Formal Verification in Early Mission Planning

Philipp M. Fischer[(1)], *Daniel Lüdtke*[(1)], *Volker Schaus*[(1)], *Olaf Maibaum*[(1)], *Andreas Gerndt*[(1)],

German Aerospace Center (DLR)[(1)]
Software for Space Systems and Interactive Visualization
Lilienthalplatz 7
38108 Braunschweig
Germany
E-mail:Philipp.Fischer@dlr.de
E-mail:Daniel.Luedtke@dlr.de
E-mail:Volker.Schaus@dlr.de
E-mail:Olaf.Maibaum@dlr.de
E-mail:Andreas.Gerndt@ddlr.de

ABSTRACT

Spacecraft are complex systems. Changing one of its design parameter can have implications on the overall design and might become a crucial factor to mission success. In the early phases of spacecraft design, parameters as well as the mission goals are likely to change. These changes have to be applied carefully and need to be analyzed in respect to the whole system and the intended mission. The software Virtual Satellite supports this analysis by using an abstract model where the engineers can enter design data of their components. It allows describing operational phases of the spacecraft by defining modes such as Recharge or Science. These operational modes can be referenced by parameters to define individual values for them. Together with their respective mode durations, it can be determined for example how much energy is consumed in a specific mode or how much is produced. But this does not consider the influence of the parameter with respect to the overall mission goals. For example having a mission life time of 20 years and a spacecraft which spends too much of that time to maintain its power state, it remains unclear if the remaining time is long enough to gather enough scientific data as demanded by the mission requirements.

This paper shows an approach to such problems based on formal verification. The data of the early phase model is used to create a state model of the spacecraft. Both, the model and the formalized requirements will be given to a model checker that automatically verifies on formal basis that the spacecraft complies with its specification. This method enables engineers to quickly check the design with respect to the mission requirements once they applied changes to it or to the requirements.

INTRODUCTION

A spacecraft is a complex system. Designing it requires the expertise and knowledge of various engineers from different domains. Most of their work and in particular the spacecraft components they are responsible for depend on each other. Changing just one parameter may have severe impact to the overall design. For example some new scientific equipment requires more power during flight. This requires a stronger power sub-system including bigger solar panels and heavier batteries. As a consequence the mass of the spacecraft rises making it too heavy for the currently selected launcher. In the early design phase, the engineers need to be aware of such dependencies and their impact to the design. Therefore the German Aerospace Center (DLR) develops the software Virtual Satellite [1] which incorporates concepts of model based systems engineering. An integrated and consistent data model covers all aspects from the design decomposition to the specification of parameters and calculations. Consequently changes to the modeled design can be traced and understood by the design team on-the-fly.

During design time it is not just the components which are changed and adjusted. The requirements may change as well. E.g. in case that the intended mission-lifetime is not long enough to gather enough scientific data, it is possible that the design team decides to extend it. Accordingly, they need to be aware of the implications these requirement changes bring along. An extended mission lifetime might raise new demands to a propulsion subsystem for example. It is inevitable for the engineers to firstly being able to decide for changes by understanding their design and its dependencies. Secondly it is important that they can easily check the feasibility of the design right after such changes.

One way to verify that the model complies with its specification and requirements is simulation [2] [3]. But during the early phase studies a lot of detail is missing that is necessary to properly parameterize the required models. Nevertheless, the early design phase allows for certain abstractions that lead to a different approach. Looking for example to the semi-conductor industry, formal methods are well established in order to check the design models of new integrated circuits against their specification. One of such methods is model checking, which extensively explores the state space of the models and check whether they comply with formalized specifications and requirements. In case conflicts are found the model checker generates a counter example highlighting the contradiction to the specification. This method helps the industry to counter the problem of rapidly growing complexity and the rising demands of maturity and quality [4].

This paper will show an approach of applying the formal method of model checking into the early spacecraft design domain. Starting with the following section, this paper will introduce the model checking methodology. In the next section it will show how the design data model works and how it can be transferred to a representation that is suitable as input to off-the-shelf model checkers. Additionally it will present how mission requirements are represented by temporal logic. Before giving a conclusion the paper will show the model checking approach by an example discussing the advantages and disadvantages.

