Verification and Validation of a Deep Space Network Scheduling Application

Mark D. Johnston and Daniel Tran

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena CA USA 91109
mark.d.johnston & daniel.tran @jpl.nasa.gov

Abstract

Scheduling NASA's Deep Space Network (DSN) presents a challenging problem due to its size, dynamic character, and collaborative approach to resolving schedule conflicts. This paper describes an ongoing project to automate the DSN scheduling process, and our approach to verification and validation of the scheduling engine component of the new system. The scheduling engine is responsible for interpreting user's requests for communications and other services from the DSN, then generating and checking schedules that achieve those requests to the greatest extent possible. It also identifies and resolves conflicts in the schedule. We describe the overall DSN scheduling domain, some of the associated V&V challenges, and recent progress towards meeting these challenges.

Introduction

The NASA Deep Space Network (DSN) consists of three large complexes of antennas, spaced roughly evenly in longitude around the world at Goldstone, California; Madrid, Spain; and Canberra, Australia. Each complex contains one 70 meter antenna along with a number of 34 meter and smaller antennas, as well as the electronics and networking infrastructure to command and control the antennas and to communicate with various mission control centers. For a more extensive background on the DSN, refer to Umbriale (2003).

All NASA planetary and deep space missions, as well as many international missions, communicate to Earth through the DSN. In some cases, missions closer to Earth also use the DSN, some routinely, others on an occasional basis. The capabilities of the DSN make it a scientific facility in its own right, so it is used for radio astronomy (including very long baseline interferometry) as well as radio science investigations. At present, there are about 40 regular distinct users of DSN, who together schedule about 500 activities per week. Over the next few decades, utilization of the DSN is expected to grow significantly, with more missions operating, higher data rates and link complexities, and the possibility of manned mission support. In addition, there is significant pressure to reduce ongoing operations costs while maintaining an around the clock operational capability.

Scheduling the DSN is an extended process (see e.g. (Clement and Johnston 2005 and references therein), driven primarily by the needs of some mission users to sequence their spacecraft sufficiently far in advance with sure knowledge of when they will have communications with the ground. A typical goal is to have an initial schedule in place 18 weeks ahead of execution, and for it to be conflict-free by 8 weeks out. Due to the effort involved, and the occurrence of unexpected events such as launch slips and spacecraft emergencies, this desired lead time is rarely achieved in practice. Unlike other NASA communications networks, the DSN is essentially scheduled by its users. What this means in practice is that users define their scheduling needs (consistent with overall service agreements), and then work together to resolve conflicts in the resulting integrated schedule. The process allows for escalation should consensus not be reached among the users, but this is extremely rare. The DSN process is in contrast with other communications networks where a central network authority resolves contention, e.g. based on a strict priority scheme.

Software support for scheduling the DSN is currently a heterogeneous collection of tools and databases. Users provide their scheduling requirements in text documents and email, and use a graphical schedule editing tool called TIGRAS for entering and editing individual scheduled communications passes, often called *tracks*. A separate program is used to generate PostScript and PDF graphical views of the schedule. The final schedule is loaded into the Service Preparation System (SPS) database for dissemination via an internet portal, and for generation of the data products that are used to actually control the antennas and

other equipment at the DSN complexes. Late changes are handled by using TIGRAS to edit the SPS version of the schedule.

The DSN is currently implementing a new scheduling system, designated *Service Scheduling Software*, or S^3. The primary objectives of new system are to:

- *consolidate and modernize the tools and databases* used in the DSN scheduling process into a single suite
- *adopt a request-driven approach to scheduling* (in contrast to the activity-driven approach used in the past)
- *provide an interactive collaborative environment* for DSN users to investigate and mutually resolve schedule conflicts

S^3 is in development now and is planned to become operational in late 2010. In the following we describe S^3 at a high level, then in more detail the DSN Scheduling Engine component that is the focus of this paper. We describe our overall verification and validation (V&V) goals, some of the challenges encountered, and the overall approach we have adopted. We then describe some of the specific tools and processes we are using during system development to accomplish our goals, and finally we offer some general conclusions.

Service Scheduling Software (S^3)

The overall architecture of S^3 is illustrated in Figure 1.

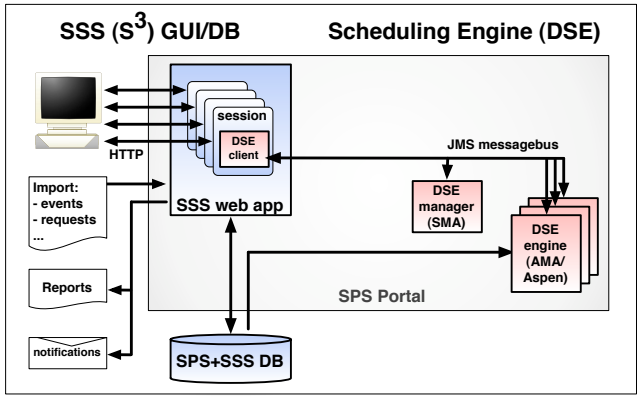

Figure 1. Architectural overview of the Service Scheduling Software (S^3) system. The Service Preparation System (SPS) Portal is an existing schedule portal with which S^3 will be integrated.

