Timed Model-based Programming: Executable Specifications for Robust Critical Sequences

Michel D. Ingham and Brian C. Williams

Space Systems and Artificial Intelligence Laboratories
Massachusetts Institute of Technology
77 Massachusetts Ave., Cambridge, MA 02139
{ingham, williams}@mit.edu

Abstract. For robotic spacecraft, robust plan execution is essential during time-critical mission sequences, due to the very short time available for recovery from anomalies. These sequences include hard-coded delays between certain actions, which implicitly capture knowledge about the state of the spacecraft or its environment. Our goal is to fold the representation of such timing constraints into the model-based programming paradigm, which provides a language for writing *executable specifications* of desired system state trajectories. These specifications are executed by a Timed Model-based Executive that is state-based and fault-aware. This paper presents the semantics of *timed model-based programs*, an implementation in terms of *timed hierarchical constraint automata*, and an execution algorithm. We demonstrate the application of our executive on a Mars atmospheric entry scenario.

1 Introduction

There is growing demand for high-reliability embedded systems that operate robustly and autonomously in the presence of tight real-time constraints. The ever-increasing complexity of such systems imposes stringent requirements on our execution technology, in the areas of software verifiability, temporal reactivity and fault management. We look to advances in embedded software, to help mitigate the risks associated with high levels of system complexity. Robotic space exploration provides an interesting domain for the study of these embedded programming issues.

For robotic spacecraft, robust plan execution is essential during time-critical mission sequences, such as planetary orbital insertion and entry, descent and landing, due to the very short time available for recovery from anomalies. Such time-critical spacecraft sequences include hard-coded delays between certain actions, which implicitly capture knowledge about the state of the spacecraft or its environment. For example, in the onboard sequence for atmospheric descent of a Mars lander, a delay of approximately 10 seconds between jettison of the lander's heatshield and deployment of its legs would be introduced, to ensure that the deploying legs would not impact the separating heatshield. Engineers choose to encode this type of engineering knowledge implicitly via a timing constraint, rather than explicitly including the relevant states in the plant model, either because it simplifies the onboard reasoning, or because of system observability limitations. For example, the heatshield separation distance from the lander is not explicitly measurable, though its expected behavior envelope has been determined from pre-launch empirical testing and statistical simulation. Robust control programs for time-critical sequences must allow for specification of such timing constraints.

This paper describes a novel approach to executing time-critical activities, with the following features:

1. *state-based specification* – We adopt the model-based programming paradigm [14], which allows the programmer to interact directly with “hidden” plant states, that is, states that are not directly observable or controllable. Since engineers prefer to reason about embedded systems in terms of state evolutions, state-based specifications provide a natural means of encoding control programs for these systems.
2. *timed specification* – We augment the basic model-based programming paradigm by introducing clock variables and timing constraints, allowing the execution of control programs to be dependent on time as well as system state. In this approach, which we call *timed model-based programming (TMBP)*, control programs set and read clock variables just as they set and read plant state variables.
3. *executable specification* – Execution of a control program requires mapping from the state goals, specified in the program, to actuator commands that achieve the goals, and from the sensor observations, to the current system state. In the TMBP approach, this mapping between states and sensors/actuators is performed automatically, by a deductive controller that reasons through a common-sense *plant model*. This model represents the set of possible behaviors and interactions of the system components.
4. *fault-aware execution* – Model-based programs must ensure correct synthesis of behavior in the presence of failures. The deductive controller performs model-based diagnosis and reactive planning to enable the executive to detect and respond to failures on-the-fly, within the state-achievement loop. These failure recoveries are executed in a manner that is transparent to the control program.

Figure 1 presents the architecture of our Timed Model-based Executive. The control sequencer executes a control program by issuing *configuration goals* for achievement by the underlying deductive controller. Execution of the control program is conditioned on time constraints, which are resolved based on input from the *system clock*, and state constraints, which are resolved based on *state estimates* returned by the deductive controller. The deductive controller reasons through the plant model to generate state estimates and commands, given observations from sensors and configuration goals to be achieved.

The following section identifies related work. Next, we provide a motivating example of TMBP. We then describe the semantics of timed model-based program execution and our executive’s implementation. Finally, we illustrate its capability by stepping through the execution of our motivating example.

2 Related Work

Our approach unifies concepts from robotic execution languages, model-based reasoning and formal modeling of real-time systems. Like other state-of-the-art robotic execution languages, such as RAPs [3] and TDL [10], we provide support for task-level

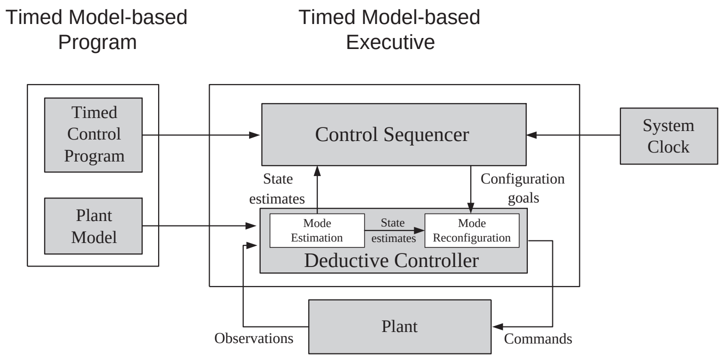

Fig. 1. Architecture for the Timed Model-based Executive.

control capabilities: task decomposition, time-keeping, preemption, iteration, parallel and sequential composition. We support both goal- and event-driven execution, where our “goals” and “events” are represented as changes to system states. Unlike these other execution languages, the mechanisms for specifying and commanding goal achievement methods are provided by the underlying deductive controller. The capacity for a Timed Model-based Executive to operate at the level of system state specifications, and to abstract away the details of how states are achieved, is considered a significant benefit.

The design of the deductive controller is based on previous work in model-based reasoning, diagnosis and reactive planning. In particular, we leverage algorithms for state inference and optimal reconfiguration used in the Livingstone [11] and Burton [12] model-based executives. By coupling deductive reasoning with the task control capabilities of the control sequencer, we provide an executive that is flexible, fault-aware and robust.

Formal approaches to real-time system modeling, e.g. Timed Automata [1] and Timed Transition Systems [5], are characterized by their clean semantics and amenability to verification via formal tools. We share various notions with these approaches, including the definition of a complex system as a composition of concurrently-operating automata, and the incorporation of clock variables and timing constraints into the semantic model. Key differences include our adoption of a hierarchical computational model, and our use of configuration goals as a mechanism for goal-driven execution.

3 Timed Model-based Programming Example

The execution of critical spacecraft sequences depends on timing conditions as well as state knowledge. As an example, consider the entry sequence for a Mars lander spacecraft, such as the Mars Polar Lander [15] (Figure 2).