MODEL CHECKING IN SYSTEMS ENGINEERING AND OTHER DOMAINS

The term *Model Checking* refers to automated techniques to verify a model against its specification on a formal basis. The model checker analyses the model and tries to find counter examples that violates the specification. It allows detecting design flaws earlier and right from the start before any code or further artifacts are generated from that model. This technique has gained great acceptance over the past years in quite different domains such as the semiconductor industry [5] [6]. Some work in that area has proven that it is possible to verify large chip design inheriting a complexity of up to 10^{20} states [7]. A selection of the dominant model checkers of the recent years are NuSMV, Spin, and Prism. NuSMV is open source and based on the early work of SMV done at the Carnegie Mellon University [6]. It incorporates different verification techniques based on binary decision diagrams (BDD) and propositional satisfiability (SAT). Additionally, it supports different dialects of temporal logic to formalize specifications. It is commonly applied to verify state machines being represented as Kripke Structures [8] [9]. Additionally NuSMV supports bounded model checking leading efficiently to minimum length counter examples [10] [4]. The model checker Spin [11] focuses on the validation of process interactions. It uses PROMELA [12] as an input language for the model whereas the specifications are defined using linear temporal logic. It has been applied to a variety of applications including bus protocols, address protocols and fragments of TCP/IP. Prism has been developed for the analysis of probabilistic systems supporting two different types of Markov models. Instead of evaluating a formula whether it performs a transition in the state machine, like it is implemented in other model checkers, it uses probabilities to decide [13].

Besides the applications in various domains, some successful applications have also been shown in the space domain. E.g. it has been successfully introduced at the Jet Propulsion Laboratories to verify the correctness of command sequences using the model checker UUPAL [14] [15]. Aim of this verification was to make sure that no command sequence can harm or destruct the satellite. As a further example, the Java pathfinder project tries to transform Java code into PROMELA which is used as input language for the model checker Spin. Different to earlier approaches where Lisp code was translated to PROMELA, this project aims to create a framework that lets the software engineers check their Java code in their development process [16]. The COMPASS project evaluated on the usage of formal modeling and analysis techniques for software of modern spacecraft. Some verification tools such as NuSMV have been applied and extended. The results of the project were evaluated on a mission of the European Space Agency (ESA). Various models were created and verified in parallel to the ongoing software development in the actual project. Issues that were detected by the verification methods were fed back into the actual development of the spacecraft [17].

Another application for model checking is *Planning*. Planning focuses on the optimization and solving of problems such as the travel-salesman and puzzle games [18]. Some work has been conducted to evaluate the efficiency and effectiveness of the model checkers Spin and SMV compared to traditional planning tools. The work showed that under certain restrictions the model checkers can replace and compete with standard planning tools [19]. Different than model checkers, planning tools have already been applied to space domain problems like data downlink scenarios [20].

MODEL CHECKING OF EARLY SPACECRAFT MODELS

In order to use model checkers to verify for the aforementioned changes to the spacecraft design or to the mission requirements, the early design data needs to be consistently transformed. The design software Virtual Satellite which is used as a modeling tool throughout concurrent engineering sessions offers this design data. Fig. 1 describes the internal data model that is used to store the design data as well as applied changes. The whole spacecraft design is stored in the *StudyRepository* which contains all relevant information of the spacecraft. The spacecraft itself is decomposed in a tree-like hierarchy using *SystemComponents*. The tree-like hierarchy is achieved by letting each *SystemComponent* reference to other *SystemComponents* declaring them as their children or subcomponents. Each *SystemComponent* contains *Parameters* which are used to represent certain aspects of a spacecraft such as a component's mass or its power demand. Additionally, the data model allows specifying operational *SystemModes* of the spacecraft. It refers to the spacecraft's operational state such as being in a special science mode, downlink mode or similar. Additionally, each mode has a duration representing the amount of time of being active. Depending on the active mode, characteristics of the spacecraft may change. This is reflected by the *Values*. Each *Parameter* has at least one *Value* which refers to a *SystemMode*. Having individual *Values* with the references to different *SystemModes* it is possible to model characteristics such as power demands depending on the operational mode of the spacecraft [1] [21].

