

Practical Results from the Application of Model Checking and Test Generation from UML/SysML models of on-board Space Applications

J. M. Faria, S. Mahomad, N. Silva

Critical Software S,A

Parque Industrial de Taveiro, Lote 48,

3045-054 Coimbra, Portugal

[jmfaria, samahomad, nsilva]@criticalsoftware.com

Abstract

The deployment of complex safety-critical applications requires rigorous techniques and powerful tools both for the development and V&V stages.

Model-based technologies are increasingly being used to develop safety-critical software, and arguably, turning to them can bring significant benefits to such processes, however, along with new challenges.

This paper presents the results of a research project where we tried to extend current V&V methodologies to be applied on UML/SysML models and aiming at answering the demands related to validation issues.

Two quite different but complementary approaches were investigated: (i) model checking and the (ii) extraction of robustness test-cases from the same models. These two approaches don't overlap and when combined provide a wider reaching model/design validation ability than each one alone thus offering improved safety assurance.

Results are very encouraging, even though they either fell short of the desired outcome as shown for model checking, or still appear as not fully matured as shown for robustness test case extraction.

In the case of model checking, it was verified that the automatic model validation process can become fully operational and even expanded in scope once tool vendors help (inevitably) to improve the XMI standard interoperability situation.

For the robustness test case extraction methodology, the early approach produced interesting results but need further systematisation and consolidation effort in order to produce results in a more predictable fashion and reduce reliance on expert's heuristics.

Finally, further improvements and innovation research projects were immediately apparent for both investigated approaches, which point to either circumventing current limitations in XMI interoperability on one hand and bringing test case specification onto the same graphical level as the models themselves and then attempting to automate the generation of executable test cases from its standard UML notation.

1 INTRODUCTION

The presented R&D work was produced in the context of an ESA sponsored ISVV "process validation" program using actual projects as subject-matter.

Model based techniques such as the semi-formal modelling languages UML and SysML is a practice with increasing application both inside and outside the field of safety-critical systems and applications.

Its widespread utilisation, although mainly for the design and development stages, brings a new set of challenges and demands of which, providing adequate validation capabilities to the model itself and to its derived products (software systems in this case) are only a few of them.

So, at the same time, many works have been done focusing on the validation of the developed UML/SysML models [1] and on the extraction of tests from such models [2][3].

In the work here reported, two complementary approaches for the verification and validation of UML/SysML models were followed. In the first, UML was combined with formal model checking [4] for the automatic validation of the correctness of the (UML) models.

In the second approach, a methodology was devised for identifying and defining robustness validation test cases from the UML/SysML model. Such methodology is inspired in other known techniques in safety-critical domains, such as Failure Mode and Effects Analysis (FMEA) [5] and Sneak Circuit Analysis (SCA) [6].

This paper describes our experience with both approaches highlighting both capabilities and limitations, and presenting a thorough analysis of possible future research work in such domains. It is our expectation to actively contribute to the development of such promising technologies.

The remaining of this paper is organized as follows: Section 2 presents an overview of the technologies associated to the work here described, namely, model checking, FMEA, and SCA. Section 3 details the methodology and results from the automatic combination of UML and model checking while Section 4 covers the methodology for extraction of robustness tests. Section 5 presents the conclusions of our work and, finally, Section 6 presents our view and expectations for future work in the field.

2 OVERVIEW OF PROCESSES AND TECHNOLOGIES

This section presents a summary description of the processes and technologies related to the work presented in this paper.

2.1 MODEL CHECKING

Model checking is an automatic technique for verifying finite-state reactive systems, such as sequential circuit designs and communication protocols [4]. It is a technique intended to prove automatically that a (set of) logical property(ies) P , holds of a system behaviour S , abbreviated $S \models P$. Logical properties are expressed in temporal logic and system behaviour as some sort of a finite state machine. The model checker generates the entire state space of the finite state machine S and analyses whether P holds for every state. There exist model checking approaches which apply different decision procedures.

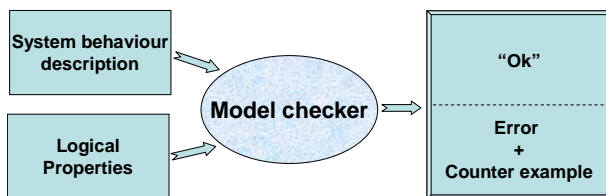


Figure 1 - Model checking principle

Typically, the system behaviour is abstractly represented by a graph, where the nodes represent the system's states and the arcs represent possible transitions between the states. Because graphs alone are too weak to provide an interesting description, they are annotated with more specific information. A common approach is to recur to Kripke structures [4].

