Mission Data System and the Future of Autonomy at Jet Propulsion Laboratory

Sanford M. Krasner

Jet Propulsion Laboratory
4800 Oak Grove Drive
4801 Pasadena, CA, USA 91109
Email: skrasner@jpl.nasa.gov

INTRODUCTION

Planetary missions require significant levels of autonomy, due to the critical, one-time-only, nature of many mission objectives, and to the long delay times for human intervention. These missions also operate in highly uncertain environments, such as planetary atmospheres or surfaces. Mission Data System (MDS) is a project to construct a reusable software architecture for planetary missions, based on identifying the states of the mission. MDS is not itself an autonomy development, but will provide a framework for incorporation of other autonomy technologies.

MOTIVATION FOR SPACECRAFT AUTONOMY

The Jet Propulsion Laboratory (JPL) is NASA's lead center for planetary space exploration. JPL has built spacecraft with increasing levels of autonomy since at least the Vikings in the early 1970s. Planetary spacecraft autonomy has been driven by the impracticality of continuous human involvement in spacecraft operations. Two-way light times within the solar system range from tens of minutes (inner solar system) to hours (outer solar system). In addition, communications time through the Deep Space Network (DSN) is a very limited and expensive resource; some missions go up to 1 week between contacts. In this environment, the spacecraft is responsible for guaranteeing its own safety without ground intervention.

Planetary missions are characterised by critical mission activities which must occur correctly at predefined times, without any opportunity for human intervention and corrections. Examples of critical activities include: orbit insertions; planetary entry, descent and landing; and planet or moon flybys. In each of these cases, there is only one opportunity to accomplish the activity – if the opportunity is missed, the mission is a failure.

In addition to critical activities, some missions are extremely limited in the time available to complete the mission. This requires that human intervention, and the round-trip time required for evaluation and planning, be minimized. For instance, the proposed Europa Orbiter has a planned orbital lifetime of approximately 30 days. Any outage while waiting for ground intervention in anomaly analysis and recovery would result in a severe impact on mission science value.

The Mars '07 rover mission is limited to ~180 days, due to seasonal power limitations and solar array dust accumulation. The mission plan calls for the rover to drive ~300 meters/day, and to drive into territory not previously seen, at ground level, by ground operators. The mission also requires the rover, starting from 10 meters away, to drive to a target rock and place science instruments without ground intervention. Limited mission duration will limit the number of pauses to wait for ground commanding.

Planetary missions also operate in highly uncertain and variable environments. During aerobraking for Mars Global Surveyor, the uncertain and rapidly variable nature of the Martian atmosphere required extensive monitoring and commanding to maintain spacecraft safety. This level of attention will be impractical for future missions when there is significant competition for DSN resources. The Mars rovers operate in a very poorly known environment; when driving beyond the area that has been observed by ground operators, the rover must be able to identify potential hazards, such as large rocks or crevasses, and to plan paths around these hazards. (The rover may also carry terrain maps, based on best-available orbiter imagery.) The Mars '07 mission anticipates collecting remote sensing data during traverses, autonomously identifying scientifically interesting targets, and revising the ground-supplied mission plan.

STATE OF THE PRACTICE IN JPL SPACECRAFT AUTONOMY

SEQUENCING

The predominant mode of planetary mission operations has been based on the use of timed sequences – scripts of commands which are executed at predefined absolute or relative times. Sequencers are relatively simple to implement and to test, but do not adapt well when activity start times or durations cannot be determined ahead of time. Non-determinism may be due to:

- uncertainty in the behavior of the spacecraft (such as precise propulsion performance during maneuvers),
- uncertainty in the environment (such as the location of the spacecraft with respect to the surface during a Mars landing, or the presence of unanticipated hazards during a rover traverse);
- time required to recover from anomalies or failures. This may include time required to shutdown failed equipment, configure and initialize new equipment, and time to recover the state assumed by the sequence.

Sequence design also assumes a single linear chain of events (such as warming up equipment before using it). If this path is changed, due to concurrent activities, failures, or uncertainties in the environment, the sequence may not execute as planned.

