Model Checking as a Service: Towards Pragmatic Hidden Formal Methods

Benedek Horváth
IncQuery Labs Ltd.
Budapest, Hungary
Johannes Kepler University Linz
Linz, Austria
benedek.horvath@incquerylabs.com

Bence Graics
Ákos Hajdu
Zoltán Micskei
Vince Molnár
Budapest University of Technology and Economics
Budapest, Hungary
{graics,hajdua,micskei,molnarv}@mit.bme.hu

István Ráth
IncQuery Labs Ltd.
Budapest, Hungary
istvanrath@incquerylabs.com

Luigi Andolfato
European Southern Observatory
München, Germany
landolfa@eso.org

Ivan Gomes
Robert Karban
Jet Propulsion Laboratory,
California Institute of Technology
Pasadena, CA, USA
{ivan.gomes,robert.karban}@jpl.nasa.gov

ABSTRACT

Executable models can be used to support all engineering activities in Model-Based Systems Engineering. Testing and simulation of such models can provide early feedback about design choices. However, in today's complex systems, failures could arise due to subtle errors that are hard to find without checking all possible execution paths. Formal methods, and especially model checking can uncover such subtle errors, yet their usage in practice is limited due to the specialized expertise and high computing power required. Therefore we created an automated, cloud-based environment that can verify complex reachability properties on SysML State Machines using hidden model checkers. The approach and the prototype is illustrated using an example from the aerospace domain.

CCS CONCEPTS

• **Software and its engineering → System modeling languages**; **Formal software verification**.

KEYWORDS

MBSE, SysML, verification, model checking

ACM Reference Format:
Benedek Horváth, Bence Graics, Ákos Hajdu, Zoltán Micskei, Vince Molnár, István Ráth, Luigi Andolfato, Ivan Gomes, and Robert Karban. 2020. Model Checking as a Service: Towards Pragmatic Hidden Formal Methods. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3417990.3421407

MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8135-2/20/10…$15.00
https://doi.org/10.1145/3417990.3421407

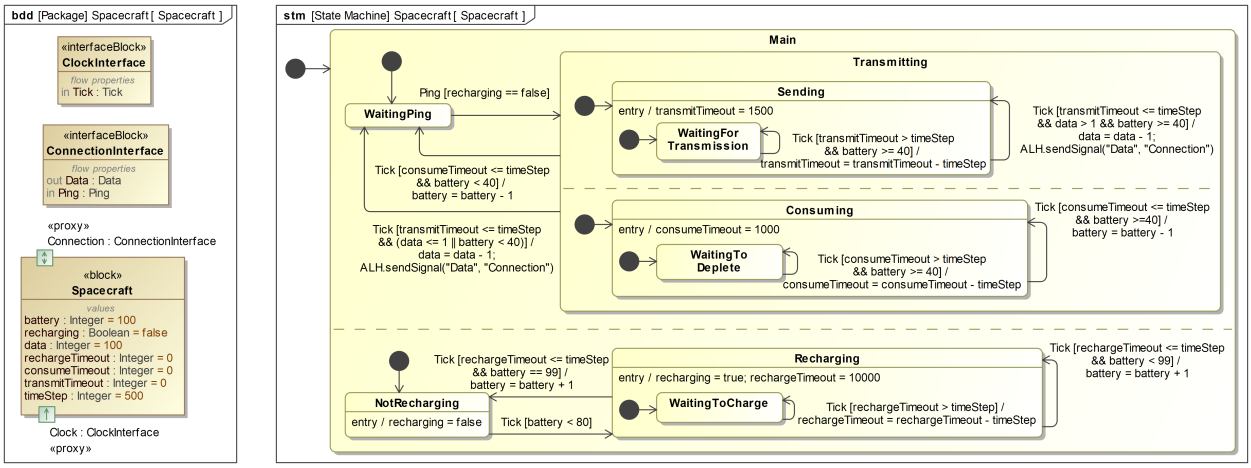
1 INTRODUCTION

Modeling is a key activity in any engineering discipline to cope with complex systems. *Model-Based Systems Engineering* (MBSE) makes models first-class citizens supporting all engineering activities from specification and design to validation, and not just documentation. A typical modeling language used in MBSE is the *Systems Modeling Language* (SysML) [20]. Recent developments in related specifications [19, 21] and tooling turned SysML artifacts into *executable models*, which can be simulated and analyzed [11] to provide early feedback about design options and decisions.