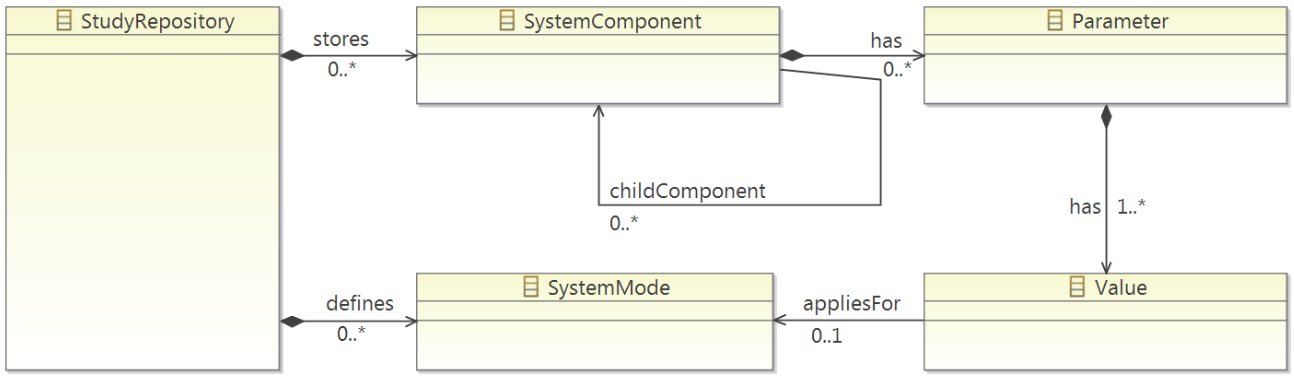

Fig. 1. The basic structure of the internal data model used in Virtual Satellite

The software Virtual Satellite offers functionality to create sequences of operational modes already. These so called schedules can be used to analyze the change of parameters over time. Therefore, certain parameters of interest can be assigned to that schedule and being initialized by a given value to represent characteristics such as a charged power system. To analyze these parameters, their mode dependent values are integrated by the time the corresponding mode is active.

Comparing these schedules to Kripke structures, which are a special representation of state machines, it becomes apparent that they suit well for model checkers and NuSMV in particular. To allow NuSMV to check the early design model it needs to be transformed appropriately. NuSMV declares variables which hold the accumulated values of the parameters of interest as they have been declared in the schedule. The operational modes are transformed to represent the states of that model. Depending on the state NuSMV uses the parameters to affect the accumulated values as well as the accumulation of time which is defined in the *SystemModes* of the data model. Fig. 2 shows a NuSMV compatible model of a simple spacecraft. The section *VAR* defines the variables of the checkable mode. The definition contains the name of the variable as well as its range of values. The variable *state* represents the operational modes of the spacecraft. The ranges for time and charge are arbitrary and are chosen as days and percent in this example. The *ASSIGN* section defines the assignments to the variables. Here they are first initialized with a given value. The first *next* block defines the possible changes of the spacecraft's state which can be either one of the enclosed enumeration. The second *next* block gives an example of how the time parameter is accumulated depending on the current state of the spacecraft. Further *next* blocks for various parameters can be added if needed. The values within the *case* block are chosen depending on the truth of the attached logical expressions. The special expression *TRUE* denotes the default case for the *case* block.

