Using Timed Automata to Check Space Mission Feasibility in the Early Design Phases

Jafar Akhundov
Matthias Werner
Chemnitz University of Technology
Straße der Nationen 62
09111 Chemnitz, Germany
Jafar.Akhundov@informatik.tu-chemnitz.de
Matthias.Werner@informatik.tu-chemnitz.de

Volker Schaus
Andreas Gerndt
German Aerospace Center (DLR)
Lilienthalplatz 7
38108 Braunschweig, Germany
Volker.Schaus@dlr.de
Andreas.Gerndt@dlr.de

Abstract— **According to the model-based systems engineering paradigm, all engineers contribute to a single centralized data model of the system. The German Aerospace Center (DLR) develops a software tool Virtual Satellite which enables the engineers to store, exchange and alter their corresponding subsystem data on base of a distributed system model and thus contribute to the overall mission design during concurrent engineering (CE) sessions. Each engineer has their own scope of responsibilities, e.g. satellite trajectory, communication, or thermal analysis. Tracking implications of design changes on the whole system and feasibility aspects of the design is not trivial. Having an automated feasibility checking mechanism as a part of CE which would run iteratively after each design change provides a useful feedback mechanism for engineers and for the spacecraft client. For the purpose of mission feasibility checking a domain specific language (DSL) has been implemented using the Xtext Java framework.**

The extended parametric data model defined in the DSL serves as an executable representation of the spacecraft mission. The idea to use such an executable model to create a preliminary mission plan and hence confirm missions feasibility during conceptual study has already been introduced by Schaus et al. at the DLR. However, the vector of values of system variables was assumed to be equivalent with the currently active component, implying that component activities are mutually exclusive. This led to over-constraining of the execution model. Our work argues that concurrency considerations are critical from the earliest design phases. Since satellite is coupled with its environment and concurrency is an intrinsic property of the physical nature, considering concurrency allows for more realistic mission plans.

The contributions of this paper are the introduction of concurrency considerations at the early space mission design phases and the use of timed automata tool (UPPAAL) for the mission feasibility check during concurrent engineering sessions. As a result, with almost no overhead, the planned mission can be analyzed in a more realistic way. Furthermore, the runtimes of the feasibility check amount to 10-100 milliseconds or less, which is also a significant improvement with respect to the previous work. This allows for more precision and fine granular modeling, and is a promising basis for model refinements in the consecutive mission design phases.

TABLE OF CONTENTS

978-1-4673-7676-1/16/$31.00 ©2016 IEEE

1. INTRODUCTION

Concurrent engineering has evolved into a crucial supporting mechanism for space mission design and analysis during the early project phases. Its purpose is to clarify the baseline requirements and design properties such as mass, power consumption, resulting costs, etc. Practice has demonstrated that up to 85% of project costs are generated in early design phases [1].

During a CE session, different spacecraft design alternatives are simultaneously created, refined and analyzed by a team of engineers. Each of the latter has their own scope of responsibilities, e.g. satellite trajectory, communication unit or thermal analysis.[1], [2] Implications of design refinements on the system-level behavior as well as satisfiability of all constraints and goals are not trivial to track. Having an automated feasibility checking mechanism as a part of CE session would provide for these necessary functionalities for both engineers and clients [3].

There are different methods to check system compliance with mission constraints while satisfying goals of the mission. Schaus et al. [3] at the German Aerospace Center (DLR) have suggested a simple execution model for satellite's subsystem activities during CE sessions at the early design phase as a part of the Virtual Satellite software tool. To describe such a satellite, dedicated domain specific language (DSL) has been implemented which can be used to define system parameters, mission goals and constraints. Such a general systemic description is then translated into an execution model which is used to generate an execution trace leading to the mission goal. An important issue in this approach is that the operational state of the system, a vector of all state variables, is assumed to be identical to the state of the currently active component, i.e. the system can only be specified with having a single active component at a time. This is an obvious over-constraining of the model, since a satellite and its environment should be inseparable and, hence, concurrent to each other. Concurrency considerations should not only be a possible, but a desirable and even a necessary property in design of systems coupled with their physical environment.

The following work introduces support for concurrency in the DSL and the implied execution model. This allows for increased expressiveness of the language and more realistic analysis results. Furthermore, with the concurrent execution model the generated mission plans are 'denser' which is why the probability of a false negative result of the feasibility checking is lower. As a formalism for missions' feasibility check Timed Automata (TA) are proposed. A comprehensive mapping of the DSL enhanced with the timing constraints and concurrent constructs to the TA and, consequently, the UPPAAL[2] tool is provided. An interface between the Virtual Satellite and the UPPAAL tool is implemented which supports exporting of the system's description in the DSL into the UPPAAL-readable form (XML), and importing of the resulting traces back into the Virtual Satellite and visualization thereof. The results of this work are demonstrated by means of a use-case study of an TET-1 observation satellite.