The components on the left (S^3 GUI/DB) include the S^3 web application through which users interact with all elements of the system. This portion of the system connects to an Oracle database to hold scheduling and all related data needed throughout the process. Users can generate reports and upload files from their local environments, using nothing more than a standard web browser to work with the system. Notifications can be viewed from within the application's web browser environment, or selectively routed to regular email.

Shown on the right of Fig. 1 is the DSN Scheduling Engine, a distributed set of server processes dynamically allocated to users as needed. The client library labeled "DSE Client" provides the link between these two major components of S^3. The DSE uses a protocol analogous to HTTP sessions so that each scheduling engine process can provide dedicated low-latency responsiveness to one user at a time. Individual servers register their availability with the DSE Manager (Schedule Manager Application, or SMA), which locates and assigns free DSE engine processes (ASPEN Manager Applications, or AMAs). To handle the expected number of simultaneous S^3 users, as many as several hundred AMA instances may be running at once, distributed over a number of the host computers that are part of the overall portal environment.

As noted above, S^3 is adopting a request-driven approach to scheduling the DSN, in contrast to current practice which can be best described as activity-driven. In an activity-driven approach, the schedule is created and managed entirely in terms of scheduled activities (communications passes), where the intent of each activity, and its relationships to other activities, is only sketchily contained in the schedule and related data. In contrast, in the request-driven approach, the scheduling request is the primary entity driving the scheduling process. Each request includes an implicit specification of activities that will be required to satisfy the request, including their allowed flexibilities in timing or resources. The advantages of a request-driven approach include:

- *leveraged effort*: one scheduling request can generate and be used to manage many scheduled activities, and a change to a request can propagate to all activities derived from it; this can significantly reduce the ongoing effort needed to generate the schedule and manage its changes
- *automated continuous schedule validation*: based on the request specification, the schedule can be continuously monitored for constraint and preference satisfaction; this can help minimize the effort to ensure that schedule changes, as they invariably occur, will not introduce undetected inconsistencies between requests and activities
- *traceability*: all activities trace to scheduling requests that describe the purpose and intent of the generated activities

The main drawback of the request-driven approach is the complexity of the request specification language. There are many options and subtleties involved in describing the constraints and preferences on DSN activities, and a sufficiently rich representation of these is necessarily large and complicated. This poses a significant challenge to the validation of the system.

S^3 is being developed as an integrated extension of the Service Preparation System (SPS), an existing operational system that provides a well-established portal to DSN users for all schedule and navigation data, and products derived from them. SPS includes a wide range of schedule validity checks to ensure that the data used to execute the schedule

is correct and accurate. Thus, for example, ensuring that scheduling errors do not drive the antennas out of their operating ranges is and remains the responsibility of SPS. Scheduling errors within S^3 will have the very negative consequence of requiring schedule rework late in the process, but will not pose a threat to the DSN assets or the missions they support.

DSN Scheduling Engine (DSE)

The DSN Scheduling Engine (DSE) is one of the central components of the S^3 system (Johnston et al. 2009). The major functional areas allocated to the DSE include:

- Schedule Conflict Checking
  - identify any activities that violate DSN feasibility rules, based solely on the schedule (i.e. independently of any scheduling requests), and characterize the nature of the rule violation
- Scheduling Request Interpretation
  - analyze scheduling requests for completeness and validity, then return diagnostics useful to the end user to help them formulate the request as intended
  - expand scheduling requests into communication passes (tracks) or other activities that satisfy the request specification
  - match tracks in the schedule to requests, accounting for the fact that users may separately edit tracks or create/delete tracks
  - determine whether activities in the schedule satisfy or not the scheduling request with which they are associated
- Schedule Repair
  - repair schedule conflicts: for activities in conflict that can be adjusted by the DSE (i.e. those not locked), use the flexibilities specified in the scheduling request to adjust the activities to eliminate or reduce conflicts
  - repair request violations: for requests that are not satisfied by their associated tracks, find a satisfying assignment of tracks without increasing schedule conflicts
- Schedule Query
  - identify temporal intervals satisfying user-specified conditions
  - identify temporal intervals where a given scheduling request can be potentially satisfied
- Schedule Optimization
  - adjust scheduled activities to better satisfy preferred request specifications

Scheduling Requests

Since scheduling requests constitute the driving data entity in S^3, this section provides an overview of what information is contained in each scheduling request. This information can be categorized broadly to include services, timing constraints and relationships, preferences, priority, and communications patterns.

