Using Hybrid Automata for Early Spacecraft Design Evaluation

Jafar Akhundov, Michael Reissner, and Matthias Werner

Operating Systems Group, TU Chemnitz, Germany
`jafar.akhundov | michael.reissner | matthias.werner@cs.tu-chemnitz.de`

Abstract. Designing a spacecraft is a costly and time-consuming activity. Engineers in this domain therefore rely on model-based analysis and verification methods, which give them feedback about expectable mission parameters already in the early design phases.
We present an operational semantics for a customized language for spacecraft design by means of linear time invariant hybrid automata. The concept allows the modelling of concurrently operating satellite components. An existing domain-specific language serves as foundation for the formalism, which ensures backward compatibility to existing models. Possible semantical ambiguities of the original syntax are discussed and possibilities of their resolution in the automata representation are given.

1 Introduction

The design of spacecraft systems is commonly regarded as a complex task. Studies have shown that up to 85% of the overall spacecraft costs result from decisions made in the early design phases [Wil15].

Aerospace industry today applies the concurrent engineering (CE) development methodology for getting early feedback about design alternatives [STF$^+$13]. In contrast to the classical waterfall development, CE encourages short iterative development cycles with constant evaluation of the system design alternatives. Such a development process demands an unambiguous description of specific spacecraft aspects, such as satellite trajectories, communication mechanisms, or thermal effects, so that an early reasoning about system properties becomes possible [SFS03]. Information such as the overall system mass, the power consumption and the costs for development must be derivable from the system models. These results are relevant for both the spacecraft designers and the customers.

The need for unambigious design descriptions already led to the utilization of formal methods in the aerospace domain. Several approaches have been discussed in earlier publications [Tru04] [Kel97] [LS15] [VEM08]. One salient description technique is the domain specific language (DSL) proposed by Schaus et al. [STF$^+$13]. It was retrospectively applied to an existing satellite project and has shown its general applicability in practice [WL99] [STF$^+$13] [Tie13] [FLK$^+$08]. The original authors, however, already identified that this ad-hoc DSL specification lacks a clear definition of semantics. This becomes especially relevant when analyzing non-functional aspects such as consistency and correctness. Furthermore, the missing relation to known formal concepts prevents the re-use of a wide set of existing tools, analytical methods and feasability checks.

Defining semantics of the given DSL by means of timed automata has already been done informally by the authors in [ATW15] and [ASGW16]. However, timed automata lack expressivity to model dynamics of second and higher order. Therefore, linear time-invariant hybrid automata (LTI-HA) have been introduced in [ATW16], which both support higher order dynamics and address the issue of composability for modelling spacecraft systems at early design stages. The hallmark of the LTI-HA is the property of superposition of continuous functions describing the continuous dynamics of different modelled components.

This article targets the question if the existing DSL by Schaus et al. [STF+13] can be mapped into a formal description that enhances the original language concepts with a clear operational semantics. Since spacecraft modules are mainly defined in terms of their static parameters and approximate dynamic executions, the set of applicable approaches includes methods such as hybrid/timed automata, timed CSP and timed Petri nets. Feedback from the aerospace domain shows that automata concepts are comparatively easy to grasp for engineers unrelated to computational logic and formal modelling, so we chose them as the target formalism.

The article starts with a general definition of the LTI hybrid automata in Section 2. The text continues with a structured presentation of the foundational DSL in Section 3. Sections 4 and 5 show how the DSL concepts can be mapped to LTI-HA formalism and discuss the implications of the different mapping approaches. This includes a discussion of how the hybrid automata semantics can be transitioned back by extending the original language.

2 Linear Time-Invariant Hybrid Automata (LTI-HA)

Hybrid automata (HA) are available in different variations from the existing related work [MMP92] [LSV03] [Hen00] [LLL09] [SL02]. Our tailored version relies on the following definitions:

Definition 1 (Valuation of a variable). *A valuation V for a set of variables X is a function $V : X \mapsto \mathbb{R}$ which assigns to each continuous variable x a value $V(x)$.*

Definition 2 (A constraint). *A constraint $\kappa(X)$ on a set X is defined as*

$$\begin{aligned}\kappa(X) ::= \quad & f(X,\mathbb{R}) = g(X,\mathbb{R}) \mid f(X,\mathbb{R}) < g(X,\mathbb{R}) \mid f(X,\mathbb{R}) \leq g(X,\mathbb{R}) \mid f(X,\mathbb{R}) > g(X,\mathbb{R}) \mid \\ & f(X,\mathbb{R}) \geq g(X,\mathbb{R}) \mid \neg\kappa_i(X) \mid \kappa_i(X) \wedge \kappa_j(X),\end{aligned}$$

where functions f, g are any functions.

Definition 3 (A satisfaction relation for constraints). *A satisfaction relation $\models \subseteq X \times C(X)$, where X is a set of variables, $C(X)$ is a set of constraints $\kappa(x)$ over X and $\kappa_i, \kappa_j \in C(X)$, is defined as follows:*

$$\begin{array}{l}\kappa(X) \models \text{true} \\ \kappa(X) \models f(X,K_f) \ = \mid < \mid \leq \mid > \mid \geq \ g(X,K_g) \text{ iff } f(V(X),K_f) \ = \mid < \mid \leq \mid > \mid \geq \ g(V(X),K_g) \\ \kappa(X) \models \neg\kappa_i \qquad \text{iff} \quad V(X) \not\models \kappa_i \\ \kappa(X) \models \kappa_i \wedge \kappa_j \quad \text{iff} \quad V(X) \models \kappa_i \wedge V(X) \models \kappa_j,\end{array}$$

where constants $K_f, K_g \in \mathbb{R}$.

Definition 4 (State of a hybrid system). *The state of a hybrid system at some point t in time is a pair $\sigma(t) = (L, V)$ consisting of two time-dependent components: the discrete state L which represents the currently active location and the continuous state V which stands for the current valuation of continuous variables.*

Definition 5 (Event). *An event is defined as a signal which carries no value but provides its information by its presence or absence at any given time:*

$$e : \mathbb{R}^+ \mapsto \{present, absent\},$$

where $\mathbb{R}^+$ represents the global time reference.

The occurrence of an event is an infinitely brief change of its value from absent to present and back again. Event is a construct needed to enable transitions, so that a hybrid system could react to changes in the environment. Generation and consumption of an event are represented by symbols "$\uparrow$" and "$\downarrow$", respectively, i.e. $e\uparrow$ is the generation of an event e and $e\downarrow$ is the consumption of an event e.