At the end of the cruise phase of its mission, as the spacecraft approaches Mars, it turns on its descent engine, putting it into standby mode. Four and a half hours later, as the spacecraft nears its entry point into the Martian atmosphere, it switches from Earth-

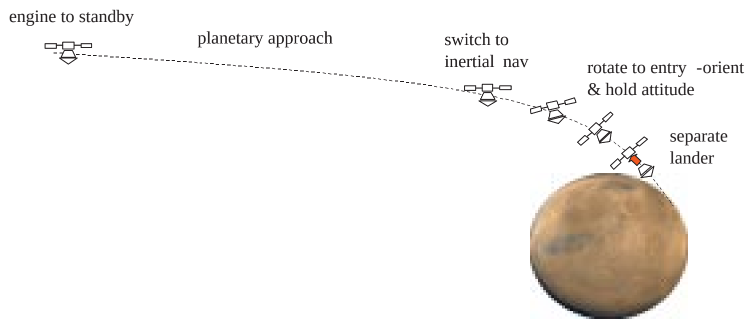

Fig. 2. Mars entry sequence.

relative navigation, using a combination of a star tracker and an inertial measurement unit (IMU), to inertial navigation using only the IMU. This navigation mode switch is necessary because, once atmospheric entry is initiated, the spacecraft will no longer be able to perform the reorientations necessary to track the reference stars. Four minutes after switching its navigation mode, the spacecraft prepares for atmospheric entry by rotating to its entry orientation. Once the entry orientation has been achieved, the lander stage of the spacecraft separates from the cruise stage and proceeds toward entry into the Martian atmosphere (all the while holding its attitude at the entry orientation). When atmospheric entry is initiated (as determined by a change in the spacecraft's acceleration due to atmospheric drag), the entry sequence ends and the spacecraft proceeds to the descent and landing phases of the mission.

In the above sequence, execution is conditioned on time conditions. For example, consider the four and a half hour delay between putting the descent engine into standby mode and switching to inertial navigation. Though it might seem more desirable to condition the sequence execution on a state of the spacecraft, in this case the relative distance of the spacecraft from the entry point, this is not an option: the spacecraft does not have access to any observations that would allow it to measure this relative distance. For this reason, the sequence must include a hard-coded delay, the length of which has been conservatively (but fairly accurately) estimated from computations performed by ground operations personnel based on the cruise trajectory of the spacecraft. The onboard executive must therefore have the ability to initiate clocks that it can reference, in order to check for satisfaction of timing conditions.

It is also important to note that this sequence specification is expressed in terms of states of the spacecraft. Many of the states in the sequence are "hidden"; that is, they are not directly observable, but instead must be deduced indirectly based on one or more sensor observations and knowledge of how these observations relate to the state of interest. For instance, the state of an engine must generally be deduced based on various measurements of electrical power, temperature, and acceleration. Similarly, hidden states are not necessarily directly controllable, but instead must be commanded indirectly, sometimes through a complex communication path conditioned on states of multiple components. For example, engine commands from the flight computer must

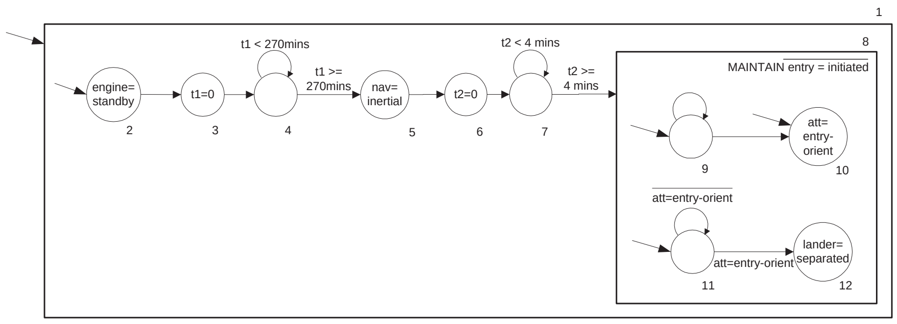

Fig. 3. THCA representation of the timed control program for the Mars entry sequence.

pass through a propulsion drive electronics module, which must be powered on for the commands to reach the engine.

This type of state-based specification is far simpler than a control program that must turn on valve drivers, open individual valves in the propulsion subsystem, and interpret readings from the various sensors in the system. Having engineers specify desired spacecraft behavior in terms of more abstract hidden states makes the task of writing the control program much easier and avoids the error-prone process of reasoning through low-level system interactions. In addition, it gives the program's execution kernel the latitude to respond to novel failures as they arise. This is essential for achieving high levels of robustness.

Timed Control Program – Figure 3 shows the graphical representation of the timed control program for the entry sequence. This representation, which we call a *timed hierarchical constraint automaton*, consists of a set of locations (represented as circles and boxes in the figure). While they are marked, locations can assert configuration goals, corresponding to states that the plant must progress toward (e.g., *engine=standby* is asserted in location 2). Locations can also assert clock initializations (e.g., location 3 initializes clock *t1* to zero). Transitions between locations (represented as arrows in the figure) can be conditioned on time and state constraints (e.g., the transition from location 4 to location 5 is conditioned on $t1 \geq 270$ *mins*). We formally define these automata in a later section of the paper. Here, we make general references to the figure, to provide an informal idea of how timed control programs are written and executed. Later, we will revisit this example in more detail.

Plant Model – The plant model is built from a set of component automata, such as those depicted graphically in Figure 4.[1] The component automata operate concurrently and synchronously. Each component (e.g., *engine*) is represented by a set of modes, corresponding to the nominal (e.g., *off, standby, firing*) and off-nominal (e.g., *failed*) states of the component. The behavior within each of these modes is described

[1] The models shown here provide a fairly abstract representation of component behavior. However, timed plant models can also be written at lower levels of abstraction; for example, a spacecraft engine could be modeled as a composition of more detailed component models of tanks, valves, thrusters, etc…

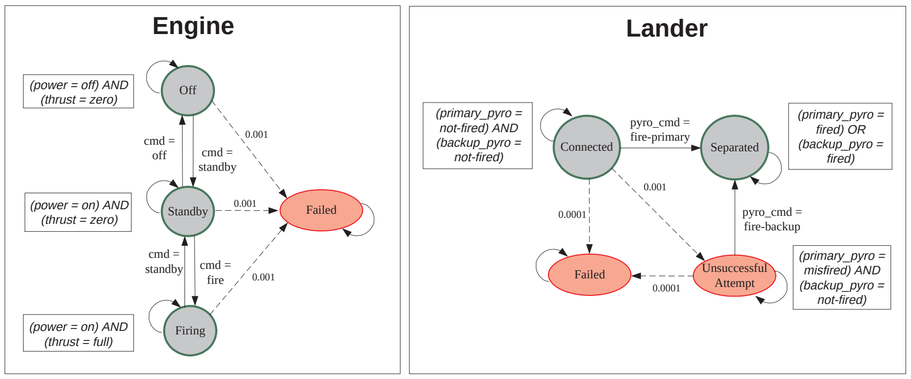
Engine
(power = off) AND (thrust = zero)
Off
cmd = off
cmd = standby
0.001
(power = on) AND (thrust = zero)
Standby
0.001
Failed
cmd = standby
cmd = fire
0.001
(power = on) AND (thrust = full)
Firing
Lander
(primary_pyro = not-fired) AND (backup_pyro = not-fired)
Connected
pyro_cmd = fire-primary
Separated
(primary_pyro = fired) OR (backup_pyro = fired)
0.001
0.0001
pyro_cmd = fire-backup
Failed
0.0001
Unsuccessful Attempt
(primary_pyro = misfired) AND (backup_pyro = not-fired)

Fig. 4. Example component automata from a simplified Mars lander spacecraft model. Nominal modes are represented as circles, and fault modes are represented as ovals. The probabilities on nominal transitions are omitted for clarity.

by a set of logical constraints on plant variables with finite domains, such as *thrust* and *power*. Stochastic component behavior is captured through a set of probabilistic transitions between modes. For example, the engine's transitions from *off*, *standby*, and *firing* to *failed* each have a 0.1% probability. Transitions between nominal modes are conditioned on plant variable assignments. For the engine, these transitions occur with probability 99.9%, immediately upon assertion of their triggering commands. The full plant model for the simplified spacecraft used in the Mars entry example would include component models for the *engine*, *nav*, *att*, *entry* and *lander* states.