Motivation. Testing and simulation, however, can explore only a handful of traces for typical interactions or calculate common values for system properties. When the modeled system is complex, failures can arise due to subtle design details or corner cases that are difficult to detect with manual review or simulation (e.g., deadlocks in protocols or a combination of specific inputs and timing could result in an unexpected error). *Formal methods* use precise formalisms, logic solvers, and search algorithms to reason about correctness properties of the modeled system. *Model checking* [4] is an automated formal verification technique that can systematically traverse all possible execution traces in the model, and therefore either prove that an undesired behavior is not possible or show a concrete execution trace that violates the property.

Research on using model checkers to verify high-level engineering models, like UML State Machines, spans several decades [12]. These methods define a mapping from the high-level modeling language to the low-level mathematical input language of a model checker (e.g., transition systems or timed automata) encoding the semantics of the modeling language. Newer approaches extended this basic idea by making verification more scalable [13] or adapting to newer specifications (e.g., to SysML [7] or fUML [15]). However, most of such works require complex toolchains and a deep understanding of the model checking tools employed, therefore prohibiting its wide-spread usage by systems engineers. To counter

Authors' manuscript. Published in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020. The final publication is available at http://dx.doi.org/10.1145/3417990.3421407.



(a) Block definition **(b) Behavior specified as state machine**

Figure 1: A simple SysML model describing a simplified spacecraft and illustrating the scope of our approach.

this challenge, the inputs and configuration parameters of model checkers can be hidden using automated transformations, an approach commonly referred to as *hidden formal methods* [24].

Recently, MBSE approaches started to utilize cloud-based, open, collaborative environments, where various modeling and analysis tools can be integrated (e.g., OpenMBEE[1]). System models are stored in versioned model stores (e.g., OpenMBEE MMS[2] or Teamwork Cloud[3]). Scalable cloud-based environments also open up the possibility to use resource-intensive analyses such as model checking. Solutions are already available to perform static error analysis on the stored models, e.g., the IncQuery Server [10] supports complex model queries with the VQL language [3]. However, to the best of our knowledge, there is no integrated solution that utilizes formal methods and model checking to verify system models.

Objective. Our objective is to transfer the extensive research results on executable MBSE and formal verification into a pragmatic, integrated, and extensible environment. Our vision is to extend the analysis capabilities of IncQuery Server with integrated and automated model checking that can be used as a turnkey solution to catch hard to detect problems in system design models.

Results. We designed a flexible architecture and created a proof-of-concept tooling that automatically transforms SysML State Machine models stored in a central model store to the input language of different model checker back-ends via an intermediate representation, while carefully preserving the semantics of the original models. End-users can define reachability properties over predicates of states and variables directly in the system models. Model checkers are executed in containerized cloud environments, and the analysis results (e.g., an execution trace demonstrating a violation) are back-annotated to the original modeling elements, therefore completely hiding formal verification. To summarize, the contributions of the paper are as follows:

- Proposed a modern, cloud-native framework for verifying systems engineering models using model checker back-ends.
- Created a proof-of-concept Model Checking as a Service (MCaaS) environment based on proprietary and open tools.
- Demonstrated that formal verification can be completely hidden from the end-user with automated transformations.

2 MOTIVATING EXAMPLE AND SCOPE

A simplified spacecraft model in Figure 1 illustrates the kind of modeling elements and properties that our MCaaS approach currently targets. The spacecraft can receive a `Ping` signal from the ground to start sending data in packets. The data transmission consumes battery power, and if the battery level falls below 80%, the spacecraft has to start recharging. If the battery level falls below 40%, ongoing data transmission is paused until a full recharge. The duration of packet sending, power consumption, and recharging activities is specified with parameters. There are several properties that the system design has to fulfill, but as an example, consider that the spacecraft (a) should only start transmitting when receiving a ping, and (b) should never transmit when the battery is below 40%.