The structure of this paper is as follows: Section II is a general problem statement which is the basis for finding an appropriate solution for the problem. The mappings of the mission model onto the timed automata (TA) and UPPAAL representations are described in Sections III and IV Building a use case study of the TET-1 satellite (also used in [3]) in the UPPAAL tool is the content of Section V. Sections VI gives provides the results of the run time measurements for the use case. Related work in this area is provided in Section VII, followed by the discussion of the result of this work and future work in Section VIII which concludes the paper.

2. Problem Statement

This section consists of three parts. The first one gives an informal overview of a general satellite model based on [4], [3], [5]. The second gives a short discussion on possible formalisms for a more precise problem definition. The third translates this model's semantics to the hybrid automata representation. The purpose of this approach is to achieve a clearer insight into the problem and find candidate solution methods.

Preliminary Definitions

Component Activities—A satellite system consists of several components, e.g. battery, communication module. Every component can change the global system's state consisting of continuous mission parameters through its activity. An example of a mission parameter could be the amount of data transmitted to Earth or the current battery charge level. Since an execution model is built based on this component-based specification of a spacecraft, only active components are of interest, i.e. components which can change mission parameters at run-time.

Every mission parameter has a certain domain which is usually constrained during the design phase of a space mission. Every system parameter and/or its time derivatives may have several constraints. The mission goal is represented by a vector of values of mission parameters representing system's state or a set of system's states. If the mission goal is achieved in the expected time, the mission is a success.

Component activities may overlap in time. Since specific overlappings of component activities might be undesirable (e.g. too high a load on a battery), it is possible to prohibit them by means of constraints.

Each component can be activated by an asynchronous event or periodically by a timer for a specified duration. Triggered components can depend on a single activation event or remain operating as long as some external interval occurrence is active. Corresponding examples are the activation of camera by an asynchronous event such as fire, or charge of the battery as long as the sun is available.

Given the above definitions, a *schedule* or a *plan* is defined as a mapping of the component activities onto the set of finite length time intervals. A *schedule (mission plan)* is defined to be *feasible* iff it satisfies both the mission constraints and the task overlapping constraints. If an optimal schedule is to be found, "goodness" of mission constraints and goals must be defined. Then an optimal schedule is a feasible schedule which, for every other feasible schedule, provides a greater goodness value.

Appropriate Formalism

As a result of the above definition, the problem of feasibility analysis is identical to the offline scheduling problem with several constraints and generalizations[6]. Hence, any formalism supporting concurrency and having explicit notion of time would be a potential formulation, analysis and solution tool, e.g. Timed CSP, Timed Petri Nets, Hybrid (HA) or Timed Automata.

There are two main reasons to chose hybrid automata over the others as a formalism for describing space system's execution model. The first is that the concepts of timed automata are easier to grasp for engineers unrelated to computational logic and formal modeling. The second is that the base model used in the implementation of the DSL used in the Virtual Satellite translates one-to-one to a HA/TA-description which will be demonstrated by the following text. Furthermore, TA - a subclass of hybrid automata [7] - have been proven to be more expressive than Timed Petri Nets [8] and just as expressive as Timed CSP [9].

Formalization of the Problem Domain

Definition 1: A Hybrid automaton is an extended labeled transition system represented by a 10-tuple [10] $(Loc, X, inv, init, flow, E, upd, jump, L, sync)$ where:

- Loc is a set of locations each representing a system mode,
- X is an ordered set of continuous variables over $\mathbb{R}$,
- $inv, init$ map the set of modes to predicates over the variables in X, where inv represents of invariants for each mode and $init(v)$ is the initial condition for $v, v \in Loc$,
- $flow$ is a mapping of Loc to predicates over $X \cup \dot{X}$, where $\dot{X}$ defines the first time derivative of all variables in X,
- $E \subseteq Loc \times Loc$ is a finite state of edges also called switches/transitions and represented by arrows between corresponding locations,
- upd maps E to the power set of X and variables in $upd(e)$ change their value when the transition e is taken (the rest of the values remain at their values before the transition),
- $jump$ is a mapping from E to transition conditions, where $jump(e)$ is a predicate over $X \cup upd'(e)$, $upd'(e)$ representing the values of variables from X after the transition is taken,
- L is a finite set of (synchronization) labels, so that automata

[2]Developed in a collaboration of the universities of AALborg and UPPsala

which have the same synchronization label in their transitions would take those respective transitions simultaneously, and

• $sync$ maps the set of switches E to $L \cup \{\tau_A\}$, where τ_A marks a transition of an automaton A which are not synchronized.

Restating semantics of the problem of feasibility analysis in terms of this formalism:

• Every component is represented by a separate hybrid automaton with two modes (Loc) of execution: active and inactive,
• X is the set of mission parameters, including global clock for the mission times and local clocks for durations,
• $inv, jump$ are defined by mission constraints and/or goals,
• $init, upd$ are the initial conditions and resets at the discrete, no-duration switches between execution modes,
• differential equations of $flow$ are given by the overlapping rates of changes of mission parameters depending on the active components,
• events and intervals are represented with separate automata, each with corresponding broadcasting $sync$ transitions, and
• $sync$ define the labels, so that synchronization occurs when a component changes mode of execution depending on the transition of some other automaton representing an event.