Executing the Model-Based Program – The execution of the timed control program for Mars entry begins by asserting the configuration goal *engine=standby* (at the location labeled "2" in Figure 3), which the control sequencer issues to the deductive controller for achievement. To determine how to achieve this goal, the deductive controller considers the latest estimate of the state of the plant. Suppose the deductive controller determines from its sensor measurements and previous commands that the engine is off. The deductive controller deduces from the model that it should send a command to the plant that will lead the engine to standby mode. Based on the *engine* model in Figure 4, the deductive controller issues the command *cmd=standby*. Based on new sensor measurements confirming that the engine is indeed powered on and that the thrust is still zero, the deductive controller tracks the engine's transition into standby mode.

With the configuration goal now achieved, the sequencer then initializes a clock variable (at location 3) and waits for the clock to read 270 minutes (at location 4). At this point, the control sequencer asserts the goal *nav=inertial* (at location 5). The deductive controller determines that the goal can be achieved by simply issuing a command that triggers the desired change in navigation mode. After receiving confirmation that the goal has been achieved, the sequencer initializes a new clock (at location 6), and waits for 4 minutes to elapse (at location 7). It then starts continuously asserting the configuration goal *att=entry-orient* (at location 10, which gets continually re-marked by the transition from location 9), which the deductive controller begins to achieve by commanding the appropriate mode switch for the attitude control system. When the entry orientation is achieved (transition from location 11 to 12), the sequencer proceeds by issuing the configuration goal *lander=separated* (location 12), all the while continuing to assert *att=entry-orient*, to maintain the spacecraft's attitude. This results in the deductive controller triggering the firing of the lander's primary pyro latches to separate it from the cruise stage (as per the *lander* model in Figure 4). The sequencer continues to hold its entry orientation until the deductive controller indicates that entry has been initiated (*entry=initiated*), based on IMU sensor measurements indicating the onset of drag due to atmospheric entry, at which point execution of the timed control program in Figure 3 terminates.

This describes a nominal (i.e. fault-free) execution of the entry sequence. However, the robustness provided by the TMBP approach is particularly emphasized in the case of off-nominal execution. One of the main strengths of TMBP is its fault-awareness, i.e. its seamless incorporation of fault diagnosis and recovery capabilities within the sense-decide-act loop. Consider a hypothetical situation where the primary latches connecting the lander to the cruise stage fail to release upon command (corresponding to the failure transition into the *unsuccessful-attempt* mode for the *lander* model in Figure 4). Since the timed control program for the entry sequence has specified the configuration goal *lander=separated* (at location 12), the control sequencer will continue to assert this goal to the deductive controller until it has been achieved, or until it has been determined that the goal state cannot possibly be achieved from the current estimated state. Presuming that the mission-critical pyro latch subsystem incorporates some redundancy, failure of the primary latches to fire open would result in the deductive controller reasoning through the *lander* model to deduce that it should fire the backup latch. Whereas current approaches to spacecraft fault protection would require explicit diagnosis and recovery actions to be built into the sequence, the Timed Model-based Executive can perform this recovery in a manner that is transparent to the control sequencer.

4 Timed Model-based Program Semantics

In this section, we present the semantic model for TMBP. We begin by describing the semantics for the two parts of the timed model-based program: the plant model and the timed control program. The semantics for the control sequencer and deductive controller modules of the Timed Model-based Executive are then presented. The section concludes with a semantic definition of the execution of a timed model-based program in terms of legal state evolutions of a physical plant.

4.1 Plant Model

A physical plant is modeled as a factored *partially observable Markov decision process (POMDP)* $\mathcal{P} = \langle \Pi, \Sigma, \mathbb{T}, \mathbf{P}_\Theta, \mathbf{P}_\mathbb{T}, \mathbf{P}_\mathbb{O}, \mathbb{R} \rangle$. Π is a set of *variables*, each ranging over a finite domain. Π is partitioned into *state variables* Π^s, *control variables* Π^c, and *observable variables* Π^o. A *full assignment* σ is defined as a set consisting of an assignment to each variable in Π. Σ is the set of all possible full assignments over Π.

A *state* s is defined as an assignment to each variable in Π^s. The set Σ_s, the projection of Σ on variables in Π^s, is the set of all possible states. An *observation* of the plant, o, is defined as an assignment to each variable in Π^o. A *control action*, μ, is defined as an assignment to each variable in Π^c.

$\mathbb{T}$ is a finite set of *transitions*. Each transition $\tau \in \mathbb{T}$ is a function $\tau : \Sigma \rightarrow \Sigma_s$; $\tau(\sigma)$ is the state obtained by applying transition τ to any feasible full assignment σ. The transition $\tau^n \in \mathbb{T}$ models the system's nominal behavior, while all other transitions model failures. $\mathbf{P}_{\mathbb{T}}$ associates with each transition τ and full assignment σ a probability $\mathbf{P}_\tau(\sigma)$. $\mathbf{P}_\tau(\sigma)$ is shorthand for $\mathbf{P}_\tau(s' \mid s, \mu)$, where $s' = \tau(\sigma)$, and s and μ are the state and control variable assignments in σ, respectively. $\mathbf{P}_\Theta(s_0)$ is the probability that the plant has initial state s_0. The reward for being in state s is $\mathbb{R}(s)$. The probability of observing o in state s is $\mathbf{P}_{\mathbb{O}}(o \mid s)$.

The key features of the plant model are: (a) it captures nominal and various off-nominal system behaviors, by defining multiple transition functions for each full assignment; and (b) it is encoded compactly using concurrency and constraints (see example in Figure 4).

In TMBP, an *interleaving* model of computation is adopted,[2] where execution proceeds in *cycles*. Each cycle i consists of an instantaneous "discrete" event and a "continuous" phase in which time advances by some amount $\delta^{(i)}$. As far as the plant model is concerned, the discrete events correspond to transitions between plant states, which are assumed to occur instantaneously at absolute system times $t^{(0)}$, $t^{(1)}$, ... The plant maintains its state between these discrete event times; that is, state $s^{(i)}$ is assumed to hold in time interval $[t^{(i)}, t^{(i+1)})$. The time step $\delta^{(i)} = t^{(i+1)} - t^{(i)}$ is not necessarily constant from one cycle i to the next; it is determined by the amount of time required for the Timed Model-based Executive to complete one cycle of control sequencer and deductive controller operations.

Given a sequence of control actions $[\mu^{(0)}, \mu^{(1)}, \ldots]$, a *legal plant trajectory* is represented by a sequence of states $[s^{(0)}, s^{(1)}, \ldots]$, such that:

1. $s^{(0)}$ is a valid initial plant state, that is, $\mathbf{P}_\Theta(s^{(0)}) > 0$;
2. each transition from $s^{(i)}$ to $s^{(i+1)}$ occurs at time $t^{(i+1)}$;
3. for each i, there is a full assignment $\sigma^{(i)} \in \Sigma$ which agrees with $s^{(i)}$ on assignments to variables in Π^s and with $\mu^{(i)}$ on assignments to variables in Π^c; and
4. $s^{(i+1)} = \tau(\sigma^{(i)})$, for some $\tau \in \mathbb{T}$ with $\mathbf{P}_\tau(\sigma^{(i)}) > 0$.

A trajectory involving only the nominal transition τ^n is called a *nominal trajectory*. A *simple* trajectory does not repeat any state.

The space of possible state trajectories for a plant can be visualized using a *Trellis diagram*, which enumerates all possible states at each time step and all transitions between states at adjacent times (Figure 5).

[2] Real-time modeling formalisms such as Timed Automata [1] and Timed Transition Systems [5] have previously adopted a similar interleaving model of concurrency; Henzinger, Manna and Pnueli [5] have shown that the interleaving model is an appropriate model for capturing most phenomena of interest occurring in the timed execution of real-time systems.