A scheduling request specifies DSN *services* required on behalf of a user, along with any associated constraints and preferences. A DSN service specifies use of any of the available capabilities of the DSN, including uplink and downlink services, Doppler and ranging (for spacecraft navigation), as well as more specialized capabilities. The details of a spacecraft's service specification depend on the onboard hardware and software (the frequency band, encoding, etc.). Along with other factors such as radiated power levels and distance from the Earth, these all determine a set of antennas and associated equipment (transmitters, receivers, etc.) that must be scheduled to satisfy the request. However, these assets are not all equally desirable, and so there are preferred choices for antennas and equipment that also need to be considered.

In addition to single antenna/single spacecraft communications, there are a variety of other DSN service types. Some missions need the added sensitivity of more than one antenna at once, and so make use of arrayed downlinks

Constraint	Description
reducible	whether and by how much the requested time can be reduced to fit in an available opportunity
extensible	whether and by how much the requested time can be increased to take advantage of available resources
splittable	whether the requested time must be provided in one unbroken track, or can be split into two or more
split duration	if splittable, the minimum, maximum, and preferred durations of the split segments; the maximum number of split segments
split segment overlap	if the split segments must overlap each other, the minimum, maximum, and preferred duration of the overlaps
split segment gaps	if the split segments must be separated, the minimum, maximum, and preferred duration of the gaps
viewperiods	periods of visibility of a spacecraft from a ground station, possibly constrained to special limits (rise/set, other elevation limits)
events	general time intervals that constrain when tracks may be allocated; examples include: • day of week, time of day (for accommodating shift schedules, daylight, ...) • orbit/trajectory event intervals (occultations, maneuvers, surface object direct view to Earth, ...) Different event intervals may be combined and applied to one request. The included events may have a preference ordering.

Table 1: Timing constraints that play a dominant role in the specification of scheduling requests.

using two or more ground antennas. For navigation data, there are special scenarios involving alternating the received signal between the spacecraft and a nearby quasar, over a baseline that extends over multiple complexes. For Mars missions, there is a capability to communicate with several spacecraft at once (called Multiple Spacecraft Per Aperture, or MSPA): while more than one may be sending down data at once, only one at a time may be uplinking.

Constraints on DSN scheduling requests fall into several broad categories. The most important is *timing*: users need a certain amount of communications contact time in order to download data and upload new command sequences, and for obtaining navigation data. How this time is to be allocated is subject to many options, including whether it must be all in one interval or can be spread over several, and whether and how it is related to external events and to spacecraft visibility. Table 1 lists a number of these factors.

A second category of constraint is that of *temporal relationships among contacts*. In some cases, contacts need to be sufficiently separated so that data collection has time to accumulate data but not overfill onboard storage. In other cases, there are command loss timers that are triggered if the time interval between contacts is too long, placing the spacecraft into safemode. During critical periods, it may be required to have continuous communications from more than one antenna at once, so some passes are scheduled as backups for others.

A third category of constraint can be called *"distribution" requirements*. These can cover an extended time span and specify constraints on certain aspects of an overall set of activities during that time. Examples include: a certain proportion of 70m contacts; ensuring that navigation passes are spread out roughly evenly between the northern and southern hemisphere complexes; ensuring that not all contacts for one mission during a week are on the same antenna.

In addition to constraints, there are numerous *preferences* that scheduling users have as to how their activities are to be scheduled. Many would prefer additional time if it is available, while at the same time are able to reduce some contact durations in order to resolve a contentious period on an antenna. There may be preferences on gap durations, whether tracks are split or continuous, for tracks to occur during day shift at a particular operations center, and so on. While some of these preferences are implicit, some must be explicit and, if they apply, need to be specified as part of the scheduling request.

Priority plays a significant role in DSN scheduling, but not the dominating role that it plays in some other systems (e.g. Calzolari et al. 2008). Critical events (launches, surface landings, planetary orbit insertions) preempt other more routine activities. Other than critical activities, missions have higher priorities during their prime mission phase than during their later extended missions. However, higher priority does not automatically mean that resource allocations are assured. Depending on their degree of flexibility, missions trade off and compromise in order to meet their own requirements, while attempting to accommodate the requirements of other users. As noted above, one of the key goals of S^3 is to facilitate this process of collaborative scheduling.

One characteristic of DSN scheduling is that, for most users, it is common to have repeated *patterns of requests* over extended time intervals. Frequently these intervals correspond to explicit phases of the mission (cruise, approach, fly-by, orbital operations). These patterns can be quite involved, since they may interleave communication and navigation requirements.

Schedule Conflicts

A schedule conflict is a violation of a DSN feasibility rule in a schedule. Table 2 provides a list of the most common conflict types for which the DSE is required to check.

Conflict Type	Description
Spacecraft	Multiple tracks of the same mission share the same temporal extent (unless tracks are arrayed, handoff, or meet other specific criteria)
Beginning of Track (BOT)	Multiple tracks start within 15 min at Goldstone and 30 min at Canberra or Madrid.
Start of Activity (SOA)	Multiple activities (pre-track setups) start within 15 min at Goldstone and 30 min at Canberra or Madrid.
Antenna (Facility)	Multiple non-MSPA (Multiple Spacecraft Per Aperture) tracks use the same antenna at one time
Equipment	Multiple tracks are scheduled to use the same equipment during the same temporal extent
Viewperiod	The spacecraft or other tracking target is out of view of the tracking antenna
Teardown	The post-track teardown time does not match the expected teardown time
Setup	The pre-track setup time does not match the expected setup time

Table 2: Major conflict types in DSN scheduling.

