File (SSF) as a result of simulation runs. SSF files are the verified product passed downstream through the uplink pipeline [13]. Prior to MER, all commands in an SSF had to be time-tagged with an absolute time. However, tagging commands with absolute times is not compatible with event driven sequencing. The SSF standard, SEQGEN, and SLINCii were updated to accept relative command timings.

B. Generating SSF files for SATF blocks

Previously, at least one command with an absolute time tag (a real-time request) in an SASF file (Spacecraft Activity Sequence File) was necessary to produce an SSF with useful content. As MSL master sequences call each other, it is possible that a sol's command load will not require real-time commanding, so the ability to generate valid SSF only from SATF files was necessary. SEQGEN now generates, and SLINCii correctly interprets SSF files from SATF files, even without a SASF request.

When an MSLICE user compiles a sequence project, the MSEQ (MSLICE Sequencing) Server returns SSF files that correspond to the sequence content of the RML files included in the request that only contain sequences and no immediate commands (SASF requests). The REST compilation call contains serialized RML sequence data, and the MSEQ Server converts this RML to SATF and SASF with MPS Editor libraries. Then, MSEQ Server will run the SATF and SASF through SEQGEN with simulation turned off using a bare-bones model, generating an SSF [14]. MPSEditor components can automatically create a bare-bones SEQGEN adaptation with no attributes and no RESULTS, from the current MSL command dictionary.

C. durations.xml Rover Sequencing and Visualization Program (RSVP) interface

MPS provides a standard user-defined function library to supplement SEQGEN functionality that missions customize, adding the functions needed. We started including XML

parsers in the standard user-defined library. The MSL SEQGEN adaptation has no geometry knowledge, so the duration of mobility and manipulation commands in Rover Planner (RP) sequences is simulated by RSVP. However, to correctly capture flight rule violations present in RP sequences, SEQGEN needs to be told how long each of the commands in those sequences takes. RSVP creates multiple durations.xml files, which are combined and then read and associated with each RSVP command with XML parsing functions in MSL's mission-specific user-defined function library. As RSVP does not invoke the MSL SEQGEN adaptation with a set of initial conditions for simulation, RSVP is unable to provide the full genealogy for each command in RP sequences in durations.xml. Instead, RSVP provides a reduced genealogy for each command in its sequences.

Genealogies do not contain actual command names; rather, they contain the names of sequences and individual command numbers. On the handover from one sol to the next, SEQGEN chops off everything in the genealogy up until and including yestersol's master sequence, so that SEQGEN genealogies and RSVP genealogies match. For example, if tosol is Sol 100, the genealogy of the first command in tosol's master sequence may be represented as $\$mars00099_XXCMD\$mars00100_1CMD$, wherein XX is the number of the command that caused the handover to mars00100. Because there is only one master sequence per sol, identifying commands with reduced genealogies like $\$mars00100_YYCMD$ still allows RSVP to match durations to the right commands. While the example above is not an actual genealogy used by MSL, it captures the essence of the genealogy structure used by MSL.

D. XMLRPC server-mode modeling

In addition to processing simulation runs in batch mode, SEQGEN simulation can also be performed interactively in server mode [14]. RSVP invoked SEQGEN interactively on MER [4] using the Data Transfer Mechanism (DTM) interface, but on MSL, RSVP uses SEQGEN in batch mode only. We added a new interface to the server mode to recent versions of SEQGEN that allows simulation requests to be performed over the wire using XMLRPC requests. These requests are processed in different threads of the same core process, but do not interact with each other as they are run inside independent simulations, so the concurrent simulations are not plagued by causality errors in typical PDES systems [15]. Running SEQGEN in XMLRPC mode does not significantly affect the runtime bottom line: in both cases, SEQGEN is able to process four hundred commands per second (Fig.11).

Even though different simulation instances on the same server process are not subject to such errors, it is possible for mission-specific model faults to emerge as MSL allows 16 threads of parallel execution at most (MER allowed 14). We use semaphores (not just locks) and aggressive runtime assertions during sol-to-sol master handover to ensure the consistency of our sequence engine simulation and activity constraint manager violation checks. The activity constraint manager is a component of MSL FSW that prevents the rover from being counterproductive. For example, the constraint manager may prevent sample processing during drives, thus preventing vibration from distorting sample processing results.

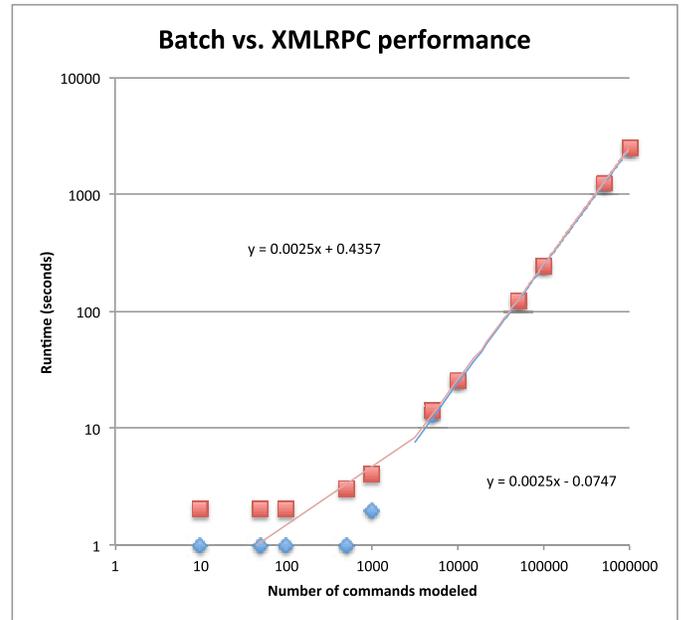


Fig. 11. SEQGEN runtime performance as a function of command load. Batch results appear in blue and XMLRPC results appear in red.

