State-Based Models for Planning and Execution

Matthew B. Bennett, Russell L. Knight, Robert D. Rasmussen, Michel D. Ingham

NASA
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{Matthew.B.Bennett, Russell.L.Knight, Robert.D.Rasmussen, Michel.D.Ingham}@ jpl.nasa.gov

Abstract

Many traditional planners are built on top of existing execution engines that were not necessarily intended to be operated by a planner. The Mission Data System has been designed from the onset to have both an execution and planning engine and provides a framework for defining state-based models that can be used to coordinate planning and execution. The models provide a basis for ensuring the consistency of assumptions made by the execution engine and planner, and a basis for run-time communications between the planner and execution engines.

Introduction

Many traditional planners are built on top of existing execution engines that were not necessarily intended to be operated by a planner. The planner must take into consideration the execution engine s behavior, must make the same assumptions about the real world (which may be hidden in the code), and must be aware of its quirks. For example, a command may be issued by an execution engine and have side effects on the execution of subsequent commands. The effects may be both on the behavior of the execution engine and on what is being controlled in the real world. A planner must be aware of these effects to generate plans that will succeed when executed. Because there may be no well-defined structure in the execution engine to model real-world effects, and because planners generally have no rigorous model of the executive s behavior, it is challenging to build planning and execution engines that work together under the same assumptions. Furthermore, it is difficult to keep them consistent in a parallel development effort.

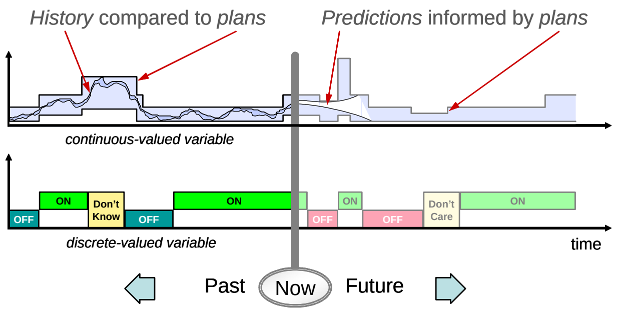
The Mission Data System (MDS) project at the Jet Propulsion Laboratory has developed a control architecture, modeling framework, and systems engineering methodology for developing state-based models of real-world behavior and effects, and execution engine behavior, which are used to inform both planning and execution. More specifically, these models provide the basis for ensuring the design-time consistency of assumptions in the execution engine and planner, and are the basis for run-time communications and coordination between planners, schedulers, and execution engines. MDS calls these models "State Effects Models" and the methodology by which they are developed "State Analysis"[2]. The modeling framework has been designed to be an open architecture for applying various formalisms and algorithms for spacecraft operations planning and execution.

In MDS, state-based models provide a basis for communication between planners, schedulers, and the engines that execute scheduled plans, as follows:

(1) MDS has clearly defined roles in the architecture for planning and execution.
(2) MDS has defined semantics of execution in terms of state-histories that are represented using state constraints (goals). The MDS architecture has well-defined interfaces for exchanging information between the execution and planning engines.
(3) MDS handles inaccuracies in modeling by explicitly representing uncertainty in state estimates produced during execution, using this uncertainty in controllers, planning for uncertainty in knowledge goals, and monitoring and enforcing the planned level of uncertainty during execution using estimators.
(4) MDS deals with uncertainty in activity duration and event timing using flexible time, uncertain time intervals, and worst-case state predictions limited by temporal constraints that impose deadlines.
(5) MDS State Analysis is a systems engineering methodology that with the MDS software frameworks bridges the gap between how execution is treated in the planning process, and what happens when the resulting plan is actually executed [2].

Execution and Planning Semantics

MDS uses constraint-based semantics to model execution behavior. A constraint (goal) on a state variable represents a set of possible state trajectories over an interval of time. A state trajectory is estimated during execution and must agree with the planned constraints for the plan to execute successfully. A state trajectory for state variable is



History compared to *plans*
Predictions informed by *plans*
continuous-valued variable
OFF
ON
Don't Know
OFF
ON
OFF
ON
OFF
Don't Care
ON
discrete-valued variable
time
Past
Now
Future

Figure 1: State value histories and plans shown as timelines

represented in MDS using a state value history (see figure 1)[4]. A state value history is updated by a state variable s estimator during execution to reflect the system s best estimate of what the state actually was. A state variable s achievers (controller and estimator) enforce the planned constraints during execution.

Plans consist of constraint networks containing both temporal and state constraints. State constraint semantics are based on set theoretic operations (such as union and intersection) over sets of possible state histories. These semantics are used by the planner to generate plans.

