Considering Concurrency in Early Spacecraft Design Studies

Jafar Akhundov, Peter Tröger, and Matthias Werner

Operating Systems Group, TU Chemnitz, Germany
`{jafar.akhundov,peter.troeger,matthias.werner}@cs.tu-chemnitz.de`

Abstract. In real-world spacecraft systems, concurrent system activities must be constrained for energy efficiency and functional reasons. Such constraints must be considered in the early design phases, in order to avoid costly reiterations and modifications of the proposed system design in later phases. Although some initial attempts for using formal specifications exist in the domain, there is a lack of concurrency support in the utilized approaches. In this paper, we therefore first formalize an existing domain-specific language for specifying spacecraft designs and their constraints. Since this language does not support the modelling of concurrency issues, we extend it accordingly and map it to standard timed automata, based on a general system model. The new-style specifications can now be processed with existing and proven automata modelling tools, which enables faster and more reliable feasibility checks for early spacecraft system designs.

1 Introduction

The model-based engineering paradigm is widely accepted in the aerospace industry. One specific implementation is "Concurrent Engineering (CE)", an approach that has evolved into an important support mechanism for early space mission design decisions. It is supposed to clarify the baseline requirements and design properties for a future spacecraft, such as overall mass, power consumption and costs for development and operation. Studies have shown that up to 85% of the overall spacecraft project costs result from decisions made in this early phase [14].

During the CE project phase, multiple spacecraft design alternatives are simultaneously created, refined and analyzed by a group of engineers. Everybody in this group has an own range of responsibilities, such as the consideration of satellite trajectories, the realization of communication mechanisms, or the consideration of thermal aspects [11]. This makes the evaluation and ranking of architecture variations a complicated issue, since local design changes always have implications on global system scope. Those modifications may even impact the feasibility of the complete system design, if not correctly tracked and understood before the next project phase. This led to the development of feasibility checking approaches as part of CE in the past. They are conducted iteratively after each major design change, to provide feedback for engineers and the spacecraft customer.

The following paper discusses the consideration of concurrency demands and restrictions in the CE project phase based on a formal specification. Section 2 first introduces an abstract system model for spacecraft systems, in order to restrict the model entities to a necessary minimum. Section 3 then explains a domain-specific language being used in a German spacecraft project for the formulation of constraints and properties. In Section 4, we show how this DSL must be enhanced to consider concurrency issues properly. Section 5 finally explains how the resulting DSL can be translated to timed automata for better analysis support from existing tools.

2 Spacecraft System Model

A spacecraft system can be modeled as a set of internal and external state variables $\mathbf{S} = \mathbf{IS} \cup \mathbf{ES}$. The internal state si is defined by all state variables that can be directly manipulated through spacecraft's activities, e.g., the charging level of the battery or the available free memory. External state variables change by themselves and are not directly affected by spacecraft activities. Examples are temperature, the time, or the availability of sun light. They constitute the external state se.

The spacecraft activity can be abstractly defined based on the idea of interacting components. One relevant property here is the consideration of their concurrent activities and their relation to time. Let $\mathcal{T} = \{\tau_1, ..., \tau_n\}, n \in \mathbb{N}$ be a set of operational states representing activities of the spacecraft components, e.g. the active charging of the satellite battery from solar panels or the usage of some data transmission down-link. I.e., for each τ_i, there exists a transition function $f_i(se, si, \Delta t) = \Delta si$ that describes the change of the internal system's state if a component is active for the time Δt.

In real-world satellite systems, concurrent component activities must be constrained for energy efficiency and functional reasons. Not all spacecraft components can be active at the same time. Let $\mathcal{C} = \{c_1, ..., c_k\}, k \in \mathbb{N}$ represent a set of constraints for the state variables. Please note that the mission goals can be seen as special mission constraints.

Given the above definitions, a *schedule* or a *plan* is defined as a mapping of the component activity set $\mathcal{T}$ onto a set of time intervals:

$$\mathcal{P} : \mathcal{T} \mapsto \mathcal{I}, \qquad (1)$$

where $\mathcal{I} = \{(b, f) : b, f \in \mathbb{R}^+, b < f\}$ is a set of finite length time intervals with b, f begin the start- and end-times of the intervals, respectively. As a side condition, all C_i must be true at any time.