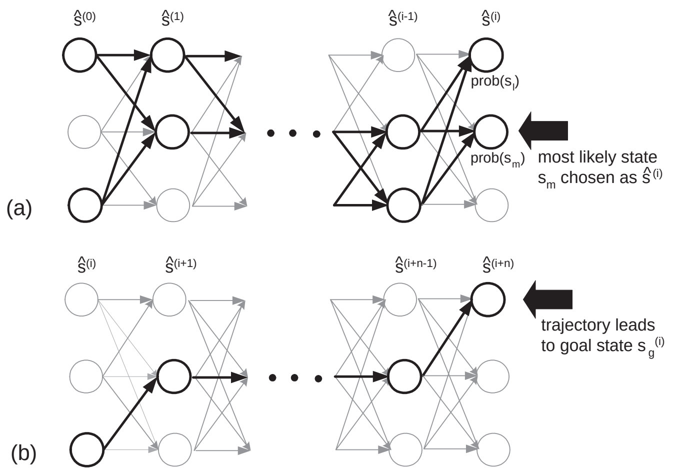

Fig. 5. A Trellis diagram depicts the plant's possible state trajectories. (a) Given the previous belief state, the latest control action and the latest observation, ME computes the current belief state, and selects the most likely state as its estimate; (b) MR chooses a path through the diagram along nominal transitions, terminating at the max-reward goal state.

4.2 Timed Control Program

In this section, the semantics of the second part of the timed model-based program, the *timed control program*, is discussed. A legal execution of a timed control program is defined in terms of a timed sequence of *control program locations*, which represent the "state" of the control program's execution at any given time, *configuration goals*, which provide the mechanism for goal-driven execution, and *clock interpretations*, which provide the mechanism for conditioning goals and activities on time constraints. The semantics of timed control programs builds on the semantics of untimed control programs [13, 14], by introducing the clock interpretations and conditioning the transitions between program locations on these clock interpretations.

We assume a dense model of time, where the time domain is taken as the set of non-negative real numbers, $\Re^+$. We define a set Π^t of *clock variables*. Clocks can be viewed as "timers," measuring the system time elapsed since their initialization. A clock interpretation ν assigns a value to each clock in Π^t. We define Σ_t as the set of all possible clock interpretations over Π^t. For $\delta \in \Re^+$, the clock interpretation that adds δ to the value of each clock variable in ν is denoted by $\nu + \delta$. The value of a particular clock x^t in a clock interpretation ν is denoted $\nu(x^t)$. We define a clock x^t to be *active* in execution cycle i if it was initialized in some earlier cycle. Conversely, if x^t has not been initialized prior to cycle i, we say that it is *inactive*. Once active, clocks cannot become inactive.

Formally, a *timed control program* is a deterministic automaton $\mathcal{TCP} = \langle L_{cp}, \lambda_{cp}, \tau_{cp}, g_{cp}, \iota_{cp}, \Sigma_s, \Sigma_t \rangle$. L_{cp} is the set of program locations, where $\lambda_{cp} \in L_{cp}$ is the program's initial location. τ_{cp} is a transition function $\tau_{cp} : L_{cp} \times \Sigma_s \times \Sigma_t \rightarrow L_{cp}$. Transitions between program locations are conditioned on plant state estimates and clock interpretations. Each location has a corresponding set of *clock initializations* $\iota_{cp}(l) \subseteq \Pi^t$, which is the set of clocks to be initialized upon transitioning to location $l \in L_{cp}$. Each location l also has a corresponding *configuration goal* $g_{cp}(l) \subset \Sigma_s$, which is the set of plant goal states associated with location l.

Similar to the plant model, the timed control program changes locations instantaneously at absolute system times $t^{(0)}, t^{(1)}, \ldots$, and maintains its location between these discrete event times. Given a sequence of most-likely state estimates $[\hat{s}^{(0)}, \hat{s}^{(1)}, \ldots]$ of a plant $\mathcal{P}$, a *legal execution* of a timed control program $\mathcal{TCP}$ is represented by a sequence of locations $[l^{(0)}, l^{(1)}, \ldots]$, clock interpretations $[\nu^{(0)}, \nu^{(1)}, \ldots]$, and configuration goals $[g^{(0)}, g^{(1)}, \ldots]$, such that:

1. $l^{(0)}$ is the initial program location λ_{cp};
2. $\nu^{(0)}$ is a valid initial clock interpretation, that is, $\nu^{(0)}(x^t) = 0$ for all clocks $x^t \in \Pi^t$, and all clocks are initially inactive;
3. $\langle \nu^{(i)}, \nu^{(i+1)} \rangle$ represents a legal clock interpretation sequence, that is:
   – for each clock x^t that is inactive in cycle i, $\nu^{(i)}(x^t) = 0$,
   – for each x^t that is active in cycle i, $\nu^{(i)}(x^t) = \nu^{(i-1)}(x^t) + \delta^{(i-1)}$, where $\delta^{(i-1)} = t^{(i)} - t^{(i-1)}$ is the same for all active clocks,
   – for each x^t that is initialized in cycle i (i.e., $x^t \in \iota_{cp}(l^{(i)})$), x^t is active for all cycles $j > i$;
4. $\langle l^{(i)}, l^{(i+1)} \rangle$ represents a legal control program transition, that is, $l^{(i+1)} = \tau_{cp}(l^{(i)}, \hat{s}^{(i)}, \nu^{(i)} + \delta^{(i)})$; and
5. $g^{(i)}$ represents a valid configuration goal, that is, $g^{(i)} = g_{cp}(l^{(i)})$.

The semantics of a timed control program can thus be considered a variant of Deterministic Timed Automata [1], with two key distinctions: first, its execution is conditioned on the hidden state of a physical plant; and second, its locations assert configuration goals intended to operate on the hidden state of a physical plant.

4.3 Control Sequencer

The control sequencer's role is to direct the closed-loop goal-driven execution, by issuing the configuration goals specified in the timed control program. In each execution cycle, it takes as input a timed control program $\mathcal{TCP}$, the plant state estimate and the time step from the system clock, and it issues a configuration goal to the deductive controller. More precisely, the control sequencer advances the control program from its current location $l^{(i)}$ to a new location $l^{(i+1)}$, by taking the transition enabled by the state estimate $\hat{s}^{(i)}$ and the clock interpretation $\nu^{(i)}$. It generates the configuration goal $g^{(i+1)}$ associated with the new program location $l^{(i+1)}$.

Formally, the semantics of the control sequencer can be described by the function *CS* (see Figure 6), which operates on the control program $\mathcal{TCP}$ defined in Section 4.2. Given an initial program location $l^{(0)}$, an initial clock interpretation $\nu^{(0)}$, and sequences of state estimates $[\hat{s}^{(0)}, \hat{s}^{(1)}, \ldots]$ and cycle time intervals $[\delta^{(0)}, \delta^{(1)}, \ldots]$, *CS* can be used to generate a legal execution of $\mathcal{TCP}$, i.e., it outputs legal sequences of program locations, clock interpretations and configuration goals (as defined in Section 4.2).

$$CS(\mathcal{TCP}, l^{(i)}, \hat{s}^{(i)}, \nu^{(i)}, \delta^{(i)}) \rightarrow \langle l^{(i+1)}, \nu^{(i+1)}, g^{(i+1)} \rangle ::$$

1. **Advance to new program location.**

$$l^{(i+1)} = \tau_{cp}(l^{(i)}, \hat{s}^{(i)}, \nu^{(i)} + \delta^{(i)}).$$

2. **Update clock interpretations.**
For each clock $x^t \in \Pi^t$:

$$\nu^{(i+1)}(x^t) = \begin{cases} 0 \text{ if } x^t \text{ is inactive;} \\ \nu^{(i)}(x^t) + \delta^{(i)} \text{ if } x^t \text{ is active.} \end{cases}$$

x^t becomes active if $x^t \in \iota_{cp}(l^{(i+1)})$.