With the checkable model using the information from the early phase spacecraft design model, the requirements to that model still need to be specified. NuSMV uses temporal logic to specify them. The section *LTLSPEC* (see Fig. 2) describes our temporal logic for checking the model. The letter G is used in temporal logic to express that a certain property has to always hold. The arrow expresses causality similar to an if-then statement [5]. The complete expression can be translated to: "If the spacecraft is always safe then it is never able to reach the mission goal safely." The expression might be confusing since it is obvious that if the spacecraft is safely designed it can fulfill its goals. But model checkers try to find a counterexample to the specification. Accordingly, the model checker will give a counterexample where the spacecraft is always safe and reaches the goal safely. This temporal formula leads to an easy and separated description of safety constraints and mission goals. Within the example two safety constraints are described by logical expressions stating that the battery and the storage device should stay within reasonable operational conditions. This simple description by using logical expressions provides easier access for engineers than temporal logic.

```
MODULE main
VAR
        state: {SCIENCE1, SCIENCE2, CHARGE, DOWNLINK};
        time :   0 .. 1000;
        charge : 0 .. 100;
        ...
ASSIGN
        init(data)    := 0;
        ...
        init(state)   := SCIENCE1;

        next(state) := case
                TRUE : {SCIENCE1, SCIENCE2, CHARGE, DOWNLINK};
        esac;

        next(time) := case
                state = SCIENCE1 : time + 2;
                state = SCIENCE2 : time + 2;
                state = CHARGE   : time + 4;
                state = DOWNLINK : time + 1;
                TRUE             : time;
        esac;

        next(charge) := case
                ...
        esac;

        ...
DEFINE
        safe := charge > 5 & charge < 100 & data >= 0 & data <= 100;
        goal := time > 50;
LTLSPEC
        G ((safe) -> G !(goal & safe))
```

Fig. 2. A sketch of a checkable model that has been created out of the information of an early phase design model

Implementation in the Design Process and Performance Considerations

To support the design sessions, the Software Virtual Satellite together with the model checker should enable engineers to quickly evaluate design or requirement changes. Within Virtual Satellite each engineer should be allowed to start the model checking tool. The results of the model checking run can influence the decisions of the engineers and accordingly influence their design.

A reasonable computational performance is necessary in order to allow such quick iterations and design verifications. The problem size that needs to be explored depends on the model, its amount of states and variables, and finally the amount of mode transitions. NuSMV utilizes various techniques to reduce the problem size; still in this case it leads to a theoretical problem size of M^S where S is the amount of scheduled operational modes and M the amount of operational modes within the spacecraft. As a consequence an increase of the requirement for the maximum mission life time leads to an exponentially increased demand for computational power and time to run the model checker. Fig. 3 shows some calculation results that were performed on a standard off-the-shelf notebook. A calculation time of some minutes sounds reasonable and applicable during design sessions but further increases in mission time will increase the computational demands dramatically. In order to keep the evaluations as quick as possible the mission time and its transitional change needs special attention. Depending on the demands of the mission, it is either possible to just verify a certain time frame of the overall mission or to decrease the level of time-detail. E.g. this can be achieved by modeling the mission time with the detail of orbit revolutions instead of hours or minutes.