While Property (a) could be checked in principle with reviews or model validation rules, Property (b) is much harder as we have to consider all feasible paths in the model. This is where our MCaaS approach can exploit the full power of formal methods.

Scope. Checking state machine models is a hard problem in general. On the one hand, the rich set of modeling elements often result in large state space (e.g., a high number of interleavings) that pose a challenge for model checkers. Furthermore, there are also various cases where different tools or standards implement different behavior for the same modeling construct. Therefore, we currently limit ourselves to a subset of SysML that can be well supported by typical model checkers but is already expressive enough to show the feasibility and usefulness of our approach.

Our prototype supports the verification of a single hierarchical state machine that is owned by a *Block*. The state machine can have

[1]https://github.com/Open-MBEE
[2]https://github.com/Open-MBEE/mms-alfresco
[3]https://www.nomagic.com/products/teamwork-cloud

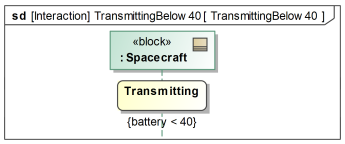


Figure 2: Reachability property.

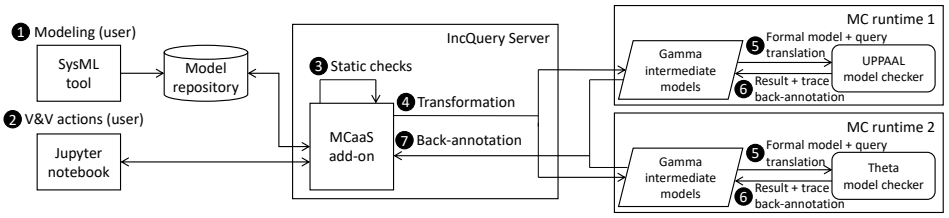


Figure 3: Overview of the MCaaS architecture.

both simple and composite *states*, consisting of *orthogonal regions* and *hierarchy*. The Block can have several *ports*. Each port is typed by an *Interface Block* that lists the allowed incoming or outgoing *signals*. Signals currently cannot have parameters. The Block can define several *Value Properties* that can be used in the state machine as variables in *guards*, *effects*, and *actions*. Currently, we support integer and Boolean types.

States can be connected by *transitions*, possibly also crossing hierarchy levels. Transitions can have a *trigger* and a list of guards and effects. Triggers can refer to signals arriving via any port of the owner Block. Guards consist of arbitrary predicates defined over the variables of the owner Block with basic arithmetic and comparison operators. Transition effects can either be assignments (with the same restrictions as for guards) or signal sending operations (implemented by the *ALH.sendSignal* helper script, specifying the signal and port name). Guards and effects must be atomic and bounded (e.g., no loops are allowed). In our implementation, we currently use a simplified JavaScript syntax, but it can later be replaced with Alf [18]. States can be associated with entry and exit *behaviors* (activated upon entering and leaving the states, respectively), which consist of assignments and signal sending operations with the same restrictions as transition effects. Do-behaviors, as they can be interrupted during execution, are currently not supported.

Reachability property. The prototype implementation supports the verification of *reachability properties* on state machines. Such properties describe *state predicates*: the configuration of the state machine and logical expressions over its variables. The purpose of the model checker is to prove if *any* execution *eventually* reaches a configuration where the predicate holds.

In our prototype, we use SysML sequence diagrams to define the reachability property, as illustrated by Figure 2. The property defines the undesired configuration where the `Transmitting` state of the state machine is active, and the `battery` level of the Spacecraft is below `40%` corresponding to Property (b) defined earlier in this section. The sequence diagram consists of a *lifeline* representing the Block whose state machine is to be verified. The lifeline can contain several SysML state invariants defining the state configuration (conjunction) to be reached. Furthermore, a simplified JavaScript syntax can be used to express logical predicates over variables.