3. **Issue configuration goal.**

$$g^{(i+1)} = g_{cp}(l^{(i+1)}).$$

Fig. 6. Control sequencer function *CS*.

4.4 Deductive Controller

The deductive controller's dual role is to (a) infer the system state based on observations from the sensors, and (b) issue control actions that achieve the configuration goals. In each execution cycle, it takes as input the plant model $\mathcal{P}$, the configuration goal from the control sequencer, and the observation from the physical plant. It generates the most likely plant state estimate and an appropriate control action. This section presents the semantics of the deductive controller by defining semantics for its two distinct capabilities, *mode estimation* and *mode reconfiguration*.

Mode Estimation - The sequence of state estimates is generated by the deductive controller's mode estimation (ME) capability. ME is an online algorithm for tracking the likelihood of each possible plant state, given the plant model, the control actions, and the observations. In each cycle, ME returns the most likely plant state as the current state estimate.

Given the semantic model of the plant as a factored POMDP, ME is framed as an instance of *belief state update*. Belief state update computes the current *belief state*, that is, the probability associated with being in each state, conditioned on the control actions performed up to the *last* cycle, and the observations received up to the *current* cycle. Exploiting the Markov property, the belief state $b^{(i+1\bullet)}[s]$ at execution cycle $i+1$ is computed from the belief state and control actions at cycle i and observations at cycle $i+1$ using the following belief state update equations:

$$b^{(\bullet i+1)}[s_k] = \sum_{j=1}^{N} b^{(i\bullet)}[s_j]\mathbf{P}_{\mathbb{T}}(s_k \mid s_j, \mu^{(i)}),$$

$$b^{(i+1\bullet)}[s_k] = b^{(\bullet i+1)}[s_k]\frac{\mathbf{P}_{\mathbb{O}}(o^{(i+1)} \mid s_k)}{\sum_{j=1}^{N} b^{(\bullet i+1)}[s_j]\mathbf{P}_{\mathbb{O}}(o^{(i+1)} \mid s_j)};$$

where $b^{(\bullet i+1)}[s_k] \equiv p(s_k^{(i+1)} \mid o^{(0)}, \ldots, o^{(i)}, \mu^{(0)}, \ldots, \mu^{(i)})$ and $b^{(i+1\bullet)}[s_k] \equiv p(s_k^{(i+1)} \mid o^{(0)}, \ldots, o^{(i+1)}, \mu^{(0)}, \ldots, \mu^{(i)})$. $\mathbf{P}_{\mathbb{T}}(s_k \mid s_j, \mu^{(i)})$ is the probability that $\mathcal{P}$ transitions from state s_j to state s_k, given control actions $\mu^{(i)}$. $\mathbf{P}_{\mathbb{O}}(o^{(i+1)} \mid s_k)$ is the probability that observation $o^{(i+1)}$ is received in state $s_k^{(i+1)}$. The initial belief state $b^{(0\bullet)}[s_k]$ is computed based on $b^{(\bullet 0)}[s_k] = \mathbf{P}_{\Theta}(s_k)$.

Belief state update associates a probability to each state in the Trellis diagram. For ME, the tracked state with the highest belief state probability is selected as the most likely state $\hat{s}^{(i)}$ (Figure 5a). Note that the size N of the state space of the factored POMDP is very large, on the order of m^n, where n is the number of components in the system, and m is the average number of modes for each component. Given the high level of reactivity required for the Timed Model-based Executive, it is necessary to approximate the belief state update process, such that only a limited number of the most likely state estimates are computed in each execution cycle. Various implementations of the ME capability have been developed, which provide tractable approximations to the belief state update computation [8, 11, 14].

Mode Reconfiguration - The sequence of control actions is generated by the deductive controller's mode reconfiguration (MR) capability. MR provides an online algorithm for finding an optimal policy that achieves the configuration goal, given the plant model and the current belief state from ME (a policy π is a state-action mapping that specifies, for each state, an action to be taken). In each execution cycle, MR returns the first control action from the optimal policy.

Given a plant model expressed as a POMDP, the semantics of MR maps to a *goal-directed decision theoretic planning* problem. In decision theoretic planning, the objective is to choose actions such that some measure of reward is maximized [7]. More precisely, decision theoretic planning computes an optimal policy π^* for the POMDP that maximizes the expected (possibly discounted by a factor γ) sum of reward over the finite discrete-time horizon of interest.

The MR problem is described as goal-directed, because the specific objective of the planning problem is to find a policy that leads to a state that satisfies the given configuration goal. As such, the reward metric $R(s)$ for the MR planning problem is defined as a sum of a goal-specific reward function, which biases the solution toward states that achieve the specified configuration goal, and the state reward function $\mathbb{R}(s)$ built into the plant model POMDP, which biases the solution toward lower-cost policies among the set of policies that achieve the configuration goal.

Assuming the system is a Markov decision process (MDP) with fully-observable states, the solution of the basic decision-theoretic planning problem is obtained by solving the Bellman optimality equations [9]:

$$V_i^*(s) = \max_{\mu} \left[R(s) + \gamma \sum_{s' \in \Sigma} \mathbf{P}_{\mathbb{T}}(s' \mid s, \mu) V_{i-1}^*(s') \right],$$

$$\pi_i^*(s) = \arg\max_{\mu} \left[R(s) + \gamma \sum_{s' \in \Sigma} \mathbf{P}_{\mathbb{T}}(s' \mid s, \mu) V_{i-1}^*(s') \right].$$

In these equations, the optimal value function in the i^{th} cycle, V_i^*, is defined inductively as the maximum of the sum of the immediate reward R and the discounted expected value of the remaining $(i-1)$ steps. The optimal policy for the i^{th} cycle, π_i^*, is defined in terms of the optimal value function V_{i-1}^* for the $(i-1)^{th}$ cycle. Common approaches for solving the Bellman equations for a MDP include *value iteration* and *policy iteration* [9].

For a plant modeled as a POMDP, the solution to the planning problem is obtained by solving the Bellman equations associated with the "belief MDP," which defines as its state space the set of all possible belief states [7]. Note that the solution of the decision theoretic planning problem associated with the belief MDP is generally intractable via exact dynamic programming algorithms, due to the continuous nature of the belief state space. Thus, numerous approaches have been developed for finding approximate solutions [7]. These algorithms take the approach of computing approximations to the optimal value function, by exploiting properties of the belief state space. For the huge factored state spaces (exponential in the number of components) of interest in TMBP, this approach does not scale well. Instead, our executive implementation, which is based on the Burton reactive planner [12], makes key assumptions that allow MR to be performed reactively by dividing the planning problem into two steps: first, find a reachable goal state that satisfies the configuration goal and maximizes reward; second, find a (possibly suboptimal) sequence of control actions that lead from the current most-likely state to the maximum-reward goal state. Thus, MR can be viewed as picking a simple path through the Trellis diagram along nominal transitions leading to the goal state, as shown in Figure 5b.

4.5 Timed Model-based Program Execution

Now that the semantics of the timed model-based program and the modules of the Timed Model-based Executive have been introduced, this section combines these semantic descriptions into an overall execution semantics for the timed model-based program.

