Modeling Relationships Using Graph State Variables[1]

Matthew B. Bennett and Robert D. Rasmussen
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
(818) 393-0836
Matthew.B.Bennett@jpl.nasa.gov and Robert.D.Rasmussen@jpl.nasa.gov

Abstract—The Mission Data System is a unified flight, ground, simulation, and test software system for space missions. Currently, its first application will be the Mars Smart Lander mission, where common MDS software frameworks will be adapted for use in interplanetary cruise, entry-descent-landing, and rover operations. A key architectural theme of MDS is explicit modeling of states. This provides a sound foundation for estimation, control, and data analysis. Certain essential states are relative rather than absolute. Relative states are defined in graph state variables (GSVs) as relationships between nodes in a graph. GSVs are a general graph-based state representation that (1) derives a state's value by combining relationships, (2) produces different results for different derivation paths, (3) handles changes to topology and relationships between nodes, and (4) represents dependencies between relationships (e.g. correlations). This paper shows example GSV representations for spacecraft orientation, location, trajectories, dynamics, and kinematics.

TABLE OF CONTENTS

1. INTRODUCTION

Future space missions are becoming more challenging and complex. Flybys of planets are now being followed with landers and in-situ mobile vehicles. Mobile vehicles have demanding objectives to achieve scientific objectives without the intervention of ground controllers. Obstacles must be avoided. Chance objects of interest need to be efficiently identified and investigated on the way to achieving specific ground-directed mission objectives. Timely communications with Earth may be impractical because of the long distances involved, or because of blocked communications. Upcoming flyby missions are also becoming more challenging. The spacecraft must autonomously compensate for uncertainties in small body trajectories that cannot be anticipated by their human operators on Earth. Missions that previously could be planned in advance as detailed sequences of commands, now must be flexible to handle uncertainties in time, trajectory, and science opportunities. Graph state variables provide a way for spacecraft to represent new targets and obstacles relative to known locations. Ground-based operators need not be in the loop; additions can be made in a timely autonomous fashion in response to events in the environment.

The availability of more capable and affordable computing power enables the above mission objectives. But this also puts more responsibility on software to implement more advanced capabilities. Further, spacecraft technology advances now make it affordable to launch many more spacecraft than in the past. We can now launch missions that are comprised of a fleet of cooperating spacecraft, putting further demands on spacecraft operations. Graph state variables can represent the trajectories of spacecraft fleets and the communication networks by which they coordinate their operations. Mission operations for this next wave of spacecraft will be prohibitively expensive if we continue to orchestrate every detail of spacecraft direction using human-intensive sequencing tools and processes. Clearly, there is a need for more spacecraft autonomy in the next generation of spacecraft to simplify human-spacecraft interactions. Graph state variables in concert with MDS goal-directed behavior will go a long way towards achieving autonomous fleet operations.

Software costs will limit the potential for taking advantage of these new opportunities in spacecraft autonomy. Currently, spacecraft software is built starting with inherited capabilities, standards, and operational paradigms from previous missions. Much, however, is re-written and validated because many requirements about the spacecraft and mission is hidden within the software code without an abstraction that facilitates reuse. Even when inherited software is well documented, the requirements and assumptions that went into its design are so interwoven into

[1] 0-7803-7321-X/01/$10.00 © 2002 IEEE

its implementation that reuse requires significant rework or discarding.

The Mission Data System (MDS) is a unified flight, ground, and test architecture under development at the Jet Propulsion Laboratory. It will provide a set of reusable software frameworks, inspectable models, and a methodology and language to identify and express requirements in their terms. This paper describes graph state variables, a specific MDS software framework designed for modeling relative states.

MDS has a number of general architectural patterns for models. One pattern is explicitly representing state as a function of time with a specified uncertainty. Many states are *absolute* (e.g. volume, pressure, current) and can be modeled as scalars with units. Other states are *relative* and are defined with respect to a reference point (e.g. voltage with respect to a ground, location with respect to an origin, orientation with respect to a coordinate frame). Graph state variables represent such reference points as nodes in a graph, and a relative states as edges between pairs of nodes in the graph. Graph state variables link and combine relative states along paths in graphs. A graph rather than a tree topology allows for alternate relative state derivations that can depend on mission envionment. Mission requirements on relative states are modeled explicitly in terms of graph state variable architectural elements. The implemented elements are easily inspected and modified from mission to mission. Further, graph state variables provide a common architectural framework for representing relative states on both ground and flight software systems. Graph state variables and the entire MDS architecture will enable future mission software specialists to concentrate on building new autonomous capabilities, rather than on re-inventing the past.

2. MDS Overview

The Mission Data System guiding belief is that software plays a central and increasingly important system role that must be reconciled with the overall systems engineering approach adopted by a project. Both software and systems engineering apply across all parts of a project and to all elements of the environment affecting the mission. Therefore, it is essential that systems and software share a common approach to defining, describing, developing, testing, operating, and visualizing what systems do. To

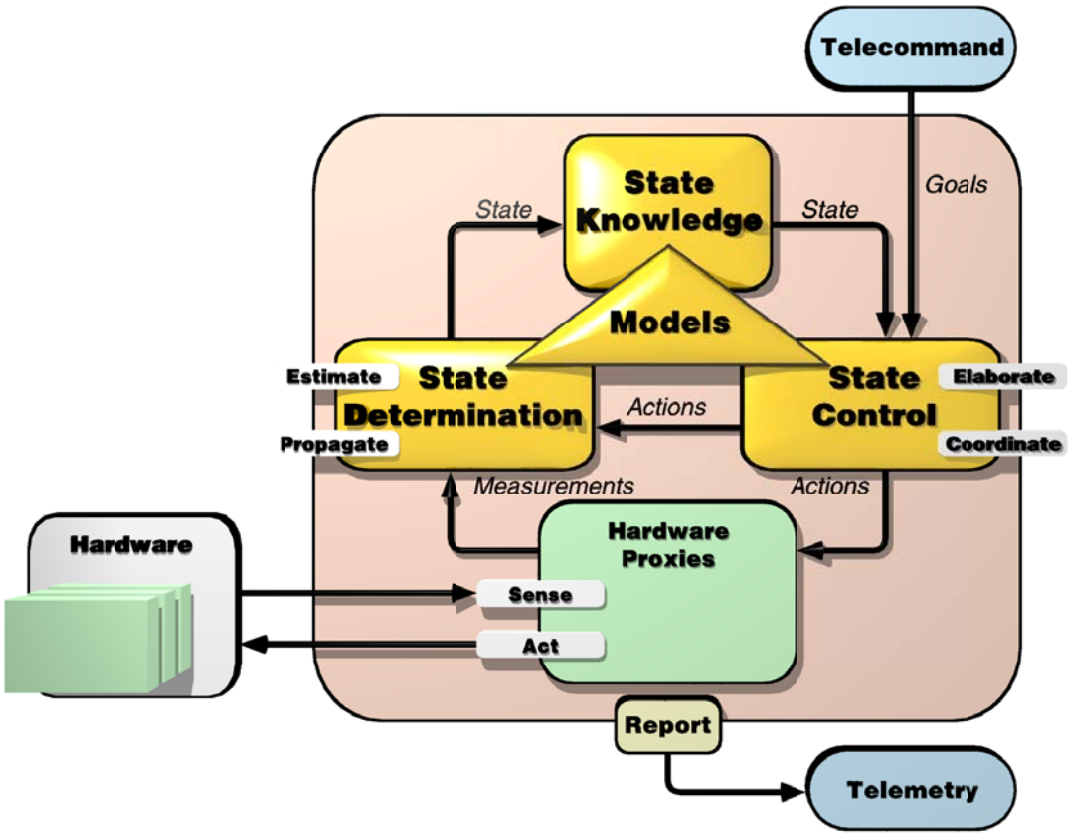

Figure 1. This diagram emphasizes several MDS architectural themes: the central role of state knowledge and models, goal-directed operation, separation of state determination from control, and closed-loop control. Graph state variables manifest the central role of state knowledge and models. Graph state variables represent relative states and model their composition into combined states.

realize this belief, MDS is founded on a set of architectural themes that shape the design [1]. These themes are emphasized because they have a broad impact on the design and they differ from earlier spacecraft engineering practices. Key themes that are manifested by graph state variables are described below.