Finally, sequence design has no representation of intent and dependency – the notion that certain activities in the sequence are dependent on the correct operation of other activities (such as maintaining a healthy propulsion system while performing a maneuver). The representation of commands does not have any notion of "persistence" – that the state established by a command (e.g. powering a piece of equipment) is intended to persist for some period of time. Since there is no notion of intended state persisting over time, there is no ability to determine if the desired state is being met over relevant time intervals.

Missions have also used varying levels of conditional and event-driven sequencing. As far back as the Galileo Jupiter Orbiter, sequences were able to operate conditionally based on information about spacecraft operations. Galileo used this capability to implement the mission-critical probe release, probe relay and Jupiter orbit insertion activities. These activities required very careful design of multiple concurrent sequences, including fault response sequences, and of the interactions between them. This process was extremely labor-intensive and required extensive rework when the sequence timing was revised.

Spacecraft Control Language (SCL) has formalized event-driven command. In general, these event-driven activities invoke predefined command scripts – the timing of these scripts is based on the conditions assumed when the scripts were designed.

There have been some attempts to add on-board evaluation, planning and script generation. This was done in the Deep Space 1 Remote Agent [1] and is proposed for the USAF Techsat-21 mission [2].

Due to these limitations on sequence implementations, some missions have chosen to design critical mission activities into their flight software. Mars Pathfinder and Mars Exploration Rover ('03) have implemented state machines to coordinate the Entry, Descent and Landing activities. These implementations are event-based, but they are labor-intensive and inflexible; reprogramming is required if activity design is changed.

ORBIT DETERMINATION AND CONTROL

Until recently, JPL missions have performed orbit determination and planned trajectory control exclusively on the ground. Orbit determination makes use of radiometric information (Doppler and ranging) measured by the DSN, as well as optical navigation using images taken onboard and downlinked to the ground. Trajectory control was done by means of individual discrete maneuvers, designed on board and uplinked to the spacecraft as timed sequences. This has been feasible for previous missions because the missions in interplanetary ballistic cruise do not require rapid turnaround to correct trajectory errors. In addition, until recently onboard computation power and processing algorithms have been too limited to implement onboard navigation.

The Deep Space 1 mission demonstrated autonomous orbit determination [3], based on estimated non-gravitational accelerations from thruster and ion propulsion engines, as well as onboard processing of optical navigation images.

DS1 demonstrated use of a low-thrust ion engine, thrusting continuously over long periods of time. Autonomous orbit determination became more important due to the continual non-gravitation forces applied. DS1 also demonstrated autonomous orbit control, with maneuver direction and magnitude computed onboard. Onboard navigation interacted with an attitude control "expert", which estimated required durations for attitude slews and maneuvers. Navigation software constructed relative-time sequences, which were implemented by the usual sequencing system.
Optical navigation was also used to estimate spacecraft time of closest approach during the Braille asteroid encounter in July '99. The encounter sequence was executed by relative-time sequences, triggered at computed times before closest approach. This demonstrated a hybrid of event-driven and time-based sequencing.

FAULT PROTECTION

Fault protection has generally been implemented at several levels. Fault detection monitors are implemented in software components throughout the system. Violations of fault thresholds are reported; in general, values of these thresholds are not directly related to ongoing spacecraft activities. Changes in threshold values must be explicitly sequenced. Fault responses may be taken locally for some types of faults. Limited diagnoses are sometimes done by combining multiple symptoms; these diagnoses are defined by system engineers during the development processes. Fault responses are implemented by predefined sequences; these are generally unrelated to the ongoing sequence. In all but critical periods, ongoing sequences are cancelled to prevent interference with fault protection responses. This requires that the fault protection response establish a safe spacecraft state, which can be maintained until the next ground contact. Again, the spacecraft lacks a representation of intent and dependency for its ongoing activity, so the only acceptable response is to override all ground-commanded activities.

The DS1 remote agent included a demonstration of model-based fault diagnosis and isolation, but this was limited to a simple simulated failure mode during a dedicated demonstration period.