Given a timed model-based program consisting of a timed plant model $\mathcal{P}$ and a timed control program $\mathcal{TCP}$, a sequence of cycle time intervals $[\delta^{(0)}, \delta^{(1)}, \ldots]$, and a sequence of observations $[o^{(0)}, o^{(1)}, \ldots]$, a *legal execution* of the timed model-based program is represented by sequences of state estimates $[\hat{s}^{(0)}, \hat{s}^{(1)}, \ldots]$ of $\mathcal{P}$, program locations $[l^{(0)}, l^{(1)}, \ldots]$ of $\mathcal{TCP}$, clock interpretations $[\nu^{(0)}, \nu^{(1)}, \ldots]$ of $\mathcal{TCP}$, configuration goals $[g^{(0)}, g^{(1)}, \ldots]$ of $\mathcal{TCP}$, and control actions $[\mu^{(0)}, \mu^{(1)}, \ldots]$, such that:

1. The initial conditions are valid, that is:
   - $\mathbf{P}_\Theta(\hat{s}^{(0)}) > 0$;
   - $l^{(0)}$ is the initial program location λ_{cp}; and
   - $\nu^{(0)}(x^t) = 0$ for all program clocks x^t.
2. The sequences of program locations, clock interpretations, and configuration goals correspond to a legal execution of $\mathcal{TCP}$, which is consistent with the semantics of the control sequencer (presented in Section 4.3).
3. If plant state $\hat{s}^{(i+1)}$ is the result of a nominal plant transition from $\hat{s}^{(i)}$, then $\hat{s}^{(i+1)}$ is the state resulting from taking $\mu^{(i)}$, the first action in an optimal policy that

achieves configuration goal $g^{(i)}$, consistent with the semantics of MR (presented in Section 4.4).

4. Given the sequence of control actions, the sequence of state estimates corresponds to a legal trajectory of $\mathcal{P}$, which is consistent with the semantics of ME (presented in Section 4.4).

As discussed above, the implementation of the deductive controller has been previously described in [11, 12]. In the remainder of this paper, we focus on the technical details of the control sequencer.

5 Control Sequencer Implementation

Engineers generally prefer to use visual representations of system specifications over textual encodings. For this reason, StateCharts [4] and similar formalisms have become standard tools in the design and analysis of embedded systems. We adopt a graphical language, called *timed hierarchical constraint automata (THCA)*, to encode our timed control programs (Figure 3). In this section, we define THCA as a specific instance of the $\mathcal{TCP}$ automaton presented in Section 4.2, and we present the execution algorithm used by our control sequencer.

5.1 Timed Hierarchical Constraint Automata

In this section, we define the THCA encodings of timed control programs. In the following we call the "states" of a THCA *locations*, to avoid confusion with the physical plant state. A *program location*, as defined in Section 4, corresponds to a set of "marked" THCA locations.

A THCA has seven main attributes. First, it composes sets of concurrently operating automata. Second, each location is labeled with a state constraint, called a *goal constraint*, which the physical plant must progress towards, whenever that location is marked. This allows the hidden state of the plant to be controlled directly. Third, each location is labeled with a set of *clock initializations*, which initialize clock variables upon transition into the location. Fourth, each location is labeled with a constraint on clock and state variables, called a *maintenance constraint*, which must hold for that location to remain marked. Fifth, transitions between locations are conditioned on time and hidden state. Sixth, automata are arranged in a hierarchy – a location of an automaton may itself be an automaton, which is invoked when marked by its parent. This enables the initiation and termination of complex concurrent and sequential behaviors. Finally, multiple outgoing transitions can be taken from a single location, allowing an automaton to have several locations marked simultaneously. This enables a compact representation for iterative behaviors, as illustrated in Figure 3.

A THCA $\mathcal{A}$ is a tuple $\langle \Sigma, \Theta, \Pi, \mathbb{G}, \mathbb{I}, \mathbb{M}, \mathbb{T} \rangle$. Σ is a set of locations, partitioned into *primitive locations* Σ_p and *composite locations* Σ_c. Each composite location corresponds to another THCA. We recursively define the *subautomata* of $\mathcal{A}$ as the set of locations of $\mathcal{A}$, and locations that are subautomata of the composite locations of $\mathcal{A}$. Graphically, primitive locations are represented as circles (locations 2-7 and 9-12 in

Figure 3), while composite locations are represented as rectangles (locations 1 and 8 in Figure 3). $\Theta \subseteq \Sigma$ is the set of *start locations* of $\mathcal{A}$, identified by short diagonal arrows in Figure 3. At cycle i the state of a THCA is the set of marked locations $m^{(i)} \subseteq \Sigma$, called a *marking*. A marking represents the current state of multiple concurrent threads of execution.

Π is a set of variables, partitioned into plant state variables Π^s and clock variables Π^t. Each $x^s \in \Pi^s$ ranges over a finite domain $\mathbb{D}[x^s]$, and each $x^t \in \Pi^t$ ranges over $\Re^+$. We use propositional state logic for our constraint system, where propositions are composed into formulae using the standard logical connectives, and ($\wedge$), or ($\vee$) and not ($\neg$). In the case of state constraints, propositions take the form $(x^s = v)$, where $x^s \in \Pi^s$ and $v \in \mathbb{D}[x^s]$; in the case of clock constraints, propositions take the form $(x^t$ ineq $t)$, where $x^t \in \Pi^t$, ineq $\in \{<, >, \leq, \geq\}$ and $t \in \Re^+$. In addition, constraints can be of the form $\models c$ ("entails c") or $\not\models c$ ("does not entail c"). In our diagrams, we use c to denote the constraint $\models c$, and $\overline{c}$ to denote the constraint $\not\models c$. When a constraint is not specified, it is taken to be implicitly **True**.

$\mathbb{G}$ associates with each primitive location $\sigma^p \in \Sigma_p$ a state constraint $\mathbb{G}(\sigma^p)$, called a *goal constraint*, that the plant progresses towards whenever σ^p is marked. $\mathbb{I}$ associates with each primitive location $\sigma^p \in \Sigma_p$ a set of *clock initializations* $\mathbb{I}(\sigma^p)$ that are performed when σ^p is initially marked. $\mathbb{M}$ associates with each location $\sigma \in \Sigma$ a *maintenance constraint* $\mathbb{M}(\sigma) = (\mathbb{M}^s(\sigma), \mathbb{M}^t(\sigma))$ consisting of a state constraint and a clock constraint that must both hold at the current instant for σ to be marked. Maintenance constraints associated with composite locations are assumed to apply to all subautomata within the composite location. In our diagrams, maintenance constraints are preceded by the keyword "MAINTAIN". For example, in Figure 3, the assignment *t1=0* in location 3 is a clock initialization, the assignment *nav=inertial* in location 5 is a goal constraint, and *MAINTAIN* $\overline{\textit{entry=initiated}}$ in location 8 is a maintenance constraint.

$\mathbb{T}$ associates with each location $\sigma \in \Sigma$ a transition function $\mathbb{T}(\sigma)$. Each $\mathbb{T}(\sigma)$ specifies a *set* of locations to be marked at cycle $i + 1$, given appropriate assignments to Π at cycle i. Transitions are conditioned on *guard conditions* that must be satisfied by the conjunction of the plant model, the estimated plant state, and the current clock interpretation. In Figure 3, the guard on the transition from location 4 to itself is the clock constraint *t1<270 mins*, and the guard on the transition between locations 11 and 12 is the state constraint *att=entry-orient*.