3 MODEL CHECKING AS A SERVICE

An overview of our cloud-native MCaaS architecture is presented in Figure 3. Users design the state machines and define the properties of interest in their modeling tool of choice, and push them to a model repository ❶. Then, users open a web browser to perform verification and validation (V&V) actions ❷ with the MCaaS add-on. The add-on first performs static checks ❸ to validate the structural integrity of the models, before translating them into an intermediate representation ❹. We use the Gamma Statechart Composition Framework [17] and its statechart language as an intermediate representation, as it is close to SysML State Machines and provides various verification features. In our current scenario, we use Gamma to translate the intermediate representation to a formal model and a query to be checked by different model checkers ❺. The model checkers analyze the query and return their result in terms of the formal model. The verification result, including a possible execution trace, is back-annotated to the Gamma representation ❻, which is further mapped back to the original SysML representation to be presented in the browser ❼. This way, the process is fully automated, and all details of formal methods and model checking are hidden from end-users.

One advantage of the MCaaS approach is the separation of concerns for the engineering and the formal verification domains: systems engineers can design both the models and the properties in a high-level, engineering language they are familiar with. Furthermore, the verification result is also presented in this format, making it easier to understand without expertise in formal methods. The second advantage of our approach is that it uses an intermediate language instead of direct translation to model checkers. This allows easy integration of new model checkers (forming a portfolio), which is a crucial feature due to the fact that model checking is a hard problem, and different tools have different strengths. Currently, Gamma supports UPPAAL [2] and Theta [23]. Furthermore, model checkers can be started in parallel with different configurations, and their results can be combined. The ability to scale-out in the cloud allows the adaptive allocation of a high amount of computational resources [22] that can address the high resource demand of model checkers. Finally, the intermediate representation can also help in achieving semantic integrity. The tool- or standard-specific semantics could be implemented as parameters of the translation to the intermediate representation.

To prove the feasibility of our approach, we implemented a working prototype. In order to utilize the benefits of the cloud, we deployed IncQuery Server [10] together with the MCaaS add-on in a Docker[4] container. We defined a web interface for the model checker (MC) runtimes and deployed them in separate containers. In the following subsections, we discuss the details of the major steps of the workflow in the context of the prototype.

[4] https://www.docker.io

Modeling and static checks. In the prototype implementation, systems engineers use Cameo Systems Modeler[5] or MagicDraw[6] to design state machines and reachability properties in SysML ❶. These models are uploaded to Teamwork Cloud (TWC), a collaborative model repository. Engineers engage with the MCaaS workflow via the Jupyter notebook in a web browser ❷. The notebook is the frontend of the MCaaS add-on of IncQuery Server (IQS) [10], a scalable query evaluation middleware on the top of collaborative model repositories in the cloud. The add-on fetches a given revision of the model from TWC and caches it in memory on IQS. Next, IQS performs static checks ❸, defined by well-formedness constraints in VQL [3], to ensure the structural integrity of both the state machine and the property (e.g., the lifeline on the sequence diagram should represent a Block whose classifier behavior is the state machine). Invalid elements or constructs are reported in the browser along with an informative error message.

Transformation to the intermediate language. The SysML State Machines are transformed to the statechart language of the Gamma framework (GSL) ❹ [17]. The GSL language features are close to SysML State Machines. Thus, the elements of the currently supported scope (see Section 2) are transformed to the corresponding elements in GSL, taking into account the PSSM semantics [21]. Furthermore, the JavaScript guards and effect behaviors of transitions and states are translated to the Gamma expression and action languages, respectively. During the transformation, a traceability model between the SysML State Machine and the Gamma statechart is built to track the mapping between the source and target models. This traceability model is also used for transforming the reachability property and back-annotating the verification results to the source domain.