MISSION DATA SYSTEM – A STATE-BASED ARCHITECTURE

Mission Data System (MDS) is a JPL project to build architecture and frameworks, which can be reused across multiple missions. [4] The MDS architecture is intended for use in flight, ground and test implementations. Reuse of the same architecture across flight and ground systems allows reuse of common designs, common implementations, and allows migration of functionality from ground to flight. MDS is not an autonomy technology development in itself, but provides a standard structure, which can support various autonomy technologies.

MDS provides a standard way of analyzing and implementing a mission software system, using a small number of standard architectural concepts. These architectural concepts are supported by framework software, which implements common interactions between architectural elements. Framework software also supports creation of software components, multiprocessing, and communication between components. Software component communications include interactions between components running in the same processor, components in different processors, and between ground and flight computers. Common representations between ground and flight implementations allow for common, symmetric uplink and downlink protocols. These protocols support reliable delivery where required.

The MDS architectural concepts are illustrated in Fig. 1. Each block on this diagram represents a type or a particular instance. "Type" and "instance" are used in the object-oriented sense. Types are identified to encourage reuse between flight and ground deployments, and reuse across multiple missions.

Since the MDS architecture is intended for use in flight, ground and test environments, the architecture includes "deployment instances". A deployment is a set of MDS-based software running in a particular processor. A mission implementation may include flight, ground and test deployments, all communicating across various communication paths. This architecture also supports, for example, multiple processors onboard one spacecraft.

The fundamental MDS architectural concept is the "state variable instance", or "state variable" or "state" for short. A state variable represents a piece of information about the mission and the environment in which it operates. A state may be as simple as an enumerated variable, such as whether a power switch is Open or Closed; a state may be as complicated as an interplanetary trajectory. An interesting concept is the state of the mission data repository, such as whether a particular science observation has been captured in the repository. Some state variables may be controllable

within a particular mission's scope (such as spacecraft attitude). Other states may be uncontrollable but predictable (such as a planetary ephemeris). Knowledge of uncontrollable states may be required for correct operation of the mission (Such as predictions of when the Sun will rise over a Mars landing site.)