Based on the given system model, we are now discussing how constraints can be formulated and checked for a specific system model instance.

3 DSL-Based Constraint Description

Different methods can be used to check a spacecraft system model's compliance with the intended constraints and goals for the planned mission time. Schaus et al. described how an expert can formulate such requirements in a dedicated *domain specific language (DSL)* and use search algorithms to check the feasibility with respect to the given constraints [10]. The proposed syntax was used to create a description for a representative flying mission called satellite TET-1 [7, 16]. The DSL syntax turned out to be detailed enough to be fed into a search algorithm that looks for an execution trace fulfilling all conditions as mission plan [12].

Originally, the proposed concept was called *continuous verification*. We avoid the term 'verification' intentionally here, since at such an early design phase there is no real detailed system specification against which the system would be formally verified. Actually, creating unambiguous specifications is the purpose of the whole preliminary analysis. Instead of 'verification', the terms 'feasibility study/check' will be used.

In the published DSL syntax, the engineer can define different system components being exclusively active (called **states**) and system parameters (called **operational state variables**) which can be changed by the components' activities. The values of these parameters define the global system state. I.e., there exist implicit constraints

$$\forall i, j, i \neq j, \tau_i \Rightarrow \neg \tau_j \tag{2}$$

The mission goal is defined as a constraint on the operational state variables. Additional constraints beside the central mission goal can also be formulated. Furthermore, for every component there are corresponding rates of changes to the system parameters. The generated mission plan is a chain of activity periods derived from this description, such as *Battery Charge* $\rightarrow$ *Collect Data with Cameras* $\rightarrow$ *Send Data to Earth* $\rightarrow$ *Idle* $\rightarrow$ ....

Since satellite systems depend not only on the internal components, but also on the execution environment, there is a possibility to describe such environmental properties in the DSL, too. This is done by parameterizing external simulation models (SGP4) [10, 6] to compute satellite trajectories and generate periodic external events that activate components.

The original DSL from [10] is translated to the grammar description in Figure 1. It contains of the following parts:

- The *solver* construct determines the actual search algorithm which is used to find an appropriate solution.
- The *state parameters* together with the *initial parameter values* define system parameters which determine spacecraft system's state.
- The *operational states* represent the components of the satellite being active.
- The *starting state* defines the first component being active.
- The *operational state changes* define rates of change of system parameters by components' 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* being defined. 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.

Since in the original language there is no component concurrency being considered, only one component resp. operational state can be active at any point in time. Consequently, there is no explicit time notion in the language, with the only exception of

Solver

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

State Parameters

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

Initial State Parameter Values

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

Operational States

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

Starting State

⟨*component-init*⟩ ::= 'Describe Starting State' {⟨*identifier*⟩} 'End'

Operational State Changes

⟨*change*⟩ ::= 'Changes Parameter' ⟨*identifier*⟩ 'by' ⟨*real-number*⟩ ';'
⟨*multi-change*⟩ ::= 'Changes Parameter' ⟨*identifier*⟩ 'by' ⟨*real-number*⟩
    'and shifts effective Value to Parameter' ⟨*identifier*⟩ ';'
⟨*state-change*⟩ ::= 'State' ⟨*identifier*⟩ {change | multichange} 'End'
⟨*state-changes*⟩ ::= 'Describe Operational State Changes' {⟨*state-change*⟩} 'End'

Operational Constraints

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

Operational Goals

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

Simulation Modules

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

Fig. 1. Syntactical structure of an existing domain-specific language for spacecraft design descriptions, derived from [10]. Some non-terminals, such as identifiers, are omitted due to space restrictions.

simulation step which is the parameter fed to the external orbit simulator. This also determines the values for the changes of state parameters and the maximum number of steps before the mission deadline.