In the MDS architecture, goal networks fully describe plans and are also directly executable. At execution time, goal nets are monitored by the execution engine, which issues goals to state variables when preconditions are met. State variables forward the goals to the appropriate achievers, which execute and enforce the constraints. Temporal preconditions are represented directly in the goal network. State preconditions are monitored by the goal checker by consulting state information stored in state value histories, and by consulting the readiness of a state s achievers to begin enforcing a constraint.

Roles and Interfaces between the Planning and Execution Engines

MDS has clearly defined roles in the architecture for planning and execution. State variables and achievers provide well-defined interfaces between the planner and the execution engine. State variables store state history during plan execution that can be inspected by the planner. State variables also provide interfaces to access plans as they are developed by the planner and to inspect plans by the execution engine. Achievers execute planned state constraints as part of the execution engine and provide interfaces that describe their own execution behavior. State variables are consulted by the planner for modeling information about the physics of the state to be controlled (state effects model) as well as for information about control system execution behavior. For states that are actively estimated or controlled, the state variable consults its achievers for determining their execution behavior; otherwise, the physics in the state effects model (discussed in the next section) is sufficient to describe a state s behavior.

The goal checker is part of the execution engine. It issues goals to state variables to be executed when certain preconditions are satisfied. Some of these preconditions are timing constraints as developed in the plan; others are state preconditions and the readiness of achievers to begin enforcing specific constraints. These other preconditions are checked through interfaces on state variables. State variables provide state information stored in their value histories for checking state preconditions. State variables consult their achievers for checking the readiness of their achievers to begin enforcing specific constraints.

The planner must be informed about the execution capabilities of achievers (such as whether or not an achiever can execute a goal). At plan time, the planner can consult a state variable to determine if a goal is achievable, if 2 goals can be achieved back to back, and what the expected state will be when the execution engine executes a particular goal. A state variable that has achievers can consult its achievers in turn to answer these questions. In this way, the planner can be informed about the execution engine s behavior.

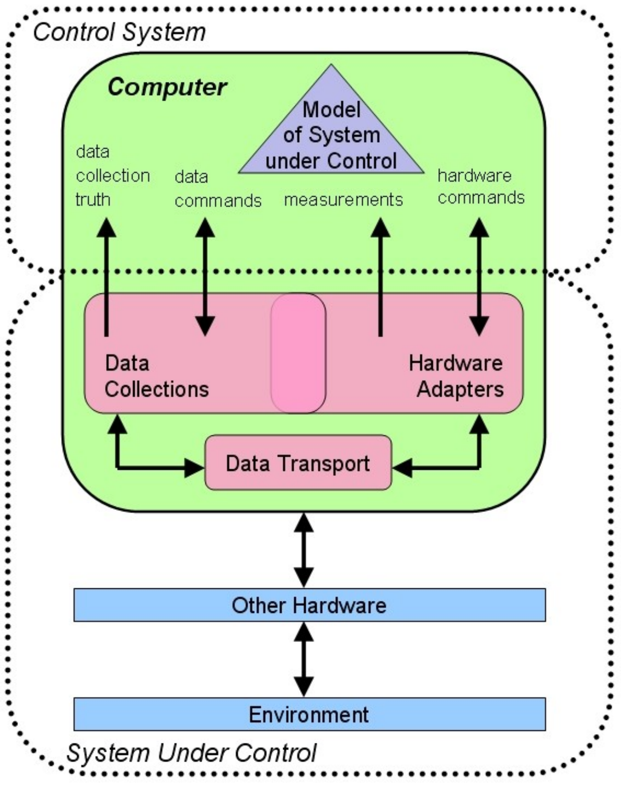


Figure 2: State effects model the physics of the system under control and the interface to the system under control in terms of commands and measurements.

State variables and achievers provide centralized places for storing modeling information that is consistent for use in both planning and execution. Rule-based and procedural based approaches can be embedded within achievers to model and implement achiever execution behavior.

State Effects Models

The MDS architecture defines the states of the system to be controlled using state effects models. The state effects model provides the basis for communication between planners and execution engines. By defining state effects to be modeled in a single unified fashion ensures that both the planner and execution engines can make the same assumptions about the system they are controlling.

State effects models define

(1) Physical models of the dynamic behavior of states, including the physical effects between states,
(2) Measurements that the control system uses to estimate the states being controlled, and how states affect measurements, and
(3) Commands that the control system uses to control the states of the system under control, and how the commands affect states.