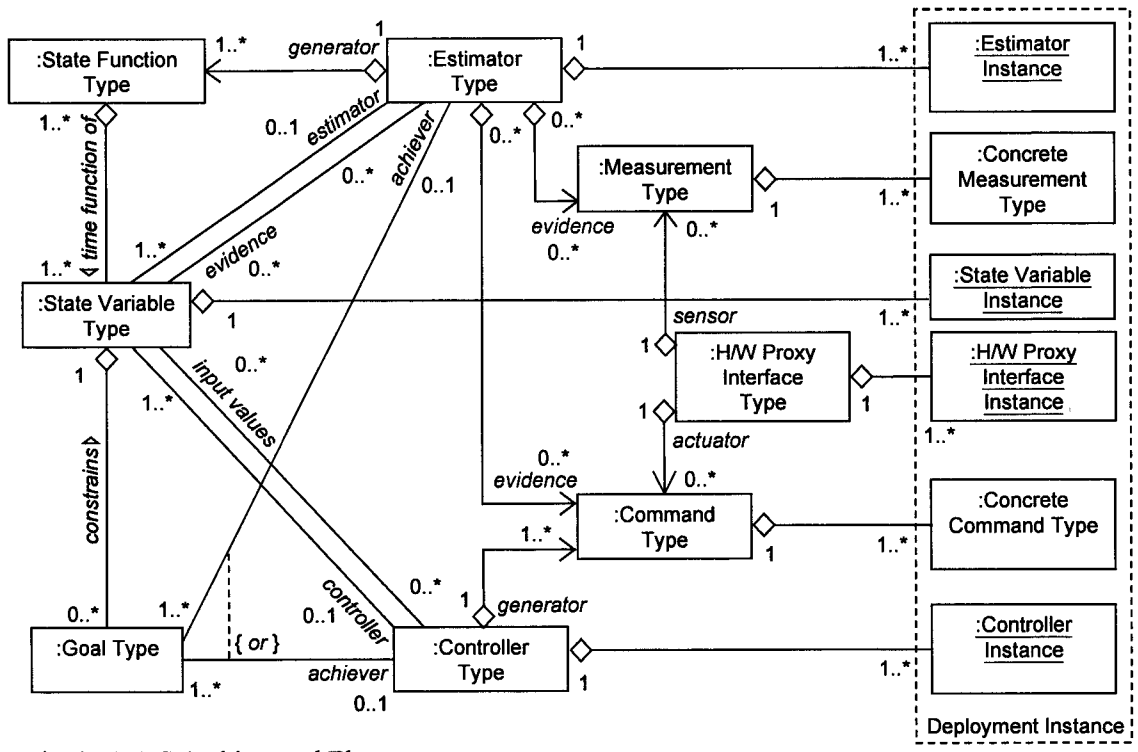

Fig. 1 – MDS Architectural Elements

Each state variable is required to have an estimator; one estimator will generally estimate more than one state variable. An estimator may make use of sensor measurements to estimate state variables; it may also make use of device commands, using "dead-reckoning" models to predict the effects of commands. The estimator is responsible for resolving any conflicts between various sources of evidence. These conflicts may also be the basis for detecting failures in sensors or actuators.

Device failure status may also be represented as a state variable of the system. Knowledge that a device has failed may be used in elaboration and planning, so that alternate devices are used. It may also be used to invalidate use of measurements from that device.

The MDS architecture makes a fundamental distinction between measurements and estimates. Each sensor in the system has a measurement model, which may be a function of several state variables. (For instance, a gyro measurement may be a function of scale factor, bias drift, and alignment, as well as gyro inertial rate.) Sensors are normally viewed as measuring only one state, with other states in the measurement models considered as nuisances to be hidden as much as possible. Explicit recognition of these additional states, as well as incorporation of calibration parameters as state variables allows all of these states to be estimated at the same time.

The MDS architecture also supports a "state variable proxy" pattern. In this pattern, a state variable may be estimated in one deployment, with the resulting estimates transported to another deployment. For instance, a planetary ephemeris may be estimated in a ground process, then transported to the spacecraft for its own uses. A symmetric pattern is used to transport onboard estimates to the ground, for telemetry analysis and display. The same software is used to access a particular state variable in all deployments.

An important notion is that the state variable is defined to have a value at all times. The software model for a state variable includes an "estimate function" to extrapolate its value over particular time intervals. For a discrete-valued

state (such as ON or OFF), this model may be as simple as a constant value over contiguous time intervals; for a trajectory state variable, this model may be as complicated as the standard navigation models, e.g. Chebyshev polynomials.

State variables are also required to represent some notion of the certainty (or uncertainty) of the state estimate. For continuous state variables, such as attitude or trajectory, there is a well-defined but complicated mathematics for computing uncertainty, based on initial uncertainty and measurement noise. For a discrete-valued state, the representation of uncertainty may be a small number of discrete gradations, such as "very certain", "somewhat certain" and "completely unknown". Representations of uncertainty are required so users of state estimates may decide how much reliance to put on a particular estimate. Various missions have experienced problems when invalid measurements and estimates have been used incorrectly, propagating erroneous information throughout the system.

In the MDS architecture, all control is implemented by means of goals. A goal is defined as "a constraint on a state variable over a period of time". A state variable may have a large range of values. (Spacecraft attitude may be anywhere within a sphere; spacecraft position may theoretically be anywhere in space.) A goal defines a limited subset of the possible range. (E.g. a goal on spacecraft attitude may be that an antenna boresight is pointing at Earth over some period of time. This goal requires additional state variables to represent antenna orientation within the spacecraft frame, and to represent the direction to Earth.)
The MDS architecture also allows for "transitioning" goals. Rather than specifying a particular value, a "transitioning" goal may specify that a state variable is changing in a particular way. (A "slewing" goal on spacecraft attitude specifies that the attitude is being slewed to a desired value. A "profiling" goal may specify that spacecraft position with respect to Mars surface is following a smooth altitude-and-velocity profile to descend to the surface.) "Transitioning" goals will often be followed by "maintaining" goals. (E.g. the "slewing" goal may be followed by a "maintaining pointing" goal. This contrasts with current command-based implementations, where the notion of transitioning to, then maintaining an achieved state is only implicit in the command definition.)