3. The Timed Automata Model

Since state space explosion is a big problem for discrete-continuous (hybrid) systems, and TA are a decidable subclass of HA, it suggests that TA should be sufficient to describe and solve the problem of achievability of mission goals with overlapping component activities without much overhead.

In contrast to the general HA defined in the previous section, timed automata only allow for a single type of continuous variables - the ones for which $\frac{dt}{dt} = 1$. These represent continuously running clocks with discrete resets at transitions defined by $init, upd$.

TA can communicate with each other through binary synchronization represented by synchronization action. Synchronization actions can be repersented by transitions which can be the input actions (*sync?*) or output actions (*sync!*) that are performed in pairs. [11], [12]

In the sequential execution model only one operational state can be active, 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.

Detailed representation of system model in timed automata is given in the following subsections.

Component Model

Each component's activity is represented as a timed automaton which can be basically in two states - Active and Inactive (Fig. 1). Inactive state is initial for all components. If some activation event enables the transition from the Inactive state to the Active, and there is no violation of system constraints for the activation of the component, such as the battery charge, the transition is taken. The component remains in the Active state either until one the following conditions holds:

• maximum duration time is reached
• some violation of system constraint occurs
• the event on which the activity of the components depends becomes inactive or unobservable.

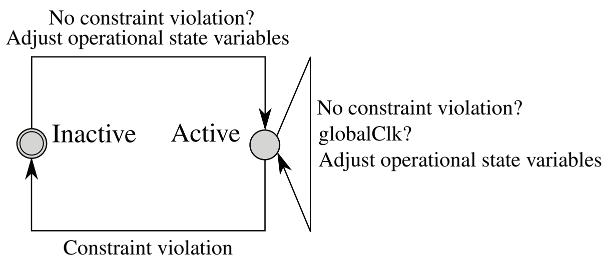

Figure 1. Example of a component with no activation trigger. It does not depend on any synchronization label to be activated - only to remain active. *globalClk?* is a synchronization label for synchronizing activity of the component with the step of the global clock.

While in the Active state, the system keeps changing its state according to the component's activities, i.e. draining battery charge or updating data. It does so in discrete steps with the global system clock. These state changes are only possible as long as no constraint is violated. Should this happen, component becomes inactive.

Event Model

An activity can be an occurrence (event) in time [13] or a span over some interval of time. Intervals are defined by having a non-zero duration, as opposed to events. In some cases duration of the intervals is not of much interest, just its starting point which would trigger the corresponding procedures. In the other cases, as e.g. in the case of battery charge, duration of an event is crucial.

Point Events—A periodic point event can be represented by a single location and a single transition which is only activated when the assigned timer hits a certain value (period). When this happens, a broadcast synchronization occurrs with all automata waiting for it, and the timer is reset. (Fig. 2)

Intervals—A periodic intervals requires two states with two clocks, one for the period and one for duration. Once the period of the event is reached, the event becomes active for the duration time. During this time, it constantly takes broadcast synchronization actions, enabling depending automata to become active for this period.(Fig. 3)

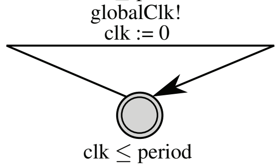

Figure 2. Periodic external point event. Every *period* amount of time a global synchronization label is activated so that other automata could fire their transitions.

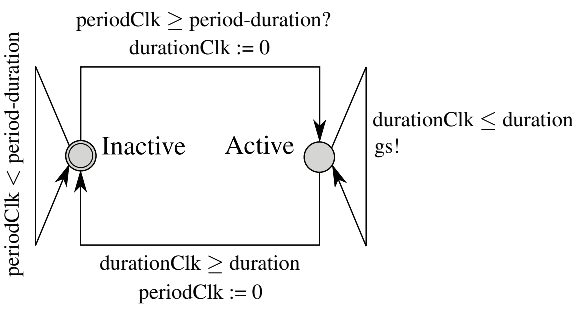

Figure 3. Periodic external interval event. Two clocks exist for this TA - a duration clock and the period clock. In every period, the automaton remains *duration* amount of time active.

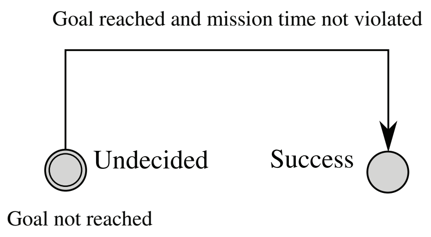

Figure 4. Reachability analysis automaton

Timing Model

Every step of the TA networks occurs synchronous with the global ”clock”. Therefore, the granularity of a single tick becomes the critical parameter. Set too high [3], the precision of the model and the delivered result may be of little use further. Set too low, and the verification performance may degrade. The global clock is represented by a single-location-automaton with one transition which is taken once every ’1’ time unit (the same as Fig. 2). The granularity and semantics of this time unit and, correspondingly, the precision of simulation/verification, is a decision made by an engineer assigned to the task of feasibility analysis. All of the system parameter rates should scale correspondingly with the tick granularity.