5.2 Executing THCA

To execute a THCA $\mathcal{A}$, the control sequencer starts with an estimate of the initial state of the plant, $\hat{s}^{(0)}$, and an empty clock interpretation, $\nu^{(0)} = \emptyset$. It initializes $\mathcal{A}$ by marking the start locations of $\mathcal{A}$ and all their starting subautomata. It then repeatedly steps automaton $\mathcal{A}$ using the function $Step_{THCA}$ (Figure 7), which maps the current state estimate, clock interpretation and marking to a next state estimate, clock interpretation, marking and configuration goal. Execution completes when no marks remain.

Our model of time is implemented via a set of active clocks. Clock assignments in $\nu^{(i)}$ are updated at the beginning of each execution cycle, and remain constant throughout. The assignment to each active clock x^t_j is computed as the difference between the

$Step_{THCA}(\mathcal{A}, m^{(i)}, \hat{s}^{(i)}, t^{init}(i), t^{abs}) \rightarrow \langle g^{(i)}, m^{(i+1)}, \hat{s}^{(i+1)}, t^{init}(i+1)\rangle$::

1. **Update active clocks.** Update each active clock variable $(x_j^t = t^{abs} - t_j^{init})$. Add these new clock variable assignments to $\nu^{(i)}$.
2. **Check maintenance constraints for marked composites.** Unmark all subautomata of any marked composite location in $m^{(i)}$ whose maintenance constraint is violated by $\hat{s}^{(i)} \wedge \nu^{(i)}$.
3. **Assert clock initializations.** For any clock initialization $(x_j^t = 0)$ asserted by currently marked primitive locations, unless t_j^{init} is already specified in $t^{init}(i)$, set $t_j^{init}(i+1) = t^{abs}$.
4. **Setup configuration goal.** Output, as the configuration goal $g^{(i)}$, the conjunction of goal constraints from currently marked primitive locations in $m^{(i)}$ whose maintenance constraints are satisfied by $\hat{s}^{(i)} \wedge \nu^{(i)}$.
5. **Take action.** Request that MR issue a command that progresses the plant towards a state that achieves the goal $g^{(i)}$.
6. **Read next state estimate.** Once the command has been issued, obtain from ME the plant's new most likely state $\hat{s}^{(i+1)}$.
7. **Await incomplete goals.** If the goal constraint of a primitive location marked in $m^{(i)}$ is not entailed by $\hat{s}^{(i+1)}$, and its maintenance constraint was not violated by $\hat{s}^{(i)} \wedge \nu^{(i)}$, then include that location as marked in $m^{(i+1)}$.
8. **Identify enabled transitions.** A transition from a marked primitive location σ^p in $m^{(i)}$ is enabled if both of the following conditions hold true:
   (a) σ^p's goal constraint is entailed by $\hat{s}^{(i+1)}$, or its maintenance constraint was violated by $\hat{s}^{(i)} \wedge \nu^{(i)}$;
   (b) the transition's guard condition is satisfied by $\hat{s}^{(i+1)} \wedge \nu^{(i)}$.

   A transition from a marked composite location σ^c in $m^{(i)}$ is enabled if both of the following conditions hold true:
   (a) none of σ^c's subautomata are marked in $m^{(i+1)}$ and none of σ^c's subautomata have enabled outgoing transitions;
   (b) the transition's guard condition is satisfied by $\hat{s}^{(i+1)} \wedge \nu^{(i)}$.
9. **Take transitions.** Mark and initialize in $m^{(i+1)}$ the target of each enabled transition. Re-mark in $m^{(i+1)}$ all composite locations with subautomata that are marked in $m^{(i+1)}$.
10. **Update list of clock references.** Add the contents of $t^{init}(i)$ to $t^{init}(i+1)$.

Fig. 7. $Step_{THCA}$ algorithm.

current absolute system time, $t^{abs}(i)$, and the absolute time when the clock was initialized, $t_j^{init}(i)$.

The $Step_{THCA}$ algorithm provides a correct implementation of the control sequencer semantics presented in Section 4.3; that is, given a correct implementation of a deductive controller, it satisfies all conditions of a legal execution of a timed model-based program in Section 4.5. The proof is omitted here. Key features of the algorithm are as follows. *Reactive preemption* is implemented in Step 2: we unmark all subautomata of composite locations whose maintenance constraints have been violated by the latest state estimate, preventing the assertion of any goal constraints by these subautomata. *Clock persistence* is ensured in Step 3: we avoid resetting an already-active clock to zero when the primitive location gets re-marked, for example due to incomplete achievement of its goal constraint. *Goal-driven execution* is provided by Steps 4 and 5, via assertion of the configuration goal for achievement by the deductive controller. *Closed-loop execution* is implemented in Steps 6 and 7, based on plant state feedback: we continue to assert a goal until it is determined to have been achieved by the deductive controller. Finally, *progress due to goal achievement or preemption* is ensured in Step 8, by enabling transitions out of locations whose goal constraints are satisfied or whose maintenance constraints have been violated.

6 THCA Execution Example

We illustrate the interaction between the control sequencer and the deductive controller, by stepping through a fault-free execution of the Mars entry sequence discussed previously.[3]

Initial State – Initially, the start locations are marked (locations 1 and 2 in Figure 3). We assume ME provides the following initial plant state estimate: $\{$*engine=off, nav=Earth-relative, att=cruise-orient, lander=connected, entry=not-initiated*$\}$. Execution will continue as long as outermost composite location 1 remains marked.

Cycle 1, $t^{abs} = 0.0$ sec – The first $Step_{THCA}$ cycle proceeds as follows. (Step 1) No clocks are active, so no clock variables are updated. (Step 2) Since no maintenance constraints are specified for marked composite location 1, it remains marked. (Step 3) No clocks are initialized by the start locations. (Step 4) The only goal constraint asserted is *engine=standby*. This state assignment is passed to MR as the configuration goal. (Step 5) MR issues the first command in a sequence that achieves the configuration goal. Based on the simple plant model shown in Figure 4, this command is *cmd=standby*. (Step 6) ME confirms that *engine=standby* is achieved. (Step 7) The goal constraint has been achieved, so location 2 will not remain marked. (Step 8) Since location 2's outgoing transition is labeled **True**, it is enabled. (Step 9) After taking this transition, locations 1 and 3 are marked. (Step 10) There are no active clocks, so no initialization times to update.

Cycle 2, $t^{abs} = 0.5$ sec – Location 3 asserts the clock initialization *t1=0*, so we note 0.5 sec as the initialization time for *t1*. No goal constraints are asserted, and ME's state estimate remains unchanged. Location 3's outgoing transition is labeled **True**, so it is enabled. After taking this transition, locations 1 and 4 are marked.

Cycle 3, $t^{abs} = 1.1$ sec – We update clock variable *t1 = 1.1 - 0.5 = 0.6 sec*. No clock initializations or goal constraints are asserted, and ME's state estimate is unchanged. Since *t1* is less than 270 minutes, only the transition from location 4 to itself is enabled. Thus, locations 1 and 4 remain marked.

Since this behavior will be repeated for multiple cycles, we skip to the next "interesting" execution cycle, N_1, which occurs 16199.5 seconds later. State estimate $\{$*engine=standby, nav=Earth-relative, att=cruise-orient, lander=connected, entry=not-initiated*$\}$ is unchanged, and locations 1 and 4 remain marked in cycles 4 to N_1.

Cycle N_1, $t^{abs} = 16200.6$ sec – We update clock variable *t1 = 16200.6 - 0.5 = 16200.1 sec*. No clock initializations or goal constraints are asserted and the new state estimate remains unchanged. Since the value of *t1* is greater than 270 minutes, only the transition from location 4 to location 5 is enabled. After taking the enabled transition, locations 1 and 5 are marked.

