Model-Driven Engineering for Design-Runtime Interaction in Complex Systems: Scientific Challenges and Roadmap

Hugo Bruneliere, Romina Eramo, Abel Gomez, Valentin Besnard, Jean-Michel Bruel, Martin Gogolla, Andreas Kästner, Adrian Rutle

Model-Driven Engineering for Design-Runtime Interaction in Complex Systems: Scientific Challenges and Roadmap*

Report on the MDE@DeRun 2018 workshop

Hugo Bruneliere[1], Romina Eramo[2], Abel Gómez[3], Valentin Besnard[4], Jean Michel Bruel[5], Martin Gogolla[6], Andreas Kästner[6], and Adrian Rutle[7]

[1] IMT Atlantique, LS2N (CNRS) & ARMINES, France
hugo.bruneliere@imt-atlantique.fr
[2] University of L'Aquila, Italy
romina.eramo@univaq.it
[3] Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya (UOC), Spain
agomezlla@uoc.edu
[4] ERIS, ESEO-TECH, Angers, France
valentin.besnard@eseo.fr
[5] IRIT (CNRS) & Université de Toulouse, France
bruel@irit.fr
[6] University of Bremen, Germany
{gogolla|andreask}@informatik.uni-bremen.de
[7] Western Norway University of Applied Sciences, Norway
Adrian.Rutle@hvl.no

Abstract. This paper reports on the first Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems (also called MDE@DeRun 2018) that took place during the STAF 2018 week. It explains the main objectives, content and results of the event. Based on these, the paper also proposes initial directions to explore for further research in the workshop area.

Keywords: Design time modeling · Runtime modeling · Interactions · Correspondences · Traceability · Feedback.

1 Introduction

Complex systems are now predominant in several domains such as automotive, health-care, aerospace, industrial control and automation [2]. Such systems call

* This workshop has been supported by the MegaM@Rt2 project. MegaM@Rt2 has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No. 737494. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation program and from Sweden, France, Spain, Italy, Finland & Czech Republic.

for modern practices, such as Model-Driven Engineering (MDE), to tackle advances in productivity and quality of these now Cyber-Physical Systems (CPSs) [4]. However, the proposed solutions need to be further developed to scale up for real-life industrial projects and to provide significant benefits at execution time. To this intent, one of the major challenges is to work on achieving a more efficient integration between the design and runtime aspects of the concerned systems: the system behavior at runtime has to be better matched with the original system design in order to be able to understand critical situations that may occur, as well as corresponding potential failures in design. Methods and tools already exist for monitoring system execution and performing measurements of runtime properties. However, many of them do not rely on models and, usually, do not allow a relevant integration with (and/or a traceability back to) design models. Such a feedback loop from runtime is highly relevant at design time, the most suitable level for system engineers to analyze and take impactful decisions accordingly. It might also be useful to let the final users have some sort of control and manipulation possibilities over elements they would not be able to access otherwise. This last benefit implies that the models at runtime might be quite different from those at design time, especially in terms of programming/engineering background.

MDE@DeRun 2018[8] has been planned as a meeting point where both researchers and practitioners on model-driven and model-based techniques and architectures for complex systems can share their experiences and thoughts on this area of work. Its main goal was to disseminate and exchange related ideas or challenges, identify current and future key issues as well as explore possible solutions. The potentially relevant topics include notably (but not only):

- Model-based techniques, methods and tools allowing any interaction between design time and runtime, possibly resulting from heterogeneous engineering practices.
- Model-based techniques, methods and tools for inferring design deviations and identifying affected elements over a possibly large spectrum of runtime system configurations or conditions.
- Methods and techniques allowing to practically integrate, possibly in different ways, any feedback collected at runtime into design level models.
- Integrated model-based methods and techniques for runtime analysis and design artifacts input collection, e.g. based on probes injection to some runtime artifacts.
- Validation and verification mechanisms for linking results of runtime analysis, e.g. from execution traces, with design models expressing systems' both functional and non-functional requirements.
- (Industrial) case studies, experience reports, literature reviews or visionary positions related to any of the previously mentioned topics.

The remainder of this paper is structured as follows. Section 2 briefly introduces the different papers accepted and presented during the workshop. Possible future

[8] https://megamart2-ecsel.eu/mde-derun-2018/

challenges on design/runtime interactions in the MDE context are then discussed in Section 3, before we finally conclude this paper in Section 4.

2 Contribution Summary

In what follows, we list the 5 papers (4 short papers and 1 long paper) that have finally been accepted and presented during the workshop. A short summary is provided for each one of them.

Aliya Hussain, Saurabh Tiwari, Jagadish Suryadevara and Eduard Enoiu: ***From Modeling to Test Case Generation in the Industrial Embedded System Domain*** — This short paper presents an on-going investigation being carried out at Volvo CE[9] to improve testing processes by using a Model-based testing (MBT) approach. The goal has been to investigate the use of MBT and the evidence on how modeling and test generation can improve the current way of manually creating test cases based on natural language requirements. The authors used the Conformiq Creator tool to model the behavior and structure of a function controlling the accelerator pedal provided by Volvo CE. The authors automatically created test cases covering the model, and compare these test cases in terms of test goal coverage and number of test cases to assess the applicability of MBT in this context. The approach has shown encouraging results.