Reachability Analysis

There is a special automaton representing a transition in the case where the goal constraints for the mission have been met. (Fig. 4) This is done explicitly only for representation purposes, since the whole query can be defined in the verifier. Once the wanted state is achieved, the automaton moves into the Success state and remains there. The corresponding verifier-query can then be represented as a reachability query for the Success state in the predefined maximal mission time. The verification query is then simply[4]

```
E<> MissionValidation.Success
      and (time < MISSION_TIME)
      and (...) ,
```

which translates as ”check if there exists a path which reaches this state in predefined time”.

For multi-goal mission, an automaton may become more complex, depending on the semantics of the goals. In the case when all goals must be fulfilled, and once the goal is reached it remains so, the automaton remains unchanged. However, one could think of the situations where certain conditions must be fulfilled when the spacecraft system is exactly in some region of space. Since position and velocity are dynamic properties, satellite may move back and forth between partial fulfillment and undecided state but never quite reach the success state. Furthermore, in such a case when a number of such goals exist, i.e. for multi-stage missions, the reachability automaton can grow big. Worse still, at every step the verificator would consider this transition as well. Hence, to minimize the effects of these drawbacks, temporal formula will determine the exact goal of the mission in the query windows of the tool.

[3]High values for a single tick correspond to the longer durations.
[4]Example is given for the UPPAAL tool

4. The UPPAAL Model

There are several model checking tools using timed automata as an underlying formal model, e.g. CMC by Laroussinie and Larsen [14], KRONOS by Yovine [15], RED by Wang [16] and UPPAAL by Larsen et al. [12], [17], [18]. These have been compared according to numerous parameters such as syntax, semantics, state-space representation, reduction techniques, etc. in several works [11], [19], [20]. The UPPAAL is verification tool [18], [12], [17] that uses timed automata [21] to verify system properties with timely behavior (e.g. real-time systems). It is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark. [12], [17] In the aerospace domain, it has been used to verify HSTS, the planner and scheduler of the Remote Agent autonomous control system deployed in Deep Space 1 [22], [23]. Many other examples of UPPAAL applications can be found in [11, p.21]. It also has the capability to support deadlock freedom verification and extends the classical timed automata with the boolean and integer typed variables.

UPPAAL also enhances the classical notion of timed automata with broadcast and urgent synchronization actions, and urgent and commited locations. Broadcast synchronization action is an action the output of which can synchronize with an arbitrary number of input actions. In an urgent synchronization action time is not allowed to pass when this action is enabled. The same is valid for the urgent location, whereas a TA in commited location must directly continue to the next transition without time delay. [11]

Mapping from the Formal Model

As a proof of concept, Table 1 conclusively demonstrates the correspondence between the formal system model from Section 2 and the UPPAAL representation. Locations in the hybrid automata representation become the UPPAAL locations. Mission parameters X can be defined as simple (ranged) integer variables. Location invariants, transitions, constraints, resets and synchronization labels translate one-to-one. Initial values of mission parameters $init$ can be defined by initializing the variables of the UPPAAL model. There are no differential equations and continuous variables in UPPAAL with the exception of clocks. The result of a verification can be stored as a simulation trace, which holds an exact path to the desired state. There are different options here, depending on the used search algorithm and the desired path, e.g. ”some path”, ”shortest path”, etc.

Although priorities have not been the focus of this paper, there are UPPAAL constructs and expressions which allow to assign priorities to the processes. For example, $<$ in the

Table 1. Correspondence between the formal and UPPAAL models

HA Element	**Semantics**	**UPPAAL Element**
Loc	Set of executable satellite components	UPPAAL locations
X	Set of mission parameters	UPPAAL variables (integers or ranged integers)
inv	Set of system constraints for modes of operation (invariants)	Location invariants
$init$	Initial values of mission parameters	Initialization of variables
$flow$	Differential equations describing continuous dynamics of the system	UPPAAL clocks with $\frac{dt}{dt} = 1$
E	Discrete transitions between modes of operation	UPPAAL transitions
upd	Mission parameter resets	Transition resets of variables
$jump$	Mission constraints, goals	Transition conditions
$sync$	Synchronization labels for synchronous transitions	UPPAAL channels

system description *system A<B* assignes a higher priority to process B.

5. USE CASE: TET-1

As an example, we have used the mission described and analyzed in [3]. The TET-1 satellite has two experimental components which gather information, a 512 MB hard drive as information storage, a downlink module for sending the data back to Earth. Furthermore, it has a battery with the predefined capacity and charge rate which can only be charged at certain periodic time intervals.