State and Models Are Central

MDS is a state-based architecture, where *state* is a representation of a momentary condition of an evolving system and *models* describe *how* states evolve and are affected (see figure 1). All clients of state access it in a uniform way through *state variables*, as opposed to a program's local variables. Figure 2 illustrates the clients of state.

A *state timeline* describes *what* a state's value is as a function of time. State timelines are a complete record of the system's history, expectations, and plans. As well as providing the fundamental coordinating mechanism on the spacecraft, state timelines are also the objects of a uniform mechanism of information exchange between flight and ground.

Graph state variables are used to represent relative states. GSVs model relative states as directed edges between pairs of nodes in a graph. Each edge represents a relative state as transitive relationship timeline between its two nodes. A graph state variable can derive a relative state by composing the relative states along a path in its graph.

Explicit Use of Models

MDS tries to express domain knowledge explicitly in inspectable models rather than implicitly in program logic. The models separate the application-specific knowledge from the reusable logic that applies domain knowledge to solve a problem. A model built for a specific mission may also be reused on future projects. The task of customizing MDS for a mission, then, becomes largely a task of defining and validating new or reused models.

Graph state variables provide a reusable framework for modeling relative states as nodes and edges in a graph. The type of a graph state variable is defined by the type of relative state that appears on its edges. The type is a transitive relationship that explicitly models how to represent a particular kind of relative state for a particular domain. The type also models how relative states are composed together to form derived relative states. The composition models are abstracted in terms of relationship operators for concatenation and inversion. Derived relationships are described in more detail in section 5.

A graph state variable type that models spacecraft attitude and trajectory is an example of a model that can be reused by subsequent missions. In this case, the relationship type is a combined translation and rotation that has applications in the navigation, guidance and control, and robotics domains.

Figure 2. System state is the MDS architectural clearinghouse for information processing.

This kind of relationship is called a six degree of freedom transformation and is discussed further below and in section 5.

Join Navigation, Attitude Control, and Robotics

MDS builds navigation, attitude control, and robotics applications from a common mathematical base. They have been built in the past as separate development efforts because they operate over different timescales or in different environments, or because their dynamics don't greatly affect each other. When they need to share information, in cases such as maneuver execution or pointing towards celestial bodies, the interfaces have been ad hoc and conversions are needed between different forms of knowledge representation.

In general, elements that are separately engineered may work by themselves, but often fail to work together. Separately engineered applications set the stage for bugs to slip through the cracks simply because of the large number of possible interactions. This is a major source of unreliability. Future missions will have greater needs to share more information between disciplines that have been separately engineered in the past. For example, rovers will need to navigate to targets identified by orbiting mapping spacecraft and will need to point antennas to Earth or orbiting relay spacecraft. Docking missions and missions to orbit small planetary bodies will require tight coupling between attitude control and navigation. The MDS solution is to build subsystems from common architectural elements, rather than the other way around. In this way, the interactions are more reliable by using common interaction management mechanisms and consistent knowledge representations for orbit dynamics, attitude dynamics, and kinematics.

Graph state variables that model six degree of freedom transformations provide a common mathematical base and modeling mechanism for location and attitude relative states needed in the domains of navigation, guidance and control, and robotics. Six degree of freedom graph state variables are discussed further in section 5.

3. STATE

State Variables

The MDS architectural element for representing state is the *state variable* [2]. All users of state knowledge get it from state variables. An *estimator* weighs in evidence from measurements, commands, and other states to calculate an estimate for a state's value. Only one version of a state's estimate is represented, and it's stored in a single state variable to discourage potentially inconsistent private estimations. A remote system, however, may need a copy of a state estimated on a different system. In this case, the estimating system sends new estimates to the remote system. The remote system accesses the copy, just as the local system accesses the original, except it is not allowed to change the copy. A state variable that contains such an immutable copy is called a *proxy state variable*. A state variable that is updated locally with an estimator is called a *basis state variable*.

Some states are computed simply as functions of other states (i.e. their estimations do not incorporate measurements or commands). Such states are called *derived state variables* and the computations of their values are called *derivations*. This concept comes in handy later during the description of graph state variables.

A state estimate is the system's best guess of the "true" physical state. MDS recognizes that state estimates are not "truth" and requires that all state estimates include an assessment of uncertainty.

MDS stores the state timeline for a basis or proxy state variable as functions of time in the state variable's *value history*. Because a state's value history implies its derivatives, MDS implicitly represents a state's derivatives in its value histories. If an application needs an explicit representation of a state's derivative, then MDS requires that the derivative be encapsulated and consistent with its value history.

Absolute and Relative States

In this section, we describe the requirements on graph state variables for representing relative states. Some states are *absolute* in nature and are independent from points of reference. Some examples of absolute states are

- whether a pyrotechnic device is fired or not,
- whether a parachute is stowed or deployed or separated,
- whether a cable is cut or not,
- whether a switch is open or closed,
- whether an instrument is powered on or off,
- the current flowing through a heater, and
- propellant volume and pressure

Certain key spacecraft states, however, represent relative rather than absolute quantities. Examples of *relative states* are

- a spacecraft basebody orientation relative to the Earth Mean Equator 2000 coordinate frame,
- the location of a spacecraft relative to the Earth,
- the direction to the Sun relative to a spacecraft basebody coordinate frame,
- the location of a rock relative to a rover arm end effector,
- the voltage of an instrument's high-voltage sensor grid relative to a spacecraft chassis,
- the output power of a transmitter amplifier relative to its input signal,
- the connectivity and data rate of a spacecraft instrument relative to an onboard communications bus,
- the connectivity and data rate of a lander relative to an orbiting communications satellite or an Earth ground station, and
- the heat transfer of an infrared detector relative to its radiator.

MDS graph state variables were developed to recognize the requirement for explicitly representing relative states with respect to points of reference. For example, one cannot understand a 3-dimensional vector representing a spacecraft's location without defining an origin (e.g. the Earth, Sun, or Mars barycenter) and a coordinate system (e.g. Earth Mean Equator 2000). Similarly, a quaternion representing a spacecraft's orientation has no physical meaning without also specifying a reference coordinate system.

It is always possible to treat such relative states as absolute by adopting a standard reference for all objects and specifying everything relative to that. This, however, is usually inconvenient. Core relationships are obscured since they are not explicitly modeled. Further, such absolute representations can be numerically imprecise. This makes the software less understandable, less reusable, and less reliable.