An important issue in the aforementioned DSL is that the operational state of the system, a vector of all state variables, is assumed to be directly bounded to the **state** of the currently active component. In other words: The system is specified with having only one active component at a time. This is an obvious over-constraining of the model, since a satellite as a system cannot be observed isolated from its environment. Concurrency consideration should be therefore not only a possible, but a desirable property. Several components of a satellite can be active simultaneously. One simple example is data being sent to earth while the battery is being charged. Furthermore, such a restrictive execution model can lead to the false negative check result, although the mission could still be possible to execute with a more "dense" plan[1].

Supporting concurrency in the DSL and the implied execution model would lead to increased expressiveness of the language and allow for more realistic derivable analysis results. Furthermore, with the concurrent execution model the generated plans are more 'dense' and thus the probability of a false negative result of the feasibility checking is lower.

4 Extended DSL Syntax

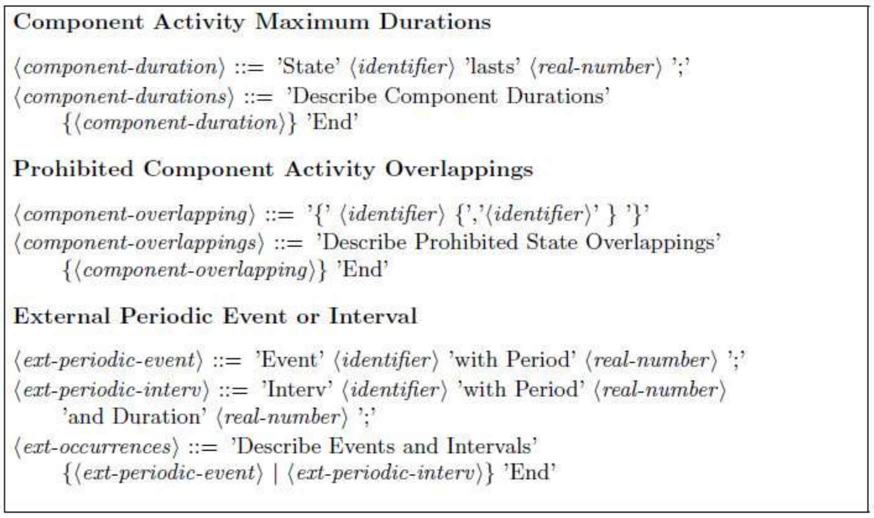
Given the formal system model from Section 2, we translate the semantics of the given DSL first into the set-theoretical constructs to understand the missing expressivity and determine the necessary enhancements:

- A *solver* is implementation-specific and defines a search method to trace the state space of a given problem. Hence, it is not part of the formal model.
- The *state parameters* are represented by the set $\mathbf{S}$ of state variables in the formal model. Their initial values are implementation-specific.
- The *operational states* are defined by the set $\mathcal{T}$ of the formal model. Starting State does not need to be determined by the engineer since a satellite can have several active components simultaneously and their enabling depends on the validity conditions.
- The *operational state changes* are also defined by the set $\mathcal{T}$ of the formal model.
- The *operational goals and constraints* are represented by the set $\mathcal{C}$ of the formal model.

There is no explicit notion of duration in the given DSL. The set of prohibited overlapped component activities is also missing in the explicit form. Implicitly, however, one could say that the set is given by the assumption in the execution model that all overlapped executions are forbidden. Furthermore, events which enable certain component activities are not explicitly present in the DSL. They are generated by the simulation modules and directly fed to the search algorithm.

[1] Here, the term "plan density" is loosely coined to describe a possibility to use overlapping time intervals to pack more activity into the mission time.

For supporting the missing concurrency semantics, the DSL is now enhanced with three syntactical constructs, as shown in Figure 2: Explicit durations per component activity, prohibited component activity overlapping, and events + durations.