Definition 6 (LTI-HA). *A linear time-invariant hybrid automaton $\mathcal{H}$ is a tuple $(\mathcal{L}, \mathcal{T}, \mathcal{X}, \mathcal{S}_I, \mathcal{S}_O, G, \mathcal{F}, \mathcal{I})$, where:*

- *$\mathcal{L} = \{L_1, \ldots, L_n\}$ is a set of discrete **locations** (or **modes**);*
- *$\mathcal{T} \subseteq \mathcal{L} \times \mathcal{L}$ is a (not necessarily complete) **transition relation**, where $\times$ denotes a cartesian product. $\mathcal{L}$ represents the nodes and $\mathcal{T}$ - the edges of a graph with possible loops, called the **control graph**;*
- *$\mathcal{X}$ is a set of continuous real-valued variables, called **state variables**. Time in this context refers to a designated variable $t \in \mathcal{X}$;*
- *$\mathcal{S}_I$ and $\mathcal{S}_O$ are two disjoint sets of **input and output events**, respectively, which define the event signature of the automaton;*
- *A set of **guards** G which are tuples of form $(\tau, C(\mathcal{X}), \mathcal{E}, \mathcal{A})$, where a set of valid constraints $C(\mathcal{X})$, a set of input events $\mathcal{E} \subseteq 2^{S_I}$ and a set of output events $\mathcal{A} \subseteq 2^{S_O}$ are associated with every transition $\tau \in \mathcal{T}$. For any transition τ, several guards can be defined;*
- *The change of any $x \in \mathcal{X}$, except for t, at any time point is described by the **flow function** f_L of the currently active location describing the continuous change of system state:*

  $$x_i(t) = f_{x_i,L}(t).$$

  We are restricting the flow functions only to those which are valid solutions for ordinary linear time-invariant differential equations of order k. This restriction guarantees that flow functions fulfil the superposition property. Therefore, for two simultaneously active locations L_1 and L_2 of two concurrent hybrid automata the resulting flow function for a global continuous variable x_i is defined as:

  $$x_i(t) = f_{x_i,L_1}(t) + f_{x_i,L_2}(t)$$

  $\mathcal{F}$ denotes the set of flow functions for every location in $\mathcal{L}$ and every variable, except for t.

Definition 7 (Initial configuration). *$\mathcal{I}$ is the **initial state/configuration** of the system $\sigma(t_0) = (L_\mathcal{I}, V_\mathcal{I})$, where $L_\mathcal{I} \in \mathcal{L}$ is the initial active mode, $V_\mathcal{I} : \mathcal{X} \mapsto \mathbb{R}$ is the initial valuation of all the variables in $\mathcal{X}$ and $V_\mathcal{I}(t) = t_0$.*

2.1 Semantics of the LTI-HA

Once an LTI hybrid automaton is initialised by some configuration $\mathcal{I}$, it is executed in the initial location $L_\mathcal{I}$, starting with the initial valuation $V_\mathcal{I}(\mathcal{X})$. If some variable is not initialised in $\mathcal{I}$ then

its initial value must be configured by another automaton.
For the ease of reference, the value of a variable $x \in X$ at the moment a location $L \in \mathcal{L}$ becomes active, will be referred to as $\bar{x}$.

Time Semantics In each active location, time elapses at the same rate: $\forall L \in \mathcal{L} : \dfrac{dt}{dt} = 1$. The evaluation of flow functions is only based on the duration of time interval spent in the corresponding location:

$$\forall f_{x_i,L_j}(t), \bar{t}_1, \bar{t}_2 \in \mathbb{R}, \Delta = t - \bar{t}_1 = t - \bar{t}_2 : f_{x_i,L_j}(t - \bar{t}_1) = f_{x_i,L_j}(t - \bar{t}_2) = f_{x_i,L_j}(\Delta)$$

Transitions are timeless. The global time reference is represented by a continuous variable $t \in \mathcal{X}$ which is progressing along with the automaton execution.

Guard Semantics If at some time point t for a system with the state $\sigma(t) = (L, V(\mathcal{X}))$, all of the constraints of a guard g_τ of an outgoing transition $\tau = (L, L')$ evaluates to *true* and all of the input events of that guard occur, L' becomes the new active location without delay and all the output events $e \in \mathcal{A}(\tau)$ occur. In such a case, a guard is called to be *enabled*. Formally:

$$\text{guard activation}\ \frac{\begin{array}{c}\sigma(t) = (L, V(\mathcal{X})) \quad \tau = (L, L') \quad \exists g_\tau = (\tau, C_\tau(\mathcal{X}), \mathcal{E}_\tau, \mathcal{A}_\tau)) \ : \\ \forall e \in \mathcal{E}_\tau : e = \text{present} \quad \forall \kappa(x) \in C_\tau(\mathcal{X}) \models \text{true}\end{array}}{\forall e \in \mathcal{A}(\tau) : \uparrow\! e \quad \sigma'(t) = (L', V(\mathcal{X}))}$$

If more than one guard of an outgoing transition is enabled, the one to use is chosen non-deterministically.

Event Semantics The output events have a one-to-all semantics, that is, every output event is broadcasted and can be (simultaneously) consumed by the concurrent LTI hybrid automata (should they exist) if the corresponding transitions are enabled and have those events as inputs. The input events can only have a single source. Transitions where the input events of one are the output events of the other are called *synchronising* transitions and can only occur simultaneously. Other transitions are *non-synchronising.* Events do not have duration.

Two non-synchronising transitions (with no intersections of the corresponding input and output events on the corresponding guards) of two hybrid automata $\mathcal{H}^1, \mathcal{H}^2$ can be either taken in any order (the first two cases), or synchronously (the third case). $\prec$-relation represents causal ordering. Synchronising transitions can only be taken synchronously (the else-case). Formally:

$$\text{transition sync}\ \frac{\begin{array}{c}\forall \text{ enabled } g^1 = (\tau^1, C^1(\mathcal{X}^1), \mathcal{E}^1, \mathcal{A}^1) \in G^1, \quad g^2 = (\tau^2, C^2(\mathcal{X}^2), \mathcal{E}^2, \mathcal{A}^2) \in G^2 \\ \text{if } (\mathcal{E}^2 \cap \mathcal{A}^1 = \emptyset \wedge \mathcal{E}^1 \cap \mathcal{A}^2 = \emptyset)\end{array}}{\begin{array}{l}\text{Either: } \mathcal{E}^1\!\downarrow\ \prec \mathcal{A}^1\!\uparrow\ \prec \mathcal{E}^2\!\downarrow\ \prec \mathcal{A}^2\!\uparrow \\ \text{or: } \mathcal{E}^2\!\downarrow\ \prec \mathcal{A}^2\!\uparrow\ \prec \mathcal{E}^1\!\downarrow\ \prec \mathcal{A}^1\!\uparrow \\ \text{or: } (\mathcal{E}^1 \cup \mathcal{E}^2)\!\downarrow\ \prec (\mathcal{A}^1 \cup \mathcal{A}^2)\!\uparrow \\ \text{else: } (\mathcal{E}^1 \cup \mathcal{E}^2) \backslash (\mathcal{A}^1 \cup \mathcal{A}^2))\!\downarrow\ \prec (\mathcal{A}^1 \cup \mathcal{A}^2)\!\uparrow,\end{array}} \tag{1}$$