Graph state variables are required to be able to derive relative states from others. For example, a voltage for a grid in an instrument is measured with respect to the instrument's chassis voltage. The instrument chassis voltage is offset from the spacecraft ground. The grid's voltage with respect to the spacecraft ground, therefore, is the sum of the two voltages. This also demonstrates the requirement for graph state variables to allow a point of reference to be used in the definition of more than one relative state. In the above

example, the instrument's chassis voltage reference is used to define the grid voltage as well as the chassis's voltage offset relative to the spacecraft ground.

A rover arm example demonstrates more compelling requirements for deriving relative states (see figure 3). A rover needs to autonomously control the relationship (**d**), the relative location of a rover's end effector (**D**) with respect to a rock on the Mars surface (**E**). This relative state may be derived from a combination of relative states for location. Some states are estimated from measurements of arm joint angles: the effector with respect to the forearm (**D** *w.r.t.* **C**), the forearm with respect to the upper arm (**C** *w.r.t.* **B**), and the upper arm with respect to the rover body (**B** *w.r.t.* **A**). The rover body location with respect to the rock (**A** *w.r.t.* **E**), which may be estimated using binocular vision algorithms, is also a constituent of the derivation. The relative state for the end effector's location w.r.t. rock may be derived as **D**->**C**->**B**->**A**->**E**. When the end effector is close to the rock, the effector's proximity sensor may provide a more accurate estimate of the rock's location. In this case, the directly measured estimate of the relative state **D**->**E** may be used instead of the longer derivation above. This demonstrates that graph state variables are required to be able to represent multiple derivations for the same relative state, and to be able to select between them. The selection between allowable derivations may be based on a property of the derivations, such as their accuracy.

A rover traversal example demonstrates that graph state variables are required to accommodate the addition of new relative states and points of reference. A rover's location state relative to a landing site may be derived from a sequence of location offsets between waypoint locations. Each offset between two waypoints is a relative location state, and each waypoint is a point of reference. New waypoints are identified as a new target for each traversal the rover makes across the Mars surface. New relative location states may be created as each new waypoint is identified. Each new relative location state is the location of the current waypoint with respect to the previous waypoint. The derivation for the rover's location with respect to the landing site is computed from the sequence of relative location states going back to the landing site.

A similar example is an interplanetary spacecraft that encounters scientific targets of opportunities during its mission. The locations of new targets are relative states that may represent the trajectory of a planetary object relative to another (e.g. a new comet relative to the Sun's barycenter, or a new moon relative to a planet's barycenter). The locations of such unplanned targets need to be added to the spacecraft's knowledge. Further, to point at a new target, the spacecraft attitude control system needs to compose the relative location of the new target with other previously known location states to derive the location of the object with respect to the spacecraft.

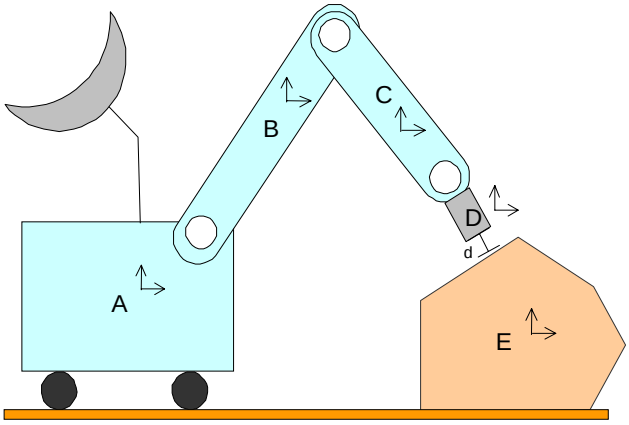

Figure 3. Rover arm example.

In addition to being a point of reference for multiple relative states of the same kind, graph state variables are required to allow a node to be a point of reference for more than one *type* of relative state. For example, a spacecraft gyro may be a point of reference for describing its location and orientation as a relative state. It may also be a point of reference for its voltage relative state. Finally, it may also be a point of reference for its connectivity relative state on the onboard communication network.

The examples above demonstrate requirements on graph state variables to define, organize, and derive relative states that represent relationships between points of reference.

4. Relative State Representation Today

Today's spacecraft represent relative states usually in an implicit rather than explicit manner. For example, simple relative states such as voltages are generally implicitly understood to be offsets with respect to the spacecraft chassis ground. Also lacking in current space applications is a systematic representation of uncertainty. Some detailed examples from real spacecraft applications follow to demonstrate other areas of improvement that are addressed by graph state variables.

Spacecraft attitude control systems typically estimate the spacecraft orientation relative state using onboard sensors that measure spacecraft rotation rates and directions to stars, the sun or nearby planetary bodies. Spacecraft orientation and directions to solar system bodies and stars are implicitly defined with respect to a coordinate frame reference (e.g. an inertial coordinate frame such as EME 2000). In many missions, there is no explicit representation in the onboard attitude control system of the spacecraft trajectory, or the orbits and rotations of the planetary bodies. When target directions are needed, the ground navigation system computes orbits and rotations using detailed ground based models and converts them into simplified propagation models. These models approximate the directions in the coordinate frame that the spacecraft implicitly uses and are uplinked to the onboard attitude control system. It in turn

uses them to produce directions at needed times for orienting the spacecraft, pointing its antennas and instruments, and avoiding harmful sun rays.

The pointing system model that resides in the Cassini spacecraft attitude control system improves on this situation [3]. It explicitly represents the orbits of planetary bodies, the spacecraft trajectory, and orientations between coordinate frames. Translations between planetary bodies and the spacecraft, and rotations between coordinate frames are organized as edges in a tree. Each edge either represents a relative position or relative rotation between nodes. The relative positions are vectors between two location nodes (e.g. a planetary body and the Cassini spacecraft.) The relative rotations are quaternions between two coordinate frame nodes (e.g. EME 2000 and the spacecraft local coordinate frame). Some nodes are used as reference points for both locations and coordinate frames. The direction of a planetary body relative to the spacecraft is computed by first finding the path in the tree between the body and spacecraft. The direction is computed by adding the sequence of vectors and applying the rotations along the path. A found path can be reused for computing a direction at a later time without having to search the tree. The onboard pointing system propagates the edge information as functions of time. The relative locations in the tree edges are uplinked conic or polynomial functions, and the relative rotations in the tree edges are linear functions using constant rotation rates. These functions and rates are commanded by the ground and are computed from more detailed ground based navigation models.

JPL's Deep Space 1 mission has an onboard navigation application called AutoNav that also contains a model of the spacecraft trajectory; however, it can autonomously update the model of the trajectory using asteroid sightings. AutoNav explicitly represents and propagates trajectory and orbit models within the navigation software, but in a form that is not directly understood by attitude control. Attitude control queries AutoNav for a direction to a planetary body, and AutoNav returns a direction that is implicitly in the Earth Mean Equator 2000 coordinate frame. Attitude control rotates this vector into the spacecraft coordinate frame using its estimate of the spacecraft's attitude. AutoNav computes the direction for specific time by evaluating a polynomial function. When the flyby geometry changes too fast for repeated AutoNav queries, AutoNav provides attitude control with a first order approximation of the dynamics of the changing direction (an initial vector and a velocity vector). In this case, attitude control rather than AutoNav propagates the direction. AutoNav receives models of planetary orbits and spacecraft trajectory from the ground in the form of ephemeris files containing the polynomial coefficients. The ground software creates these by reducing a yet more accurate orbit representation in the ground navigation system. These more detailed ground representations are composed of sets of trajectory, ephemeris, and orientation files that represent relative location and orientation states. These relative locations and orientations are defined in terms of a set of coordinate frames and fixed locations corresponding to the spacecraft and planetary bodies. The relations are organized using a tree topology, where each coordinate frame or location is identified as a node in a tree. Relative translations and rotations are organized as edges between the nodes in two trees. One tree contains all the translations and another contains all the rotations. These translations and rotations can be composed to derive new relative states by following paths within the trees.