Saurabh Tiwari, Emina Smajlovic, Amina Krekic and Jagadish Suryadevara: ***A System Modeling Approach to Enhance Functional and Software Development*** — This short paper presents a SysML-based modeling approach to enhance functional and software development process within Volvo CE. The increased complexity of embedded software demands for new development methodologies to address flexible functional development, enhance communication among development teams, and maintain traceability from design concepts to software artifacts. The discussed approach has been experimented in the context of developing a new transmission system (partially electrified) and its features. While the underlying modeling approach is still work-in-progress, both initial success and existing gaps have been highlighted.

Daniel Zimmermann: ***Automated Consistency Preservation in Electronics Development of Cyber-Physical System*** — This short paper presents an automated strategy to ensure consistency between two widely used categories of software tools in electrical engineering: an Electronic Design Automation application (EDA) for designing Printed Circuit Boards (PCBs) and an electronic circuit simulator tool to predict system behavior at runtime. Coupling these two types of tools provides the developers with the ability of efficiently testing and optimizing the behavior of the electric circuit during the PCB design process; to avoid the disadvantages of ambiguous heuristic matching methods, a strategy ensuring a reliable assignment

[9] Volvo Construction Equipment AB, Sweden

of these model elements is needed. The approach has been implemented by using Eagle CAD as the PCB software and Matlab/Simulink with the Simscape extension as the simulation tool.

Valentin Besnard, Matthias Brun, Frédéric Jouault, Ciprian Teodorov and Philippe Dhaussy: ***Embedded UML Model Execution to Bridge the Gap Between Design and Runtime*** — This long paper proposes a solution to bridge the gap between design and runtime aspects in model-based software development. In fact, with classical model-driven development techniques, developers start by building design models before producing actual code. Although various approaches can be used to validate models and code separately, models and code are however separated by a semantic gap. This gap typically makes it hard to link runtime measures (e.g., execution traces) to design models. The approach presented in this paper avoids this semantic gap by making it possible to execute UML design models directly on embedded microcontrollers. Therefore, any runtime measure is directly expressed in terms of the design model.

Andreas Kästner, Martin Gogolla, Khanh-Hoang Doan and Nisha Desai: ***Sketching a Model-Based Technique for Integrated Design and RunTime Description*** — This short paper sketches a UML- and OCL-based technique for the coherent description of design time and runtime aspects of models. The basic idea is to connect a design model and a runtime model with a correspondence model. The authors show two simple examples, one for structural modeling and one for behavioral modeling, that introduce the underlying principles. As all three models are formulated in the same languages—UML and OCL—one can reason about the single models and their relationships in a comprehensive way.

3 Discussion: Challenges and Roadmap

Although many contributions could be achieved in the last decade in the MDE community, there are still several open challenges towards a complete and relevant integration between runtime and design aspects in complex systems. Firstly, explicit correspondences and/or traceability links are needed between runtime and design models. Secondly, a better understanding of the nature of the available runtime information (and its possible impacts on the design information) is required. Thirdly, the objectives and benefits of leveraging such correspondences and information need to be defined. We foresee a set of challenges that can be used as a research roadmap. They are introduced in what follows.

Correspondences/traceability between runtime and design models — The aim to match the system behavior at runtime with the original system design can be achieved in several ways. This is mainly related to the concept of traceability. As widely treated in the literature, traceability relationships may help designers to understand the associations and dependencies that exist among heterogeneous models and their correspondences [5, 6].

In MDE, a *trace link* is a relationship between one or more source model elements and one or more target model elements, whereas a *trace model* is a structured set of trace links, e.g., between source and target models. Trace links may be defined between entire artifacts (e.g., a requirements document and a design model) or between parts of artifacts.
The correspondence between runtime and design models might also take advantage of the MDE capabilities. For instance, in the case of (automated) model transformations, the traceability links are not only obvious but also allow some syntactic adaptation (e.g., different levels of abstraction) as well as some semantic adaptation (e.g., different viewpoints) on the way.
In order to integrate runtime and design aspects of the system several aspects need to be considered.

1. *Types of correspondences* — Correspondences between models could be defined through the following means: *(a)* traceability link, *(b)* consistency specification, *(c)* (bidirectional) model transformation, *(d)* model viewpoints and views. *(e)* megamodeling.
2. *Approaches* — Correspondences between models can be defined by means of the following approaches: *(a)* by integrating correspondences inside models, that implies a modification of the original models, or *(b)* by defining external correspondences between models, in this case the consistency of the original models is preserved (no modifications).
3. *How correspondences are produced* — Correspondences can be defined both in a manual manner, requiring engineers and domain experts, or automatically, starting from executable correspondence specifications. There can also exist mixed approaches where correspondences are automatically initiated/proposed and refined manually.
4. *When correspondences are produced* — Correspondences can be produced *(a)* at design-time (e.g., when creating the design model), between design-time and runtime phases (e.g., by applying some processes/transformations on the design model), *(b)* at system initialization (e.g., by creating all traceability links), or *(c)* on the fly at runtime (e.g., by creating a new trace link for each new runtime object created/used).