Component Activity Maximum Durations

⟨*component-duration*⟩ ::= 'State' ⟨*identifier*⟩ 'lasts' ⟨*real-number*⟩ ';'
⟨*component-durations*⟩ ::= 'Describe Component Durations'
{⟨*component-duration*⟩} 'End'

Prohibited Component Activity Overlappings

⟨*component-overlapping*⟩ ::= '{' ⟨*identifier*⟩ {','⟨*identifier*⟩' } '}'
⟨*component-overlappings*⟩ ::= 'Describe Prohibited State Overlappings'
{⟨*component-overlapping*⟩} 'End'

External Periodic Event or Interval

⟨*ext-periodic-event*⟩ ::= 'Event' ⟨*identifier*⟩ 'with Period' ⟨*real-number*⟩ ';'
⟨*ext-periodic-interv*⟩ ::= 'Interv' ⟨*identifier*⟩ 'with Period' ⟨*real-number*⟩
'and Duration' ⟨*real-number*⟩ ';'
⟨*ext-occurrences*⟩ ::= 'Describe Events and Intervals'
{⟨*ext-periodic-event*⟩ | ⟨*ext-periodic-interv*⟩} 'End'

Fig. 2. DSL Enhancements for Concurrency Support

As a result of the reformulation and enhancement of the DSL grammar, the problem of feasibility analysis can now be reformulated as an off-line scheduling problem with several constraints [8]. Therefore, any formal modeling method supporting concurrency and having explicit notion of time would be a potential solution helper for the problem. Well-known examples are Timed CSP, Timed Petri Nets, Timed Automata, or Hybrid Automata.

We decided to map the enhanced DSL to Timed Automata [1, 2] for several reasons:

– The concepts of timed automata are easier accepted by engineers unrelated to computational logic and formal modeling.
– The structure of the DSL introduced in [10] translates one-to-one to a automata description.
– Timed automata have been proven to be more expressive than Timed Petri Nets [5] and just as expressive as Timed CSP models[9]

Although hybrid automata are more general and more expressive than timed automata [13], we sticked with pure time automata eventually, since they provide enough modeling capabilities for the given concurrency analysis problem.

5 Determining Feasibility

5.1 Timed Automata Representation

Mapping the enhanced DSL to timed automata constructs enables the system modeler to generate mission plans with appropriate existing tools, based on a transformed description with the DSL. Generally, the following definition holds:

Definition 1. *A timed automaton $\mathcal{A}$ is a tuple $(L, l_0, X, Inv, T, \Sigma)$ where:*

- *L is a finite set of states (locations)*
- *l_0 is the initial state*
- *X is a finite set of synchronously running real-valued clocks*
- *$T \subseteq L \times C(X) \times \Sigma \times 2^X \times L$ is a finite set of transitions $l \xrightarrow{g,a,r} l'$, where g is the guarding condition for the transition, a is the action and r is a set of clocks which are reset by the transition*
- *$Inv : L \rightarrow C_<(X)$ represents an invariant for every corresponding location*
- *Σ is an alphabet of all possible actions.*

Timed automata can model communication through binary synchronization, represented here by synchronization actions that are represented by according transitions. They can be either input actions (*sync?*) or output actions (*sync!*) that are performed in pair. [15, 3]

In the sequential execution model of the original DSL, only one operational state was allows to be active at a time. Hence, in such a representation, the whole set of components can be modeled as a single automaton with single inactive state (for idle mode) and several active states - one for each component. The values of system parameters will change with the given rates in some finite fixed step size which will be measured by a global clock.

Mapping the enhanced DSL description to the timed automata formalism is now straightforward. An operational state or component activity can be represented by a single automaton consisting of two locations - *Active* and *Inactive.* Validity conditions for the component to become active are represented by guards in the language of the TA. Clocks represent a maximum duration, if any is given. A component can remain active or inactive so long as the invariants are correspondingly fulfilled.

Mission goals and constraints can be embedded in two ways. Constraints are simply the guards on the transitions or location invariants which cannot be violated. To each goal there exists a dedicated automaton which also consists of two states. Once the goal is reached this automaton moves from one location to the the second and remains there representing a fulfilled goal. If all of the goals are satisfied before the global clock reaches some predefined value (mission time constraints), then the mission is feasible.

