International Journal on Software Tools for Technology Transfer (2021) 23:679–684
https://doi.org/10.1007/s10009-021-00632-0

GENERAL

Special Issue: MeTRID

On methods and tools for rigorous system design

Simon Bliudze[1] · Panagiotis Katsaros[2] · Saddek Bensalem[3] · Martin Wirsing[4]

Accepted: 8 June 2021 / Published online: 22 June 2021

Abstract
Full a posteriori verification of the correctness of modern software systems is practically infeasible due to the sheer complexity resulting from their intrinsic concurrent nature. An alternative approach consists of ensuring *correctness by construction*. We discuss the *Rigorous System Design* (RSD) approach, which relies on a sequence of semantics-preserving transformations to obtain an implementation of the system from a high-level model while preserving all the properties established along the way. In particular, we highlight some of the key requirements for the feasibility of such an approach, namely availability of (1) methods and tools for the design of correct-by-construction high-level models and (2) definition and proof of the validity of suitable domain-specific abstractions. We summarise the results of the extended versions of seven papers selected among those presented at the 1st and the 2nd International Workshops on Methods and Tools for Rigorous System Design (MeTRiD 2018–2019), indicating how they contribute to the advancement of the RSD approach.

Keywords System design · High-level modelling · Correct-by-construction · Domain-specific abstraction

Rigorous System Design

Modern software systems are inherently concurrent. They consist of components running simultaneously and sharing access to resources provided by the execution platform. For instance, embedded control software in various domains, ranging from household robotics through operation of smart power-grids to on-board satellite software, commonly comprises, in addition to components responsible for taking the control decisions, a set of components driving the operation of sensing and actuation devices. These components interact through buses, shared memories and message buffers, leading to resource contention and potential deadlocks compromising mission- and safety-critical operations. Similar problems are observed in various kinds of software, including system, work-flow management, integration software, web services, etc. Essentially, any software entity that goes beyond simply computing a certain function necessarily has to interact and share resources with other such entities.

The intrinsic concurrent nature of such interactions is the root cause of the sheer complexity of the resulting software. Indeed, in order to analyse the behaviour of such a software system, one has to consider all possible interleavings of the operations executed by its components. Thus, the complexity of software systems is exponential in the number of their components, making a posteriori verification of their correctness practically infeasible. An alternative approach consists of ensuring correctness by construction, through the application of well-defined design principles [4,20], imposing behavioural contracts on individual components [7] or by applying automatic transformations to obtain executable code from formally defined high-level models [33].

✉ Martin Wirsing
wirsing@ifi.lmu.de

Simon Bliudze
simon.bliudze@inria.fr

Panagiotis Katsaros
katsaros@csd.auth.gr

Saddek Bensalem
Saddek.Bensalem@univ-grenoble-alpes.fr

1 University of Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, 59000 Lille, France

2 School of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

3 Université Grenoble Alpes, CNRS, Grenoble INP, Verimag, 38000 Grenoble, France

4 Ludwig-Maximilians-Universität München, Munich, Germany

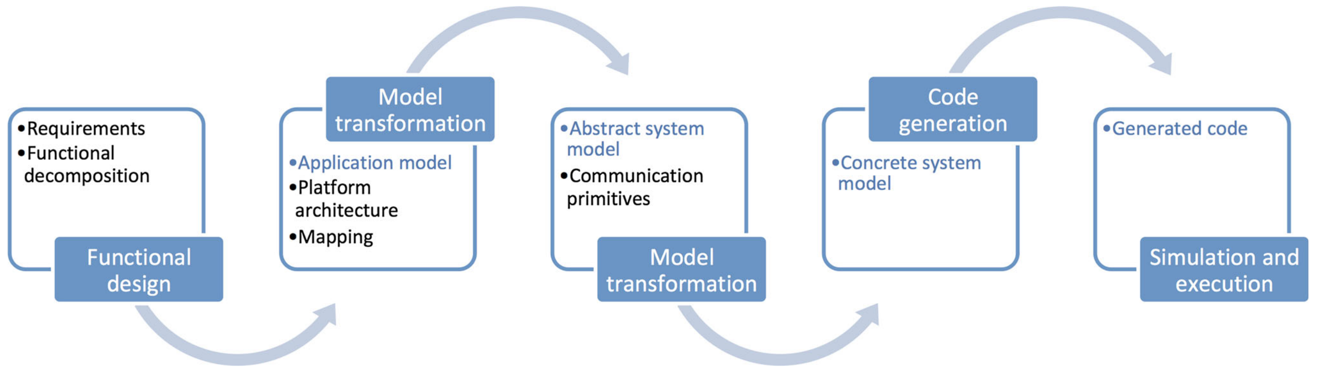

Fig. 1 A simplified example of the RSD flow instantiation (the blue items are the result of the previous stage; the black ones are provided as new input at the current stage)

The Rigorous System Design (RSD) [33] approach enforces multiple levels of separation of concerns. It relies on a sequence of semantics-preserving transformations to obtain an implementation of the system from a high-level model while preserving all the properties established along the way.