The goals of the system, the mission time, as well as some global constraints have been defined as constants. For each module, there exists an array with the parameter changes that the module can make to all of the operational state variables (e.g. data amount) and an array representing the constraints for these changes. Furthermore, two functions have been defined for the automata transitions, a guard function which checks if the constraints are not violated, and an update function, which changes the corresponding variables. The code of the body of the update function is given below.

```
int i;
for (i = 0; i < numMissionParams; ++i)
        params[i] = params[i] + changes[i];
```

There is, however, a special case for the battery, since there exists a possibility that, although at the time of transition check there is enough battery charge for this particular activity to be executed, the global drain of the battery combined will violate the critical battery level by an unknown ammount (depending on the number of active components). Thus, some global knowledge should be provided, so that this situation never occurs. The first and the trivial solution would be to set the critical battery level high enough so that to avoid such an occurrence. However, this raises the question, how much is "high enough", precisely. The other solution would be keep track of battery drain rate, that is, a global knowledge of how many components are active and using the battery charge. This can be simply implemented by using a special variable which is increased before the activation of each component by the amount of battery change rate that is specified for it. Once the component becomes inactive, the battery change rate is correspondingly reduced. The body guard function which checks if the battery rate is safe is:

```
return (batteryRate + batteryChange
        <= battery - criticalBattery),
```

where batteryRate is the current battery draining rate, the battery change is the change that the component would make in each step to the battery (additionally to other components). The battery variable represents the current battery charge level, and criticalBattery the corresponding critical charge.

The BatteryCharge automaton represents the timed behavior of the satellite's battery (Fig. 5). There are three clocks which constitute the temporal progress of the automaton - the period, duration and the waiting clock. The module is activated once the period clock has hit a certain predefined value, that is, sun is available and the battery can be charged. However, the charge time is limited, and this limitation is represented by the duration clock which is reset once the automaton moves into the Active state and starts ticking. There is one special case, when the battery can be further charged, but is already full - for this case a special timer is set which counts until the rest of the "duration" time is out, after which the automaton moves into the Inactive state.

The ground station is modeled by a simple duration event represented in (Fig 3). If a ground station is available, it constantly sends an "enable" signal to the components which are waiting for it, in this case it is the downlink component. The downlink component waits for any of the ground stations to become visible, and once it is, it becomes active, if the corresponding system constraints are not violated, e.g. there should be data available to send and enough battery charge.

The idle state is redundant in this model, since the system state is represented by all automata being inactive and waiting for the constraints to be changed. This raises a question if a deadlock situation is possible. Once the battery is charged again, there will come a point when other components will be active again, e.g. data experiments, and they will, in turn, provide data to send back to earth.

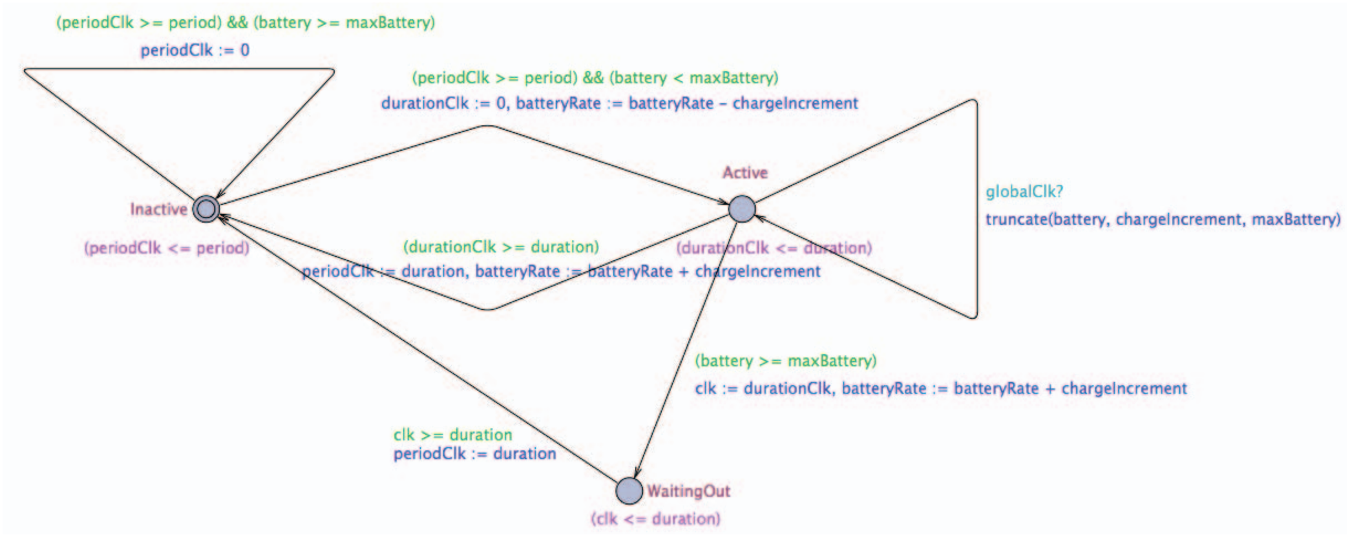
(periodClk >= period) && (battery >= maxBattery)
periodClk := 0
(periodClk >= period) && (battery < maxBattery)
durationClk := 0, batteryRate := batteryRate - chargeIncrement
Active
globalClk?
truncate(battery, chargeIncrement, maxBattery)
Inactive
(periodClk <= period)
(durationClk >= duration)
periodClk := duration, batteryRate := batteryRate + chargeIncrement
(durationClk <= duration)
(battery >= maxBattery)
clk := durationClk, batteryRate := batteryRate + chargeIncrement
clk >= duration
periodClk := duration
WaitingOut
(clk <= duration)