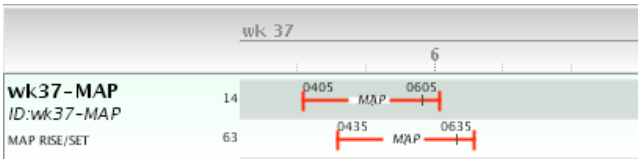

Figure 2. An example of a spacecraft conflict: multiple non-arrayed antennas (14 and 63) are scheduled for the MAP mission at the same time.

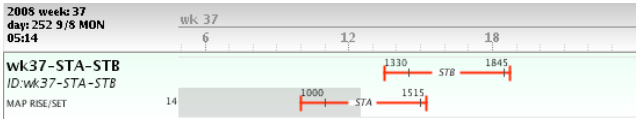

Figure 3. An example of a facility conflict: multiple non-MSPA missions (STA and STB) are placed on antenna 14.

Scheduling Request Violations

In contrast to conflicts, which are defined on scheduled *activities*, scheduling violations refer to *requests* that are not satisfied by activities on a particular schedule instance. Table 3 provides a list of the most important types of violations that the DSE checks for when determining whether scheduling requests are satisfied.

Violation Type	Description
Track Quantization	The track start or end time violates the request quantization constraint. For example, requests can specify that tracks start or end only at 5 minute intervals.
Track Separation	If the request is splittable, the separation time between two tracks violates the split segment overlap or split segment gap constraint.
Track Duration	If the request is splittable, the track duration violates the request split duration constraint.
Service Specification	The track violates the request service specification, i.e. the antenna or equipment allocated does not match the requested service.
Total Track Duration	The total track duration does not meet the requested duration
Number of Tracks	The number of tracks for the request violates the maximum. For a non-splittable track, this limit is 1; for a splittable track, the limit may be specified.
Track Temporal Extent	The track start or end time falls outside the scheduling request's time interval.
Viewperiod Reference	The track time interval violates the specified antenna in view marker.
Event Reference	The track time interval violates the intersection of the event time intervals referenced by the scheduling request.
Request Reference	The track time interval violates the scheduling request's temporal constraint link to other requests.

Table 3: Scheduling request violations.

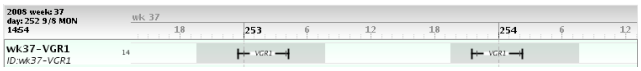

Figure 4. An example of a request reference violation: though both tracks fall with in their viewperiods, the required separation interval between the tracks must be at least 24 hours. In this example the tracks are separated by approximately 12 hours.

DSE Architecture

There are two basic design principles around which the DSE has been developed, derived from its role as provider of intelligent decision support to DSN schedulers:

- *no unexpected schedule changes*
  - all changes to the schedule must be requested, explicitly or implicitly
  - the same sequence of operations on the same data will always generate the same schedule (a principle that has important implications for testability)
- *always return something "reasonable"*, even for infeasible scheduling requests, possibly by relaxing aspects of the request
  - if a request is determined to be infeasible, return a diagnosis of the nature of the infeasibility: this provides a valuable starting point for users to diagnose the problem

The DSE is implemented with ASPEN (Chien et al. 2000) at its core, as indicated in Figure 5, lowest level. At the next lowest level is an ASPEN DSN domain model expressing resource availability and reservations in terms of timelines, driven by a schedule model API that incorporates the DSE search and other algorithms. Above this layer we can distinguish between DSE processing that requires comprehension of scheduling request specifications (right) versus those functions that can be conducted without scheduling requests. The latter include 1) identifying schedule conflicts, and 2) schedule query functionality (e.g. "find schedule gaps longer than 3 hours in duration on any 70 meter antenna on day 147 of 2009").

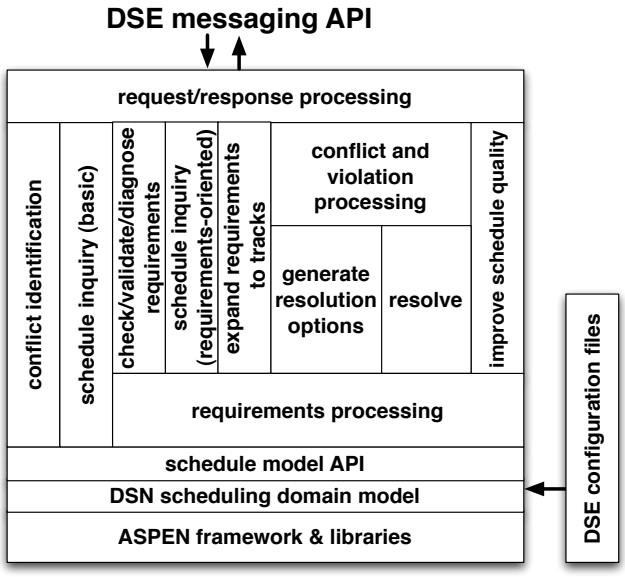

Fig. 5. Major functional areas of the DSN Scheduling Engine, showing schematically how they are layered.