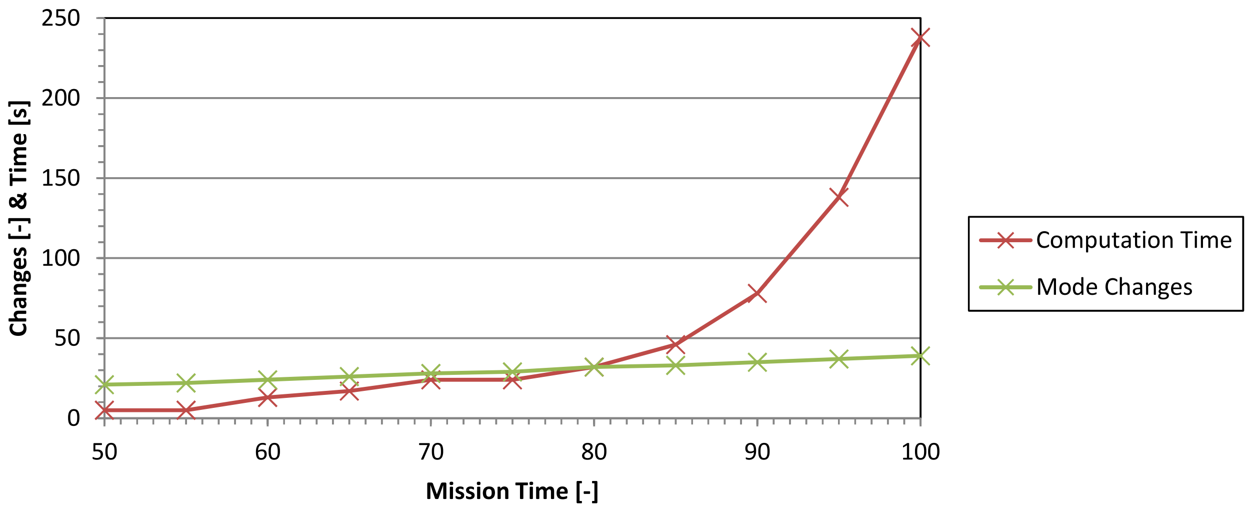

Fig. 3. Influence and exponential increase of the computational time versus the mission time as described in the mission goal

In conjunction with the computational power it is important to keep in mind that this type of verification is not complete because it is based on bounded model checking. Bounded model checking iteratively increases the amount of transitions from state to state until it reaches a counterexample [4]. In certain cases it could be possible that the whole problem space needs to be explored before finding a counterexample or not. Therefore, NuSMV restricts the maximum length of the counterexample. This boundary can be parameterized and needs to be set according to the problem. In the given example it needs to allow for enough changes to satisfy the mission time goal criteria. In case that the parameter is too small, the model checker will evaluate all possible state transitions without finding a counterexample. First of all this will consume a lot of computational time and second it might lead to a possible false assumption about the feasibility of the mission.

The third concern that has to be highlighted is the abstraction of the spacecraft. In order to allow for a useful application of model checkers it is not only necessary to abstract the mission time. The *DownLink* mode handles the amount of data stored on the spacecraft but not the actual data transmitted to the ground. It would be possible to easily model the amount of data that is transferred to the ground, but it becomes difficult to model correct contact times. Within our simple model the transition into the downlink mode is applicable at any time whereas in reality a contact between the satellite and a ground station is required. Since it is not possible to implement a detailed orbit model and ground contact model, it is still possible to abstract the ground contact on a stochastic basis. Fig. 4 shows a simple example how it can be implemented as a further mission constraint. Assuming that time reflects full orbit revolutions, the variable *orbit* calculates the modulo of the time to the basis of ten. The following *downlink_constraint* which is part of the safety expression restricts the transition into the *DownLink* mode to the first four out of ten orbits.

```
DEFINE
        orbit := time mod 10;
        downlink_constraint := (state = DOWNLINK) & orbit < 4;

        safe := downlink_constraint & ...
```

Fig. 4. Definition of a restriction to the transition to the downlink mode

CONCLUSION

This paper presents that it is possible to use the early design data of a spacecraft and to transform it into a state model. This state model can be used to understand and in particular verify that the designed spacecraft fits to the mission requirements. But for the purpose of verification, the model needs to be accompanied by a specification and a model checking tool. The model checking tool is the central element to compare the model to its specification. The specification which is normally provided as temporal logic can be abstracted to two types of logical expressions. The first one describes the mission goal like the expected mission duration, whereas the second one describes operational constraints such as data storage capacity. This separation provides easier access for the engineers than using temporal logic itself. In order to apply this method to the early design process it needs to be possible to evaluate each change to the spacecraft’s parameters or to the mission requirements. Therefore the verification needs to be executed easily and quickly. As a consequence different levels of abstraction are needed and have to be balanced between detail and computational power demands. In particular, verifying that the satellite design is feasible over the entire mission duration becomes impossible on a reasonable level of abstraction. In this case it is necessary to consider the important parts of the mission.

All together this work shows that it is possible to use formal model checking techniques in the early design phase of a spacecraft. This enables the engineers to quickly verify that a spacecraft meets the requirements of the intended mission obeying lack of detail due to certain abstraction.

REFERENCES

[1] V. Schaus, P. M. Fischer, D. Lüdtke, A. Braukhane, O. Romberg and A. Gerndt, "Concurrent Engineering Software Development at German Aerospace Center - Status and Outlook," in *Proceedings of the 4th International Workshop on System & Concurrent Engineering for Space Applications (SECESA 2010)*, Lausanne/Switzerland, 13.- 15. October 2010.