A controller will enforce the goal constraint by querying the state variable for estimated value, and issuing hardware commands to implement the necessary commands. Many goals will have effects similar to commands in current missions. Defining goals adds two very important notions: persistence and intent. The goal defines a condition that will persist over the lifetime of the goal; this condition can be monitored to detect a failure to maintain the constraint. The goal also makes the desired condition explicit. The maintaining activity is not implicit as something to be continued once a desired state has been achieved.

Knowledge goals may be issued to estimators. These goals define required levels of estimate certainty, and may result in goals to have particular sensors powered.

Goals are issued from a temporal constraint network. The time interval for each goal is defined by start and end timepoints; the goal constraint is delivered to a controller and enforced when the start timepoint "fires". Each timepoint has a time window defined by an earliest and latest start time. These times are defined by a network of temporal constraints, including minimum and maximum offsets relative to other timepoints. A special "Epoch" timepoint ties this networks to absolute time frames. This networks supports absolute time sequencing, relative time sequencing, required minimum durations, required maximum durations, etc.
Timepoints are fired when a timepoint is between its start and end time, and when the "starting constraint" for a goal is recognized. This allows the goal network to implement event-based control, so that particular goals are initiated when required precursor states are achieved. This also allows timepoint firing to be delayed until a particular absolute or relative time is reached. Time-out failures are detected by firing a timepoint when its end-time is reached; in most cases, this will result in goal failure.
For instance, a parachute deployment may be triggered to occur at a particular function of spacecraft altitude and velocity; Altitude and velocity are estimated states of the system. Image taking may be triggered when pointing has stabilized at the desired orientation. Note that attitude and velocity may not be directly controlled, but they are required state variables in order to understand and control the system. The architecture supports "event" goals, which allow detection of certain events without directly controlling them.

The goal network is constructed by a process of goal elaboration. (Fig. 2) Introduction of a goal into the network by ground operators (e.g. "antenna pointing at Earth") may elaborate into other subgoals or "children" goals. These

subgoals may occur before the start of the parent goal (e.g. thrusters may be warmed for 90 minutes before they are used). Subgoals may occur concurrently with parent goals (e.g. thruster power must be maintained while thrusters are being used for control), or may occur after the parent goal. (E.g. clean up after an activity). Required duration of these activities is represented as temporal constraints within the network. Subgoals may have their own elaborations. The goal network is built by together by linking these successive levels of goals. Elaboration bottoms out in a set of executable goals, which may be delivered directly to controllers or estimators.