The state effects model is used during planning and execution to

(1) Decompose the user s intent into a plan of coordinated constraints on affecting states needed to achieve the intent,
(2) Validate the plan against predictions of states based on initial conditions and predictions of affecting states, and
(3) Generate state predictions to be checked and optionally enforced during execution.

Each of these uses is discussed below in greater detail.

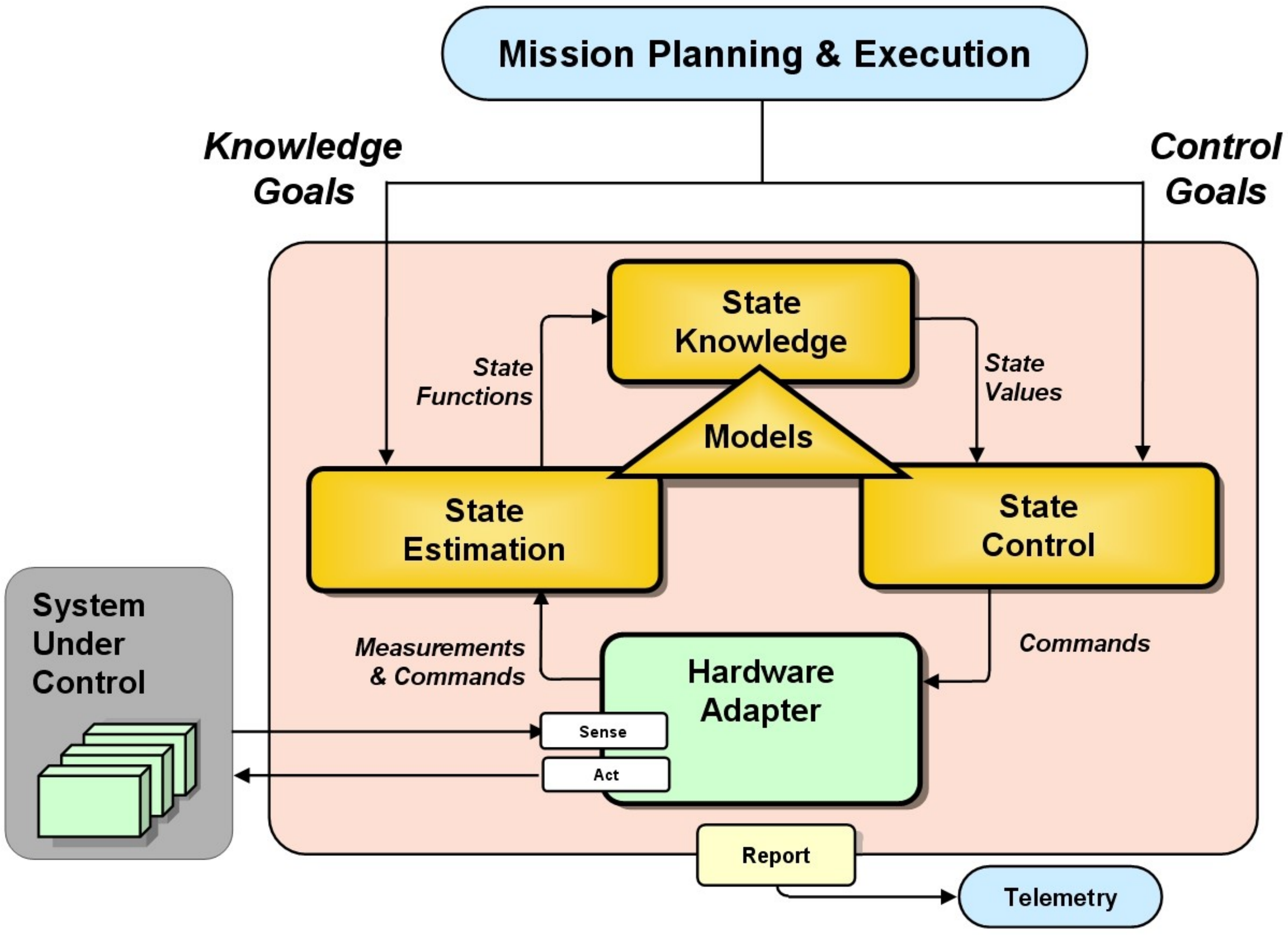


Figure 3: The MDS control system architecture.

Plan Decomposition

In MDS, decomposition of user intent is done through a process of *goal elaboration*. Each high-level goal on a state variable elaborates to a supporting network of goals on states that affect the state, as defined in the state effects model. For example, if the user intent is to have a new picture, then this high-level goal may elaborate to goals on the power state of the camera, the operational mode state of the camera, the data storage resource state, etc.