Request comprehension is necessary for violation checking, for request-based schedule queries, and for expanding requests into candidate activities. When searching for reso-

lutions for conflicted schedules or for unsatisfied requests, the DSE makes full use of the alternatives and flexibilities expressed in the scheduling request specification. The DSE provides a family of parameterized strategies that users can invoke independently or chained together. Among the options that users can control are:

- the order and repetitions of the different search phases
- how much time to invest in each solution search phase
- whether to focus on resolving conflicts or on request violations
- which parameters and in what order to relax scheduling request parameters if the DSE cannot find a satisfying allocation to tracks
- whether to consider adjusting tracks to improve schedule preferences, e.g. after a conflict resolution selection that reduced tracks from nominal to minimal durations

In addition, users can control the focus of the DSE (in temporal extent or on a subset of all mission users, or both.

V&V for the DSN Scheduling Engine

Some of the goals and challenges of DSE V&V include the following:

- Given the usage of the DSE as integrated with an encompassing web application being developed completely independently, how can we test it with as much similarity to that deployment environment as possible?
- How can we channel the expert knowledge of DSN scheduling end users to a) validate the scheduling request specification language, and b) provide a progressive suite of test cases?
- User scheduling requests can have a complex structure with many interacting elements: how can we validate that such requests are in fact feasible?
- How can the existing heterogeneous scheduling tool collection be used to best advantage in validating the new system?
- Since ASPEN is used by numerous projects and the core ASPEN code is shared across them all, how can we ensure that ASPEN core changes do not impact DSN scheduling engine functionality and behavior?
- How do we ensure that the DSE distributed server infrastructure will withstand the expected load in the S^3 operational environment?

We discuss each of these topics in the following subsections.

Testing the DSE

The DSE interface to the S^3 web application (See Figure 1) is defined as a set of XML messages sent over a messaging middleware layer (currently JMS). The structure of these messages is fully defined by an XML schema document. In order to simplify this interface for the purposes of integration with the remainder of S^3, we have defined a client Java library that can be incorporated directly in the S^3 web application, through which all messages to the DSE are routed. Based on this, we have developed a test harness to drive the DSE exactly as it will be driven when integrated with the web application.

The DSE test harness is controlled by an XML script that specifies:

- the test runtime environment, including environment variable settings
- a sequence of ingoing XML messages representing requests made of the DSE
- a companion set of response messages that represent the expected responses

The harness loads and runs the test script in a fresh session of the DSE server. It captures the responses and the log output, then scans the logs for warnings, errors, and exceptions, filters out trivial changes (e.g. run dates) and compares the result messages against a baseline. The results are reported to HTML files that can be viewed in a standard web browser. Each test can be run separately as a JUnit test, e.g. within an IDE, and the entire suite can be run from an ant build script at the command shell. The latter is also run from the DSE CruiseControl instance, which builds and tests the system with each change to the source code repository. All of the test scripts, input message files, and baseline output messages are themselves checked into the repository.

Along with the harness we developed a test management tool that can be run at any time to provide the status of all the individual tests. This GUI highlights tests that are not passing, shows the differences graphically, and allows for immediate re-baselining of the response messages when a change in the results has been verified as expected.

A typical test sequence would incorporate several elements, depending on the intent of the test:

- for testing schedule conflicts, a set of scheduled activities specifying their time and resource assignments, along with corresponding visibility intervals
- for testing treatment of scheduling requests, including expansion or violation detection, one or more scheduling requests, and visibility or other constraining event intervals
  - optionally, a pre-existing schedule could be provided, in case the interaction between request expansion and existing activities is of interest
- for testing conflict resolution, schedule queries, and similar functions, both requests and pre-existing schedules would need to be sent to the DSE

A request message sent to the DSE can ask for any of scheduled activities, conflicts, and request violations to be returned. As a result, all of these can be captured after each request message, and added to the baseline test result set.

There currently exists a collection of ~40 example tests, but this set is intended to be replaced with a more systematic and extensive set as development proceeds. Some scheduling request language features have changed and are expected to continue to evolve significantly through the first few pilot releases of S^3. All of the request language

features of Table 1 (and more not listed), as well as the schedule conflict and violation types listed in Tables 2 and 3 will have corresponding tests. We expect the final test set to number in the hundreds.