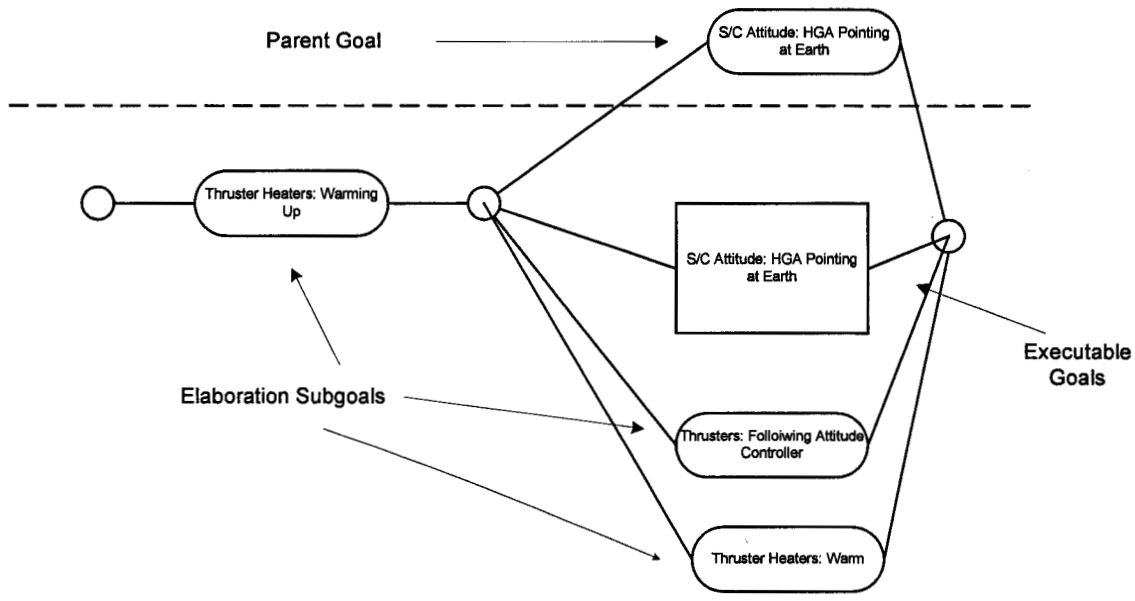

Fig. 2 – Sample Elaboration Diagram

The goal network explicitly represents precedence, concurrence and dependency among goals in the network. Persistence of goals is represented by the intervals between start and end timepoints. The "parent" goals are dependent on successful maintenance of the children goals; failure of a child goal is reported to the parent goal for possible response. This forms the basis for fault protection within the MDS architecture. [6]

Elaboration may result in simultaneous constraints on a particular state variable. The elaboration process invokes compatibility–checking models on each state variable. These models determine if the proposed goal network is consistent – if all concurrent goals can be accomplished. For instance, individual goals on pointing may be incompatible; depending on geometry, it may not be possible simultaneously to maintain antenna pointing at Earth and solar array pointing at the Sun. This consistency checking also provides a mechanism for resource management and allocation. Goals to allocate power to individual devices can be compared to total power available, to determine if constraints on available power margin are met.

As mentioned above, goal failure is an important basis for fault detection. A parent goal may take one of several responses when a child goal fails:

a) ignore the failure if the parent goal can be accomplished by other means (e.g. use a redundant device if it is already powered);
b) invoke a different elaboration to accomplish the same parent goal (e.g. power on a redundant device if the prime device fails);
c) fail the parent goal. This invokes the same set of options on the next higher-level of the goal dependencies. Failure of a goal also relinquishes all child goals in the network. "Safety-net" activities such as spacecraft safing may be implemented by default goals, which are invoked when all other goals fail.

Goals may be prioritized so that a conflict of two goals will cause the lesser-priority goal to fail, with corresponding failure responses. This allows a mission plan to be "oversubscribed" – deliberately overplanned to take advantage of

better-than-nominal performance, with the expectation that some lower-priority goals may not be accomplished. This capability allows ground operators to maximize mission return.

The MDS architecture is being applied to system engineering and requirement definition through a process of "state analysis". State analysis involves identifying state variables referenced or controlled in mission scenarios. Additional states are discovered through identifying goals, states to be estimated, required hardware, measurements and commands, etc. This results in system requirements and design, which are expressed in MDS architectural elements.

Current MDS Status

MDS has been baselined as the software architecture for the Mars '07 Lander/Rover mission. The MDS project is completing a simplified one-dimensional prototype of a Mars Entry, Descent and Landing scenario. This scenario was undertaken primarily to drive framework development; frameworks and adaptations have been implemented for state variables, estimators, controllers, measurements, commands, executable goals and uplink and downlink transport of state variable and measurement information. Flight, simulation and ground telemetry display deployments have been developed as part of this scenario. As of the time of this conference, goal elaboration, scheduling and goal-failure detection will have been demonstrated.

The MDS project is performing state analyses on scenarios for Mars '07 Rover Operations and for Entry, Descent, and Landing. These analyses will lead to incremental implementations of these mission scenarios within the MDS architecture. These activities will aim towards a June, '03 technology demonstration gate, leading to development of the Mars '07 mission software.

MDS INTEGRATION TO AUTONOMY TECHNOLOGIES

The MDS architecture is intended to support incorporation of autonomy technologies. It is anticipated that some autonomy technologies will provide implementations for particular architectural elements. The MDS architecture will provide organizing principles and architectural roles for these technologies. Of course, integration of these technologies may point out deficiencies in the MDS concepts, leading to a stronger, more applicable architecture.