Events are never buffered and can only be consumed if $\mathcal{E}^i \cap \mathcal{A}^j \neq \emptyset \wedge \mathcal{E}^i \not\prec \mathcal{A}^j \wedge \mathcal{A}^j \not\prec \mathcal{E}^i$. That is, events generated by one transition in an automaton cannot be consumed by the next transition of the same automaton even if it becomes enabled with no time delay.

Execution Semantics At any time point, exactly one location is *active*, beginning with $L_{\mathcal{I}}$.

Definition 8. *Execution semantics of an LTI-HA can be given as an operational semantics with two rules, for instantaneous discrete steps(ds) and continuous time steps(ts), respectively:*

$$ds\;\frac{\sigma(t) = (L, V(\mathcal{X})) \quad \exists((L,L'), C_\tau(\mathcal{X}), \mathcal{E}_\tau, \mathcal{A}_\tau)) : \ \forall e \in \mathcal{E}_\tau : e = \mathit{present} \quad \forall \kappa(x) \in C_\tau(\mathcal{X}) \models \mathit{true}}{\sigma(t) = (L, V(\mathcal{X})) \xrightarrow{\mathcal{A}_\tau} (L', V(\mathcal{X})) = \sigma'(t)}$$

$$ts\;\frac{\forall x \in \mathcal{X} : f_{x,L}(0) = V(x) \quad \forall x : V'(x) = \bar{x} + f_{x,L}(\varDelta) \quad \mathit{in}\ [\bar{t}, \bar{t} + \varDelta] : \nexists\ \mathit{enabled}\ g_{(L,L')}}{\sigma(t) = (L, V(\mathcal{X})) \xrightarrow{\varDelta} (L, V'(\mathcal{X})) = \sigma'(t + \varDelta)}$$

with $V(\mathcal{X}), V'(\mathcal{X})$ being valid valuations over the set $\mathcal{X}$.

An execution step $\rightarrow = \xrightarrow{\mathcal{A}_\tau} \vee \xrightarrow{\varDelta}$ is thus either a discrete or continuous step. An execution is a sequence $\pi = \sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow ...$ with $\sigma_0 = \mathcal{I}$. A state is **reachable** from the initial configuration $\mathcal{I}$ if there is an execution of $\mathcal{H}$ leading to it. Such an execution can be controlled externally by timely enabling of the input events of $\mathcal{H}$. Since an LTI-HA can be non-deterministic, many different executions are possible. Generating a control sequence of external events $\mathcal{C} : \mathbb{R}^+ \mapsto S_I$ is not a part of the model but a task for an external solver which lies outside of the scope of this text.

3 DSL-Based Spacecraft Description

In the published DSL syntax [STF+13], an engineer can define different system modules and system parameters (called **operational state variables**) which can be changed by the modules' activities. The values of these parameters, at any time point, define the global system state at that moment.

- The *solver* construct determines the actual search algorithm which is used to find an appropriate solution in the state space of all possible system states.
- The *state parameters* together with the *initial parameter values* define system parameters which determine spacecraft system's continuous state and its initial value.
- The *operational states* represent the modules of the satellite being active which represent the discrete state of the system.
- The *starting state* defines the initial active module
- The *operational state changes* define rates of change of system parameters by modules' activities.
- For some of the state parameters, there could exist upper or lower bounds that are determined in the *operational constraints* construct.
- The goal of the whole mission can be defined as a set of state parameter values which can be set in the *operational goals* section.
- External events and intervals are determined by using external *simulation modules* that are parameterized to compute satellite trajectories and generate periodic external events for module activation [STF+13] [DCy10]. However, those events are not explicitly specifiable by the language. Only the construct for a simulation step is given, since the rest is initialization of the satellite's trajectory and coordinates of the ground stations for down-link and up-link communications. This simulation step length determines the time interval during which the system parameters change by the amount given in the operational state changes description.

Given the DSL syntax concepts as a foundation, a satellite system is now a collection of subsystems - *modules*, e.g. battery, communication module, payload, etc. Each module can be described in terms of system parameters like mass, dimensions and parameter changes that module exhibits on global system state per unit time if active. Each module may be activated periodically or asynchronously by predefined set of events and remain active as long as none of the constraints is violated, or/and as long as the triggering interval occurrence has not been deactivated. The current implementation (hence, semantics as well) imply that each module can be only in two modes, On, in which case it changes the system's parameters according to the change rates defined in the Operational State Changes construct, and Off, in which case no changes occur.

Module activities can be scheduled in a sequence or concurrently, depending on the assumption made about the underlying execution model. Initially, Schaus et al. [STF$^+$13] used a sequential execution model to generate and traverse system's state space.

Drawbacks of this approach have been recognized by the authors [STF$^+$13], and concurrent semantics has been introduced and discussed in [ATW15] [ASGW16]. Support for concurrency is natural and leads to an increased expressiveness of the model and allow for more realistic derivable analytical results. Also, concurrent model leads to 'denser' plans, and thus the probability of a false negative result of the feasibility checking is lower [ATW15] [ASGW16].

The mission goal is represented as a predicate over the vector of mission parameters which defines the desirable mission state. Should this state be reachable by module activities in the mission time, the mission is said to be a success. Given that, a *schedule* or a *mission plan* is a mapping of module's changes of system parameters onto the time axis. The plan is said to be feasible if it satisfies both mission constraints and achieves mission goals in a given time interval, i.e. if the result of the plan execution is a mission success.

4 Operational DSL Semantics

Starting from the informal discussion in the last section, we are are now formulating the formal semantics of the existing DSL with the help of LTI-HA. There are at least two possible interpretations for the intended DSL semantics with respect to the potential system states. In both cases, a system is described by a single LTI-HA.