Recent versions of MPSEditor [16] include a simple sequencing perspective that builds on existing MSEQ Eclipse RCP functionality and additionally allows simulation to be performed on any SEQGEN process running in XMLRPC server mode, including local processes. Thus, the combination of SEQGEN run in XMLRPC mode and MPSEditor allows ad-hoc modeling and sequencing to be performed locally.

V. FUTURE WORK

Running SEQGEN locally may represent a large time savings, as approximately half the time it takes to model sequences in an RML file in MSLICE is spent sending the sequences to and retrieving the SEQGEN-simulated results from the MSEQ server [13].

Currently, MSLICE is exploring the use of Sequence Revitalization (SEQR) [17] activity timelines to persist and version activity plans. SEQGEN now affords the ability to read and write SEQR timelines through a command line option. These activity plans could be used by the MSL SEQGEN adaptation to implement more thorough ground constraint checks. Furthermore, work is underway to store the final spacecraft and ground states of a SEQGEN simulation run in SEQR state timelines, in addition to storing them in an output file. These final states could be corrected by a Timeline Component System (TCS) component that reads the latest available telemetry data from AMPCS.

ACKNOWLEDGMENT

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. 2013 California Institute of Technology. Government sponsorship acknowledged.

REFERENCES

- [1] G. Reeves and J. Snyder, "An overview of the mars exploration rovers' flight software," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 1, 2005, pp. 1–7 Vol. 1.
- [2] A. Murray, M. Schoppers, and S. Scandore, "A reusable architectural pattern for auto-generated payload management flight software," in *Aerospace conference, 2009 IEEE*, 2009, pp. 1–11.
- [3] B. Streiffert and T. O'Reilly, *The Evolution of Seqgen - A Spacecraft Sequence Simulator*. American Institute of Aeronautics and Astronautics, 2013/06/14 2008.
- [4] K.-M. Cheung, A. Ko, D. Page, J. Bixler, and S. Lever, "Design and architecture of planning and sequence system for mars exploration rover (mer) operations," in *AIAA SpaceOps Conference*, Montreal, Canada, 2004.
- [5] L. Needels, *Multi-Mission Sequencing Software*. American Institute of Aeronautics and Astronautics, 2013/06/14 2002.
- [6] B. Streiffert and T. O'Reilly, *Sequence System Building Blocks: Using a Component Architecture for Sequencing Software*. American Institute of Aeronautics and Astronautics, 2013/06/14 2006.
- [7] T. W. Starbird, J. R. Morris, K. S. Shams, and M. W. Maimone, "Rapid diagnostics of onboard sequences," Jet Propulsion Laboratory / California Institute of Technology, Pasadena, CA, United States, NASA Tech Brief, December 2012.
- [8] C. Grasso, "The fully programmable spacecraft: procedural sequencing for jpl deep space missions using vml (virtual machine language)," in *Aerospace Conference Proceedings, 2002. IEEE*, vol. 1, 2002, pp. 1–75–1–81 vol.1.
- [9] A. Mishkin, D. Limonadi, S. Laubach, and D. Bass, "Working the martian night shift - the mer surface operations process," *Robotics Automation Magazine, IEEE*, vol. 13, no. 2, pp. 46–53, 2006.
- [10] K. Novak, Y. Liu, C.-J. Lee, and S. Hendricks, *Mars Science Laboratory Rover Actuator Thermal Design*. American Institute of Aeronautics and Astronautics, 2013/06/14 2010.
- [11] Msl reports / getseqtree script output for sols 10-250.
- [12] A. A. et al., "Mars science laboratory mission system mos & gds mbse effort," February 2011, internal Presentation.
- [13] T. Crockett, K. Shams, and J. Morris, "Telerobotics as programming," in *Aerospace Conference, 2011 IEEE*, 2011, pp. 1–7.
- [14] J. Salcedo and T. Starbird, "Seq_gen: A comprehensive multimission sequencing system," Jet Propulsion Laboratory / California Institute of Technology, Tech. Rep., April 1994.
- [15] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.
- [16] B. Streiffert and T. O'Reilly, *MPS Editor - An Integrated Sequencing Environment*. American Institute of Aeronautics and Astronautics, 2013/06/14 2010.
- [17] S. Chung, *Timeline-based Mission Operations Architecture*. American Institute of Aeronautics and Astronautics, 2013/06/14 2012. [Online]. Available: <http://dx.doi.org/10.2514/6.2012-1269750>