[2] O. Kalden, P. Fritzen and S. Kranz, "SimVis - A Concurrent Engineering Tool for Rapid Simulation Development," in *Proceedings of Recent Advances in Space Technologies (RAST'07)*, Instanbul/Turkey, June 2007.

[3] M. Berlin, A. Berres, K. Bries and H. Kayal, "SMASS - A Lightweight Satellite Simulation Framework for Concurrent Engineering in Education," in *3rd International Workshop on System & Concurrent Engineering for Space Applications (SECESA 2008)*, Rome/Italy, 2008.

[4] E. Clarke, A. Biere, R. Raimi and Y. Zhu, "Bounded Model Checking Using Satisfiability Solving," in *Formal Methods in System Design*, Netherlands, 2001.

[5] M. Y. Vardi, "Branching vs. Linear Time: Final Showdowm," in *Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, 2001.

[6] K. L. McMillan, Symbolic Model Checking: An Approach to the State Explosion Problem, Carnegie Mellon University: Doctoral Thesis, May 1992.

[7] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill and L. Hwang, "Symbolic Model Checking: 10^20 States and Beyond," in *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90)*, Philadelphia/USA, June 1990.

[8] A. Cimatti, E. M. Clarke, F. Giunchigalia and M. Roveri, "NuSMV: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, 2000.

[9] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Proceeding of International Conference on Computer-Aided Verification (CAV 2002),* Copenhagen/Denmark, June 2002.

[10] A. Biere, A. Cimatti, E. Clarke and Y. Zhu, "Symbolic Model Checking without BDDs," in *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99*, Amsterdam/Netherlands, 1999.

[11] G. J. Holzmann, "The Model Checker SPIN," *IEEE Transactions on Software Engineering,* vol. 23, no. 5, May 1997.

[12] "Spin - Formal Verification," 30 Juli 2012. [Online]. Available: http://spinroot.com/spin/whatispin.html. [Accessed 31 August 2012].

[13] M. Kwiatkowska, G. Norman and D. Parker, "PRISM: Probabilistic Symbolic Model Checker," *Proceedings of TOOLS 2002,* April 2002.

[14] C. A. Stone, A. Carter, H. L. Justice, R. M. Keller and Y.-W. Tung, "Improved Modeling and Validation of Command Sequences Using a Checkable Sequence Language," in *Proceedings of the 2012 IEEE Aerospace Conference,* Big Sky/USA, 2012.

[15] "UPPAAL Home," Uppsala University and Aalborg University, 17 April 2012. [Online]. Available: http://www.uppaal.org. [Accessed 31 August 2012].

[16] K. Havelund and T. Pressburger, "Model Checking Java Programs Using Java PathFinder," *International Journal on Software Tools for Technology Transfer (STTT),* vol. 2, no. 4, 25 March 2000.

[17] M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma and Y. Yushtein, "Formal Correctness, Safety, Dependability and Performance Analysis of a Satellite," in *34th Int. Conf. on Software Engineering (ICSE 2012)*, 2012.

[18] H. P. Gumm, Philipps-Universität Marburg, "Das SMV System," Lecture Notes Winter Semester 2011/2012. [Online]. Available: http://www.uni-marburg.de/fb12/formalemethoden/vorlesungen/modelchecking11/skript/kapitel02.pdf. [Accessed 31 August 2012].

[19] D. Berardi und G. D. Giacomo, „Planning Via Model Checking: Some Experimental Results,“ Roma/Italy, 2000.

[20] D. Long and M. Fox, "The 3rd International Planning Competition: Results and Analysis," *Journal of Artificial Intelligence Research,* vol. 20, pp. 1-59, 2002.

[21] P. M. Fischer, V. Schaus, D. Lüdtke and A. Gerndt, "Design Model Data Exchange Between Concurrent Engineering Facilities by Means of Model Transformation," in *Proceedings of the 13th NASA-ESA Workshop on Product Data Exchange (PDE 2011),* Los Angeles/USA, 2011.