In the first interpretation, every active discrete state in $\mathcal{L}$ corresponds to a single active satellite module. This interpretation represents the *modular view* of an engineer and corresponds to the initial implementation and application of the DSL syntax [STF$^+$13].

The Idle state represents a special activity when none of the satellite modules is enabled and therefore must be in the location set $\mathcal{L}$ of the automaton. For reachability analysis, the set of mission goal predicates is $\mathcal{Z}$. Building the HA follows then from the inference rules in Figure (2). It is assumed that for unspecified flow functions the rate of change is zero, and for guards unspecified for certain state parameters evaluate to *true*.

There are several considerable disadvantages of this interpretation. The first is lack of support for concurrency, that is, satellite modules can only be active exclusively at any time unit which is not realistic [ATW15], [ASGW16]. The second is that language itself does not provide any information on the binding of events $\mathcal{S}$ to automata transitions, although configuration for external event generator can be provided (orbit parameters and simulation step). In the initial language implementation [STF$^+$13] most of this information is provided ad hoc at the phase of translation of the DSL representation into an executable model. That is, the language is not complete or self-contained. The third problem is the HA-formalism artifact. If a HA has a pair of transient events

Solver

⟨*solver*⟩ ::= 'Set Solver' ⟨*solver-path*⟩ 'End'

State Parameters

⟨*state-parameter*⟩ ::= 'Parameter' ⟨*id*⟩ ';'
⟨*state-params*⟩ ::= 'Describe State Parameters' {⟨*state-parameter*⟩} 'End'

Initial State Parameter Values

⟨*state-parameter-init*⟩ ::= 'Parameter' ⟨*id*⟩ 'Set Value' ⟨*real-number*⟩ ';'
⟨*state-params-init*⟩ ::= 'Describe Initial Parameter Values' {⟨*state-parameter-init*⟩} 'End'

Operational States

⟨*module-activity*⟩ ::= 'State' ⟨*id*⟩ ';'
⟨*module-activities*⟩ ::= 'Describe Operational States State Idle ;' {⟨*state-parameter*⟩} 'End'

Starting State

⟨*module-init*⟩ ::= 'Describe Starting State' ⟨*id*⟩ 'End'

Operational State Changes

⟨*change*⟩ ::= 'Changes Parameter' ⟨*id*⟩ 'by' ⟨*real-number*⟩ ['do not exceed' ⟨*real-number*⟩] ';'
⟨*multi-change*⟩ ::= 'Changes Parameter' ⟨*id1*⟩ 'by' ⟨*real-number*⟩ ['do not exceed' ⟨*real-number*⟩]
    'and shifts effective Value to Parameter' ⟨*id2*⟩ ';'
⟨*state-change*⟩ ::= 'State' ⟨*id*⟩ {change | multichange} 'End'
⟨*state-changes*⟩ ::= 'Describe Operational State Changes' {⟨*state-change*⟩} 'End'

Operational Constraints

⟨*interval-constraint*⟩ ::= 'Constraint' ⟨*idc*⟩ 'for Parameter' ⟨*idp*⟩
    'Between' ⟨*real-number*⟩ 'And' ⟨*real-number*⟩ ['Accounts For State' ⟨*ids*⟩] ';'
⟨*bound*⟩ ::= 'Constraint' ⟨*idc*⟩ 'for Parameter' ⟨*idp*⟩
    'Below'|'Above' ⟨*real-number*⟩ ['Accounts For State' ⟨*ids*⟩] ';'
⟨*constraints*⟩ ::= 'Describe Operational Constraints' {⟨*interval-constraint*⟩ | ⟨*bound*⟩} 'End'

Operational Goals

⟨*bound-goal*⟩ ::= 'Parameter' ⟨*id*⟩ 'Below'|'Above' ⟨*real-number*⟩ ';'
⟨*op-goals*⟩ ::= 'Describe Operational Goals' {⟨*bound-goal*⟩} 'End'

Simulation Modules

⟨*time*⟩ ::= 'Executing Time' ⟨*id*⟩ 'with StepSize' ⟨*real-number*⟩ ';'

Fig. 1: Syntactical structure of the existing DSL for spacecraft design descriptions, derived from [STF+13]. Some non-terminals, such as identifiers, are omitted due to space restrictions.

(L_i, L_j) with two enabled transitions between them, then such an automaton can experience Zeno behavior, that is an infinite amount of timeless discrete transitions can occur in a finite amount of time. This is possible to avoid if e.g. the automaton is forced to remain for at least Δ amount of time in every mode.

The less significant drawback is redundancy of the language. For example, the introduction of the "multi-change" construct which can be expressed by two "change" constructs. In the same way, the optional syntactical construct "do not exceed ⟨*real-number*⟩" in the "change" and "multi-

$$\text{operational-states}\,\frac{\text{'State'}\langle id\rangle}{id \in \mathcal{L}} \qquad \text{transitions}\,\frac{}{\forall l, k \in \mathcal{L}, l \neq k : (l, k) \in \mathcal{T}}$$

$$\text{state-parameter}\,\frac{\text{'Parameter'}\langle id\rangle}{id \in \mathcal{X}}$$

$$\text{state-parameter-init}\,\frac{\text{'Parameter'}\langle id\rangle\text{'Set Value'}\langle num\rangle \qquad id \in \mathcal{X} \qquad num \in \mathbb{R}}{V_{\mathcal{I}}(id) = num}$$

$$\text{module-init}\,\frac{\text{'Describe Starting State'}\langle id\rangle\text{'End'} \qquad id \in \mathcal{L}}{L_{\mathcal{I}} = id}$$

$$\text{state-change}\,\frac{\text{'State'}\langle \textit{id-state}\rangle\{\text{change}|\text{multi-change}\}\text{'End'} \qquad \textit{id-state} \in \mathcal{L}}{\begin{array}{c}\forall\{\text{change}\} \in \{\text{change}|\text{multi-change}\} : \text{change}\,\frac{\textit{id-state}}{} \\ \forall\{\text{multi-change}\} \in \{\text{change}|\text{multi-change}\} : \text{multi-change}\,\frac{\textit{id-state}}{}\end{array}}$$

$$\text{change}\,\frac{\text{'Changes Parameter'}\langle id\rangle\text{'by'}\langle num\rangle \qquad id \in \mathcal{X} \qquad num \in \mathbb{R}}{f_{id,\textit{id-state}}(t) = num \cdot t}$$