Temporal logic is a class of modal logic. It extends propositional logic to incorporate time operators, in the sense that formulas can evaluate to different truth values over time. There are different types of temporal logic notations that correspond to different views of time (branching vs. linear, discrete vs. continuous, past vs. future, real time). Examples of temporal logic formal languages are Linear Temporal Logic (LTL), Computational Tree Logic (CTL) and Timed CTL.

Temporal logic is a powerful tool to express several properties about systems. Examples of properties that can be expressed (and consequently verified by model checkers) are:

- Reachability: some particular situation can be reached;
- Safety: something never occurs;
- Liveness: something will ultimately occur;

- Fairness: something shall (or shall not) occur infinitely often;
- Deadlock-freeness: the system can always evolve to a successor state.

While a violation of a safety property can be detected by a finite sequence of executions steps in the system, a violation of a liveness property may be detected only by an infinite execution of the system. Consequently, liveness properties are harder to be verified by model checkers.

Examples of existing model checkers are Alloy, FDR, Kronos, Maude, NuSMV, SPIN, TLC, and UPPAAL.

2.2 FMEA

This FMEA overview is based on [5].

Failure modes and effects analysis (FMEA) is the analysis by which each potential failure mode in a product (or function or process) is assessed to determine its effects. The potential failure modes are classified according to their severity [IEC 60050-191].

The failure modes and effects analysis or failure modes, effects and criticality analysis (FMEA/FMECA) process, performed in a timely and iterative way, is relevant in the decision making process. It is more effective to improve design or processes if implemented at an early stage.. As soon as preliminary information is available at high level the FMEA/FMECA process can be initiated and then extended to lower levels as more detailed artefacts become available.

As described in [5], the following steps are the essence of this analysis:

1. Define the product (i.e. function or hardware) to be analysed. This step comprises identification of interface functions, expected performance, product restraints and failure definitions. Functional descriptions of the product shall include all tasks to be performed for each mission, mission phase and operational mode. Functional analysis can be used as input for product definition.
2. Prepare functional and reliability block diagrams showing the operation, interrelationships and interdependencies of the product items. All product interfaces shall be indicated.
3. Identify all potential failure modes for each item and investigate their effect on the item and on the product and operation under study.
4. Evaluate each failure mode according to the worst potential consequences and assign a severity category.
5. Assess the probability of occurrence of each identified failure mode and assign a criticality category when possible.
6. Identify failure detection methods and existing compensating provisions for each failure mode.

7. For all critical items propose corrective or preventive actions (such as operator actions) required to eliminate the failure or to mitigate and control the risk.

8. Document the analysis and summarize the results and the problems that cannot be solved by the corrective actions. Derive a list with critical items.

2.3 SNEAK CIRCUIT ANALYSIS

This SCA overview is based on [6].

Sneak Analysis is used to identify “sneak circuits”, i.e. unexpected paths for a flow of mass, energy, data or logical sequence that under specific conditions can initiate undesired functions or inhibit other functions. Sneak circuits are not the result of failure, but latent conditions, existing in the system.

Since system failures can occur as a result of design errors and in the absence of component failures, SCA can be very useful.

Sneak analysis is a generic term used for the analytical techniques employed to methodically identify sneak circuits and design concerns in a system. Sneak path analysis is a sneak analysis technique that relies on the identification of paths between “targets” and “sources” and the use of clues, which supports the design concern analysis.

The SCA is composed by preparatory tasks that include definition of the analysis scope, gathering of the data for the subsequent steps, decomposing the design in “blocks” according to the functions of the part of the system under analysis and documenting functional inputs and outputs matrix.

The core sneak analysis consists of the following steps:

1. Sneak path analysis. Identification of sneak paths, sneak timings and sneak indications through: identification of targets and sources, tracing of paths between them, and application of “component + path” clues to the components that constitute the path;
2. Design concern analysis. Identification of sneak labels and design concerns through application of “component” clues;
3. Assessment of the consequences of sneak circuits and design concerns up to the highest level of design decomposition decided.
4. Documentation. Sneak circuits and design concerns are documented together with the recommendations to eliminate them as well as input data, interim results and conclusions.

3 UML AND MODEL-CHECKING

3.1 METHODOLOGY DESCRIPTION

The methodology explored combined UML/SysML modelling and model checking in an automatic manner, i.e., the input for the model checker was generated automatically from the UML/SysML models. For that, it was necessary to have a format converter tool, capable of performing such

translation. Once the models are translated, they can be verified by the model checker. [Figure 2](#) depicts the methodology.