DSE Test Client GUI

From the initiation of the S^3 project, precise requirements for the scheduling engine component of the system have been difficult to define. As a combined new *software system* and *user process*, the DSN scheduling automation project has had to work on both aspects at the same time, recognizing that the two are tightly connected. The scheduling engine component has been particularly difficult to characterize, since users interact with it only indirectly through the S^3 web application, and there is no existing DSN scheduling request specification language to serve as a point of comparison.

In order to elicit timely feedback from expert scheduling users, we have taken the approach of developing a standalone test GUI for the DSE. DSN scheduling users can directly use this GUI to explore various aspects of DSE behavior, including:

- the scheduling request specification language — for expressiveness and completeness
- expanding scheduling requests into empty and crowded schedules
- conflict resolution when resource contention is high
- repairing scheduling requests that are in violation of the request specification

The DSE test client is a Java application that allows users to interactively define all of the separate elements that can be used in scheduling requests, including viewperiods, events, constraints and preferences, relationships among requests, etc. Once a request is defined, it can be sent to the scheduling engine for expansion into specific tracks. Users can build up collections of requests and test them in isolated schedules, or in combination on crowded schedules where conflicts are certain to occur. They can then run the DSE conflict resolution algorithms, and edit tracks to set up "what if" scenarios. An example screen from the test GUI is shown in Figure 6. The DSE test client uses the same client library as the S^3 web application, and thus serves as a proxy for that component of the system while it is under development.

As a pathway for S^3 users to gain understanding of the functionality of the DSE and to provide feedback to the development group, the test client has been very successful. To date, two versions of the test client have been provided to DSN scheduling users, in January and July 2009, from which a great deal of useful feedback has been obtained, well over a year before deployment of the S^3 system. This has helped reduce the risk that key functionality is being missed, and has also helped prompt consideration of scheduling process changes that will be required as S^3 moves toward deployment.

One of the functions of the DSE test client is to capture test configurations that can then be converted into test harness cases, as described above. The "messages" tab of the test GUI provides a running history of all communications with the DSE, and any message or sequence of messages can be captured for scripting in the test harness. The visual feedback provided by the request editor and schedule views are very useful in verifying that the test configuration is as expected.

By placing the DSE test client in the hands of users, with a sufficiently high level of functionality, we had also in-

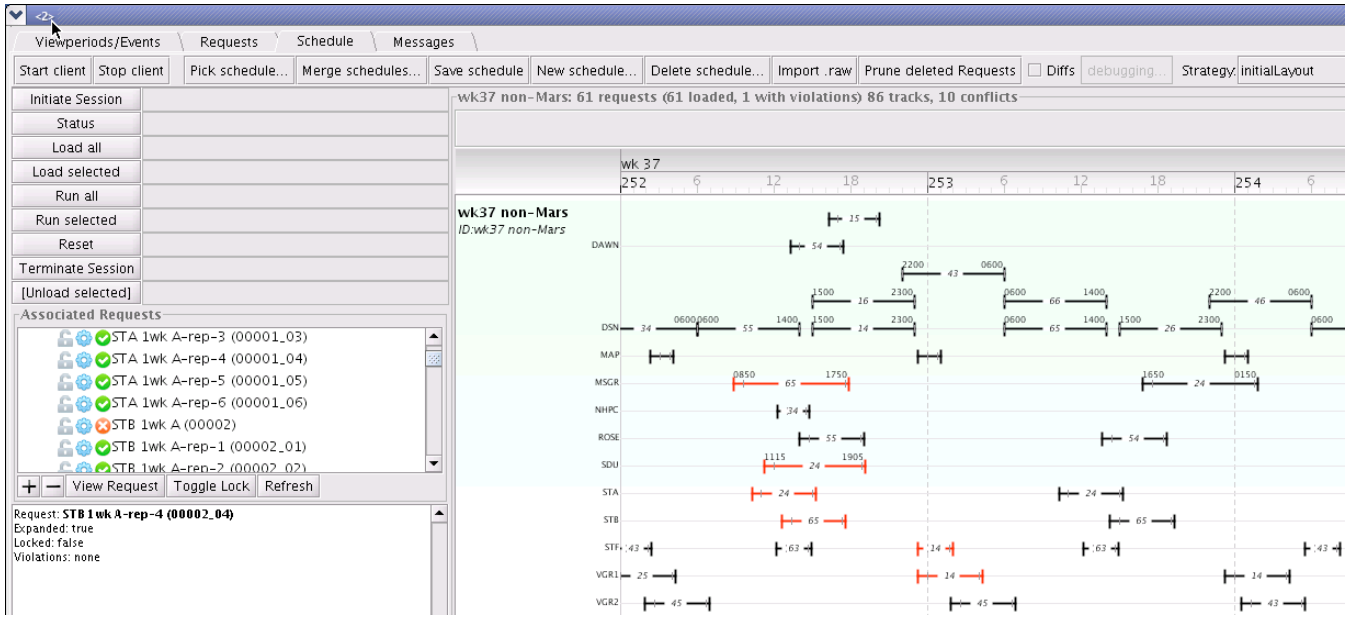

Figure. 6: a sample screen from the DSE test client, showing a schedule (left, Gantt view) generated from a list of scheduling requests (right). Tracks shown in red are in conflict, while requests flagged with a red "X" are in violation. Users can choose from a variety of strategies to repair the schedule, and can explicitly edit tracks to set up test situations to examine.