$$\text{multi-change}\,\frac{\begin{array}{c}\text{'Changes Parameter'}\langle id_1\rangle\text{'by'}\langle num\rangle\text{'and shifts effective Value to Parameter'}\langle id_2\rangle \\ id_1, id_2 \in \mathcal{X} \qquad\qquad num \in \mathbb{R}\end{array}}{f_{id_1,\textit{id-state}}(t) = num \cdot t, f_{id_2,\textit{id-state}}(t) = -num \cdot t}$$

$$\text{interval-c}\,\frac{\begin{array}{c}\text{'Constraint'}\langle id_c\rangle\text{'for Parameter'}\langle id_p\rangle\text{'Between '}\langle num_1\rangle\text{'and'}\langle num_2\rangle \\ id_p \in \mathcal{X} \qquad num_1, num_2 \in \mathbb{R} \qquad (num_1 < num_2)\end{array}}{\begin{array}{l}\forall L \in \mathcal{L} : f_{id_p,L}(t) : \dot{f}_{id_p,L} \neq 0 : \\ \quad \forall L_{\text{from}} \neq L \in \mathcal{L} : \quad \exists((L_{\text{from}}, id_l)\{id_p > num_1, id_p < num_2\}, s_i, s_o) \\ \quad \forall L_{\text{to}} \neq L \in \mathcal{L} : \quad \exists((L, L_{\text{to}})\{id_p \leq num_1, id_p \geq num_2\}, s_i, s_o)\end{array}}$$

$$\text{interval-c-state}\,\frac{\begin{array}{c}\text{'Constraint'}\langle id_c\rangle\text{'for Parameter'}\langle id_p\rangle\text{'Between '}\langle num_1\rangle\text{'and'}\langle num_2\rangle\text{'Accounts For State'} < id_l > \\ id_p \in \mathcal{X} \qquad num_1, num_2 \in \mathbb{R} \qquad (num_1 < num_2) \\ id_l \in \mathcal{L} \qquad \exists f_{id_p,id_l}(t) : \dot{f}_{id_p,id_l}(t) \neq 0\end{array}}{\begin{array}{l}\forall L_{\text{from}} \neq id_l \in \mathcal{L} : \quad \exists((L_{\text{from}}, id_l)\{id_p > num_1, id_p < num_2\}, s_i, s_o) \\ \forall L_{\text{to}} \neq id_l \in \mathcal{L} : \quad \exists((id_l, L_{\text{to}})\{id_p \leq num_1, id_p \geq num_2\}, s_i, s_o)\end{array}}$$

$$\text{bound}\,\frac{\begin{array}{c}\text{'Constraint'}\langle id_c\rangle\text{'for Parameter'}\langle id_p\rangle\text{'Below'}|\text{'Above'}\langle num\rangle \\ id_p \in \mathcal{X} \qquad num \in \mathbb{R}\end{array}}{\begin{array}{l}\forall L \in \mathcal{L} : \text{sgn}(f_{id_p,L}(t)) == 1|\text{sgn}(f_{id_p,L}(t)) == -1)) : \\ \quad \forall L_{\text{from}} \neq L \in \mathcal{L} : \exists((L_{\text{from}}, L), \{id_p < num|id_p > num\}, s_i, s_o) \\ \quad \forall L_{\text{to}} \neq L \in \mathcal{L} : \exists((L, L_{\text{to}}), \{id_p \geq num|id_p \leq num\}, s_i, s_o)\end{array}}$$

$$\text{bound-for-state}\,\frac{\begin{array}{c}\text{'Constraint'}\langle id_c\rangle\text{'for Parameter'}\langle id_p\rangle\text{'Below'}|\text{'Above'}\langle num\rangle\text{'Accounts For State'} < id_l > \\ id_p \in \mathcal{X} \qquad num \in \mathbb{R} \qquad id_l \in \mathcal{L} \qquad \text{sgn}(f_{id_p,id_l}(t)) == 1|\text{sgn}(f_{id_p,id_l}(t)) == -1))\end{array}}{\begin{array}{l}\forall L_{\text{from}} \neq id_l \in \mathcal{L} : \exists((L_{\text{from}}, id_l), \{id_p < num|id_p > num\}, s_i, s_o) \\ \forall L_{\text{to}} \neq id_l \in \mathcal{L} : \exists((id_l, L_{\text{to}}), \{id_p \geq num|id_p \leq num\}, s_i, s_o)\end{array}}$$

$$\text{guards}\,\frac{}{\forall L_1, L_2 \in \mathcal{L} : ((L_1, L_2), C(\mathcal{X}), s_i, s_o) = ((L_1, L_2), \bigcup_{g_{(L_1,L_2)}} C(x_i \in \mathcal{X}), s_i, s_o)}$$

$$\text{goal}\,\frac{\text{'Parameter'}\langle id\rangle\text{'Below'}|\text{'Above'}\langle num\rangle \qquad id \in \mathcal{X}}{(id, < | >, num) \in \mathcal{Z}}$$

Fig. 2: Inference rules for the modular view of the spacecraft

change" rules is redundant to the constraints which can be defined explicitly for modes ("Accounts for State"). Also, "interval-constraint" is nothing more than syntactical sugar and can be represented by two constraints, one for the upper and lower bounds, respectively.

In the second possible interpretation, an engineer abstracts away from isolated components and thinks in terms of the whole spacecraft system. Thus, every mode of the automaton corresponds to all possible *composed* activities (flows) of satellite modules ($M \subseteq 2^{\mathcal{M}} \cup \text{Idle}$). The corresponding inference rules for translation of the DSL-description into a HA are the same as in the previous case, (Fig. 2) but now every operational state is not a disjunct execution of a single satellite module but one of all possible combinations of overlapping executions. Composition of all possible module executions is formally defined by the composition operation for LTI-HA[ATW16] but has to be done by the engineer himself. The problem of this approach is the exponential growth of automaton modes (combinations of executions of satellite modules) that must be computed by hand and described in the language. This approach would demand a considerable amount of the engineer's time and effort which is usually limited by a single CE session and would violate the separation of concerns paradigm.

