Data Management in the Mission Data System

David A. Wagner
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 301-270
Pasadena, CA, USA
david.a.wagner@jpl.nasa.gov

Abstract - *As spacecraft systems evolve from simple embedded devices to become more sophisticated computing platforms with complex behaviors it is increasingly necessary to model and manage the flow of data, and to provide uniform models for managing data that promote adaptability, yet pay heed to the physical limitations of the embedded and space environments. The Mission Data System (MDS) defines a software architecture in which both control theory and end-to-end data management provide the primary guiding principles. This paper describes how the MDS architecture facilitates data accountability and storage resource management.*

Keywords: data management, data accountability, resource management, mission data system, MDS.

1 Introduction

As space missions become more complex with ever greater demand for automation, increasingly tight interoperability requirements between robotic vehicles, and unprecedented levels of human-robot interaction, the size and complexity of the data systems needed to manage them must also evolve.

An embedded control system, and particularly one that is remotely controlled, has two primary responsibilities: to effectively translate the intent of remote operators into actions, and to return information describing what happened. For scientific spacecraft, this can be summed up in a single statement: make good science observations, and return the results to where we can use them. While control is central to this process, it is equally a problem of data management. The data management challenge is to produce and store data, process it to convert it to more usable forms, move it across space and time and keep track of it in the process, and manage limited storage and transport resources. This paper will describe the Mission Data System data model and explain how it works with the goal-oriented MDS control architecture to enable precision data management.

The Mission Data System (MDS) defines a software system architecture for remotely operated embedded control systems. Based in control theory, it defines entities and relationships needed for state estimation and control. As the name implies, providing a robust and flexible framework for data management was one of the key objectives of the MDS. The references list several other papers that describe aspects of the MDS control architecture in more detail [2,3,4].

2 Data Management

The purpose of a robotic observation platform is to use its instruments to collect data and return that data to the humans who run it, usually across unreliable, intermittent, and latent transport media. The data transport connection presents a limiting factor to the science the system can do, particularly in deep space missions where the distances involved put extreme limits on the amount of data that can be exchanged, and create signal latencies measured in hours. Thus, it becomes critical to design a system that can effectively manage the process of making the observations in the first place, and ensuring that the resulting measurements are returned intact. The system has to be robust enough to get the maximum data back even in the face of a variety of possible failures such as a lost communications session, various hardware failures, or a system reboot. Because of these constraints, observations are usually planned out in great detail.

Even when execution proceeds nominally it is difficult to determine if all the data that could have been recovered was recovered. This turns out to be a tricky problem in traditional robotic data systems for a couple of reasons. One is that telemetry systems often collect and transport data in unlabelled pieces mixed in with other data, so without intimate knowledge of workings of the instrument, the observation, and the telemetry system, it is difficult to know whether the recovered data are complete. Specifically, it is difficult to know what was lost.

Secondly, and perhaps more important, is the fact that typical command-oriented control systems do not retain any information about the intent of the observations. This is because mission planners distill activity plans into commands, and then merge and sequence those commands with ones from other activities into a compact executable command sequence. This process removes information about what was really intended and where one observation ends and another begins. In space systems this usually happens on the ground. As an analogy, consider that it is difficult to infer a programmer's intent from looking at the compiled machine code. If everything works as planned, it may be possible to determine that the original plans were

achieved. But, when the inevitable anomaly occurs, it becomes very difficult to infer what part of the plan was or was not completed or what happened to the data. These were the types of problems that the Mission Data System was designed to solve.

3 Mission Data System

The Mission Data System was designed as a framework for building complete end-to-end data and control systems – not just the embedded part. Three top-level design principles were brought to bear:

- A control architecture based on explicit representation of controllable state [6, fig. 1].
- An architecture for the expression of control intent through the use of goals [3].
- A layered, integrated symmetrical data management and transport system.

The following section will briefly describe these principles, how they apply to the MDS system, and how they combine to enable data accountability.