Gamma is a statechart composition framework, thus the statechart has to be placed in a wrapper, which defines scheduling and interactions with the environment. We create the wrapper based on the Block owning the state machine. The ports of the wrapper are connected to the ports of the state machine, simply forwarding the signals between the environment and the state machine. We use the synchronous composition semantics of Gamma, which means the statechart execution is periodically scheduled by an external trigger. In each execution cycle, the statechart consumes the input signals, changes its internal state (the state configuration or the values of its variables), and produces the output signals.

The reachability property is transformed into a temporal logic expression[7] in Gamma. The states are transformed to state references of the Gamma statechart, and the logical predicates are transformed to Gamma guard expressions. Finally, a reachability query is formed from the conjunction of the translated terms.

Translation to model checkers. Gamma supports UPPAAL [2] and Theta [23] as model checker back-ends ❺. Our prototype currently only integrates UPPAAL, where the statechart is translated to timed automata, and the reachability property to a liveness query [1] in CTL [4]. More information on the translation can be found in [9]. If the (reachability) property holds for the automata, a trace (execution path) leading to the target state is returned.

Back-annotation. The MCaaS approach performs back-annotation both from the result of the model checker to the intermediate (Gamma) representation ❻ and to the original SysML model in the form of a sequence diagram ❼ (see Figure 4). Each step of the execution trace contains all the relevant information to fully reproduce the execution: (1) the active state configuration, including the values of variables, the (2) set of input signals that were received, and the (3) set of output signals that were sent by the statechart. The sequence diagram is returned to the user, and it can be simulated in a tool of choice, e.g., the Cameo Simulation Toolkit,[8] helping engineers to inspect the execution in detail or to derive test cases.

Verifying the example. When checking Property (b) on the motivating example (Figure 1), the prototype confirms reachability of the undesired configuration and returns a trace (Figure 4). The state machine and its ports are represented by lifelines. Rounded rectangles depict active states, values of variables are shown between curly braces (in terms of the statechart in Figure 1b). Outgoing and incoming arrows, w.r.t. the `spacecraft` lifeline, represent sent and received signals, respectively. The trace proves that Property (b) is *violated*, because in the last state the `battery` is 39% and the Spacecraft is still transmitting. Using the trace, engineers can debug the model and find that wrong boundaries in the guards cause the issue: *battery* >= 40 and *battery* < 40 should be replaced by *battery* > 40 and *battery* <= 40, respectively. After fixing the guards, the prototype can *prove* that the undesired state is not reachable anymore.

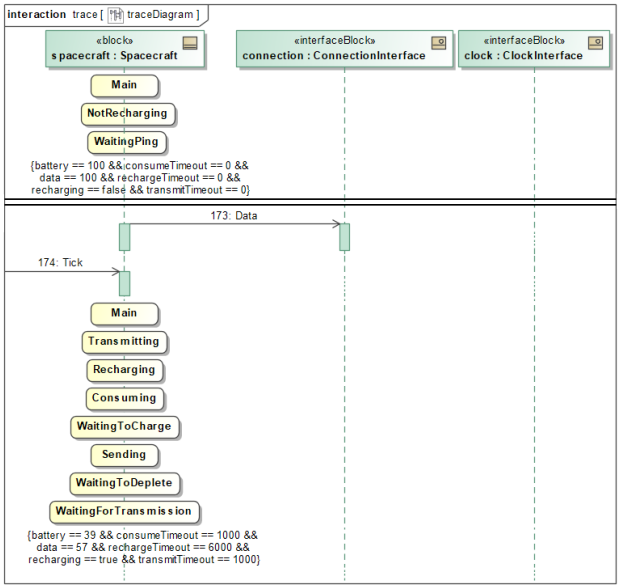


Figure 4: Trace showing that Property (b) is violated.

4 RELATED WORK

Gibson et al. used the Java Pathfinder model checker to verify multiple collaborating SysML State Machines including do-behaviors [7, 8]. However, the property to be checked must be given (as an assertion) in the generated Java code, guards were transformed manually,

[5] https://www.nomagic.com/products/cameo-systems-modeler
[6] https://www.nomagic.com/products/magicdraw
[7] Gamma supports a subset of CTL [4], including reachability (EF op.) as a special case.
[8] www.nomagic.com/product-addons/magicdraw-addons/cameo-simulation-toolkit

and the result (trace) was not annotated back. The *ProMoBox framework* [14] enables the automatic verification of Domain Specific Modeling Languages (DSMLs) by generating specialized metamodels allowing temporal properties in the language. They focused on providing verification extensions for any DSML, while we focused specifically on SysML. *RoboChart* [16] is a DSML for robotic applications, adopting the minimalist core of the UML state machine notation, while also supporting collaborating state machines and timing aspects. The formal properties are defined in a textual DSL with verification-specific keywords. The models and properties are translated into a CSP problem and solved by the FDR [6] refinement model checker, implemented in a desktop tool. *Zalila et al.* verify software processes modeled in the SPEM language [25] and properties defined in the temporal extension of OCL (TOCL). Both the SPEM model and the TOCL query are translated to a FIACRE model and properties in LTL. In contrast, we reuse elements of the source modeling language for the properties. *PLCverif* [5] also supports multiple model checker back-ends via an intermediate representation for PLC codes. Formal methods are hidden by writing requirements using English sentence templates (to be translated to CTL or LTL). PLCverif however, focuses on PLC codes in a desktop IDE, compared to our cloud-based solution targeting SysML State Machines. *Sharifloo and Metzger* proposed a cloud-based framework for checking run-time properties of adaptive systems [22], focusing on cloud resource allocation prediction based on model complexity and run-time measures (memory, CPU) from past executions. Our approach can also utilize such aspects in the future.

5 CONCLUSION

In this paper, we proposed a cloud-based, "push-button" verification workflow for SysML State Machines and reachability properties using the Gamma intermediate language and different model checkers. All details of formal methods and model checking are fully automated and hidden from the engineers via translations, including the back-annotations of the resulting trace.

In the future, we plan to add further model checkers to the workflow and experiment with their adaptive scalability by adopting and extending the work of Sharifloo and Metzger [22]. Moreover, we are planning to extend the supported state machine elements and languages for guards, effects, and do-behaviors described with activities to make it more useful for systems engineers, while preserving the *semantic integrity* of the workflow. Finally, although the workflow is fully automated and no formal methods knowledge is required, we plan to define points where human intervention and assistance is possible. This can help in cases where human expertise is needed to optimize a long-running verification job by modifying the engineering model or configuring the model checker.

ACKNOWLEDGMENTS

This research was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA). This work partially received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884. The authors are grateful for the valuable advice of Péter Bokor and Ákos Horváth.

REFERENCES

[1] Bowen Alpern and Fred B. Schneider. 1985. Defining Liveness. *Inform. Process. Lett.* 21, 4 (1985), 181–185.
[2] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. 2006. UPPAAL 4.0. In *Proc. of the 3rd International Conference on the Quantitative Evaluation of Systems*. IEEE, 125–126.
[3] Gábor Bergmann, Zoltán Ujhelyi, István Ráth, and Dániel Varró. 2011. A Graph Query Language for EMF Models. In *Proc. of the 4th Intl. Conference on Theory and Practice of Model Transformations (LNCS, Vol. 6707)*. Springer, 167–182.
[4] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick P Bloem. 2018. *Handbook of model checking*. Springer.
[5] Dániel Darvas, Borja Fernández Adiego, and Enrique Blanco Viñuela. 2015. PLCverif: A tool to verify PLC programs based on model checking techniques. In *Proc. of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems*. JACoW, 911–914.
[6] Thomas G.-Robinson, Philip J. Armstrong, Alexandre Boulgakov, and A. W. Roscoe. 2014. FDR3 - A Modern Refinement Checker for CSP. In *Tools and Algorithms for the Construction and Analysis of Systems*. LNCS, Vol. 8413. Springer, 187–201.
[7] Corrina Gibson, Robert Karban, Luigi Andolfato, and John Day. 2014. Formal Validation of Fault Management Design Solutions. *Softw. Eng. Notes* 39, 1 (2014), 1–5.
[8] Corrina Gibson, Robert Karban, Luigi Andolfato, and John C. Day. 2014. Abstractions for Executable and Checkable Fault Management Models. In *Proc. of the Conference on Systems Engineering Research*. Elsevier, 146–154.
[9] Bence Graics. 2016. *Model-Driven Design and Verification of Component-Based Reactive Systems*. Bachelor's thesis. Budapest University of Technology and Economics.
[10] Ábel Hegedüs, Gábor Bergmann, Csaba Debreceni, Ákos Horváth, Péter Lunk, Ákos Menyhért, István Papp, Dániel Varró, Tomas Vileiniskis, and István Ráth. 2018. Incquery Server for Teamwork Cloud: Scalable Query Evaluation over Collaborative Model Repositories. In *Proc. of the 21st International Conference on Model Driven Engineering Languages and Systems*. ACM, 27–31.
[11] Robert Karban, Frank G. Dekens, Sebastian Herzig, Maged Elaasar, and Nerijus Jankevičius. 2016. Creating system engineering products with executable models in a model-based engineering environment. In *Modeling, Systems Engineering, and Project Management for Astronomy VII*, Vol. 9911. SPIE, 96–111.
[12] Diego Latella, István Majzik, and Mieke Massink. 1999. Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker. *Formal Aspects of Computing* 11, 6 (1999), 637–664.
[13] Yael Meller, Orna Grumberg, and Karen Yorav. 2014. Verifying Behavioral UML Systems via CEGAR. In *Proc. of 11th International Conference on Integrated Formal Methods (LNCS, Vol. 8739)*. Springer, 139–154.
[14] Bart Meyers, Romuald Deshayes, Levi Lucio, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. 2014. ProMoBox: A Framework for Generating Domain-Specific Property Languages. In *Software Language Engineering (LNCS, Vol. 8706)*. Springer, 1–20.
[15] Zoltán Micskei, Raimund-Andreas Konnerth, Benedek Horváth, Oszkár Semerath, András Vörös, and Dániel Varró. 2014. On Open Source Tools for Behavioral Modeling and Analysis with fUML and Alf. In *Proc. of the 1st Workshop on Open Source Software for Model Driven Engineering*, Vol. 1290. CEUR, 31–41.
[16] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. 2019. RoboChart: modelling and verification of the functional behaviour of robotic applications. *Softw. and Sys. Model.* 18, 5 (2019), 3097–3149.
[17] Vince Molnár, Bence Graics, András Vörös, István Majzik, and Dániel Varró. 2018. The Gamma statechart composition framework: Design, verification and code generation for component-based reactive systems. In *Proc. of the 40th International Conference on Software Engineering*. ACM, 113–116.
[18] OMG. 2017. *Action Language for Foundational UML (Alf)*. formal/17-07-04.
[19] OMG. 2018. *Semantics of a Foundational Subset for Executable UML Models (fUML)*. formal/18-12-01.
[20] OMG. 2019. *OMG System Modeling Language (SysML)*. formal/19-11-01.
[21] OMG. 2019. *Precise Semantics of UML State Machines (PSSM)*. formal/19-05-01.
[22] Amir Molzam Sharifloo and Andreas Metzger. 2013. MCaaS: Model Checking in the Cloud for Assurances of Adaptive Systems. In *Software Engineering for Self-Adaptive Systems III (LNCS, Vol. 9640)*. Springer, 137–153.
[23] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. 2017. Theta: a Framework for Abstraction Refinement-Based Model Checking. In *Proc. of the 17th Conference on Formal Methods in Computer-Aided Design*. 176–179.
[24] Willem Visser, Matthew B. Dwyer, and Michael Whalen. 2012. The hidden models of model checking. *Software and Systems Modeling* 11 (2012), 541–555.
[25] Faiez Zalila, Xavier Crégut, and Marc Pantel. 2016. A DSL to Feedback Formal Verification Results. In *Proc. of the 13th Workshop on Model-Driven Engineering, Verification and Validation*, Vol. 1713. CEUR, 30–39.