5 Extended Semantics

Given the unsatisfying possible interpretations of the existing language, we are now proposing an enhanced automata definition. This extension targets the direct support for modeling concurrently working satellite components. At the same time, the mappings defined so far should only be changed to a minimal extent, in order to still allow the backward translation into the original DSL syntax.

$$\text{operational-states}\;\dfrac{\text{'State'}\langle id\rangle}{\begin{array}{l}\exists H^{id}(\mathcal{L}^{id} = \{\text{Active, Inactive}\}, \mathcal{T}^{id}, \mathcal{X}^{id}, \mathcal{S}_I^{id}, \mathcal{S}_O^{id}, G^{id}, \mathcal{F}^{id}, \mathcal{I}^{id} = (\text{Inactive}, V_I^{id})) \in \mathcal{H}_s,\\ \mathcal{T}^{id} = \{(\text{Inactive}, \text{Active}), (\text{Active}, \text{Inactive})\}\end{array}}$$

$$\text{state-parameter}\;\dfrac{\text{'Parameter'}\langle id\rangle}{\forall \mathcal{H}^i \in \mathcal{H}_s : id \in \mathcal{X}^i}$$

$$\text{state-parameter-init}\;\dfrac{\text{'Parameter'}\langle id\rangle\text{'Set Value'}\langle num\rangle \qquad num \in \mathbb{R}}{\forall \mathcal{H}^i \in \mathcal{H}_s : V_{\mathcal{I}}^i(id) = num}$$

$$\text{module-init}\;\dfrac{\text{'Describe Starting State'}\langle id\rangle\text{'End'} \qquad \mathcal{H}^{id} \in \mathcal{H}_s}{\forall \mathcal{H}^i \neq \mathcal{H}^{id} \in \mathcal{H}_s : L_{\mathcal{I}}^i = \text{Inactive} \wedge L_{\mathcal{I}}^{id} = \text{Active}}$$

$$\text{state-change}\;\dfrac{\text{'State'}\langle \textit{id-state}\rangle\{\text{change}\}\text{'End'} \qquad \mathcal{H}^{\textit{id-state}} \in \mathcal{H}_s}{\forall \{\text{change}\} \in \{\text{change}\} : \text{change}\;\dfrac{\textit{id-state}}{}}$$

$$\text{change}\;\dfrac{\text{'Changes Parameter'}\langle id\rangle\text{'by'}\langle num\rangle \qquad id \in \mathcal{X}^{id-state} \qquad num \in \mathbb{R}}{f_{id,\text{Active}}^{\textit{id-state}}(t) = num \cdot t}$$

$$\text{bound}\;\dfrac{\begin{array}{c}\text{'Constraint'}\langle id_c\rangle\text{'for Parameter'}\langle id_p\rangle\text{'Below'}|\text{'Above'}\langle num\rangle\\ \forall \mathcal{H}^i \in \mathcal{H}_s : id_p \in \mathcal{X}^i \qquad\qquad num \in \mathbb{R}\end{array}}{\begin{array}{c}\forall \mathcal{H}^i \in \mathcal{H}_s \ni \text{sgn}(f_{id_p,\text{Active}}^i(t)) == 1 \mid \text{sgn}(f_{id_p,\text{Active}}^i(t)) == -1 :\\ \exists((\text{Inactive}, \text{Active}), \{(id_p < num \mid id_p > num\}, s_I, s_O)\\ \exists((\text{Active}, \text{Inactive}), \{id_p \geq num \mid id_p \leq num\}, s_I, s_O)\end{array}}$$

$$\text{bound-for-state}\;\dfrac{\begin{array}{c}\text{'Constraint'}\langle id_c\rangle\text{'for Parameter'}\langle id_p\rangle\text{'Below'}|\text{'Above'}\langle num\rangle\text{'Accounts For State'} < id_l >\\ \exists \mathcal{H}^{id_l} \in \mathcal{H}_s : id_p \in \mathcal{X}^{id_l} \qquad num \in \mathbb{R} \qquad \text{sgn}(f_{id_p,\text{Active}}^{id_l}(t)) == 1 \mid \text{sgn}(f_{id_p,\text{Active}}^{id_l}(t)) == -1\end{array}}{\begin{array}{c}\exists((\text{Inactive}, \text{Active}), \{(id_p < num \mid id_p > num\}, s_I, s_O)\\ \exists((\text{Active}, \text{Inactive}), \{id_p \geq num \mid id_p \leq num\}, s_I, s_O)\end{array}}$$

$$\text{guards}\;\dfrac{}{\forall L_1, L_2 \in \mathcal{L} : ((L_1, L_2), C(\mathcal{X}), s_i, s_o) = ((L_1, L_2), \bigcup_{g_{(L_1,L_2)}} C(x_i \in \mathcal{X}), s_i, s_o)}$$

$$\text{goal}\;\dfrac{\text{'Parameter'}\langle id\rangle\text{'Below'}|\text{'Above'}\langle num\rangle \qquad \forall \mathcal{H}^i \in \mathcal{H}_s : id \in \mathcal{X}^i}{(id, < \mid >, num) \in \mathcal{Z}}$$

Fig. 3: Extended inference rules for a spacecraft system

Given that restriction, we rely our extension on the definition of a separate automaton $\mathcal{H}^i$ for each of the satellite modules. Following the DSL syntax and informal semantics defined in Section 3, each of those automata can only be in two states, *Active* and *Inactive*, combined with different

$$\text{periodic-event}\;\frac{\text{'Event'} < id > \text{'with Period'} < num >}{\begin{array}{l} \{H^{id}(\{\text{Active, Inactive}\}, \mathcal{T}^{id}, \mathcal{X}^{id}, \mathcal{S}_I^{id}, \mathcal{S}_O^{id}, G^{id}, \mathcal{F}^{id}, \mathcal{I}^{id} = (\text{Inactive}, V_I^{id}))\} \in \mathcal{H}_e \\ \mathcal{T}^{id} = \{(\text{Inactive}, \text{Active}), (\text{Active}, \text{Inactive})\} \\ \mathcal{X}^{id} := \{c, t\} \\ f_{c,\text{Inactive}}^{id}(t) := t \\ G = \{((\text{Inactive}, \text{Active}), \{c\text{mod } num == 0\}, \{\}, \{e^{id}\}), ((\text{Active}, \text{Inactive}), \{\}, \{\}, \{\})\} \\ S_O^{id} := \{e^{id}\} \\ S_I^{id} := \emptyset \end{array}}$$

$$\text{periodic-interval}\;\frac{\text{'Event'} < id > \text{'with Period'} < num_1 > \text{and Duration'} < num_2 > \qquad num_1 > num_2}{\begin{array}{l} \{H^{id}(\{\text{Active, Inactive}\}, \mathcal{T}^{id}, \mathcal{X}^{id}, \mathcal{S}_I^{id}, \mathcal{S}_O^{id}, G^{id}, \mathcal{F}^{id}, \mathcal{I}^{id} = (\text{Inactive}, V_I^{id}))\} \in \mathcal{H}_e \\ \mathcal{T}^{id} = \{(\text{Inactive}, \text{Active}), (\text{Active}, \text{Inactive})\} \\ \mathcal{X}^{id} := \{t_P, t_D\} \\ f_{t_P,\text{Inactive}}^{id}(t) := t, f_{t_D,\text{Inactive}}^{id}(t) := 0, f_{t_P,\text{Active}}^{id}(t) := t, f_{t_D,\text{Active}}^{id}(t_D) := t, \\ G = \{((\text{Inactive}, \text{Active}), \{t_P \text{ mod } num_1 == 0\}, \{e^{id}\}), \\ ((\text{Active}, \text{Inactive}), \{t_D \text{ mod } num_2 == 0\}, \{\neg e^{id}\})\} \\ S_O^{id} := \{e^{id}, \neg e^{id}\} \\ S_I^{id} := \emptyset \end{array}}$$