tended that it could be used to define and capture realistic scheduling requests for flying missions. This would allow us to validate that the overall scheduling request language and DSE functionality is appropriate for DSN user needs. In this it is been successful beyond any initial expectations: it was adopted in January 2009 by one of the DSN scheduling teams, the JPL Multi-mission Resource Scheduling Services (MRSS) team, responsible for scheduling 20 of the active users of the DSN. It has replaced a paper-and-pencil process used before it was available, and has been used to provide the basis for the initial integrated DSN schedule for an increasingly large set of missions since January 2009. As of July 2009 the test client is being used to generate the initial integrated schedule for *all* DSN missions. In addition to providing a useful stepping stone towards the full S^3 functionality, this has provided an extensive body of scheduling scenarios and test data encompassing all DSN users that will enable a much broader level of testing than would have been otherwise possible. As one indication, over 5,000 scheduling requests have been created using the DSE test client by the MRSS team, and every actual and future DSN schedule since week 17 (late April) of 2009 has been generated with input from the DSE test client.

Validating Scheduling Requests

One of the challenges faced is validating scheduling requests. As users are developing scheduling requests, requests are susceptible to being infeasible and having no solution. The set of reasons why requests may be infeasible is too large to enumerate, but we provide several examples that have already been encountered as users worked with the test client:

- The requested tracking time exceeds the available antenna view interval. A user may request 10 hours of tracking time, but due to the specified quantization in the request, the available maximum antenna view interval is 9 hours, 55 minutes. Therefore, the request cannot be strictly satisfied.
- The intersection of the events intervals and available viewperiod intervals may be empty.
- During continuous tracking, where one contact must overlap with another contact, the required overlap time can exceed the available overlap between antennas.

The feasibility of these requests is evaluated by the DSE test client library and by the scheduling engine. Since the former has data structures required to display viewperiod intervals and event intervals to the user, it is also capable of performing simple request validation checks. In particular, the test client can be used to determine if the maximum duration antenna in view interval is shorter the minimum request time. It can also be used to determine if the intersection of event intervals results in no valid intervals. By performing these checks in the client code, it is easier to provide immediate feedback to users about request validation problems while they are still editing them.

These simple validation checks can also be done in the DSE, along with validating more complex sets of requests. For requests with timing constraints to other requests, the DSE can be used to schedule these requests in isolation. If the engine is not able to generate a set of legal tracks in an isolated schedule, it is unlikely to find a solution in a more congested schedule. If no solution is found, requests are then flagged as being in violation and the user can diagnose the problem. The DSE is able to provide some feedback to the user to assist with this diagnosis, including the specific timing and duration parameters that are not satisfied. However, in cases where multiple factors conspire to make a request infeasible, the engine does not currently perform any kind of root cause analysis to determine which aspects of the request are overconstraining.

DSE and existing tools

In today's DSN scheduling process, existing tools are responsible for some functions that will be taken over by S^3 when it is deployed. One of the challenges will be to ensure that these existing tools and the DSE are consistent. Scheduling requests are a new concept introduced to the DSN scheduling process; therefore, there are no existing tools to validate requests or check that they are satisfied. There are however, tools available to check the schedule for conflicts (see Table 2). The TMOD Integrated Ground Resource Allocation System) (TIGRAS) (Borden et al. 1997) is a tool developed at JPL and currently used to to display and edit schedules, as well as generate schedule conflict reports. TIGRAS supports exporting and importing schedules in a custom XML format. This format encapsulates the schedule time bounds as well as each track in the schedule.

The DSE test client will be capable of exporting the DSE generated schedule into this XML format, which can then be imported into TIGRAS. In order to preserve the relationship between tracks and requests, this mapping is also exported to a persistent field. TIGRAS can then be used to check the detected conflicts. If changes to the schedule are performed in TIGRAS, the DSE test client can also import schedules exported from TIGRAS. Once imported into the DSE, tracks can then be validated against their originating scheduling requests. The overall effect of this round trip import/export to legacy tools is to increase confidence that the new system is consistent and correct.

Shared ASPEN core testing

The underlying implementation of the DSE makes use of the ASPEN framework and libraries. ASPEN is a generic reusable automated planning and scheduling framework that has been deployed for multiple projects, ranging from traditional ground planning to planetary surface rovers to onboard reactive planning (see e.g. Knight 2008; Estlin et al. 2007; Chien et al. 2005). Each deployment shares the same code base, with custom model-specific files developed for each mission. During the development phase of the DSE, the latest version of ASPEN is used. There are

advantages and disadvantages to this approach. One main advantage is that it allows us to leverage any improvements to ASPEN developed for other projects. However, it may introduce problems. Since the ASPEN development team consists of members not associated with the DSE, we must ensure that changes to the core do not adversely affect the DSE deployment, as well as other deployments. This is addressed with daily builds and test runs, as well as using source control tools to tag and branch specific version of the software.