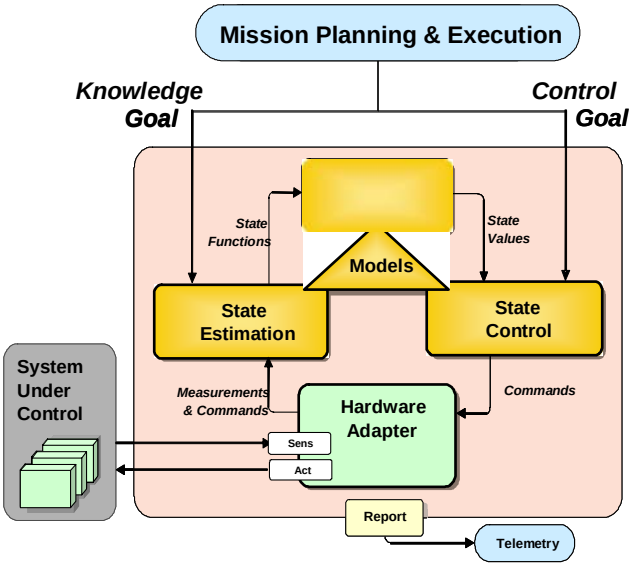

Figure 1 MDS Control Architecture

The Mission Data System's control architecture centers around the concept of state variables which provide common access to data representations of the physical states being controlled, including abstractions such as position and orientation, down to such primitive states as the position of a switch.

A state variable must be able to represent a given state continuously over time for any time of interest to the control system. Ultimately, the purpose of the control system is to cause this *state timeline* [fig. 2] to coincide with an *intent timeline* as expressed through the goals. That potentially means that a state variable needs to be able to represent a state over an infinitely long timeline from the distant past to the distant future. The representational requirements for any specific state variable depend on how the state is to be controlled. In practice, for the purposes of control, it may only be necessary for a state variable to provide a representation of the latest estimate of state. However, achieving a goal can sometimes require looking at estimates of state over an interval of time into the past. To allow planning into the future a state variable might have to provide information about extrapolated or predicted states, or *state projections*. State variables can also provide a historical record of what actually happened in the past in order to provide evidence for the estimation of some other state. For example, a record of thruster firings might be needed to compute trajectory. This historical record is also useful, in the form of telemetry, for later analysis of what actually happened.

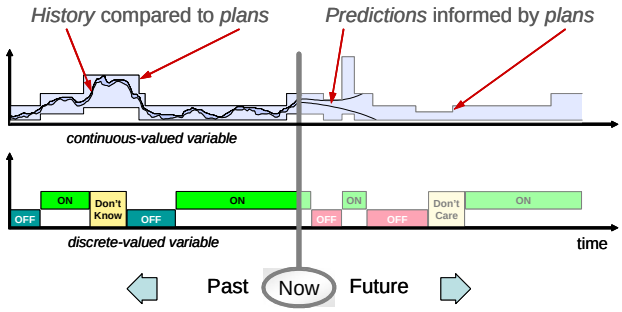

Figure 2 State Timeline

Control intent is expressed in MDS through the use of *goals*, which are defined as constraints on particular states over time. Through the use of goals, state variables can not only represent values that record past history, they can also represent intended future values as expressed in the goals. So, a state variable can be queried about future intended states: given your current set of goals, what state do you intend to be in at future time t?

The MDS control architecture requires that a given state variable must have exactly one estimator whose job it is to determine the state value as a function of the available evidence (other state values, measurements, and commands that have been issued by controllers).

The abstraction of the value function allows a given state variable to represent its values using different kinds of value representations over time. Different value functions can provide different levels of quality (precision, accuracy, uncertainty, or viewed from a data management perspective, size and complexity) over different periods of time. A state variable can provide high-resolution descriptions of state when high-precision control is needed, and lower-resolution representation when lower-resolution control is required. It can provide data when the state is being actively controlled, and no representation at all (i.e., unknown) otherwise.