Figure 5. Battery automaton

6. Results

The tests have been carried out on a platform with Core i7 running at native 3 Ghz with 8 Gb DDR 2 RAM. The UPPAAL version used was 4.0.18 installed on a Windows 7 OS. There were several possible outcomes for the verification time, depending on the algorithm used and other options. The results are provided in table (2). The values have been normalized for 20 successful runs in each measurement. Occasionally[5], some feasibility check would hang and not provide any result in a reasonable amount of time. Since the model of the satellite was checked to be deadlock free, it is assumed that this occurs because of the deadlocks occurring during the simulation. For all the cases, state space reduction has been set to conservative, because no other option provided any improvement on the run times. State space representation was "Difference Bounded Matrices", although "compact data structure" delivers the same results. The diagnostic trace was set to "some", since we were interested only in the existence of the solution and not in finding the optimal one. The other options remained at their default values.

Table 2. Feasibility check run times

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

The granularity of the system tick was the same was in the initial paper [3], i.e. 1.44 minutes. All of the parameters have been scaled so that no overflow would occur with the integer values, since UPPAAL allows does not support floating numbers but only integers with the range of $[-32768..32767]$. It is possible to overcome this limitation by using ranged integers, however, this has resulted in an unacceptable growth of memory and time consumption. Furthermore, in [3] three different ground stations in an externally coupled simulation (SGP4) were used to compute exact positions of the satellite and the respective visibility of the ground station. In this paper no external simulation is coupled with the feasibility check process, and the availability of a ground station was modeled as a periodic interval event, whose duration was equal to the union of three intervals of three ground stations. Orbit perturbations influence are, as well, not considered, since the worst case values for the stations availability are taken for the feasibility check.

[5]Once in 15-25 runs

Three search algorithms are provided by the UPPAAL tool: breadth first, depth first and random depth first. Breadth first algorithm is exhaustive and always provides a solution, if there exists one. However, the performance of this algorithm was not acceptable in any reasonable simulation. Both depth first and randomized depth first performed equally well providing the simulation trace and the verification success within a quarter of a second.

All of the algorithms provided the results of at most several days for the needed mission time. This is a considerable reduction of this value in comparison to [3] which is explained by the fact that data is constantly being gathered and there is no prohibited constellation of states meaning that the overlappings are allowed in all possible combinations.

The comparison of run times with respect to [3] are provided in Figure (6). The actual implementation from [3] is provided by two green bars, VirSat_int for the initial implementation where ground stations were always available to the satellite and VirSat_ext where the external SGP4 simulation has been used to provide feasibility check process with precise orbits and ground stations availability times. UPPAAL (the orange bar) beats both times on average.

7. Related Work

Initially, the concept of mission feasibility analysis by means of simulation of an execution of the system model has been introduced in [3] and [24]. Whereas [3] introduces the conceptual idea of continuous verification[6] of space mission design, the [24] concentrates on the implementation of various search algorithms, heuristics and optimizations of

[6]In this text, the term 'verification' has been intentionally avoided, since at such an early design phase there is no system specification against which the system would be formally verified. What's more, creating specification is the purpose of the whole preliminary analysis. Instead of 'verification', 'feasibility study/check' is used.

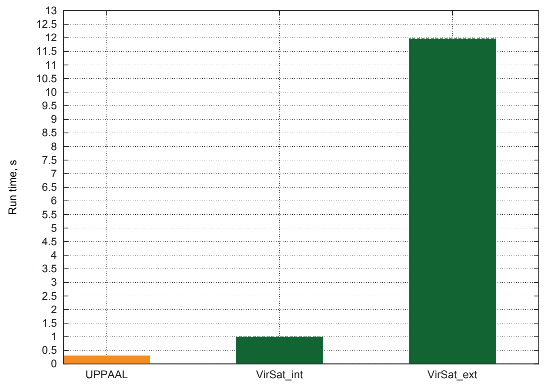

Figure 6. Comparison of run times of feasibility check

feasibility check run times for different kinds of 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 that work 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.