Figure 1 illustrates a simplified instantiation of the RSD flow. One starts by designing the *application model*. The application model is verified to prove the elementary properties that are not assured by construction, such as absence of local deadlock, and satisfaction of basic requirements. These elementary properties, serve as a basis for the proof of global properties, obtained by construction.

The application model is then extended with additional components modelling the target platform to obtain the *system model*, which is used to perform platform specific analyses and the optimisation of performance through the exploration of the design space (memories, buses, mapping of software components to hardware elements, etc.).

Finally, the model is enriched with platform specific information (e.g.communication primitives) and, after removing components modelling hardware elements, executable code is automatically generated.

Proving that the assumptions made at the modelling level to justify the separation of concerns hold, indeed, at the platform level, guarantees that all the properties established throughout the design process also hold for the generated code.

Thus, the RSD approach applies—on the higher abstraction level of the system design process—the same principles as those provided by standard compilers for the generation of machine-executable code from programs written in languages such as Java or C++. It consists in decomposing the argument justifying the correctness of the entire process into several independent arguments justifying the correctness of individual transformations. Furthermore, it provides flexibility w.r.t.the target platforms by postponing design choices as far as possible and allowing for different transformations of the same model to be applied at every design stage. However, in drawing this parallel, it is appropriate to differentiate between commonly used compilers such as `gcc`, where the public trust originates mainly from the extensive usage experience, and verified compilers such as `CompCert` [29], where preservation of semantics at the various stages is formally verified. Although the second scenario is currently preferable for RSD tool sets due to lack of usage history comparable to that of compilers such as `gcc`, both can be relevant in practice.

Applications. Although the RSD approach as formulated by Sifakis [33] originates from the development of the BIP framework for the Embedded Software design (see, for example, [5,6]), it is applicable in a much broader variety of domains, whereof we will mention below just a few.

Key issues of applying the RSD approach to the Cyber-Physical Systems (CPSs), which comprise components with both discrete and continuous underlying dynamics, are discussed in [11]. The authors argue that the objective of cyber-physical system modelling is twofold: providing the means for (1) the validation of the system design through simulation of such models and (2) the generation of executable code for the discrete control sub-system. The key point here is that the two design artefacts, i.e.the simulator and the code of the control sub-system are obtained through *two branches of the design flow sharing a substantial prefix*. Thus, the generated control sub-system is equivalent to—i.e.satisfies the same properties as—the corresponding components of the simulator *by construction*.

Expanding on the above idea, it is clear that the RSD approach can be of great benefit for system design and operation involving the so-called Digital Twins where simulation becomes “a core functionality of systems by means of seamless assistance along the entire life cycle” [32].

Autonomous and (self-)adaptive systems constitute another significant domain for the application of the RSD approach. These include, for example, autonomous vehicles, Cloud and IoT applications. Indeed, such systems must react to

changing environmental constraints and user requirements. Therefore, they are characterised by high dynamicity both of their behaviour and their structure. In particular, this implies that many of the underlying verification problems are undecidable [12,19] emphasising the need for by-construction correctness provided by the RSD approach.
Key elements. The RSD approach relies on several fundamental elements, among which we highlight two. Firstly, *methods and tools for the design of correct-by-construction high-level models* are necessary to initiate the process. Although it is difficult to imagine a unique approach that would fit all the various application domains, it seems reasonable to expect that these approaches will share a common core, comprising, at the very least, some form of

(1) Requirement elicitation and formalisation and
(2) (semi-)automatic synthesis of parts of the models in order to discharge these requirements [34].

Secondly, *defining and proving the validity of suitable domain-specific abstractions*, such as architectures, protocols or design patterns (e.g.[1,30,31]), is key for implementing an RSD flow. Such abstractions are used to facilitate the design of correct-by-construction high-level models. The transformations along the RSD flow can then rely on platform-specific implementations of these abstractions to automatically generate refined models or executable code. Although these platform-specific implementations would have to be proven correct, such proofs need only be carried out once, reducing the overall complexity of the process and ensuring the trustworthiness of the resulting artefacts.