Plan Validation

A plan is validated against its predictions by using the state effects model to compute projections on each state variable s timeline. Projections are represented as state constraints. For example, a state effects model for a battery state of charge is defined as being affected by the all the power sink and source states. Thus, the projection for the battery state of charge is computed according to a state effects model where the battery state is equal to the integral of the sum of all power sources minus the integral of the sum of all power sinks. The plan for power sources and sinks is integrated and checked against an overall constraint on the battery state of charge to be above 10%.

Prediction Enforcement

During execution, this projection can be checked as a constraint on the battery state of charge. If it appears that the charge is falling below its projection, the execution engine can decide whether or not to recompute the projection to see if there is a real problem, or to exercise a another plan.

State Estimation

Estimators in the execution engine use state effects models for measurements, commands, and state-to-state effects to estimate state variables. The execution engine checks the state estimates against the plan to ensure that the plan for having proper state knowledge is executing properly, and to determine when the plan can progress. An example state effects measurement model is a camera s power switch measurement that is affected by its corresponding switch open/closed state. The estimator for the power switch state uses this measurement to estimate the state variable of the power switch. In the absence of a measurement, the estimator could use the power switch s state effects command model. In this case, the state effects model describes how the state of the power switch is affected by the power switch command. The estimator could use this model to estimate the state of the power switch based on the last power switch command issued (and potentially the health state of the power switch). In the absence of either a command or a measurement, the state estimator could infer the state of the switch by consulting the camera operating mode state variable. It would be correct do this by reasoning from the following chain of state-to-state effects in the state effects model: the power switch state affects the power use, which in turn affects the camera operating state.

State Control

State controllers in the execution engine use the state effects models for commands to determine when to issue commands. When the execution engine issues constraints to a controller to change a state in the system under control, the controller responds by issuing the proper commands. Per the example above, the state effects model for the power switch command would show that the power switch command effects the power switch. The power switch state controller would use this model to determine that it needs to issue the power switch command to change the state of the power switch.

Refinement of Projections to Incorporate Execution Engine Behavior

In addition to the state effects model, projections can reflect the behavior of the execution engine. The planning engine consults state variables to compute projections, which in turn can consult their achievers in the execution engine. The achievers model their behavior when executing goals, and can use this along with the state effects model to refine a state projection that would otherwise be based purely on the state effects model. This refined state projection reflects not only the state constraint, and the state effects model, but also what the achiever does to the physical state when the acheiver executes the state constraint. This refined state projection should be a subset of the original state constraint if the plan is valid, otherwise the constraint is noted as unachievable, and an alternative plan is considered. In this way the execution engine models its capability to execute constraints and provides this information to the planner. By using the modeling information, the planner can insure that the plans are consistent with the capabilities of the execution engine.

State Uncertainty

MDS handles inaccuracies in modeling by explicitly representing uncertainty in state estimates produced during execution. A state variable s history contains uncertainty associated with an estimate for each point in time. An estimator always updates the history with a measure of uncertainty.

A state s controller has access to this uncertainty, and must control to bounds specified in the planned state constraints. The bounds specified in planned state constraints are checked by the controller against the current estimated state and its uncertainty. The controller takes actions to ensure that the bounds are met given the uncertainty in the state estimate. For example, if a control constraint is to keep an actuator s position within a deadband, and the current uncertainty is expressed as a range, then the controller must actually control to a narrower range to account for the uncertainty in the knowledge of the position.

If a given control constraint requires a certain state uncertainty, then the planner s elaboration of the control constraint includes a constraint on the uncertainty of the knowledge of the state. This sort of uncertainty constraint is called a knowledge goal, and is executed by the estimator that is responsible for estimating the state. It is the responsibility of the estimator to achieve the planned level of uncertainty during execution. This is monitored by the execution engine, which will flag deviations to the planner if the knowledge constraint is not being met.

Timing Uncertainty

MDS deals with uncertainty in the timing of plans using flexible time. Each time point in the planned goal network has a range of possible times, either to allow for the uncertainty in execution times of constraints or to accommodate flexibility in the execution system. Uncertain temporal intervals are appropriately labeled. The state projections produced by the planner take into account the range of times. The planner assumes times that would produce the worst-case state projections. The worst-case projections are bounded by the operators by imposing deadlines on activities in the form of temporal constraints. An example is a rover traverse followed by a fixed time Earth communication window. The energy use during the traverse is limited by the deadline imposed by the communication window, when the rover must be immobile. If the traverse is not completed by the communication window, the traverse constraint ends early, and is replanned to restart after the communication window.