The Mars Exploration Rover is planning to use a set of frames for translating between the current rover location on the surface, its previous Mars surface positions, and its next target location. The frames also represent the rover's orientation with respect to the Mars surface. The rover's arm also has a system for representing the relative states between its elements and science targets.

Common to these applications is the observation that only a single derivation is allowed because of the use of tree topology to represent relationships. In addition, there may not be a systematic representation for the same relative states shared between various mission applications in the same or neighboring system. Further there is no explicit representation of uncertainty in the applications' software architecture. MDS graph state variables provide the same functionality as these applications while also overcoming their shortcomings.

5. MDS GRAPH STATE VARIABLES

MDS graph state variables (GSVs) provide a general graph-based state representation for expressing relative states and their uncertainties in an explicit fashion. A summary definition of graph state variables is described below and explained further in following subsections.

Definitions

The following is a summary definition of GSVs. A *node* within a graph state variable names point of reference. A *relationship* within a graph state variable is a relative state between two nodes. Relationships have uncertainty and direction and relationships are transitive and invertible. A relationship's direction starts at its "from" node and ends at its "to" node. A *direct relationship* within a graph state variable is abstracted as an edge between two nodes, where the edge references a state variable that holds or derives the value and uncertainty of the relationship between the two nodes. The collection of edges and nodes in a graph state variable form a graph. A *path* within a graph state variable is a sequence of nodes, where adjacent nodes in the sequence must have an edge between them in the graph, and no two adjacent pairs of nodes can be repeated. A *derived relationship* within a graph state variable is a relationship computed by concatenating the relationships along a path that has 3 or more nodes. A *derivation* is cached path that a graph state variable can re-evaluate to compute a derived relationship. There are three kinds of graph state variables:

basis, proxy, and composite. A *basis graph state variable* is comprised of direct relationships that are computed from basis or derived state variables. A *proxy state variable* is comprised of direct relationships that are computed from proxy state variables. A *composite graph state variable* is comprised of graph state variables that are connected by sharing one or more nodes. *Dependencies* represent correlations of knowledge between direct relationships in a graph state variable and are stored in a *dependency state variable*. The Unified Modeling Language diagram in figure 4 depicts these definitions as architectural elements within the context of MDS estimators.

Nodes and Direct Relationships

A graph state variable knows about a set of reference points, called *nodes*. Each node in a graph state variable has at least one *direct relationship* with another node in the same GSV. Each direct relationship is a relative state with an uncertainty and is abstracted as a directed edge between two nodes in the graph state variable's graph. A direct relationship's direction starts at its "from" node and ends at its "to" node. The set of nodes and edges in a graph state variable comprise a graph state variable's graph.

A graph state variable only knows about direct relationships of a certain type. For example, a voltage graph state variable only knows about relative voltage states between its nodes. Other types of graph state variables may be for telecommunications gain relationships, network connectivity relationships, or location or rotation relationships. If a node is used as a point of reference for more than one type of relative state, then the same node may be known by graph state variables of different types. For example, a node representing a spacecraft's gyroscope may be used in both a rotation graph state variable and a communications connectivity graph state variable.

The rover arm example in section 3 can be described in terms of a *6 degree of freedom* (6 DOF) graph state variable. Each of its direct relationships is a 6 DOF transformation. A 6 DOF transformation contains both a translation, a rotation, and their derivatives. Three degrees are the x, y, and z coordinates of its location. The other three represent the rotation as a unit vector and the rotation angle about it. A 6DOF relationship in the GSV represents a combined relative location and orientation state that has direction and uncertainty, and is transitive and invertible. Each node in this GSV is a 6 DOF reference point for its object, where the object is a rigid body. 6 DOF reference points are called *frames*. Each has a coordinate frame with:

(1) The coordinate frame's origin fixed to a location for its frame's object, and
(2) The coordinate frame's axes fixed within the frame's object.

A 6 DOF GSV for the rover arm example knows about nodes for the end effector frame (**D**), the forearm frame (**C**), the upper arm frame (**B**), the rover body frame (**A**), and the rock frame (**E**). These frames can be seen as nodes in figure 5. The relationship *d* is the 6DOF transformation from the end effector frame (**D**) to the rock frame (**E**). Similar relationships exist from frame A to frame B (denoted R_{AB}), as well as R_{BC} , R_{CD} , and R_{AE}. If two of the nodes' coordinate frames are in alignment, for example the Mars landing site and the rock, then the 6 DOF direct relationship between them would be a translation with a null rotation.

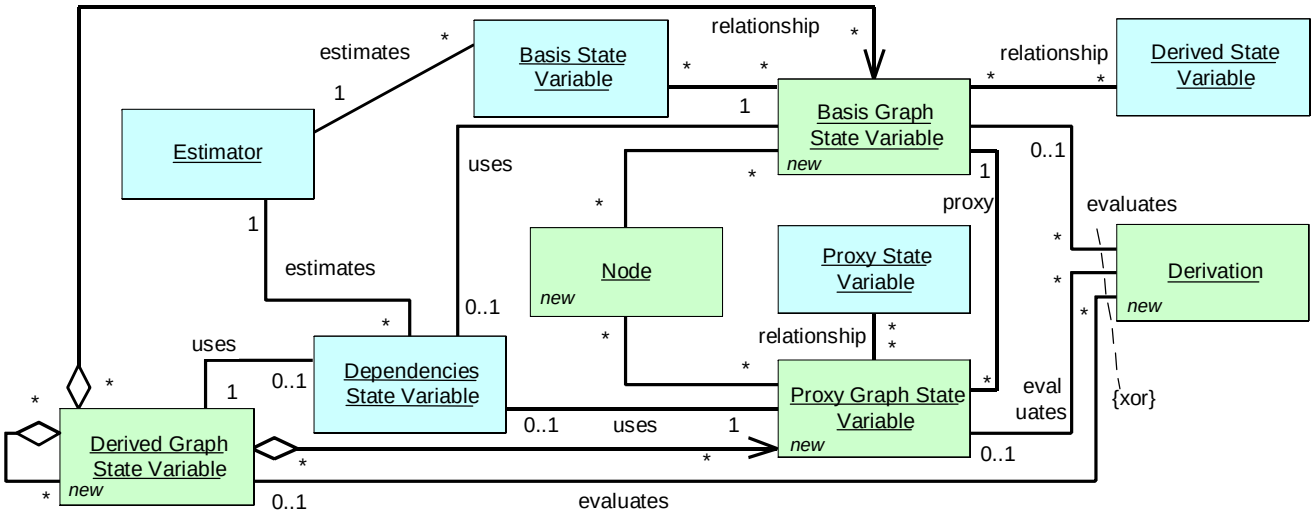

Figure 4. Unified Modeling Language (UML) representation of graph state variable architectural elements (green) in the context of other MDS architectural elements (blue).