Beside acting as a container for the state representation, the state variable is also responsible for providing transformational methods, or views, that allow users of that state to access it efficiently. Views make it

possible to keep the internal state representation distinct from what is presented to external users. For example, position along with all of its derivatives (velocity, acceleration) would typically be represented as a single complex state. If there were users who only needed to see the acceleration, a view could be provided to extract that element of the value and return it separately, possibly providing a more efficient interface for that user. A view could also apply a static transformation such as a units conversion.

3.2 Value History

A value history is simply a container for the data representation of a state value timeline within a state variable or a measurement or command value timeline. Value histories can be specialized to efficiently represent just the minimal extent of a state timeline that might be needed for a given state. Value histories can be specialized for continuously-represented states (intervallic histories), or for sample histories represented by a series of discrete values (discrete histories). Discrete value histories can be used in components other than state variables, such as hardware adapters, to record discrete samples such as science or engineering measurements, or commands.

Abstractly, the value history also provides an interface to deep storage and data transport. As such it can be viewed as a simple database mechanism. It provides a cache of values for immediate use, and a mechanism for pushing those values out to more efficient long-term storage. It provides the discrete values or value functions (and the query mechanisms for retrieving them from timeline containers) to service the component's interface requirements, and it provides the mechanism to bundle segments of the timeline into products for storage and transport.

One of the main jobs of the value history is to maintain a balance between the amount of data being cached in active memory in a form most readily useful to the control system versus the amount of data stored as products in more compressed or more persistent storage. In many cases the control system might need only a current estimate of the state under control, so the value history can push all older values out to storage for transport, or delete them as needed. But the abstraction allows for more complex interactions as needed. A value history of camera observations could, for example, immediately copy all images out to persistent storage and then retrieve them individually for analysis. This simple interface allows the system to easily move data between different storage media to manage resources.

3.3 Product-Oriented Data Management

Products bundle state data into manageable chunks that are meaningful to the application, and that are relatively efficient to store and transport. These bundles can incorporate compression and can be sized arbitrarily to optimize the particular transport or storage properties of the particular system. Thus, a product is essentially a file with some associated metadata describing the data it contains.

Product-oriented data management is already coming into common use in space missions as exemplified by the Mars Exploration Rovers (MER), and Deep Impact missions. While neither of these missions explicitly use any of the other MDS control architecture features, they both produce data products onboard, keep track of the content and meaning of those products through the use of external metadata, and employ a transport protocol to copy those products to the ground. The Consultative Committee on Space Data Systems (CCSDS) has gone so far as to develop an international standard product (file) transport protocol (CCSDS File Delivery Protocol, or CFDP [5]), suggesting a certain level of maturity and industry convergence around this concept.

The principle advantage of product-oriented data management is that it allows developers and operators to produce data in a form more naturally related to the observations and activities being performed by the spacecraft. Specifically, products tend to allow for more flexible content organization, eliminating most of the strict packaging constraints of traditional telemetry. Secondly, when transport bandwidth is at more of a premium than local storage, products can be stored and analyzed and subsequently transported or deleted based on the analysis. MER, for example, takes high-resolution images of a new location, stores them, and then derives thumbnail versions of each for downlink. Operators on the ground then review the thumbnail images and decide the disposition of the original images: if they look interesting they plan to send the full images; otherwise they delete them, or possibly save them to decide later. One can envision a future system in which this triage is performed onboard using a feature-detecting algorithm. This kind of analysis is impossible in systems where data is stored in the form of transport frames or packets because information about the aggregate structure and content of the data is not available at that level.

3.4 Goal-Oriented Control

As described in [2,3], state control in MDS is achieved through the use of goals, which express intent in the form of constraints on states. Goals that constrain the system to have knowledge of a given quality about a state of interest (so the state can be controlled) are generally referred to as *knowledge goals*. These are distinguished from *control goals* that are used to constrain external physical states of the system represented by state variables.