Cycle ($N_1 + 1$), $t^{abs} = 16201.2$ sec – We update clock variable *t1 = 16201.2 - 0.5 = 16200.7 sec*. Location 5 asserts the goal constraint *nav=inertial*. This state assignment is passed to MR as the configuration goal. MR issues the first command in a sequence that achieves the configuration goal. ME confirms that *nav=inertial* is achieved as a result of this command. The goal constraint for location 5 has been achieved, so it will

[3] System time values associated with each cycle are for illustrative purposes only, and do not reflect the actual rate of execution of the control sequencer.

not remain marked in the next cycle. Since location 5's outgoing transition is labeled **True**, it is enabled. After taking the enabled transition, the marked locations are 1 and 6.

Cycle ($N_1 + 2$), $t^{abs} = 16201.8$ sec – We update clock variable *t1 = 16201.8 - 0.5 = 16201.3 sec*. Location 6 asserts the clock initialization *t2=0*, so the current system time of 16201.8 sec is stored as the initialization time for *t2*. ME's new state estimate remains unchanged from the previous cycle. Since location 6's outgoing transition is labeled **True**, it is enabled. After taking the enabled transition, the marked locations are 1 and 7.

Cycle ($N_1 + 3$), $t^{abs} = 16202.3$ sec – Both clock variables *t1 = 16202.3 - 0.5 = 16201.8 sec* and *t2 = 16202.3 - 16201.8 = 0.5 sec* are updated. Since the value of *t2* is less than 4 minutes, only the transition from location 7 to itself is enabled. Thus, in the next cycle, locations 1 and 7 remain marked.

Again skipping to the next "interesting" execution cycle, N_2, which occurs 239.9 seconds later, the state estimate $\{$*engine=standby, nav=inertial, att=cruise-orient, lander=connected, entry=not-initiated*$\}$ is unchanged. Locations 1 and 7 remain marked from cycles ($N_1 + 3$) to N_2.

Cycle N_2, $t^{abs} = 16442.2$ sec – Clock variables *t1 = 16442.2 - 0.5 = 16441.7 sec* and *t2 = 16442.2 - 16201.8 = 240.4 sec* are updated. Since the value of *t2* is greater than 4 minutes, only the transition from location 7 to composite location 8 is enabled. After taking the enabled transition and marking location 8's starting subautomata, the marked locations for the next cycle are 1, 8, 9, 10, and 11.

Cycle ($N_2 + 1$), $t^{abs} = 16442.7$ sec – We update the two clocks. The maintenance constraint corresponding to non-entailment of *entry=initiated* on location 8 holds for the current state estimate, so all its subautomata remain marked. The only goal constraint asserted is *att=entry-orient*. After MR issues the first command that progresses toward the goal, ME indicates that *att=entry-orient* is not yet achieved. Consequently, location 10 remains marked in the next cycle. The transitions from locations 9 to 9, 9 to 10, and 11 to 11 are enabled by the current state. After taking these transitions, locations 1, 8, 9, 10 and 11 remain marked.

It takes 10 seconds for the spacecraft to turn to the entry orientation, so again we skip to the next "interesting" cycle, N_3, which occurs 10.2 seconds later.

Cycle N_3, $t^{abs} = 16452.9$ sec – We update the two clocks. The maintenance constraint on location 8 still holds, and the only goal constraint asserted is *att=entry-orient*. This time, ME confirms achievement of this goal. Taking the enabled transitions leads to a new set of marked locations: 1, 8, 9, 10 and 12. Note that location 10 remains marked despite achievement of its goal constraint, due to re-marking by the transition from 9 to 10. This reflects a desire to hold the spacecraft at the entry orientation.

Cycle ($N_3 + 1$), $t^{abs} = 16453.3$ sec – We update the two clocks. The maintenance constraint on location 8 still holds. The configuration goal passed to MR now consists of *att=entry-orient* and *lander=separated*. The following ME update confirms achievement of both goal states. Since location 12 has no outgoing transitions, its thread of execution dies, leading to marked locations 1, 8, 9, and 10.

Remaining Cycles – Execution continues with locations 1, 8, 9, and 10 remaining marked (and *att=entry-orient* continuing to be asserted as a goal) until ME indicates that *entry=initiated*. At this point, location 8's maintenance constraint becomes violated. It has no outgoing transitions, so its thread of execution dies. Since none of location 1's subautomata remain marked, it also becomes unmarked, and execution of the Mars entry THCA terminates.

7 Discussion

Our Timed Model-based Executive has been implemented in C++, as an extension of the Titan model-based executive [14]. The current implementation of our deductive controller is based on the OpSat system, which was flown on NASA's Deep Space One mission as the core of the Livingstone executive. The performance of the deductive controller is documented in [11, 12].

In this paper, we have presented the semantics for a Timed Model-based Executive that operates on a plant model represented as a factored POMDP. A more general specification of TMBP is provided in [6], where the semantics of a physical plant is modeled as a factored partially observable *semi-Markov* decision process. This leads to a more powerful Timed Model-based Executive, capable of reasoning about time-dependent plant behaviors, such as non-deterministic transition latencies (e.g., the engine takes between 30 and 60 seconds to heat up before it reaches standby mode). Reference [6] also discusses a demonstration of the Timed Model-based Executive performed in the context of a full Mars entry, descent and landing scenario.

Ongoing work in TMBP is directed toward applying and adapting our executive to other aerospace applications and scenarios, such as surface rover operations. In particular, we are in the process of integrating our TMBP technology into the Mission Data System [2], a multi-mission information and control architecture for the next generation of robotic exploration spacecraft, currently under development at NASA JPL. This infusion effort is being carried out in the context of the Mars Science Laboratory, a rover mission scheduled for launch in 2009.

8 Acknowledgments

We would like to thank Paul Elliott and Seung Chung for their contributions to the implementation of the Timed Model-based Executive. We also extend thanks to Greg Sullivan for his feedback on drafts of this paper. Finally, we would like to acknowledge the rest of the Model-based Embedded and Robotic Systems group at MIT and JHU Applied Physics Lab, for their past and ongoing efforts in the development of model-based programming. This work was supported by NASA's Cross Enterprise Technology Development Program under contract NAG2-1466.

References

1. R. Alur and D.L. Dill. A theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

2. D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks. Software architecture themes in JPL's Mission Data System. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Portland, OR, 1999.
3. R.J. Firby. The RAP language manual. Animate Agent Project Working Note AAP-6, University of Chicago, March 1995.
4. D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
5. T.A. Henzinger, Z. Manna, and A. Pnueli. Timed Transition Systems. In *Proceedings of the REX Workshop on Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer-Verlag, 1992.
6. M.D. Ingham. Timed model-based programming: Executable specifications for robust mission-critical sequences. Doctoral Thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge, MA, June 2003.
7. L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.
8. J. Kurien and P. Nayak. Back to the future for consistency-based trajectory tracking. In *Proceedings of AAAI-00*, pages 370–377, 2000.
9. M.L. Puterman. *Markov Decision Processes*. Wiley Interscience, New York, 1994.
10. R. Simmons and D. Apfelbaum. A Task Description Language for robot control. In *Proceedings of the Conference on Intelligent Robotics and Systems*, Vancouver, Canada, October 1998.
11. B.C. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI-96*, pages 971–978, 1996.
12. B.C. Williams and P. Nayak. A reactive planner for a model-based executive. In *Proceedings of IJCAI-97*, pages 1178–1185, 1997.
13. B.C. Williams and M.D. Ingham. Model-based programming: Controlling embedded systems by reasoning about hidden state. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 508–524. Springer-Verlag, 2002.
14. 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.
15. T. Young et al. Report of the Mars Program Independent Assessment Team. Technical report, NASA, March 2000.