Exploring the Concept of Abstract State Machines for System Runtime Enforcement

Elvinia Riccobene¹ and Patrizia Scandurra²

¹ Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy

elvinia.riccobene@unimi.it

² Department of Economics and Technology Management, Information Technology and Production, Università degli Studi di Bergamo, Bergamo, Italy

patrizia.scandurra@unibg.it

Abstract. Modern intelligent software systems are rapidly growing in complexity and scale, and many real usage scenarios might be impossible to reproduce and validate at design-time. As envisioned by the Models@run.time research community, the use of formal models at runtime are fundamental to address this challenge. In this paper, we explore the concept of *ASM@run.time* and put this definition into the context of the *runtime enforcement* technique to address the runtime assurance of software systems. This is a work-in-progress research line.

1 Introduction

Modern intelligent software systems, such as those employed in smart infrastructures using big data, AI and IoT technologies, are rapidly growing in complexity and scale, and many real usage scenarios might be impossible to reproduce and validate at design-time. To address this challenge, the Models@run.time research community [4] has identified a reference architecture to equip a software system with a model running in tandem with the system to address software runtime assurance. Similar ideas have been proliferating in other contexts, such as *Digital twins* in the manufacturing domain [11], and *Living models* [9] in the field of Computer Automated Multi-Paradigm Modelling.

Among the different approaches and techniques proposed in literature that exploit the concept of model@runtime, *runtime enforcement* [6] is a runtime verification method that focuses on steering system executions with the goal of preventing and reacting to misbehaviours and failures. Runtime enforcement techniques enforce the software system to run according to its specification, for example the specification of safety assertions that describe situations (states) or actions that must be avoided (e.g., a train must not open its doors when moving). When a new (input) event occurs that may change the state of the software system, the model, if available, is used to evaluate safety assertions and prevent the system change if it violates an assertion on the runtime model of the system. This enforcement mechanism can be, therefore, used for *input sanitisation* [6] to

protect the system from its (untrusted) environment. All inputs to the system shall enter first the enforcement mechanism which filters out those that could harm the system or ensure that all the necessary inputs are provided to the system. While classical runtime verification approaches (like [2, 5] and [7] to name a few) generally focus on the *oracle problem*, namely assigning verdicts to a system execution, runtime enforcement focuses on ensuring the correctness of the sequence of events by possibly modifying or preventing the system execution [6].

In this paper we present some preliminary results of our work-in-progress investigation on the use of *Abstract State Machines* as models@run.time for runtime enforcement. In particular, we present the architecture of a runtime enforcement tool we have been developing within the ASMETA framework¹ – a set of tools for the ASM formal method – to check safety assertions of software systems at runtime. This mechanism exploits the concept of executable ASM models and it is based on a new component, the `AsmetaS@run.time`, that simulates the ASM models in tandem with the real software systems. We also envision some real scenarios in the context of safety-critical systems where we are applying the ASM@run.time enforcement approach.

2 Runtime Enforcement with AsmetaS@run.time

We here present a conceptual view of a runtime enforcement for *input sanitisation* [6] to protect the system from its (untrusted) environment. The proposed mechanism exploits the runtime simulator for ASMs, namely `AsmetaS@run.time`. This last tool was recently developed as part of the ASMETA toolset to allow the use of ASM models as runtime models. It supports simulation *as-a-service* features including model roll-back to the initial state after a failure of the model execution (e.g., invariant violations, inconsistent updates, ill-formed inputs, etc.) while processing an input event.

The intent of the proposed runtime enforcement mechanism is to evaluate safety assertions when there is a new (input) event that may change the state of the system and prevent the change if it violates an assertion on the ASM runtime model of the system. As shown in Fig. 1, every attempt (or only those considered critical) to change the system state is mediated by a process (the *enforcer*) that decides whether the change is safe. To make this decision (one per each observed event), the enforcer process evaluates the effect of the event on an ASM model of the behavior of the system (or a subpart of it dealing with the most critical requirements) that runs on-board the system as runtime model. If we are in a safe state (both the system and its runtime ASM model) and there is a new (input) event that may change this state, the enforcer performs first the state transfer on the ASM model (by feeding the input event to the ASM in terms of a monitored function value) and makes sure that the transition (that may take several machine steps – an ASM run) will take the ASM to a state without violating an invariant or generating an inconsistent update. If the ASM will produce a (safe) change of state, the enforcer confirms the state transfers also to

¹ <http://asmeta.sourceforge.net/>.