The simplest kind of knowledge goal is one that simply requires the state to be known (i.e., other than unknown), which can be achieved by enabling an estimator to determine a current value of the state. This sort of control makes it possible to disable the production of state estimates when they aren't needed, possibly saving power, memory and processor resources.

An example of a knowledge goal would be a constraint to have an estimate of a given temperature with a standard deviation of less than 1 deg C within a given time period, or to have knowledge of a switch position with 95% certainty in a given time period.

The distinction between knowledge goals and control goals is made mostly for descriptive purposes because in practice they're usually expressed as part of the same goal. That is, for many control goals where the knowledge quality can vary it doesn't make sense to express a control goal without explicitly qualifying a knowledge requirement as part of the goal. So, a goal to point the camera at a target with an accuracy of 0.1 degrees only makes sense if you ensure that you actually know the pointing state well enough to control it to that degree of accuracy. Knowledge goals are primarily used to control the estimators that produce state information.

3.5 Goal-Oriented Data Control

Details describing how much data has actually been created, and where and how it is stored are called the *data state.* Because these details are inherently part of the control system rather than the system under control, they represent a special case of states that need to be controlled. thus, data states are represented by specialized state variables that cannot have any history or meta state of their own, but they can be controlled using specialized *data goals*.

The concept of data state can include information about how and where the data are stored, what downstream processing such as compression has been performed. It can include metadata labels describing transport intent (where it needs to go, how important it is to get it there), and transport status (whether, or how completely transport has been accomplished). So, a data goal might constrain a value history to produce telemetry, or to store certain values persistently. Data goals might also constrain the amount of memory available to a given value history.

Not all state variables have or require any data state. In many cases it is possible to design a control loop where the state representation and the quality of this representation are static. For very simple control systems this might not even compromise efficiency. For example, a thermostat may only really need to know a reasonably current value of the temperature in order to control it in a realtime control loop. Thus, the implementation of this state variable might reasonably optimize away its ability to answer questions about values in the past or future, or to represent values with different precisions. If the state is simple enough and no telemetry is needed, then no data state is needed.

Management of the amount of state data that is collected for transport is done using data goals, or constraints on the data states of value histories. Data goals specifically constrain attributes of the data state that are affected after the data is produced, such as how long the information is retained in memory, or how much of it should be packaged for transport.

Data goals can be used to constrain the amount of history that is remembered for a given state, measurement, or command, or the quality of the data persistence (e.g., how recoverable is that state value after a processor reset). They can constrain activities that package the values for transport, and processes that would compress the data to maintain a given quantity of history within a given storage constraint. Data goals can indirectly influence data transport, but because of the complexity of the transport process they have to do so in a particular way, as discussed in the following section.

3.6 Data Transport

Data transport is in general an unreliable process because bits sent across any medium can be lost or corrupted in transit. This is particularly true of space links where extremely long distances are involved. The goal of data transport is primarily to have the data (all the data, and exactly the data) at the receiving end of the link. This requires coordinated actions on both ends of the link. Ideally, an automatic reliable transport protocol would be used to send the data between deployments. But the long latencies and bandwidth limits of space communications often make this impractical. A goal-based system would use a similar protocol except that the decision to request retransmission of missing pieces would be under the control of another goal. Thus, the system could reason about the relative priority of completing one goal versus proceeding on to something else (and on the ground it would possibly give human operators a chance to adjust the priorities). So, the goal to have data on the ground would elaborate into some goals on the ground system (to wait for the data), and some goals on the spacecraft to try to send the data. The only control actions available on the spacecraft are to try to send the data and then wait for some external acknowledgment that it was received, or that the goal failed, Once the transport goal is completed (positively or negatively), the system is free to recover the storage space by deleting the local copy of the data.