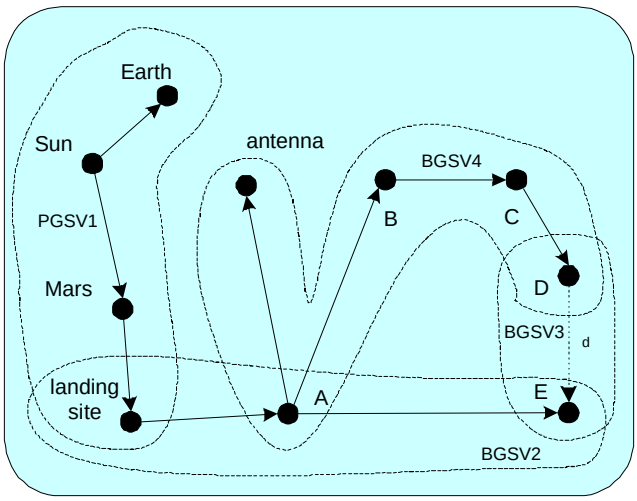

Figure 5. Rover arm example graph state variable.

Each GSV direct relationship/edge may be explicitly represented in the software as its own state variable. On the other hand, some direct relationships/edges may be may be represented in a more implicit fashion. For example, each may be computed from a different element in the same state variable. This case is discussed further in the derived relationships section below.

Derived Relationships

A graph state variable computes the relative state between any two of its nodes. The nodes may be non-adjacent; they may not have a direct relationship between them. A GSV can derive a relative state's value for non-adjacent nodes in the GSV by concatenating the direct relationships along a path between the two nodes. A *path* within a graph state variable is a sequence of nodes, where adjacent nodes in the sequence must have an edge between them in the graph, and no two adjacent pairs of nodes can be repeated. GSVs require that a relative state relationship type be invertible and transitive. For a relationship type to be invertible, a value of this type for the relationship from node **A** to node **B** implies that a relationship value from node **B** to node **A** can be computed. Note that an inverted relationship value is allowed to be "unknown" (i.e. having and uncertainty of 1) and still be defined. For a relationship type to be transitive, instances of it concatenated together must also be an instance of the relationship type. For example, if a direct relationship instance R_{AB} exists from (**A**) to (**B**), and if another R_{BC} exists from (**B**) to (**C**), and both are instances of a transitive relationship, then a *derived relationship* instance R_{AC} exists for the relationship from (**A**) to (**C**). A derived relationship is computed using a concatenation operator (•) defined for each relationship type. Thus $R_{AC} = R_{AB} \bullet R_{BC}$, where R_{AC} is of the same type as R_{AB} and R_{AC}.

In the rover arm example above, the relationship type, a 6 DOF transformation, is invertible and transitive. The relative state for the end effector's (**D**) position and orientation with respect to the rock (**E**) can be a derived relationship. Its derivation is the concatenation, and inversion where needed, of the direct relationships R_{CD}, R_{BC}, R_{AB}, and R_{AE}. More explicitly, a derivation for R_{DE} is $(R_{CD})^{-1} \bullet (R_{BC})^{-1} \bullet (R_{AB})^{-1} \bullet R_{AE}$.

Note that a direct relationship may itself be a derived state variable. In a new graph state variable example, a rover's traversability relationship between locations on a map may be computed from a sequence of moves within a grid. Each move has a traversability relationship. The traversability for any particular move may be computed from an NxN elevation map state that is estimated from camera observations. It would make no sense to pre-compute the traversability relationship of each possible move. The number of direct traversability relationships would be very large. Further, many are in the wrong direction or may be eventually discounted as being too difficult to cross. One approach is to compute traversability relationships only as needed by accessing a derived state variable that computes the traversability between two adjacent locations in the elevation map. In this case, a GSV may represent direct relationships as queries on the derived state, rather than by storing each direct relationship within the GSV.

Alternative Derivations

A graph state variable can select between alternative derivation paths. A graph can have loops that produce multiple paths between two nodes. This creates the possibility of two or more potentially different relationships between two nodes (given that the information in the GSV is never perfect). This is useful when there needs to be more than one possible way to derive a relative state. In this case, an application includes a *path arbitration criterion* that the graph state variable uses to select between possible derivations. If the user needs only "reasonable" derivations, the adapter can provide a *pass/fail criteria*. In this case, the graph state variable will only return derivations that pass the criteria. If a user needs only the *n* "best" derivations, then the adapter can provide a *utility function* that the GSV uses to return the *n* derivations that have highest value.

The rover arm example above has two possible derivation paths for the distance, **d**, between the end effector (**D**) and the rock (**E**): **D**->**E** and **D**->**C**->**B**->**A**->**E**. The GSV should return the distance directly sensed by a proximity sensor on the end effector (**D**->**E**) when it is close to the rock. In this case the discriminating factor may be the derivation's uncertainty. The directly sensed distance may have a much smaller uncertainty that the indirect longer derivation when the end effector is close to the rock. The utility function would simply return the certainty for each derivation passed to it. The GSV would choose one derivation (i.e. n=1) that has the best value (e.g. greatest certainty).

Graph state variables also allow for the use of both a utility function and a pass/fail criteria together. In this way an adapter can have the GSV return the *n* best derivations that satisfy a pass/fail criteria.

Basis and Proxy Graph State Variables

A GSV is composed of a collection of nodes and the direct relationships that connect them. Each direct relationship is stored in a state variable. GSVs whose direct relationships are basis or derived state variables are called *basis graph state variables* (BGSVs). Each basis graph state variable contains direct relationships that are estimated in the local deployment by the same estimator. The proxy pattern for state variables is repeated here for GSVs. A *proxy graph state variable* (PGSV) is composed of direct relationships that are estimated in a separate deployment by the same estimator. Each direct relationship referenced in a proxy GSV is stored in a proxy state variable. In a sense, a proxy GSV is a mirror of a basis GSV.

In the rover arm example discussed earlier, basis graph state variable BGSV2 contains rover estimates of the 6 DOF relationship between its location (**A**) and the landing site. BGSV2 also contains estimates of the location of the rock (**E**) relative to the location of the rover (**A**). Similarly, the rover estimates the 6DOF relationships for its articulated antenna and upper arm (**B**) relative to the location and coordinate frame of the rover body (**A**) and stores the estimates in BGSV4. The 6DOF transformations between the rover arm elements are also estimated by the rover and stored in BGSV4. All transformations for the articulated elements are stored in the same BGSV because they are correlated and estimated by the same estimator. Correlated relationships will be discussed further in the dependencies section below. The rover's estimate of the rock's (**E**) 6 DOF relationship relative to the end effector (**D**) using the proximity sensor is stored in BGSV3. What remains are the relationships that are estimated on the ground and copied into the proxy graph state variable PGSV1. These relationships include orbits for the Earth and Mars relative to the Sun, and the location of the landing site on Mars relative to the Mars frame.

Composite Graph State Variables

A GSV may be composed of other GSVs. Such a GSV is called a *composite graph state variable* (CGSV). A CGSV can derive new relationships from the relationships that are in its various constituent GSVs. Common nodes that appear in more than one of the constituent GSVs provide the bridge that links them together. This allows relationship derivation paths to extend over different GSVs. A composite GSV can link sets of direct relationships that are estimated in both local (Basis GSVs) and remote (Proxy GSVs) deployments using this mechanism. Further, a composite GSV allows for a hierarchical organization of sets of relationships. A composite GSV is not limited to be composed of BGSVs or PGSVs. It may have as constituents other composite GSVs. A hierarchy of collections of relationships can be created, by having CGSVs, within CGSVs, within CGSVs, etc.

The rover arm example above has a single composite graph state variable named the Frame Composite GSV. It is composed of 4 constituent GSVs: PGSV1, BGSV2, BGSV3, BGSV4. The landing site node links the proxy graph state variable for ground estimated navigation 6DOF states (PGSV1) and the rover estimated navigation 6DOF states (BGSV2). The rover body node (**A**) links the rover estimated navigation states with the 6DOF relationships between the articulated elements (BGSV4). Finally, BGSV3 is a bit of a special case. It contains the directly measured relationship between the end effector (**D**) and the rock (**E**). Thus, it provides two links which allow comparison of the estimate of the rock's location (**E**) using navigation and vision algorithms (BGSV2) and rover arm kinematics (BGSV4) with the directly measured relationship (BGSV3).

Dependencies

GSVs use *dependencies* to compute the uncertainties of derived relationships. Dependencies represent correlations of knowledge between the direct relationships in a graph state variable. Dependencies are stored or derived in a state variable referenced by its GSV.

For example, suppose two direct relationships in a GSV are *A* and *B*, where each has an uncertainty that is represented as a normal distribution with variances σ_A^2 and σ_B^2, respectively. Then the dependency in the GSV may be the covariance of *A* and *B*, denoted as *cov*(*A*,*B*). If the derived relationship A•B is computed using subtraction then A•B = A-B and the GSV computes the uncertainty of A•B as $\sigma_A^2+\sigma_B^2-2cov(A,B)$. Other types of dependencies are possible, depending on the nature of the relationships.

The rover arm example above contains 3 direct relationships in BGSV4 that represent relative locations and orientations of the rover body, upper arm, forearm, and end effector. Suppose each direct relationship is estimated using measurements from different potentiometers, each of which measures a particular joint angle. If the same A-to-D converter digitizes the voltages from all potentiometers, then a bias in the A-to-D converter affects all conversions. Thus, each estimate of a rover arm direct relationship is offset by the same bias and is correlated with the other rover arm direct relationships. In this case, the dependency state variable contains the correlations needed to compute the uncertainties of the derived relationship **D**->**A**.

Dynamic Topology

Some missions need to add relationships and nodes as new objects of interest in the environment are identified. Examples include science targets identified by a rover along the surface of Mars, opportunistic spacecraft observations during flybys, and asteroids added for optical navigation purposes. Once these targets have been observed, they may no longer be relevant to a spacecraft's future operations. In

these cases their relationships and nodes should be deleted. In some cases, the set of direct relationships changes without adding new nodes. For example, spacecraft position may be estimated with respect to the Earth immediately after launch. Later during interplanetary cruise, it is estimated with respect to the Sun (or the Solar System Barycenter). Finally, it may be estimated with respect to a planetary body for an upcoming flyby or orbit insertion. These planetary objects are typically identified and tracked prior to launch; however, a new direct relationship to represent trajectory may need to be established when a spacecraft comes under the gravitational influence of a nearby planetary object.

These examples demonstrate there is need for graph state variables to accommodate the addition and removal of relative states and reference points. To meet these necessities, graph state variables include methods for adding and deleting nodes and direct relationships. The underlying MDS component software architecture provides for adding and connecting to a GSV new state variables that correspond to new direct relationships. It also allows for disconnecting and removing state variables. GSV path arbitration criteria that favor certain derived relationships will guide the use of a new direct relationship rather than less desirable derivations.

Points of Reference

Each node in a GSV may be used as a point of reference by other states. For example, the direction of a camera boresight and the force direction of a thruster are states that need to be defined in terms of a coordinate frame point of reference. 6 DOF graph state variables that have frame nodes for the instrument and the thruster provide frames of reference for their directions. A 6 DOF GSV may be used to rotate the camera's boresight direction into the correct frame for comparison against the Sun's direction. Hazard avoidance algorithms may use this to prevent pointing the camera into the Sun. Similarly, a 6 DOF GSV may be used to rotate the thruster's force direction into the same frame as the spacecraft center of mass for computing imparted torques. In general, 6 DOF GSVs provide a mechanism to relate vectors (and tensors for moments of inertia) that are defined in terms of different frames of reference.

Performance

The search for a path between two nodes may be too slow for applications containing large graphs or high rate derivations. Graph state variables have a number of features that allow an adapter to reduce the time for deriving a relationship.

Cached Paths—Graph state variables provide a capability for returning derived relationships in a small amount of time for frequently repeated queries. If a repeated derivation is along the same path, an application may save the path returned by a query. A saved path is called a *derivation*. A derivation may be re-evaluated in subsequent relationship queries to avoid the overhead of searching the graph. Derivations may be automatically invalidated and a new path computed when a direct relationship along the path changes or a change in the graph's topology warrants it. A graph state variable may also keep relationship data about "now" more quickly accessible that data for other, less frequently requested times.

Domain Specific Search Algorithms—In addition to providing generic search algorithms for finding paths, graph state variables allow an application to specify its own search algorithm. This enables an application to use knowledge of its domain to improve its graph search performance.

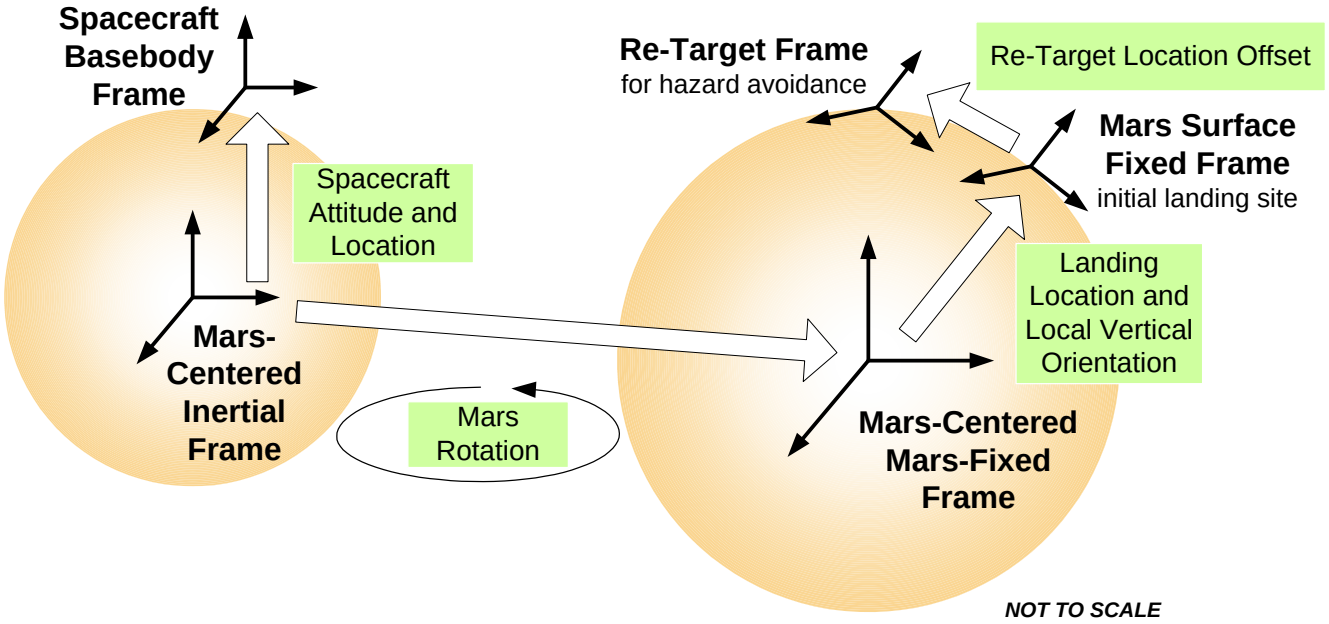

Figure 6. 6 degree of freedom navigation frames and relationships for a Mars lander prototype.

Search Algorithm Hints—Graph state variables provide a capability that allows an adapter to provide search hints. A hint can be specified in a relationship query and used by the search algorithm to direct searches. A hint may take a variety of forms. For example, the search algorithm may interpret a path arbitration criterion as a search hint.

Another may be a partial path screening criterion. A GSV may use it to eliminate candidate partial paths during each step of its graph search algorithm. A partial path screening criterion is like a pass/fail path arbitration criteria and it applies to partial paths. Many possible criteria are possible. Examples are functions of the path's nodes, direct relationships, or the relationship the partial path represents.

Another type of hint is a partial path prioritization function. The search algorithm uses it to select the next path explored during each step of a graph search algorithm. A partial path prioritization function is like a path arbitration criteria utility function except it applies to partial paths.

6. MARS EDL EXAMPLE

6DOF navigation relationships in a Mars entry, descent, and landing GSV prototype adaptation is depicted in figure 6. The spacecraft attitude and location is a relative state that represents the translation and rotation of the spacecraft basebody frame with respect to the Mars-centered inertial frame. The Mars-centered Mars-fixed frame is a rotating frame with respect to the Mars-centered inertial frame. The Mars surface fixed frame is the initially targeted landing site and is related to the Mars-centered Mars-fixed frame by a location offset from the center of Mars, and by a rotation that has the Z-axis vertical and X and Y axes along Mars longitude and latitude lines, respectively. The re-target frame is a location offset from the Mars surface fixed frame used if a hazard is detected during landing.

In addition to these navigation frames, there are frames to represent locations and orientations for the following spacecraft elements:

- the location of the spacecraft center of mass,
- the location of spacecraft roll thrusters,
- the location of descent engines, and
- the location and orientation of an ideal sensor that measures the spacecraft location and orientation.

(A true flight application would contain separate frames for

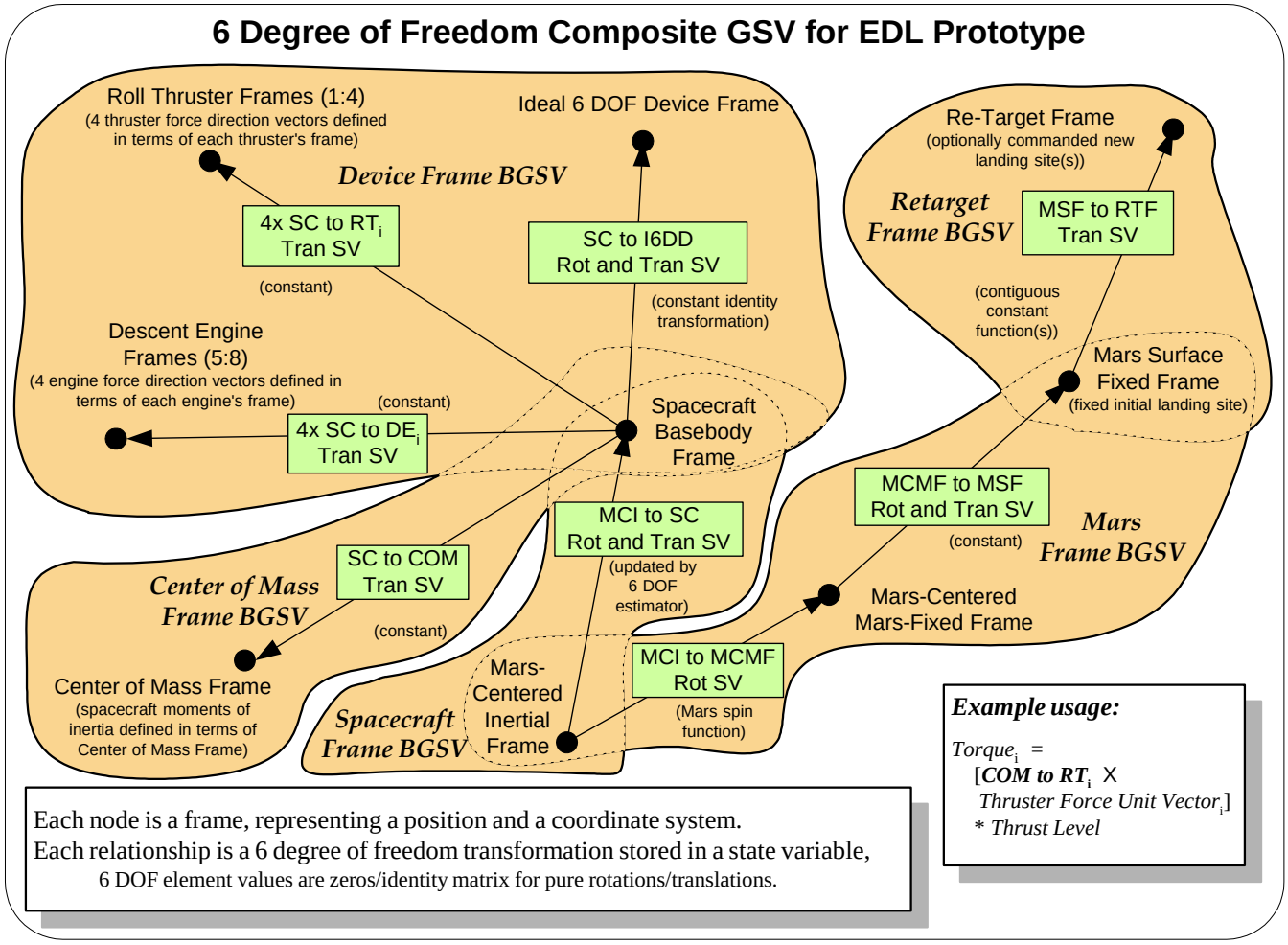

Figure 7. Graph state variables for a Mars lander prototype.

rotation measurement devices (e.g. gyroscopes), accelerometers, a star tracker, radar and lidar. These have been abstracted for the purposes of the prototype and will be added in future versions.) The spacecraft center of mass frame relative to the spacecraft basebody is a location offset represented as a 6 DOF relationship with a null rotation. Similarly, the thruster and descent engine frames are defined in terms of relative states that are only translations with respect to the spacecraft basebody frame. Finally, the ideal sensor is just a stand in for more realistic sensors, so its frame's relationship with the spacecraft basebody frame is a constant identity transformation (null rotation and translation).