Nightly builds of ASPEN, across multiple operating system and target platforms are performed to ensure the code is always in a buildable state. In conjunction with this, we also perform nightly regression runs of ASPEN across a set of sample models. These regression tests ensure that changes to ASPEN maintain the same behavior for each model, and also monitor for performance impacts of any algorithm changes.

ASPEN also uses several third-party tools to analyze the code. The Rational tool suite of Purify, Quantify, and PureCov is used to identify any memory issues, identify runtime bottlenecks within the code, and ensure that all sections of the code are executed. We also perform nightly runs of the Coverity, a static code analyzer used to identify sections of code that may pose a risk.

During integration testing, we will need to limit the changes to the ASPEN core code to ensure tests are reproducible and fixes are focused. We will take the standard approach of branching the code from the mainline. This allows development to continue on the mainline, while testers have a static version to test and identify problems. Changes to the branched version can be made to resolve any problems that arise, and these changes can then be selectively merged back into the mainline. Likewise, selected fixes and improvements on the mainline can be incorporated into the branch.

DSE Testing Under Operational Loading Levels

The deployment environment planned for the DSE includes a number of high-end server hosts that will be part of the SPS internet portal (see Figure 1 for an illustrative view). Current plans call for hosts in the class of Sun Fire X4450 servers, each with 24 total cores and 128GB of memory. Each host of this class is expected to support in the range of 50-100 separate DSE server (AMA) instances. One of the current development challenges is assessing the performance under load of a configuration containing several of these host servers.

The DSE infrastructure design allows for server instances to register their availability with the Schedule Manager Application (SMA), which then balances and assigns client session initiation requests to available servers. Once assigned, the client communicates directly with the assigned engine instance, using the messaging middleware (currently JMS). A typical usage pattern would be for a client to load a schedule and all of its auxiliary data (viewperiods, events, requests, etc.), and then perform incremental operations on the resulting in-memory model (resolve conflicts, modify requests or tracks, etc.) Server sessions time out after some adjustable period of idle time, and can also be terminated explicitly, e.g. when a user logs out or closes the last view of a particular schedule.

While there exist load testing tools for web browser based server applications, there are no comparable tools for applications with custom middleware infrastructure such as the DSE. Consequently we are developing a loading test driver that can simultaneously initiate many sessions and funnel requests to the DSE, verify responses, and record timing data for analysis. Preliminary results suggest that the JMS messagebus may act as a bottleneck, but more detailed performance monitoring is needed before definitive conclusions can be drawn.

Conclusions

We have described a NASA Deep Space Network scheduling application, currently under development for operational deployment at the end of 2010. We have focused on the scheduling engine component of the overall system, as primarily responsible for the interpretation and processing of user scheduling requests. The development, integration, and deployment of this system faces numerous challenges in order to be ready for operational use when planned. Since the DSN is a current operational system, we have demonstrated compatibility with existing tools to validate schedules and thus build confidence in the system. Also, active engagement with future users of the system, along with access to prototype system elements, has been extremely fruitful in terms of making sure that the system development is on a path to meet user needs.

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. We gratefully acknowledge the contributions and feedback from the DSN scheduling community and especially the MRSS team.

References

Borden, C.; Wang, Y.-F.; and Fox, G. 1997. Planning and scheduling user services for NASA's deep space network. In 1997 Int. Conf. on Planning and Scheduling for Space Expl.

Calzolari, G.; Beck, T.; Doat, Y.; Unal, M.; Dreihahn, H.; and Niezette, M. 2008. From the EMS concept to operations: First usage of automated planning and scheduling at ESOC. In SpaceOps 2008.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. ASPEN - automating space mission operations using automated planning and scheduling. In SpaceOps 2000.

Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, A., Davies, D., Mandl, D., Frye, S., Trout, B.,

Shulman, S., and Boyer, D. 2005, Using Autonomy Flight Software to Improve Science Return on Earth Observing One, Journal of Aerospace Computing, Information, and Communication.

Clement, B. J., and Johnston, M. D. 2005. The deep space network scheduling problem. In Innovative Applications of Artificial Intelligence (IAAI). Pittsburgh, PA: AAAI Press.

Estlin, T. , Gaines, D., Chouinard, C. , Castano, R., Bornstein, B., Judd, M., Nesnas, I. and Anderson, R. C., 2007, Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2007), Rome, Italy.

Imbriale, W. A. 2003. Large Antennas of the Deep Space Network. Wiley.

Johnston, M.D., Tran, D., Arroyo, B., and Page, C. 2009, Request-Driven for NASA's Deep Space Network. In International Workshop on Planning and Scheduling for Space (IWPSS), Pasadena, CA.

Knight, R. 2008, Automated Planning and Scheduling for Orbital Express, International Symposium on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS)