MODEL-CHECKING FOR TASTE DESIGNED SPACE SOFTWARE SYSTEMS: RESULTS AND LESSONS LEARNED

I. Dragomir [(1)], C. Redondo [(1)], T. Jorge [(2)], L. Gouveia [(2)], M. Bozga [(3)], I. Ober [(4)], M. Perrotin [(5)]

[(1)] GMV Aerospace and Defence, Isaac Newton 11, PTM, Tres Cantos, 28760, Spain, Email: {idragomir, carlos.redondo.aparicio}@gmv.com

[(2)] GMV, Alameda dos Oceanos, 115, Lisbon, 1990-392, Portugal, Email: {tmdasilva, lasequeiragouveia}@gmv.com

[(3)] VERIMAG, CNRS, 700 Av Centrale, 38401 St. Martin d'Hères, France, Email: marius.bozga@univ-grenoble-alpes.fr

[(4)] ISAE, 10 avenue Edouard Belin, 31055 Toulouse, France, Email: iulian.ober@isae-supaero.fr

[(5)] European Space Agency (ESA), ESTEC, Keplerlaan 1, PO Box 299, Noordwijk, The Nertherlands, Email: maxime.perrotin@esa.int

Objective: O-4

ABSTRACT

Model-Based Systems Engineering (MBSE) is an adopted modelling and development approach for correct-by-construction of complex software systems, such as space applications. TASTE [1] is a pragmatic and mature MBSE toolset supported by ESA that enables and provides automation for most of the phases of software system development: (i) heterogeneous system design through several modelling and programming languages (e.g., ASN.1, AADL, SDL, C/C++), (ii) code generation, build and deployment of the binary application(s), (iii) validation through static analysis and simulation, and (iv) formal verification of properties by model-checking. The formal verification capabilities have been recently added to the TASTE toolset in the ESA project *Model-Checking for Formal Verification of Space Systems* (MoC4Space) and validated on two real-life case studies. Within this paper we report on the results and lessons learned during the project.

1 INTRODUCTION

The development of large and complex software systems is a challenging task. MBSE, in general, allows various engineering teams to design such systems with minimal effort and cost and to obtain an integrated system conforming to its requirements. With respect to software systems, MBSE allows to produce correct applications for target platforms, provided the unambiguous and consistent software design and the incremental and iterative verification and validation of desired properties (e.g., typing, behaviour).

TASTE [1] is an open-source MBSE toolset supported by ESA for software systems design, validation and verification, and binary application(s) generation. The software design involves defining the data structures in ASN.1 [2], the hierarchical system architecture in a flavour of AADL [3], the system behaviour either in a modelling language such as SDL [4] or in a programming language such as C/C++, and the system deployment on dedicated target platforms (e.g., x86 Linux, RTEMS). The software validation involves a plethora of techniques such as static type analysis, real-time scheduling analysis, guided/random simulation, debugging and testing. The software verification involves a dedicated model-checker that exhaustively analyses the system behaviour and gives a verdict for property satisfaction.

The dedicated model-checker is based on the IF toolset [5] and has been recently developed and integrated in the TASTE toolset in the ESA *Model-Checking for Formal Verification of Space Systems* (MoC4Space) project[1]. This paper reports on the project results – the open-source developed tool and its use to two flight software representative case studies, as well as some lessons learned.

2 A MODEL-CHECKER FOR TASTE

Model-checking ([6], [7]) is a well-known formal verification technique for establishing the system correctness with respect to a set of properties. It consists of the exhaustive exploration of all the system behaviours and checking if the properties hold on all them. The advantages of model-checking are that it is fully automated and it produces diagnostic traces when a property is violated. However, the model-checking can become intractable even for reasonably complex systems due to all possible behaviours to explore. The model-checker developed in this activity, described in [8], is based on the IF toolset [5], an open-source toolset for model-checking of real-time systems including state-of-the-art model exploration and analysis techniques.

The TASTE model-checker takes as input the design of the system of interest (SOI) in TASTE including data view (data structures in ASN.1), interface view (hierarchical architecture in AADL), and function behaviour (one implementation per function in either SDL, C/C++ or as a GUI – a model of the environment of the SOI). It supports three formalisms for

[1] This project is funded by the European Space Agency under contract number 4000133658/21/NL/CRS.

modelling properties: Boolean Stop Conditions (BSCs), Message Sequence Charts (MSCs) and Observers. BSCs are the simplest property type that describe undesired behaviour of the system in the form of a state condition. When creating a BSC property, the tool automatically generates the skeleton for it in the form of an observer that the user completes with the desired condition using OpenGEODE (see Figure 1-(b)). MSCs describe (un)desired behaviour of the system as a sequence of input/output events between the system's functions. MSC properties are created using the integrated MSC Editor, defining their type through a dedicated property-type comment. Examples of MSC properties are provided in Figure 2 and Figure 3. Finally, observers are the most complex properties that describe (un)desired behaviour in the form of state machines by monitoring and altering the system state and events. Observer properties are created using OpenGEODE. Examples of observer properties are provided in Figure 2 and Figure 3. For further details on the property language the reader is referred to [8].

The Model-Checker Wizard is integrated within the TASTE development environment – *spacecreator*, and provides the user interface to control and run the model-checking. The wizard provides the features to create, edit and/or update properties as defined above and to possibly restrict the scope of the model-checking e.g., by restricting the system's environment through subtyping or the set of active functions for a certain property. All these features are provided in the Configuration tab illustrated in Figure 1-(a). In particular, the *subtyping* consists of restricting the domain of ASN.1 values sent over the required interfaces of GUI functions modelling the system's environment. An example is provided in Figure 1-(c). The user can also decide that no subtyping is needed, and in this case the tool generates by default the subtypes identical to the original types.

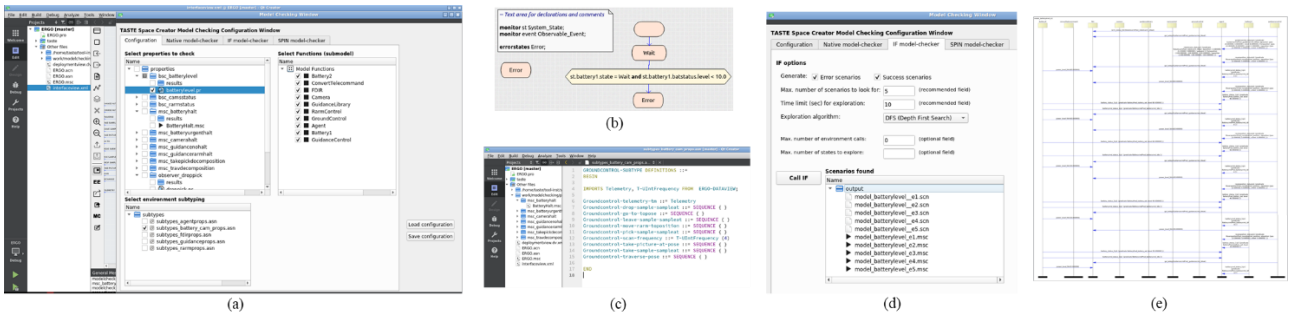

Figure 1. (a) Model-checking Wizard – Configuration, (b) Boolean Stop Condition (BSC) property, (c) Subtyping, (d) Model-Checking Wizard – IF model-checker options and results, (e) Message Sequence Chart illustrating a diagnostic trace for the BSC property.

Furthermore, the Model-Checker Wizard further allows to select the specific options and to invoke the underlying IF model-checker as illustrated in Figure 1-(d). Such options cover the algorithm for model exploration (depth-first or breadth-first search), the limits for exploration (time, number of states, number of inputs from environment) and the generation of diagnostic scenarios (success or error, maximum number of scenarios). Upon invocation of the underlying model-checker through the Call IF button, the TASTE *kazoo* build system is first used to transform the TASTE design and property into their IF counterparts and then to run the IF model-checker on them. The IF model-checker executes on the setting specified and provides a result for property satisfaction – yes, no, or inconclusive in case one of the limits has been reached before the exploration finished, together with diagnostic scenarios. The diagnostic scenarios, if any, are automatically transformed into MSC, that can be analysed by the user.

The tool is fully implemented, open source and available through the TASTE toolset in [9].

3 VERIFICATION RESULTS

The model-checking tool has been validated on two industrial case studies: the IXV mission and the ERGO planetary exploration demonstrator [10]. We describe here the case studies and verification results obtained, which can also be found in [9].

3.1 The IXV Case Study

The IXV mission aimed to define the basic needs for re-entry from Low Earth Orbit. The case study considered in this activity is a subset of the fully automated on-board software that focuses on the flaps control system, illustrated in Figure 2-(a). Two main functional chains have been modelled: the flaps positioning sequence implemented by ASW GNC CON upon reception of launch vehicle separation signal from Environment, and the flaps FDIR sequence and deactivation implemented by SL_DevExternal which programs the actions, SL_Svc_Cmd which executes the actions and SL_Svc_EM which stores the different values of the parameters involved. In total, the model has 5 functions, all modelled with SDL, except the Environment defined as a GUI function, embodying the launch vehicle and the flaps.

The aim of the IXV case study is to provide a design representative of flight software which is simple enough to validate the model-checking tool. A total of 12 properties have been defined for the case study in all three formalisms. Figure 2-

(b) illustrates a BSC property which checks whether three consecutive failure *flapstatus* values can be sent by the Environment, this triggering the execution of the FDIR process. The verification result confirms the nominal behaviour of the system design by producing diagnostic scenarios accordingly. Figure 2-(c) illustrates an MSC property which checks an undesired behaviour of the system design consisting in an incorrect triggering of the FDIR process (i.e., after reception of a *healthy flapsstatus* value from Environment). The model-checking of this property discards any incorrect behaviour of this type, as no error diagnostic scenarios have been found, thus confirming the expected system response. Finally, Figure 2-(c) illustrates an observer property which checks whether the system is capable of recovering the flaps after the FDIR sequence is completed, or deactivating the flaps. In this case the model-checker generates diagnostic scenarios for both the FDIR and flaps deactivation sequences that confirm the expected results.

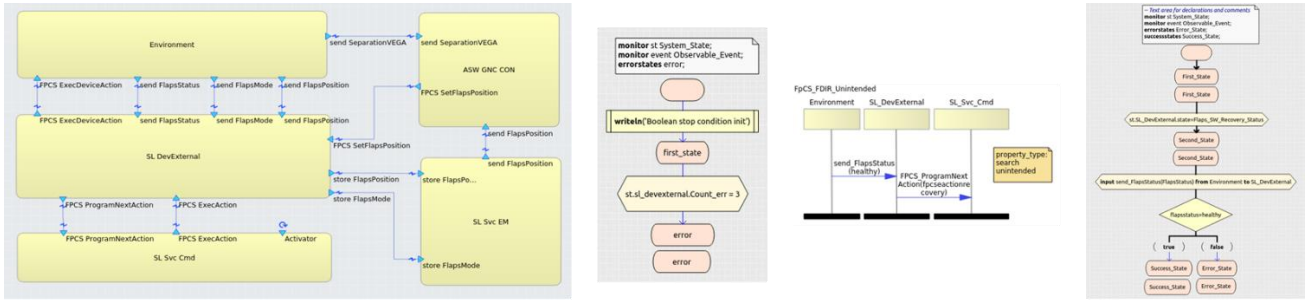

Figure 2. IXV case study: (a) interface view, (b) BSC property modelling the reception of 3 consecutive failure flapsstatus, (c) MSC property modelling an incorrect triggering of the FDIR process, (d) observer property modelling the flaps recovery after FDIR process.

All the verification results obtained were expected, including the deliberate modelling of errors, thus fully validating the tool. Moreover, the verification took in average 10 seconds (most of the properties required a verification time below 10 seconds), with a maximum of 91 seconds for a single property. The only limitation encountered with this case study is related to the expressiveness of the supported MSC formalism for specifying properties, which does not provide the means to specify interactions that should not be assessed by the model-checker.

3.2 The ERGO Case Study

The ERGO planetary exploration demonstrator is inspired by the Mars Sample Return (MSR) mission that covers the concepts and requirements of the Martian Long Range Autonomous Scientist. The case study considered in this activity consists of a subset of functionalities, "simulating" (simplified) traverses, sample collections, and image acquisitions, in E1 (telecommanding) and E4 (goal commanding) autonomy modes. The case study consists of a GroundControl for commanding, an Agent modelling deliberative and scheduling capabilities, GuidanceControl for traverse, RarmControl for sample caching, Camera for imaging, a 2-Battery system and FDIR. In total, the case study has 10 functions all modelled in SDL, except the GroundControl that is a GUI and GuidanceLibrary that supports the computations for GuidanceControl and is implemented in C++. The communication is done via 46 sporadic interfaces and 1 protected interface, and the functionalities require the periodic activation of 5 cyclic interfaces.

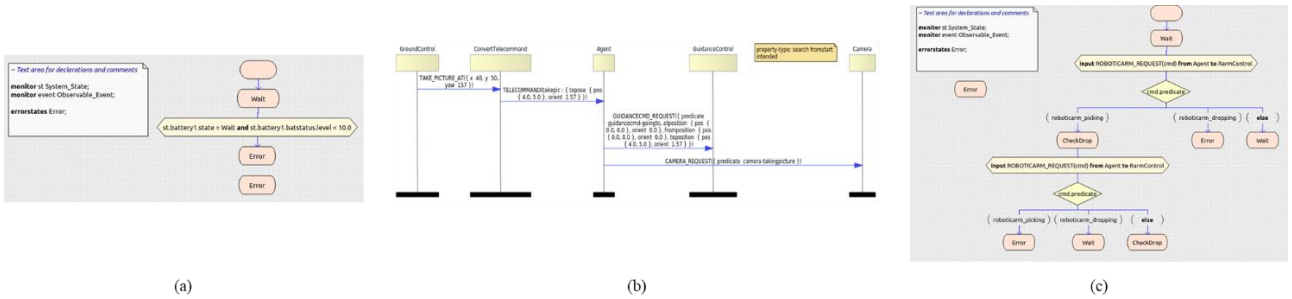

Figure 3. ERGO case study: (a) BSC property modelling that the battery level shall not drop below 10, (b) MSC property modelling the decomposition of a TAKE_PIC_AT high-level goal, (c) observer property modelling that a drop operation shall not be performed a pick.

A total of 16 properties have been defined for the case study in all three formalisms. Figure 3-(a) illustrates a BSC property which models that the system's Battery1 level shall not drop below 10 units. The verification results show that this property is not satisfied as the Battery1 level drops periodically and reaches a level below 10 units if no request to recharge is issued by GroundControl. The model-checker produces 2100 error diagnostic scenarios, which shows the complexity of this case study. Figure 3-(b) illustrates an MSC property which models the decomposition of the takepictureat high-level goal, which implies first reaching the requested pose, and upon successful completion taking an image. The model-

checker found 48 success diagnostic scenarios on which this pattern occurs, hence validating the property. Finally, Figure 3-(c) illustrates an observer property which models that only sequences of pick and drop are correct to be executed by the robotic arm, and any initial request to drop or two consecutive requests to pick are errors. The model-checker found 1016 error diagnostic scenarios, all of which starting with a drop operation being requested to the robotic arm, as the logic of which operations are available is not part of the design.

The ERGO case study considers a complex TASTE design aimed to identify the limitations of the tool. A first limitation is related to the TASTE design itself, which had to be adapted for verification: the interfaces with the environment were changed to easily specify and use the subtyping, the components have their functionalities enabled by Boolean variables and their status is published only upon change. A second technical limitation is on the use of C/C++ implementation in the design, that involved the modification of the generated IF model. This feature is not fully supported and it is part of future work. A third limitation is related to the supported MSC property language and its expressiveness. For example, properties that start with a conjunction of interactions cannot be semantically represented and it is recommended to use the observer formalism to express complex properties. Finally, the explosion of the state space (possible behaviours) has been noticed, as for one property the model-checker was not able to conclude in a 1-hour time limit. In average the time for verification is 20 minutes per property, with noticeable difference between the time representation used. All the results obtained were as expected, including the detection of the specific modelling errors added to the design.

4 DISCUSSION

In this paper we described the TASTE model-checking tool developed in the MoC4Space project, based on the IF toolset. The tool is a user friendly open-source implementation for model-checking, already available within the TASTE toolset.

The tool has been validated with two case studies, the IXV case study aiming to validate the approach and implementation for model-checking and the ERGO case study aiming to identify the limitations of the model-checker and propose some guidelines for models to be amenable to formal verification. The main limitations are related to the supported language for system and properties modelling and its expressiveness, and the state space explosion problem. In consequence, it is suggested that the system design is performed after the requirements are defined, such that the functions involved in the requirement satisfaction can be easily identified and used for verification. Additionally, the design should enable a fine-grain control of the environment of the system e.g., by specifying the available functions, interfaces, and values. The design should minimize the number of timers or cyclic interfaces used, as they are a source for the state space explosion. Hence it is suggested to group and use one timer for timed behaviour whenever possible, and also to trigger periodic functionalities with the same cyclic interface (which inherently uses a timer). With respect to the timed behaviour, the timing constraints shall be also simplified to increase the reactivity of the system with respect to the environment, and hence reduce the exploration. As for the communication between the functions via sporadic interfaces, the signals shall be sent when the information exchanged is relevant. Finally, it is suggested to use the observer formalism for modelling properties, unless they are very simple. It has been identified that the MSC formalism is not rich enough to express complex interaction conditions, e.g., the property to be checked starts as a conjunction of interactions.

Future work will address the limitations identified on the supported language for system and property modelling.

5 REFERENCES

[1] TASTE: The Assert Set of Tools for Engineering. Available at: https://taste.tools/

[2] ITU-T recommendation X.680 : Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation. 02/2021.

[3] Architecture Analysis & Design Language (AADL). SAE standard AS5506D. https://www.sae.org/standards/content/as5506d/. 04/2022.

[4] ITU-T. Specification and Description Language – SDL 2010. Available at https://www.itu.int/rec/T-REC-Z.100/en

[5] The IF toolset. Available at http://www-verimag.imag.fr/~async/IF/

[6] Queille, J. P.; Sifakis, J. (1982), "Specification and verification of concurrent systems in CESAR", *International Symposium on Programming*, Lecture Notes in Computer Science, 137: 337–351, doi:10.1007/3-540-11494-7_22, ISBN 978-3-540-11494-9

[7] Clarke, E. M.; Emerson, E. A.; Sistla, A. P. (1986). *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Transactions on Programming Languages and Systems, 8 (2): 244, doi:10.1145/5397.5399

[8] Dragomir, I., Bozga, M., Ober, I., Silveira, D., Jorge, T., Alana, E., Perrotin, M. (2021) *Formal Verification of Space Systems Designed with TASTE*, Model Based Space Systems and Software Engineering, 2022

[9] MoC4Space GitLab. https://gitrepos.estec.esa.int/taste/if-model-checking.git

[10] European Robotic Goal-Oriented Autonomous Controller (ERGO) H2020 project. https://www.h2020-ergo.eu/