Runtime information — Runtime information can be considered as any software, architectural information or model of the runtime system that can be obtained during the system execution. For instance, through observation and instrumentation, logs and metrics (that can be also considered as kinds of runtime traces), runtime information can be collected to enable comprehension of the inner workings of already deployed software system [3].
Such models containing runtime information should not be confused with models@run.time [1] that, in general, aims at applying model-driven techniques for adapting and evolving software behavior while it is executing. On the contrary, we are interested in exploiting information collected only at runtime. This information can then be used offline to improve the initial system design through trial and error, eventually with the help of verification and validation tools (for instance).
In the following, we describe several aspects we believe important to consider.

1. *Types of runtime information* — Runtime information can be of different types, such as simulation models, executable models, model representing logs/traces, model representing states or configurations of the system, models expressing dynamic information or runtime measures on design models, test models.
2. *How they are obtained* — Runtime information can be collected by means of various mechanisms, such as simulation, monitoring, execution, debugging, profiling, verification.
3. *How they are represented* — Runtime information can be represented by: *(a)* specific models representing runtime information (i.e., using a common and/or a general metamodel); or *(b)* measures that are directly expressed in terms of the design model.
4. *How are they visualized* — Runtime information can be visualized over sequence diagrams, graphical diagrams of the design model (e.g., with particular tools like Papyrus), state-space graphs, or various textual representations (using some DSLs). These models give either a snapshot of the system execution, a representation of the current execution trace, or a representation of the whole execution history (i.e., a part of the system state-space corresponding to all explored execution traces).
5. *Who uses runtime information* — Runtime information should take into account the users; e.g., end-users, architects, designers, developers of the system, and also "test engineers" in charge of verifying and validating the system. This will have a strong impact on the type of chosen runtime models.
6. *Viewpoints* — A same runtime information can take on different roles depending on the context/perspective from which it is analyzed (e.g., business, system, technology). In the same vein, some software artifacts (or parts of them) can be considered as design time or runtime ones depending on the specific viewpoint from which they are observed.

Objectives — The vision underlying the integration of design and runtime models is to create awareness of problems in design or critical situations that may occur. The understanding of this class of problems can be exploited for different purposes.

1. *Using/Analyzing correspondences* — Correspondences (i.e., traceability relationships) between elements in models can be exploited to perform operations on models. Some of the key operations are: *(a) match*, that takes two models and returns a mapping between them; *(b) compose*, that composes a pair of correspondences; *(c) merge*, that uses correspondences between two models to create a new model that is the merge of them; and *(d) set operations* on models, such as *union*, *intersection*, *difference*. Such correspondences can also be used to build views combining together several models that can possibly conform to different metamodels. This can be realized according to corresponding viewpoints specifying the nature/type of these correspondences at metamodel-level.

Furthermore, correspondences can be used to feed both functional (e.g., consistency, requirement traceability) and non-functional analysis (e.g., performance, reliability, availability, security).
2. *Inference capabilities* — Correspondence between design and runtime information can be used to achieve inference capabilities, discovering the system properties deviations and affected design components based on trace analysis. For instance, inference methods offer a control loop across the whole design chain between runtime and design time of the system, including non-functional aspects. This way, additional information from runtime models can be used to enhance system/design models.
3. *Requirements* — Correspondences can be used to reconcile the requirements and the systems runtime behavior in case of system deviations from the initial requirement specification.
4. *Reverse engineering* — Going backwards through the development cycle, correspondences can be used in reverse engineering guiding the specification of the system design from the runtime behavior.

4 Conclusion

Achieving an efficient integration between the design and runtime aspects of complex systems proves to be a relevant challenge for MDE methods and tools. The International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems (MDE@DeRun 2018) aims at providing a place for the community to share ideas and results in this research area we believe important. This paper summarized the main objectives and contributions of this first edition. Furthermore, it discussed and proposed some first directions for further research in this area, which we plan to explore in the future in our respective works.

Acknowledgements

We would like to thank everyone who took part in the success of this first edition of the workshop, including the program committee members, the paper authors and everyone who attended the workshop or took part in the interesting discussions we had.

References

1. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10), 22–27 (Oct 2009). https://doi.org/10.1109/MC.2009.326
2. Boccara, N.: Modeling Complex Systems. Graduate Texts in Comtemporary Physics, Springer (2004)
3. Cito, J., Leitner, P., Bosshard, C., Knecht, M., Mazlami, G., Gall, H.C.: Performancehat: augmenting source code with runtime performance traces in the IDE. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018. pp. 41–44 (2018)

4. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling cyberphysical systems. In: Proceedings of the IEEE. pp. 13–28 (2012)
5. Paige, R.F., Drivalos, N., Kolovos, D.S., Fernandes, K.J., Power, C., Olsen, G.K., Zschaler, S.: Rigorous identification and encoding of trace-links in model-driven engineering. Software and System Modeling **10**(4), 469–487 (2011)
6. Winkler, S., Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development **9**(4), 529–565