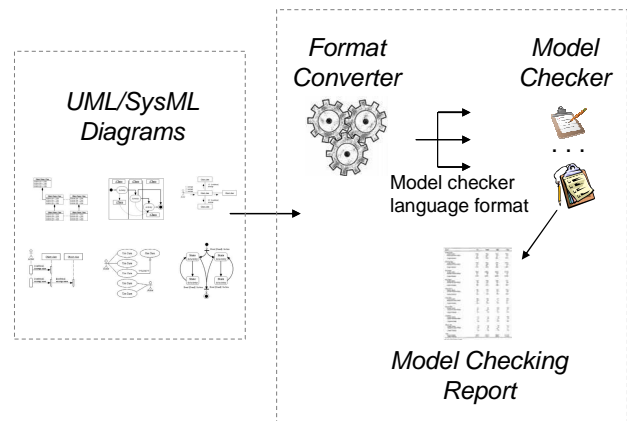


Figure 2 - Automatic combination of UML/SysML and model checking

3.2 TOOL SUPPORT

The crux of this methodology is to have a tool capable of automatically translating the information in UML diagrams into the input language of model checkers. Hugo/RT [7] was the tool found for such. It can generate input for both SPIN and UPPAAL.

The UML models are read in XMI (XML Metadata Interchange) format. Hugo/RT takes the XMI file (generated by the UML modelling tool), translates it to its internal format UTE and, subsequently, to the model checker.

Not every UML diagram can be mapped to Hugo/RT’s output. In fact, it is important to recognize that UML modelling and model checking are quite different technologies and, therefore, it is not possible to have a direct mapping between the two and to translate every expressible element in UML into some part of the system to be subject to model checking.

The input of a model checker comprises a system behaviour description and a set of properties. To translate a UML model with Hugo/RT, firstly, state machine diagrams may be associated with UML classes to specify the states that an object can visit and to describe its reaction to incoming events. Second, communication diagrams describe how the objects of a system may interact [1]. The model checker can then verify whether the interactions expressed by a communication diagram can indeed be realized by a set of state machines.

3.3 EXPERIMENTATION RESULTS

In our experiments, Hugo/RT was used to translate UML diagrams to the SPIN model checker. Proceeding as just explained in the paragraphs above indeed allowed us to validate the methodology and behaviour of the tools. However, a number of issues that lessen the applicability of the methodology in industrial projects were found. Namely:

- Hugo/RT can take as input UML models in (the OMG standard) XMI 1.0-1.2 format. However, it was observed that the XMI output format is not standard, i.e., different UML modelling tools build the XMI export file with different layouts.
- Hugo/RT is tailored to deal with the XMI output format generated by the ArgoUML tool and, because of the non-standardization, it is not capable of interpreting XMI files from other UML modelling tools.
- ArgoUML does not correctly support communication diagrams; making that objects interactions must be specified directly in Hugo/RT's internal UTE format.

4 ROBUSTNESS TESTS DEFINITION

4.1 CONTEXT & MOTIVATIONS

The work shown was the result from the validation test identification and definition from [prior] UML/SysML software reference model. Emphasis was given to robustness or dependability test cases while avoiding overlap with regular functional testing already produced by existing development and ISVV processes.

Finally, the focus of the work package should be placed on the process for identification of test cases and not on its automation.

As such, a first approach for a methodology was created, combining certain elements from FMEA and SCA which are established engineering practices in the safety critical area (the ECSS standards) and applying them to a process to extract dependability test cases from UML/SysML models.

4.2 METHODOLOGY DESCRIPTION

The methodology can be summarized as identifying every failure and sneak paths from a given model diagram and propagate failure until the limit of selected scope. It is comprised of the following steps:

Context Evaluation: Preparatory stage gathering information from prior V&V stages like review item discrepancies (RID) from design reviews and outputs from FMECA analysis which should serve to identify the optimum initial starting points i.e. model elements/diagrams for the test case extraction process. Analysis should then cover the remainder of the design model.

Test case extraction from model: Review the selected model diagram applying software oriented FMEA and SCA techniques and also with assistance from clues list (checklist) for generic software problems as well as safety critical design concerns. Only when a failure mode has been detected whose handling is not clear in the model is translated into a test case.

Test case definition: test cases were then specified in high-level informal textual form and save for the test setups involving specific failures requiring fault injection, they assumed a black-box type of interaction with the system.

Failure Propagation: whenever a problem has been found in a model, the analysis will try to perform failure propagation by analysing other connected or related UML/SysML models and presuming the failure mode being propagated.

4.3 RESULTS

The application of the methodology over a period of two weeks and focusing only on a subset of the software models resulted in sixty robustness test cases being defined.

Test cases provided only partial functional coverage, so in order to produce a test plan ensuring full functional coverage with both positive and negative verifications other more efficient methods are available and should be used.

This methodology however, did produce some unconventional robustness test cases that to our knowledge provide good added value to the validation process.

5 ANALYSIS AND CONCLUSIONS

5.1 MODEL CHECKING

The combination of UML and model checking seems to be a very interesting mechanism for the verification of UML models. Hugo/RT is a tool that is capable of automatically translating UML models information into the input language of the model checkers SPIN and UPPAAL, allowing the establishment of a complete tool chain for the implementation of such methodology.