All frames and relationships are collected into 5 basis state variables within the frame composite graph state variable shown in figure 7. Each of the relationships is stored in a basis state variable. The thruster, descent engine, and ideal sensor frames and their relationships are gathered into the same basis GSV because these represent calibration parameters for device alignments. Such parameters are typically estimated on the ground and uplinked together to the spacecraft. Thus, they are collected into the same GSV. If they were actually uplinked from the ground, these relationships would be stored in proxy state variables and collected into the same proxy GSV. The center of mass offset from the origin of the spacecraft basebody frame would be estimated from fuel depletion in its own estimator and is, therefore, in its own basis GSV. The center of mass offset would be derived from two component state variables: one a static uplinked parameter based on the spacecraft dry mass, and the other derived onboard from a propellant state variable. The spacecraft attitude and position 6 DOF relationship (MCI to SC) is in its own basis GSV because it is estimated by the spacecraft in a separate 6DOF estimator. The relationships between the Mars inertial and fixed frames are in the same basis graph state variable because they would be estimated on the ground by the navigators. Again, in a real application, these would also be proxies. The re-target frame position offset is within its own basis GSV because a separate onboard algorithm for hazard avoidance will eventually estimate it using radar and lidar measurements.

Thruster and descent engine force direction states are defined as unit vectors in terms of reference points for their individual frames. The torque for roll thruster "i" is computed by the following formula:

$Torque_i$ =
[COM to RT_i X $Thruster_i$ Force Direction] * $Thrust\ Level_i$

COM to RT_i is the derived location relationship along the path from the Center of Mass Frame, to the Spacecraft Basebody frame, and then to the frame for Roll Thruster i. $Thruster_i$ Force Direction has been rotated into the Center of Mass Frame using the inverse of the derived 6 DOF transformation along the same path.

The descent engine torques are computed in the same fashion. The spacecraft moment of inertia is another state that is defined in terms of a frame reference point. In this case it is the center of mass frame. This along with the state for the spacecraft mass defines the spacecraft inertial properties.

The spacecraft position and orientation controllers control the relationship of the spacecraft basebody frame w.r.t. the re-target frame. Given an orientation and position profile goal, they compute the thrust levels using the previously described torque and inertial properties models and transformations in the 6 DOF composite GSV.

7. Systems Engineering of Relative States

MDS state analysis is a model-based process to aid systems and software engineering. State analysis prompts systems engineers to perform a methodical discovery process and rigorous analysis of mission requirements in terms of MDS architectural elements. These architectural elements have a one-to-one correspondence to specific MDS software frameworks that need to be adapted by software engineers to meet the requirements in the state analysis. In this way, the software implementation has a much better chance of meeting mission requirements.

Graph state variables extend MDS state analysis to provide both a systems engineering methodology for explicitly describing required relative states and an unambiguous way for transforming the requirements into elements to be adapted in the software architecture. The GSV extensions to state analysis are described below.

The system engineer determines the set of relative states required to meet mission requirements. The system engineer identifies the types of relationships that model the relative states required. MDS plans to develop relationship types for relative states that are common to space applications. If, however, a required relationship type is not already provided by MDS, the systems engineer determines the mathematical form for describing the relationship. In addition, he determines the formulas for inverting and concatenating relationships. These will be used in the adaptation of the MDS provided graph state variable framework to compute derived relationships. The software engineer creates a relationship software type that meets the relationship's mathematical properties and implements the invert and concatenation operators.

The system engineer identifies the points of reference (nodes) for each of the relative states required to be modeled. The system engineer identifies the direct relationships required to represent basis relative states, and the derived relationships that use them required for representing derived relative states. The system engineer determines for each direct relationship whether it will be represented explicitly between pairs of reference points as individual states. If there are many possible direct

relationships, he may determine an algorithm for deriving them from a composite state. In this case, the software engineer codes the algorithm used by the GSV for deriving direct relationships. If the direct relationships will be represented as individual states, the software engineer identifies each state. The software engineer instantiates each of these as a basis (or derived) state variable in the deployment (spacecraft and/or ground) that estimates (or derives) it. The system engineer identifies the basis state variables that are needed as direct relationships in other deployments, and the software engineer instantiates each as a proxy state variable in each deployment that needs it.

For each estimator in a deployment, the system engineer groups the direct relationships that are estimated by it, and the nodes the direct relationships share into a basis graph state variable for that deployment. Similarly, the system engineer groups the direct relationships stored in proxy state variables within the same deployment and the nodes they share into proxy graph state variables for that deployment. The software engineer creates software instances of these basis and proxy graph state variables within the deployments they belong. Finally, the software engineer builds a composite GSV in each deployment from the basis and proxy GSVs of the same relationship type.

8. Conclusions

The MDS architecture and graph state variables are a foundation for spacecraft software that can meet the ambitious autonomy challenges of future space exploration missions. Graph state variables meet key MDS architectural themes to explicitly model states and join attitude control, navigation, and robotics. They provide a common reusable framework that will allow easy management of attitude control, navigation, and robotics interactions and provide for shared systematically engineered data. Graph state variable six degree of freedom relationships provide a common mathematical base for representing relative states for spacecraft orientation, location, trajectories, dynamics, and kinematics. In addition, because graph state variables have been designed in an adaptable abstract model, they can be adapted for the power, telecommunications, and networking domains, and possibly others. Graph state variables provide a common framework for unambiguously and systematically describing system engineering requirements on relative states and a process for implementing them reliably in software.

In summary, graph state variables:

(1) explicitly represent relative states as relationships between reference points,
(2) explicitly model the composition of derived relative states from elemental relative states,
(3) combine multiple sets of relative states (sub-graphs) estimated separately or copied from remote locations,
(4) produce different results for different derivation paths and select between them,
(5) accommodate graph topology changes - addition or deletion of reference points and relative states,
(6) represent dependencies between relative states (e.g. correlations).

9. Acknowledgements

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

The authors thank the entire MDS team at JPL for their contributions, without which this work could not have been done. We are especially appreciative of the following MDS team members for invaluable graph state variable discussions: Daniel Dvorak, Sandy Krasner, Michael Lisano, Steve Peters, Zahidul Rahman, Nicolas Rouquette, and Marcel Schoppers.

References

[1] D. Dvorak, R. Rasmussen, G. Reeves, A. Sacks; "Software Architecture Themes in JPL's Mission Data System," *2000 IEEE Aerospace Conference Proceedings*, March 2000.

[2] D. Dvorak, R. Rasmussen, T. Starbird; "State Knowledge Representation in the Mission Data System," *2002 IEEE Aerospace Conference Proceedings*, March 2002.

[3] R. Rasmussen, G Singh, D. Rathburn, G. Macala; "Behavioral Model Pointing on Cassini Using Target Vectors," *18th Annual AAS Rocky Mountain Guidance and Control Conference*, February 1995.

Matthew Bennett *is a senior engineer in the Avionics Systems Engineering section of the Jet Propulsion Laboratory, California Institute of Technology. He has interests in spacecraft autonomy and has broad experience in spacecraft development, test, and operations, including the Galileo Jupiter and Cassini Saturn orbiter missions. He has developed mission software for fault protection, guidance and control, science data collection, performance analysis, and simulation. He holds an MS from the University of Washington in Computer Science, and a BS from the University of California at San Diego in Computer Engineering.*

Robert Rasmussen *is a principal engineer in the Information Technologies and Software Systems division of the Jet Propulsion Laboratory, California Institute of Technology, where he is the Division Technologist and the Mission Data System architect. He holds a BS, MS, and Ph.D. in Electrical Engineering from Iowa State University. He has extensive experience in spacecraft attitude control and computer systems, test and flight operations, and automation and autonomy — particularly in the area of spacecraft fault tolerance. Most recently, he was cognizant engineer for the Attitude and Articulation Control Subsystem on the Cassini mission to Saturn.*