Fig. 1. Runtime safety assertion enforcement by `AsmetaS@run.time`

the system, otherwise (the change is considered unsafe) it prevents the system to react to the event and a state rollback of the ASM model is performed to move it back to its previous (safe) state before the input event was processed. Therefore, the system is allowed to react to an event only when the model successfully checked the safety constraints for the event.

We assume that there exists a *catalog of safety assertions* (expressed in any suitable language) describing all possible situations that may produce a violation of safety, and that these assertions have been expressed in the ASM runtime model in terms of ASM invariants. The catalog may be dynamically updated at runtime in case dangerous situations have not been foreseen at design time or because of unanticipated changes in the requirements when such changed requirements are added at runtime as effect of unanticipated adaptation (such as service-based applications plugging in new services and components discovered at runtime to improve quality of service). We assume that the invariants corresponding to the assertions added to the catalog dynamically are added to the ASM runtime model dynamically, as well.

The runtime enforcement technique could be useful to prevent the execution of unsafe commands in cyber physical systems where the environment is only partially observable [8], and, in general, in any safety-critical system, where the effects of not enforcing the safety assertions would lead to human hazards, as it happens for medical software [1]. We do not target hard real-time systems since these systems require dedicated solutions (e.g., real-time operating systems) and pose specific challenges.

3 Conclusion

In this paper, we have presented our long-term vision of using the ASM executable models as formal support to the runtime enforcement technique to assure

safe execution of a software system. Our short-term plan is to complete the implementation of the other components (the **Assertion Catalog** and the **Assertion Enforcer**) of the proposed runtime enforcement mechanism based on the new **ASmetaS@run.time** component. We also want to test its effective operation in the area of safety-critical systems, as for example those in the medical software domain.

In the future, we plan to extend the **ASM@run.time** enforcement approach in the context of self-adaptive systems [3,10]. Our long term goal is to develop a complete framework able to deal with requirements changes also affecting the model behavior, and therefore providing model adaptation features at runtime.

References

1. Alemzadeh, H., Kalbarczyk, Z., Iyer, R., Raman, J.: Analysis of safety-critical computer failures in medical devices. *IEEE Secur. Priv.* **11**(4), 14–26 (2013). <https://doi.org/10.1109/MSP.2013.49>
2. Arcaini, P., Gargantini, A., Riccobene, E.: CoMA: conformance monitoring of Java programs by abstract state machines. In: Khurshid, S., Sen, K. (eds.) *RV 2011*. LNCS, vol. 7186, pp. 223–238. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_17
3. Arcaini, P., Riccobene, E., Scandurra, P.: Formal design and verification of self-adaptive systems with decentralized control. *ACM Trans. Auton. Adapt. Syst.* **11**(4), 25:1–25:35 (2017)
4. Bencomo, N., Götz, S., Song, H.: Models@run.time: a guided tour of the state of the art and research challenges. *Softw. Syst. Model.* **18**(5), 3049–3082 (2019). <https://doi.org/10.1007/s10270-018-00712-x>
5. Calinescu, R., Kikuchi, S.: Formal methods @ runtime. In: Calinescu, R., Jackson, E. (eds.) *Monterey Workshop 2010*. LNCS, vol. 6662, pp. 122–135. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21292-5_7
6. Falcone, Y., Mariani, L., Rollet, A., Saha, S.: Runtime failure prevention and reaction. In: Bartocci, E., Falcone, Y. (eds.) *Lectures on Runtime Verification*. LNCS, vol. 10457, pp. 103–134. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_4
7. Liang, H., Dong, J.S., Sun, J., Wong, W.E.: Software monitoring through formal specification animation. *ISSE* **5**(4), 231–241 (2009). <https://doi.org/10.1007/s11334-009-0096-1>
8. Pinisetty, S., Roop, P.S., Smyth, S., Allen, N., Tripakis, S., von Hanxleden, R.: Runtime enforcement of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.* **16**(5s), 178:1–178:25 (2017). <https://doi.org/10.1145/3126500>
9. Tendeloo, Y.V., Mierlo, S.V., Vangheluwe, H.: A multi-paradigm modelling approach to live modelling. *Softw. Syst. Model.* **18**(5), 2821–2842 (2019). <https://doi.org/10.1007/s10270-018-0700-7>
10. Weyns, D., Iftikhar, M.U.: ActivFORMS: a model-based approach to engineer self-adaptive systems. *CoRR abs/1908.11179* (2019). <http://arxiv.org/abs/1908.11179>
11. Zhuang, C., Liu, J., Xiong, H.: Digital twin-based smart production management and control framework for the complex product assembly shop-floor. *Int. J. Adv. Manuf. Technol.* **96**(1), 1149–1163 (2018)