In Hugo/RT, model information is first translated to UTE and then to the model checker language. In the experiments made, the translation from UTE to Promela, the input language of the model checker SPIN, posed no problems. The central limitation of the global approach is, presently, in the translation of the UML model information into UTE. UML models are imported in XMI format, but it was observed that each UML modelling tool builds its own XMI file with different layouts and, consequently Hugo/RT is not capable of interpreting the XMI files from every tool. In fact, Hugo/RT has been tailored to work with the XMI generated from ArgoUML. ArgoUML, on the other hand, is still a limited tool and does not allow the correct construction of communication diagrams, which should be used to express properties to verify of the system.

In conclusion, the approach is interesting but not yet mature for industrial use. It clearly needs some effort on the front-end, i.e., the connection from UML to UTE.

5.2 ROBUSTNESS TESTS

The quality of the results shows that the proposed methodology approach deserves continued consolidation.

The robustness test cases derived from application of the method constitute an excellent complement to automated and formal model checking.

Other merits include being easier to visualize and collaborate on the analysis due to its reliance on graphical notations like UML/SysML, allowing scope and depth of analysis to vary according to needs, and also for permitting systematization of the analysis to be performed, which allows engineers with less system knowledge or experience to perform the analysis.

The method is also highly applicable with goals other than the extraction of test cases, such as in early design stages in an iterative way as a design improvement or validation process.

6 DIRECTIONS FOR FUTURE RESEARCH

6.1 MODEL CHECKING

Due to the significant potential of combining UML and model checking for the verification of UML models, future research in this field is expected. Most likely, similar tools to Hugo/RT will rise in mid-term future. Even so, simply taking Hugo/RT and trying to solve its current limitations (the explained problems in the front-end) is probably the most cost-effective approach and can produce good results in short-term future. Overcoming such difficulties should comprise:

- Adopting more powerful UML modelling tool than ArgoUML;
- Extending Hugo/RT's *understandability* of XMI files; or tailoring XMI files into ArgoUML-like XMI format;
- Implementing the automatic translation of communication diagrams into UTE format.

One other possibility for the extension of this methodology is the support for OCL, a formal language used to write annotations on UML models [8]. Like so, OCL could be used to express properties in the model that could also be automatically translated and verified by the model checker.

6.2 ROBUSTNESS TESTS

Adding to the test case extraction methodology the capability to visually express them, and automatically have them in an executable form (or even close to it), could be a very powerful evolution and capability.

Visual expression of test cases: Since the methodology can already be used from the early design stages, and even advance along the design iterations, many advantages stand to be gained by elevating the test case representation from an informal textual form to an accompanying graphical notation on par with the model itself. The advantages include: more formal test case expression, visual test case representation

aiding interpretation and collaboration, leveraging the modelling tools, automation capability, etc.

Applied R&D for such an evolution comprises using UML Test profile (U2TP), for graphical notation of test cases derived from the methodology above and applied to safety critical software systems, and assessing its benefits and disadvantages against purely textual representation and management of test cases.

Automation capabilities: Since U2TP contains mappings to an internationally standardized testing language, Testing and Test Control Notation Version 3 (TTCN-3), automatic test case generation from U2TP graphical tests should be attainable.

Evolutions in this area comprise the following applied R&D actions:

- Establishing a fully operational TTCN-3 environment for a given SVF
- Migrating real test cases into TTCN-3 environment and assess its capabilities
- Express the same real test cases in U2TP and assess automatic TTCN-3 test case generation capabilities

7 REFERENCES

- [1] T Schäfer, A Knapp, and S Merz. "Model Checking UML State Machines and Collaborations." *Electronic Notes in Theoretical Computer Science*, Vol. 55, No. 3. (2001), pp. 357-369. [<http://www.loria.fr/~merz/papers/sw-mc01.pdf>]
- [2] <http://www.conformiq.com/>, date of visit: 29-01-2009.
- [3] <http://www.leirios.com/>, date of visit: 29-01-2009.
- [4] E.M. Clarke, O. Grumberg, D. Peled. "Model Checking", MIT Press, December 1999. ISBN 0-262-03270-8.
- [5] ECSS-Q-30-02A Failure modes, effects (and criticality) analysis (FMEA/FMECA). ECSS Secretariat – ESA/ESTEC, 2001.
- [6] ECSS-Q-40-04A Part 1, Sneak analysis - Part 1: Method and procedure, ECSS Secretariat – ESA/ESTEC, 1997.
- [7] <http://www.pst.informatik.uni-muenchen.de/projekte/hugo/>, date of visit: 29-01-2009.
- [8] <http://www.omg.org/technology/documents/formal/ocl.htm>, date of visit: 29-01-2009.