$$\text{state-activation}\;\frac{\text{'State'} < id_l > [\text{'Activated By'} < id_{e,cur} > \{\text{','} < id_{e,next} >\}] \qquad \exists \mathcal{H}^{id_l} \in \mathcal{H}_s \qquad \exists \mathcal{H}^{id_{e,cur}} \in \mathcal{H}_e}{\begin{array}{l} \mathcal{S}_I^{id_l} := \mathcal{S}_I^{id_l} \cup \{e^{id_{e,cur}}\} \\ ((\text{Inactive}, \text{Active}), C^{id_l}(\mathcal{X}^{id_l}), e^{id_{e,cur}}, s_O) \\ \exists(\neg e^{id_{e,cur}}) \in S_O^{id_{e,cur}} : ((\text{Active}, \text{Inactive}), C^{id_l}(\mathcal{X}^{id_l}), \{\neg e^{id_{e,cur}}\}), s_O) \end{array}}$$

Fig. 4: Event semantics for arbitrarily defined syntactic construct

sets of flow functions. The set of continuous variables is the same for all automata and is equivalent to the set of system parameters. Shared variables are allowed for concurrent LTI-HA as long as their initial values do not contradict one another [ATW16, Def. 12]. Each of the modules can become active, should the guard conditions hold (if any present) and synchronization events occur (if any present). There are two types of constraints in the DSL, one is global for all possible operational states, and the other ones are only valid for a specific operational state (which is represented by the *Active* location of the corresponding LTI-HA). Thus, guards are built in such a way that negation of the constraint leads out of the *Active* location, and satisfied a constraint allows the location to be activated. Combinations of all possible bounds for continuous variables build the guards leading to and from the *Active* locations of each LTI-HA describing the corresponding satellite component.

In this representation, most of satellite's modules (as long as there is no explicit dependence between them, represented by events) can become active independently of one another, that is, concurrently. Furthermore, all possible overlappings of module activities can be automatically generated by using the composition operator defined in [ATW16]. The inference rules for such an extended semantic are provided in Figure 3 and Figure 4.

As in the previous section, it is assumed that an absent definition of a flow function in some location L for particular continuous variable $x_i \in \mathcal{X}$ implies $\dot{f}_{x,L}(t) = \bar{x}$ and that guards unspecified for certain state parameters or unspecified domains of state parameters evaluate to *true*.

Figure (4) provides inference rules for periodic point events and periodic intervals. Periodic events are built using one counter variable c which has the rate of change $= 1$. Once the counter reaches its period, event is generated once over the transition from the *Inactive* location to the *Active* location. Since the guard in the opposite direction does not have any constraints or input events, it is always enabled and is taken immediately, leading the LTI-HA back to the *Inactive* location without delay.[1] That is, the *Active* location is called to be *transient* in such a case. Intervals are designed by providing two point events, for start and end of the interval, respectively. For this purpose, two timers are needed, one for the period and one - for the duration. In this case the *Active* location is not transient, since the some time ($=num_2$) is spent in the *Active* location. Also, a state-activation inference rule is given which describes which events and/or intervals are bound to which events.

The whole system is then built by applying the composition operation[ATW16] to all LTI hybrid automata describing the satellite component and events.

5.1 Possible DSL Extension

The back-porting of the extended formal semantics into the original DSL is possible in multiple ways. For supporting the missing concurrency semantics, the DSL can be enhanced with three syntactical constructs, as shown in Figure 5. This allows to define activation events and model satellite's environment in the language. Furthermore, a ⟨*module-activation*⟩ non-terminal can be introduced that defines a dependency of the module activity on a list of corresponding events or intervals. This is shown in the last rule of Figure 5. To eliminate redundancies, the "multi-change" syntactic rule as well as "do not exceed"-part in the "change" rule can be left out from the original language syntax.

⟨*ext-periodic-event*⟩ ::= 'Event' ⟨*id*⟩ 'with Period' ⟨*real-number*⟩ ';'
⟨*ext-periodic-interv*⟩ ::= 'Interval' ⟨*id*⟩ 'with Period' ⟨*real-number*⟩ 'and Duration' ⟨*real-number*⟩ ';'
⟨*ext-occurrences*⟩ ::= 'Describe Events and Intervals' {⟨*ext-periodic-event*⟩ | ⟨*ext-periodic-interv*⟩} 'End'
⟨*module-activation*⟩ ::= 'State' ⟨*id*⟩ ['Activated By' ⟨*id*⟩ {',' ⟨*id*⟩}] ';'

Fig. 5: DSL Enhancements for Concurrency Support: External Periodic Event or Interval and its Binding to a Satellite Module

6 Conclusion

Given the formal semantics definition of the last section, it is now possible to verify system model soundness and completeness, absence of deadlocks, create and optimize execution traces, etc. Schaus et al. [STF+13] introduced the discussed DSL for reachability analysis, thereby providing the participant engineers of a CE session with a feasability feedback on the mission. In order to define reachability of a specific system state, it is necessary to introduce the concept of a trace.

As next steps, we intend to work on the reasoning part for the operational semantics. With the formal semantics of a satellite defined, one can formally verify consistency, completeness of the corresponding model, reachability of the desirable state, absence of deadlocks and Zeno behaviours, create and optimize execution traces, etc. All of these points are equally important as a future work. As a next step, formally defining execution traces of LTI-HA as well as implementing a solver for finding an optimal schedule against predefined criteria will be considered. Furthermore, applicability of our formal approach to different kind of missions (deep space, rovers, swarms, etc) is yet to be investigated.

[1] The same semantics is also possible to achieve without the explicit *Active* state with a single loop transition having the same guard semantics as the first guard of the initial model. For the sake of uniformity with the rest of the models, the first approach was chosen.