5.2 Application in Case Study

Based on the original DSL expressiveness, [12] researched implementations of various search methods, heuristics and optimizations of verification run times for different kinds of space missions. Depending on the complexity of the system and on the chosen technique, execution times varied from fractions of a second up to several minutes on an i7 running at 3 Ghz with 8 Gb of RAM. The main challenge of [12] has been to find an optimal value function and heuristics to minimize the effects of state space explosion. However, the system schedule did not make any statement beyond satisfiability of mission constraints and mission feasibility for the case where system components are activated in a sequential manner.

Using the proposed enhanced DSL and its mapping to an automata formalism, the TET-1 mission was again modelled by the authors in the UPPAAL tool [3, 4]. Specifically , it allowed us to perform a reachability analysis and to create a simulation trace as representation of a feasible mission plan with concurrent component activities.

We investigated a major improvement of the analysis run-time in comparison to the original behavior, which is shown in Table 1. Feasibility check times are now in the sub-second area. In comparison to the sequential methods used in [10], this introduces an improvement of almost two orders of magnitude of run-time with a more realistic, concurrent execution model.

Table 1. Performance of the feasibility check with UPPAAL 4.0.18. The average results are given for three different search methods available in the tool.

Search Order	**10 Gb Data**
Breadth first	out of memory error
Depth first	0.35 s
Randomized Depth first	0.34 s

6 Summary

In the earliest design phases of spacecraft systems, concurrent of component activities must already be considered to create optimized but still feasible system designs and mission plans. The article discussed a specific domain specific language being used for describing such system designs and their constraints. We showed how the expressiveness of the DSL can be extended to describe concurrent system activities. We furthermore presented a mapping to standard timed automata, which allows the re-use the broad existing body of tools for evaluation and analysis. Our enhanced description methodology has shown to be applicable for real-world space mission systems, extending the state of the art in early spacecraft design methodologies.

References

1. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming. pp. 322–335. Springer-Verlag, New York, NY, USA (1990), `http://dl.acm.org/citation.cfm?id=90397.90438`

2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
3. Behrmann, G., David, R., Larsen, K.G.: A tutorial on UPPAAL (2004)
4. Bengtsson, J., Larsson, F., Pettersson, P., Yi, W., Christensen, P., Jensen, J., Jensen, P., Larsen, K., Sorensen, T.: UPPAAL: a tool suite for validation and verification of real-time systems (1996)
5. Cassez, B.B.F.: Comparison of the expressiveness of timed automata and time petri nets. In: Proceedings of FORMATSŠ05. LNCS, vol. 3829, pp. 211–25. Springer (2005)
6. Dong, W., Chang yin, Z.: An Accuracy Analysis of the SGP4/SDP4 Model. Chinese Astronomy and Astrophysics 34(1), 69–76 (2010)
7. Foeckersperger, S., Lattner, K., Kaiser, C., Eckert, S., Bärwald, W., Ritzmann, S., Mühlbauer, P., Turk, M., Willemsen, P.: The modular german microsatellite TET-1 for technology on-orbit verification. In: Proceedings of the IAA Symposium on Small Satellite Systems and Services (2008)
8. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
9. Ouaknine, J., Worrell, J.: Timed csp = closed timed automata. In: Proceedings of EXPRESS'02. ENTCS, vol. 38 (2002)
10. Schaus, V., Tiede, M., Fischer, P.M., Lüdtke, D., Gerndt, A.: A continuous verification process in concurrent engineering. In: AIAA Space Conference (September 2013), `http://elib.dlr.de/84093/`
11. Stark, J.P.W., Fortescue, P.W., Swinerd, G.: Spacecraft systems engineering (2003)
12. Tiede, M.: 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 für angewandte Wissenschaften (2013)
13. Tripakis, S., Dang, T.: Modeling, verification and testing using timed and hybrid automata. Model-Based Design for Embedded Systems pp. 383–436 (2009)
14. Walden, D.D., Roedler, G.J. (eds.): INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities. Wiley (2015)
15. Wardana, A.: Development of Automatic Program Verification for Continuous Function Chart Based on Model Checking. Embedded Systems II Forschung, Kassel University Press (2009)
16. Wertz, J., Larson, W.: Space Mission Analysis and Design. Space Technology Library, Springer Netherlands (1999)