Determining that a plan with flexible time will execute successfully boils down to determining dynamic controllability of the temporal constraint network [6]. How this execution actually ensues is called "timepoint firing," and is described in detail in [12].

State Analysis

State Analysis [2] improves on the current state-of-the-practice by producing requirements on system and software design in the form of explicit models of system behavior, and by defining a state-based architecture for the control system. It provides a common language for systems and software engineers to communicate, and thus bridges the traditional gap between software requirements and software implementation.

State Analysis provides a uniform, methodical, and rigorous approach for:

(1) discovering, characterizing, representing, and documenting the states of a system under control,
(2) modeling the behavior of states and relationships among them, including information about hardware interfaces, operations, and achiever behavior,
(3) capturing the mission objectives in detailed scenarios motivated by operator intent,
(4) keeping track of system constraints and operating rules, and
(5) describing the methods by which objectives will be achieved.

The state analysis methodology recognizes the need for specifying execution behavior and planning specifications in terms of common models. State effects models are developed in a spiral process of state discovery until all of the states of the system to be controlled are known and their physical models are well understood. The execution and planning engine software is then specified in terms of these physical models. This includes specifications for estimation and control algorithms, elaborations, constraint semantics, projection algorithms, and the other information exchanged between the execution and planning engine as discussed in the previous sections.

Conclusion

We have discussed the MDS control architecture, modeling framework, and systems engineering methodology for developing state-based models of real-world behavior, effects, and execution engine behavior, which are used to inform both planning and execution. These models provide a basis for ensuring the design-time consistency of assumptions in the execution engine and planner, and are the basis for run-time communications and coordination between planners, schedulers, and execution engines. The modeling framework has been designed to be an open architecture for applying various formalisms and algorithms for spacecraft operations planning and execution.

In summary,

(1) MDS has clearly defined roles in the architecture for planning and execution
(2) MDS has defined semantics of execution in terms of state-histories that are represented using state constraints (goals). The MDS architecture has well-defined interfaces for exchanging information between the execution and planning engines, including methods on achievers called by the planner.
(3) MDS handles inaccuracies in modeling by explicitly representing uncertainty in state estimates produced during execution, using this uncertainty in controllers,

planning for uncertainty in knowledge goals, and monitoring and enforcing the planned level of uncertainty during execution using estimators.

(4) MDS deals with uncertainty in activity duration and event timing using flexible time, uncertain time intervals, and worst-case state predictions limited by temporal constraints that impose deadlines.

(5) MDS State Analysis is a systems engineering methodology that with the MDS software frameworks bridges the gap between how execution is treated in the planning process, and what happens when the resulting plan is actually executed.

Acknowledgments

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. We wish to thank the rest of the Mission Data System development team, and the Mars Science Laboratory mission personnel who have participated in the maturation of MDS.

References

[1] D. Dvorak, R. Rasmussen, G. Reeves, and 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.

[2] M. Ingham, R. Rasmussen, M. Bennett, and A. Moncada, "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," *Proceedings of the 1st AIAA Intelligent Systems Technical Conference*, number AIAA-2004-6518, 2004.

[3] A. Barrett, R. Knight, R. Morris, and 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, and T. Starbird, "State Knowledge Representation in the Mission Data System," *Proceedings of the IEEE Aerospace Conference*, 2002.

[5] B.C. Williams, M. Ingham, S. Chung, and P. Elliott, "Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers," *Proceedings of the IEEE*, 91(1):212-237, 2003.

[6] P. Morris, N. Muscettola, and T. Vidal, "Dynamic Control of Plans with Temporal Uncertainty," *Proceedings of the 17th International Joint Conference on A. I. (IJCAI-01)*, Seattle, WA, 2001.

[7] A. Meiri, R. Dechter, and J. Pearl, Temporal Constraint Networks, *Artificial Intelligence*, 49:61--95, 1991.

[8] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," *International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS 1999)*, Noordwijk, The Netherlands, June 1999.

[9] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," *International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*, Breckenridge, CO, April 2000.

[10] A. K. Jonsson, P. H. Morris, N. Muscettola, K. Rajan, and B. Smith, "Planning in Interplanetary Space: Theory and Practice," *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, 177-186.

[11] N. Muscettola, P. Morris, B. Pell, and B. Smith, Issues in temporal reasoning for autonomous control systems, In F. Anger, editor, Working Notes from the *AAAI workshop on Spatial and Temporal Reasoning*, 1997.

[12] R. Knight, "Evaporating tasks during execution of dynamically controllable networks," *AAAI Workshop on Plan Execution*, 2005.