This issue

This special issue contains the extended versions of selected papers presented at the 1st and the 2nd International Workshops on Methods and Tools for Rigorous System Design (MeTRiD 2018–2019) held as satellite events of the corresponding European Joint Conferences on Theory and Practice of Software (ETAPS 2018–2019).

The goal of the MeTRiD workshop is to promote cross-fertilisation between research in academia and practical applications in the industry. On the one hand, we hope that, through the publication of research and tool papers, the workshop will contribute to raising awareness of the methods and tools available among the industrial players. On the other hand, presentation and exchange of realistic case studies should allow academic researchers to better fit their tools to industrial needs, thereby improving the dissemination of results.

The first, 2018 edition was a traditional workshop with peer-reviewed proceedings published as volume 272 of the Electronic Proceedings in Theoretical Computer Science (EPTCS) [10]. MeTRiD 2018 accepted three categories of papers: regular, tool and case-study papers. MeTRiD 2019 was by invitation with presentations of, both, already published results and ongoing work, and no proceedings.

This issue comprises 7 papers addressing the key elements outlined above. Papers [3,18] contribute to the design of correct-by construction high-level models by defining a high-level modelling formalism [18] and by providing an approach for debugging CPS models [3]. Papers [9,14,22,24,28] contribute to the design and proof of domain-specific abstractions. They provide techniques for ensuring the correctness of randomised consensus protocols [9], program block parallelisation [14], usage control policies [22], and for ensuring optimality of partition schedules [24] and energy consumption [28].

Each of these papers was reviewed by at least three reviewers and extends one of the papers presented at MeTRiD 2018 or 2019.

– The paper *"Specifying and verifying usage control models and policies in TLA$^+$"* by Grompanopoulos, Gouglidis, and Mavridou [22] is an extension of the MeTRiD 2018 paper by the same authors [21]. It presents a case study on specification and model checking of usage control models and policies. The focus of the paper is on demonstrating how to express control policies in TLA$^+$. By doing so the authors also introduce their own policy model called UseCON.
– The paper *"Programming dynamic reconfigurable systems"* by El Ballouli, Bensalem, Bozga, and Sifakis [18] is an extension of the ISoLA 2018 and FACS 2018 papers [16,17] by the same authors. The paper focuses on programming dynamic reconfigurable systems. It presents an extension of Behaviour-Interaction-Priority (BIP) framework called Dynamic-Reconfigurable BIP (DB-RIP), which allows dealing with reconfigurable systems including different types of dynamism. The technical contribution of the paper consists of the formal definition of DR-BIP, i.e. its syntax and semantics. In addition, the paper describes a prototype implementation of the language in Java and illustrates the application of the approach to model 3 different examples from different domains.
– The paper *"Model-based optimization of ARINC-653 partition scheduling"* by Han, Zhai, Nielsen, and Nyman [24] is an extension of the MeTRiD 2018 paper [23], by the same authors. This work introduces a framework for generating optimal ARINC-653 partitioned schedules, i.e.system partitions scheduled in mutual exclusion in order ensure temporal isolation of applications in safety-critical systems. The main challenge in synthe-

sising optimal period and budget parameters for the partitions is the scalability problem, since the parameter space for larger systems rapidly grows in size and cannot be exhaustively explored. The proposed framework excludes, as early as possible, non-schedulable combinations of parameters by applying (in order of sequence) global schedulability tests, statistical model-checking and model-checking in Uppaal. Finally, an evolutionary algorithm for parameter exploration is applied that generalises the yes/no verdict of the schedulability question into a numeric fitness evaluation.
– The paper *"Correct Program Parallelisations"* by Blom, Darabi, Huisman, and Safari [14] is an extension of the FASE 2015 and NFM 2017 [13,15] papers by the first three authors. This paper presents a verification technique to reason about the correctness of compiler directives indicating which program blocks may potentially be executed in parallel without changing the behaviour of the program. To this end, the authors introduce an intermediate language for representing such programs, its formal semantics and a soundness proof for deductive proof rules that build on that semantics. The paper also discusses how OpenMP programs can be translated into the intermediate language and how the verification methodology can be implemented on top of the VerCors/Viper verification tool infrastructure.
– The paper *"Energy characterization of IoT systems through design aspect monitoring"* by Lekidis, and Katsaros [28] is an extension of the MeTRiD 2018 paper [27] and a related presentation in MeTRiD 2019, by the same authors. This work proposes a model-driven approach for the energy cost estimation of Internet of Things design aspects (availability, reliability, dynamicity and security), by monitoring them. This paves the way to identify feasible architecture solutions that satisfy energy footprint requirements.
– The paper *"CPSDebug: Automatic failure explanation in CPS models"* by Bartocci, Manjunath, Mariani, Mateis and Ničković [3] is an extension of the SEFM 2019 paper [2] by the same authors. In particular, the authors introduce an approach for the detection of faults and their localisation in Stateflow/Simulink models of cyber-physical systems. This is challenging, when there are mixed discrete and continuous signals, while the faults may not physically manifest immediately. The method rests on simulation traces that satisfy or violate desired properties expressed in Signal Temporal Logic. Those traces that satisfy the properties are then used to mine additional specifications, and the method produces explanations from the analysis of failed traces with respect to the properties mined.
– The paper *"Verification of Randomized Consensus Algorithms under Round-Rigid Adversaries"* by Bertrand, Konnov, Lazić, and Widder [9] is an extension of paper [8] by the same authors. This work is focused on obtaining a fully automated proof of correctness—encompassing validity, agreement and almost-sure termination—of randomised consensus algorithms involving arbitrarily many (faulty) processes and rounds under round-rigid adversaries, i.e.adversaries that are weakly fair and that select actions in a "round-based" manner. The approach is based on the threshold automata formalism introduced in [25,26].