References

[ASGW16] Jafar Akhundov, Volker Schaus, Andreas Gerndt, and Matthias Werner. Using timed automata to check space mission feasibility in the early design phases. In *IEEE Aerospace 2016 Proceedings*, Big Sky, Montana, USA, March 2016.

[ATW15] Jafar Akhundov, Peter Tröger, and Matthias Werner. Considering concurrency in early spacecraft design studies. In *CS&P 2015 Proceedings*, pages 22–30, Rzeszow, Poland, 9 2015.

[ATW16] Jafar Akhundov, Peter Tröger, and Matthias Werner. Superposition principle in composable hybrid automata. In *CS&P 2016 Proceedings*, Rostock, Germany, September 2016.

[DCy10] Wei Dong and Zhao Chang yin. An Accuracy Analysis of the SGP4/SDP4 Model. *Chinese Astronomy and Astrophysics*, 34(1):69–76, January 2010.

[FLK$^+$08] S. Foeckersperger, K. Lattner, C. Kaiser, S. Eckert, W. Bärwald, S. Ritzmann, P. Mühlbauer, M. Turk, and P Willemsen. The Modular German Microsatellite TET-1 for Technology On-Orbit Verification,. In *Proceedings of the IAA Symposium on Small Satellite Systems and Services*, 2008.

[Hen00] Thomas A. Henzinger. The Theory of Hybrid Automata. In M.Kemal Inan and RobertP. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series*, pages 265–292. Springer Berlin Heidelberg, 2000.

[Kel97] John C. Kelly. *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems Volume II: A Practitioner's Companion*, volume 2. July 1997.

[LLL09] Jan Lunze and Franoise Lamnabhi-Lagarrigue, editors. *Handbook of hybrid systems control : theory, tools, applications.* Cambridge University Press, Cambridge, UK, New York, 2009.

[LS15] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach.* http://leeseshia.org, second edition edition, 2015.

[LSV03] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, August 2003.

[MMP92] Oded Maler, Zohar Manna, and Amir Pnueli. From Timed to Hybrid Systems. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 447–484, London, UK, UK, 1992. Springer-Verlag.

[SFS03] John P. W. Stark, Peter W. Fortescue, and Graham Swinerd. *Spacecraft systems engineering /.* 2003.

[SL02] Ying Shang and Michael D. Lemmon. The controlled composition analysis of hybrid automata, June 2002.

[STF$^+$13] Volker Schaus, Michael Tiede, Philipp M. Fischer, Daniel Lüdtke, and Andreas Gerndt. A Continuous Verification Process in Concurrent Engineering. In *AIAA Space Conference*, September 2013.

[Tie13] Michael Tiede. Evaluation of Search Algorithms for Formal Verification of Space Missions (german: Evaluierung von Suchalgorithmen zur formalen Verifikation von Raumfahrtmissionen). Bachelor's Thesis 83865, Ostfalia Hochschule fr angewandte Wissenschaften, 2013.

[Tru04] Walt Truszkowski. A Survey of Formal Methods for Intelligent Swarms. Technical Report 20050156631, NASA Goddard Space Flight Center; Greenbelt, MD, United States, December 2004.

[VEM08] S. Venigalla, B. Eames, and A. McInnes. A Domain Specific Design Tool for Spacecraft System Behavior. In *8th OOPSLA Workshop on Domain-Specific Modeling (DSM'08)*, Nashville, TN, USA, October 2008. Nashville, TN, USA: 8th OOPSLA Workshop on Domain-Specific Modeling (DSM'08).

[Wil15] Wiley. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities.* Wiley, 2015.

[WL99] J.R. Wertz and W.J. Larson. *Space Mission Analysis and Design.* Space Technology Library. Springer Netherlands, 1999.

Appendix: A DSL-to-LTIHA Translation Example

To demonstrate the application of semantic rules, this appendix provides a small example describing a satellite consisting only of one component - an experimental camera. The corresponding DSL snippet is provided in Figure 6.

,

```
 1 Describe Operational States
 2      State Experiment_Camera ;
 3 End
 4 Describe State Parameters
 5      Parameter experimentDataCamera ;
 6      Parameter massStorage ;
 7      Parameter batteryCharge ;
 8 End
 9 Describe Initial Parameter Values
10      Parameter experimentDataCamera Set Value 0.0 ;
11      Parameter massStorage Set Value 0.0 ;
12      Parameter batteryCharge Set Value 200.0 ;
13 End
14 Describe Starting State
15      Experiment_Camera
16 End
17 Describe Operational State Changes
18      State Experiment_Camera
19           Changes Changes Changes
20                Parameter experimentDataCamera by 4.0 ;
21                Parameter massStorage by −4.0;
22                Parameter batteryCharge by −0.2 ;
23      End
24 End
25 Describe Operational Constraints
26      Constraint  BatteryChargeConstraint for Parameter batteryCharge Above 40.0  ;
27      Constraint  massStorageConstraint for Parameter massStorage Above 0.0 ;
28      Constraint  DataCamera for Parameter experimentDataCamera Below 7200.0
29                  Accounts For State ExperimentCamera;
30
31 End
```

Fig. 6: Example: Camera Satellite Experiment

Applying rule "operational-states" to the line 2, an automaton is created with the *Active* and *Inactive* locations. Further, by rule "state-parameter" lines 5-7 transform into the three continuous variables - for the data gathered by the camera, used up data storage and the battery charge - which, for the sake of brevity, are called e, d and c, respectively. Rules "state-parameter-init" and "module-init" initialise the automaton's state to $(Active, (e = 0.0, d = 0.0, c = 200.0))$ (lines 10-12 and 15). The "change rule" translate lines 20-22 into the flow functions for the *Active* location. Last, the rules "bound" and "bound-for-state" add guards to the transitions of the LTI hybrid automaton from the DSL code lines 26-27 and 28-29, respectively. Since there is a single module in the system, there is no apparent difference between the last two rules. If there was, however, another component the guard constraint for the e-value would have only been added to this automaton, and none other. The resulting automaton is presented in Figure 7.

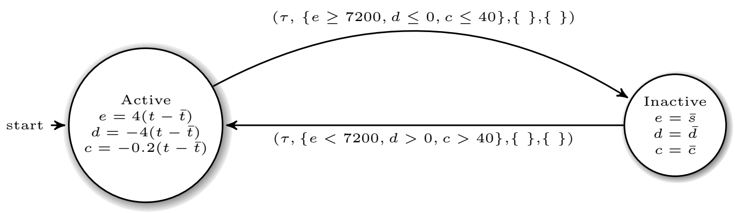

Fig. 7: An LTI-HA Describing Experimental Payload of a Satellite (Camera)