Creating mission plans is also possible with two actively researched and used planners from NASA: EUROPA [25] and SPIFe [26]. SPIFe is a tool supporting user-friendly drag-and-drop plan building with automatic constraint solvers checking if the user-created plan violates any of them. EUROPA, on the other hand, is a fully automatic tool for creating plans and schedules incorporating constraint solvers, but lacks direct GUI-support (but can still be integrated with other tools providing necessary interfaces). Whereas creating a full mission plan in SPIFe can become a tedious task, doing so in EUROPA assumes constraint solving skills and explicit heuristics to optimize searching methods.

There have been other attempts to analyze systems at the early stages, such as [27] or [28]. [27] was an attempt to use the communicating sequential processes (CSP) to verify system-level behavior for spacecraft at the earliest design phases. [28] has presented a systems engineering framework that places design requirements at the core of the design activities and uses genertic algorithms to trade off the considerable number of system design variables. None of these works have actually aimed at providing the engineers or the clients with the timed mission plan or feasibility statement.

8. Conclusion and Future Work

This paper has introduced the use of timed automata for feasibility study of a space mission at the earliest design phases. It has conclusively demonstrated, that adding as much as non-deterministic parallel activity of spacecraft components can lead to considerable computational efforts in comparison to [3] in some cases. However, in the most cases the feasibility checking of the new model has been up to two orders of magnitude faster than that of [3] and [24].

As a future work, several directions will be considered. The first concentrates on the further expansion of the mission verification at the conceptual analysis phase of mission design. In this respect, supplying satellite activities with priorities, both static and dynamic, may be one of the first steps. Also, using complex hybrid automata for even more realistic spacecraft execution model seems to be a challenging endeavor. Minimizing the state space and number of possible transitions for solving differential equations during overlapping component activities must be investigated. Furthermore, considering the popularization of swarms of smaller satellites, the problems of scaling and distributed state within the communication network is to be considered. Another important point is specification, integration and simulation of a realistic fault models and fault injections at all of the mission stages.

The focus of second research direction is on the complexity and composition. As an example, one could mention dividing the satellite activities into sub-states and their interactions. It should also be noted, that the current approach is greedy, so there is no ”intellectual” sophisticated planning behind the process. That is, if any action can be taken, it is taken.

Furthermore, in the recent work, global clock cycle is assumed to be existing which makes the whole system synchronous. While it is not critical at the Phase 0 of mission analysis where the whole system consists of components being either active or inactive, it may be a limitation at the later phases where each component is represented by a state machine and an asynchronous communication between the components is to be analyzed.

References

[1] Wiley, *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Wiley, 2015. [Online]. Available: https://books.google.de/books?id=8-cbBgAAQBAJ

[2] J. P. W. Stark, P. W. Fortescue, and G. Swinerd, “Spacecraft systems engineering /,” 2003.

[3] V. Schaus, M. Tiede, P. M. Fischer, D. Lüdtke, and A. Gerndt, “A continuous verification process in concurrent engineering,” in *AIAA Space Conference*, September 2013. [Online]. Available: http://elib.dlr.de/84093/

[4] S. Foeckersperger, K. Lattner, C. Kaiser, S. Eckert, W. B¨arwald, 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.

[5] J. Wertz and W. Larson, *Space Mission Analysis and Design*, ser. Space Technology Library. Springer Netherlands, 1999. [Online]. Available: https://books.google.de/books?id=veyGEAKFbiYC

[6] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[7] J. Lunze and F. Lamnabhi-Lagarrigue, Eds., *Handbook of hybrid systems control : theory, tools, applications*. Cambridge, UK, New York: Cambridge University Press, 2009. [Online]. Available: http://opac.inria.fr/record=b1135348

[8] B. B. F. Cassez, “Comparison of the expressiveness

of timed automata and time petri nets," in *In Proc. FORMATS05, vol. 3829 of LNCS*. Springer, 2005, pp. 211–225.

[9] J. Ouaknine and J. Worrell, "Timed csp = closed timed automata," in *IN PROCEEDINGS OF EXPRESS 02, VOLUME 38(2) OF ENTCS*, 2002.

[10] O. Mller and T. Stauner, "Modelling and verification using linear hybrid automata – a case study," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 6, no. 1, pp. 71–89, 2000. [Online]. Available: http://dx.doi.org/10.1076/1387-3954(200003)6:1;1-Q;FT071

[11] A. Wardana, *Development of Automatic Program Verification for Continuous Function Chart Based on Model Checking*, ser. Embedded Systems II Forschung. Kassel University Press, 2009.

[12] G. Behrmann, R. David, and K. G. Larsen, "A tutorial on uppaal," pp. 200–236, 2004.

[13] H. Kopetz, M. Paulitsch, C. Jones, M.-O. Killijian, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, and R. Stroud, "Revised version of dsos conceptual model," Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, Project Deliverable for DSoS (Dependable Systems of Systems), Research Report 35/2001, 2001.