3.7 Data Management

Data goals provide a mechanism to achieve two key requirements that enable data accountability: they can provide information to associate the transportable data to the observation requests, and they associate requests with models that can be used to predict how much data to expect when a given goal is accomplished. For example, a goal to take a picture implies a subgoal to store the image data whose size can be inferred from the characteristics of the camera or the picture-taking goal. Every goal created in the planning process and uplinked to the spacecraft has a unique identifier. Data products created as the result of a goal execution can be explicitly and persistently labelled as being that goal's products, allowing the ground system to unambiguously associate them with the original observation request. Goals can directly or indirectly cause

data to be produced. This information by itself allows the products to be identified and reconstructed on the receiving side.

In the case of science observations where the point of the observation is to return data, the goal can explicitly specify the products to be returned. Goals such as this make the planning process much simpler because they elevate data production and associated resource usage from a side-effect to a first-class intent. If the original goals are met, the expectations expressed in the goal can be matched up with the received data to determine overall success.

Indirect data production can result from goals whose main purpose is to effect some physical control on the system, or perform some high-level operation. The MDS goal-oriented planning process allows high-level goals to be elaborated into supporting lower-level goals. The hierarchy of parent-child associations is explicitly tracked to enable traceability of intent. Thus, if telemetry visibility is required for a control maneuver, the top-level goal for that maneuver might elaborate a subgoal to produce the required engineering data products. Background or default goals could also be used to produce telemetry products as a default case when no other explicit goals are in effect.

When the top-level intent is expressed as a data goal, resource management also becomes simpler because the amount of storage and transport bandwidth that will be needed is a direct function of the size of the product, which is explicit in the goal. Similarly, the goals can express relative priorities which can apply transactionally to entire goals. So, when a resource conflict arises the information is available to completely stop achieving one goal or the other, rather than partially failing both.

4 Conclusions

Preserving intent in the control system through the use of goals helps to make the control system more robust and it can provide a way to unambiguously account for the data. The approach is more robust because it provides explicit information to guide the system's ability to reason about the relative priorities data producing activities that use resources like storage or bandwidth, and it provides an explicit way to associate all of these sub-activities with the parent goal of getting the observations to the users.

These additional capabilities come with some cost, of course. The metadata required to associate data with goals adds overhead against already extremely limited storage and transport resources in an embedded system. But these capabilities provide ways to automate expensive human-intensive processes that currently are among the largest and most critical parts of operating a robotic space mission. For a given system a balance must be reached between the cost to develop the goals and the additional frameworks needed to manage data against the cost savings and potential for more science that can be achieved with the same class of platform. For missions that depend on reliable and verifiable data, it seems like a good investment.

5 Acknowledgements

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. I wish to thank the rest of the Mission Data System development team who have participated in the maturation of the described architectures. In particular I would like to thank Michel Ingham, Daniel Dvorak, and Robert Rasmussen who provided inspiration and technical guidance for this paper.

6 References

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns”, Addison-Wesley 1995.

[2] M.D. Ingham, R.D. Rasmussen, M.B. Bennett, A.C. Moncada, “Engineering Complex Embedded Systems with State Analysis and the Mission Data System”, American Institute of Aeronautics and Astronautics, Conference on Intelligent Systems, Sep. 2004.

[3] A. Barrett, R. Knight, R. Morris, R. Rasmussen, “Mission Planning and Execution Within the Mission Data System”, *Proceedings of the International Workshop on Planning and Scheduling for Space*, 2004

[4] D. Dvorak, R. Rasmussen, G. Reeves, A. Sacks, “Software Architecture Themes in JPL's Mission Data System,” *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, number AIAA-99-4553, 1999.

[5] Consultative Committee for Space Data Systems, File Delivery Protocol standard. http://www.ccsds.org, search: “File Delivery Protocol”.

[6] D. Dvorak, R. Rasmussen, T. Starbird, “State Knowledge Representation in the Mission Data System”, *Proceedings of the IEEE Aerospace Conference*, 2002