Model-based reasoning provides a technology for explicitly representing and using system models. Model-based Mode Identification and Recovery Planning was demonstrated on the DS1 Remote Agent. Within the MDS architecture, model-based failure diagnosis may be used as the estimator implementation for device health states; models of device behavior may compared with sensor measurements to determine if hardware devices are behaving as modeled.

Model-based representations of system states and transitions may be used in the goal-elaboration process. Within the MDS architecture, controller modes will correspond to different types of goals on the state variable being controlled; each controller mode will represent a different way of constraining the state variables. If the controller mode transitions are represented as a state transition diagram, transition events will be implemented as triggering of goals. A model-based representation of the state transition diagram could be queried to find the actions necessary to reach the desired control mode. These actions can be represented as an elaboration diagram of the corresponding goals.

Goal elaboration includes a scheduling capability. The scheduler invokes models to check compatibility of current goals on the same state variable, and adjust temporal constraints to build an achievable network. The current scheduler algorithms are based on the CASPER planning and scheduling technology. MDS anticipates that CASPER or other planning technologies may be integrated into the goal-elaboration and scheduling design. A particular concern is the amount of time required in planning and replanning; the continuous, incremental planning capabilities of CASPER may be applicable.

The goal network allows MDS adaptations to take advantage of temporal flexibility. Ground operators will want to do some level of goal-net validation prior to uplink, to give confidence that the desired plan will be achieved. Validation of current sequence-based plans is greatly simplified by that only one sequence timeline must be examined. Using goal networks with varying temporal relationships will require a significantly different approach to validation. A further complication is added by the use of oversubscription – deliberately planning more activities than are expected to be

accomplished, to take advantage of better-than-predicted performance. In this context, some aspects of the plan are expected to fail – what is an "acceptable" level of predicted failure?

Autonomous orbit determination and trajectory control will become important in future missions, particularly for missions around small bodies. Image capture and orbit determination will be implemented within the MDS measurement and estimator frameworks; trajectory control will probably require goal elaboration to establish the necessary configuration, as well as goals on spacecraft trajectory.

A particularly interesting challenge will be to adapt rover technologies, including image processing, hazard recognition, and path planning, into the MDS technology. Analysis is currently in progress to integrate these technologies in support of the Mars '07 mission.

SUMMARY

The Mission Data System provides a state- and goal-based architecture that will provide the basis for future planetary missions. The goal-based aspects of the architecture make explicit the intent, persistence, and dependencies of ground-defined mission activities. The MDS architecture will enable the reuse of designs between flight and ground systems, and across multiple missions. This architecture will encourage integration of autonomy technologies into complete, operational mission software systems.

REFERENCES

[1] A.. Jonsson, P. H. Morris, N. Muscettola, K. Rajan, B. Smith, "Planning in Interplanetary Space: Theory and Practice", 2nd NASA International Planning and Scheduling Workshop, San Francisco, CA Mar. 16-18, 2000

[2] S. Chien, "The Techsat-21 Autonomous Sciencecraft Constellation Demonstration", ESA Workshop on On-Board Autonomy, 17-19 Oct. 2001,ESTEC, Noordwijk, The Netherlands

[3] J. Riedel, "Using Autonomous Navigation for Interplanetary Missions: Mission Operations with Deep Space 1 AutoNav", ESA Workshop on On-Board Autonomy, 17-19 Oct. 2001,ESTEC, Noordwijk, The Netherlands

[4] D. Dvorak, R. Rasmussen, G. Reeves, A. Sacks, "Software Architecture Themes in JPL's Mission Data System", IEEE Aerospace Conference, 1999

[5] R. Rasmussen, "Goal-Based Fault Tolerance for Space Systems Using the Mission Data System", IEEE Aerospace Conference, 2000

[6] Krasner, "A Reusable State-Based Guidance, Control and Navigation Architecture for Planetary Missions", IEEE Aerospace Conference, 1999