[14] F. Laroussinie and K. Larsen, "Cmc: A tool for compositional model-checking of real-time systems," in *Formal Description Techniques and Protocol Specification, Testing and Verification*, ser. IFIP The International Federation for Information Processing, S. Budkowski, A. Cavalli, and E. Najm, Eds. Springer US, 1998, vol. 6, pp. 439–456.

[15] S. Yovine, "Kronos: a verification tool for real-time systems," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 123–133, 1997.

[16] F. Wang, "Region encoding diagram for fully symbolic verification of real-time systems," in *Computer Software and Applications Conference, 2000. COMPSAC 2000. The 24th Annual International*, 2000, pp. 509–515.

[17] J. Bengtsson, F. Larsson, P. Pettersson, W. Yi, P. Christensen, J. Jensen, P. Jensen, K. Larsen, and T. Sorensen, "UPPAAL: a tool suite for validation and verification of real-time systems," 1996.

[18] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[19] B. Bérard and L. Sierra, "Comparing Verification with HyTech, Kronos and Uppaal on the Railroad Crossing Example," Laboratoire Spécification et Vérification, ENS Cachan, France, Research Report LSV-00-2, Jan. 2000, lSV.

[20] P. Matouek, "Tools for parametric verification. a comparison on a case study," vol. 10, no. 10, pp. 1469–1494, oct 2004.

[21] R. Alur and D. L. Dill, "Automata for modeling real-time systems," in *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*. New York, NY, USA: Springer-Verlag New York, Inc., 1990, pp. 322–335. [Online]. Available: http://dl.acm.org/citation.cfm?id=90397.90438

[22] L. Khatib, N. Muscettola, and K. Havelund, "Verification of plan models using uppaal," in *Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers*, ser. FAABS '00. London, UK, UK: Springer-Verlag, 2001, pp. 114–122. [Online]. Available: http://dl.acm.org/citation.cfm?id=646168.681011

[23] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams, "Remote agent: To boldly go where no ai system has gone before," *Artif. Intell.*, vol. 103, no. 1-2, pp. 5–47, Aug. 1998. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(98)00068-X

[24] M. Tiede, "Evaluation of search algorithms for formal verification of space missions (german: Evaluierung von suchalgorithmen zur formalen verifikation von raumfahrtmissionen)," Ostfalia Hochschule fr angewandte Wissenschaften, Bachelor's Thesis 83865, 2013.

[25] J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, J. Ong, E. Remolina, T. Smith *et al.*, "Europa: A platform for ai planning, scheduling, constraint programming, and optimization," *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)–The 4th International Competition on Knowledge Engineering for Planning and Scheduling*, 2012.

[26] A. Aghevli, A. Bencomo, and M. Mccurdy, "Scheduling and planning interface for exploration (spife)."

[27] A. McInnes and U. S. University, *A Formal Approach to Specifying and Verifying Spacecraft Behavior*. Utah State University, 2007. [Online]. Available: http://books.google.de/books?id=UJ1RP38kIzEC

[28] R. A. Hassan, "Genetic algorithm approaches for conceptual design of spacecraft systems including multi-objective optimization and design under uncertainty," Ph.D. dissertation, West Lafayette, IN, USA, 2004, aAI3150771.

BIOGRAPHY

Jafar Akhundov *received his B.S. and M.S. degrees in Computer Science from Chemnitz University of Technology in Germany. He is currently a research scientist at the operating systems group at the same university. His current research activities include space mission specification, verification of non-functional system properties in aerospace design and modeling of temporal properties in complex systems.*

***Matthias Werner** received his diploma degree in Electrical Engineering in 1993 and a Ph.D. in Computer Science from the Humboldt University in Berlin in 1999. He is currently a professor of Operating Systems at the Chemnitz University of Technology. His research interests include non-functional system properties, cyber-physical systems, real-time systems, mobility, distributed and fault-tolerant systems.*

***Volker Schaus** holds a Master Degree in Aerospace Engineering. He is currently employed at the DLR Institute for Simulation and Software Technology and is responsible for the Virtual Satellite project. Before joining the DLR in 2010 he worked in VIP outfitting projects of wide body aircrafts as Mechanical Engineer. His research interests are digital product design and evaluation, Model-based Systems Engineering and Concurrent Engineering.*

***Andreas Gerndt** is the head of the department Software for Space Systems and Interactive Visualization at the German Aerospace Center (DLR). He received his degree in computer science from Technical University, Darmstadt, Germany in 1993. In the position of a research scientist, he also worked at the Fraunhofer Institute for Computer Graphics (IGD) in Germany. Thereafter, he was a software engineer for different companies with focus on Software Engineering and Computer Graphics. In 1999 he continued his studies in Virtual Reality and Scientific Visualization at RWTH Aachen University, Germany, where he received his doctoral degree in computer science. After two years of interdisciplinary research activities as a postdoctoral fellow at the University of Louisiana, Lafayette, USA, he returned to Germany in 2008.*