We would like to thank all the authors of these papers for their contributions and the reviewers that we have solicited for their thorough evaluations. Furthermore, we would like to acknowledge the help of the OCS team and, particularly, Markus Frohme, whose help and reactivity throughout the preparation of the issue were crucial to its successful realisation.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Attie, P., Baranov, E., Bliudze, S., Jaber, M., Sifakis, J.: A general framework for architecture composability. Formal Aspects Comput. **18**(2), 207–231 (2016). https://doi.org/10.1007/s00165-015-0349-8
2. Bartocci, E., Manjunath, N., Mariani, L., Mateis, C., Ničković, D.: Automatic failure explanation in CPS models. In: P.C. Ölveczky, G. Salaün (eds.) Proceedings of the 17th International Conference Software Engineering and Formal Methods (SEFM 2019), *Lecture Notes in Computer Science*, vol. 11724, pp. 69–86. Springer (2019). https://doi.org/10.1007/978-3-030-30446-1_4
3. Bartocci, E., Manjunath, N., Mariani, L., Mateis, C., Nickovic, D.: CPSDebug: Automatic failure explanation in CPS models. Int. J. Software Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-020-00599-4
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 3rd edn. SEI Series in Software Engineering. Addison-Wesley Professional (2012)
5. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: 4^{th} IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM06), pp. 3–12 (2006). https://doi.org/10.1109/SEFM.2006.27. Invited talk

6. Basu, A., Gallien, M., Lesire, C., Nguyen, T.H., Bensalem, S., Ingrand, F., Sifakis, J.: Incremental component-based construction and verification of a robotic system. In: ECAI, pp. 631–635 (2008)
7. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T., Larsen, K.G.: Contracts for system design. Research Report RR-8147, INRIA (2012). https://hal.inria.fr/hal-00757488
8. Bertrand, N., Konnov, I., Lazic, M., Widder, J.: Verification of randomized consensus algorithms under round-rigid adversaries. In: W.J. Fokkink, R. van Glabbeek (eds.) 30th International Conference on Concurrency Theory, (CONCUR 2019), *LIPIcs*, vol. 140, pp. 33:1–33:15. Schloss Dagstuhl — Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.33
9. Bertrand, N., Konnov, I., Lazic, M., Widder, J.: Verification of randomized consensus algorithms under round-rigid adversaries. Int. J. Software Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-020-00603-x
10. Bliudze, S., Bensalem, S. (eds.): Proceedings of the 1st international workshop on methods and tools for rigorous system design, MeTRiD@ETAPS 2018, *EPTCS*, vol. 272. Thessaloniki, Greece (2018). https://doi.org/10.4204/EPTCS.272
11. Bliudze, S., Furic, S., Sifakis, J., Viel, A.: Rigorous design of cyber-physical systems: linking physicality and computation. Int. J. Software Syst. Model. **18**(3), 1613–1636 (2019). https://doi.org/10.1007/s10270-017-0642-5
12. Bloem, R., Jacobs, S., Khalimov, A., Konnov, I., Rubin, S., Veith, H., Widder, J.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool, California (2015)
13. Blom, S., Darabi, S., Huisman, M.: Verification of loop parallelisations. In: A. Egyed, I. Schaefer (eds.) Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering (FASE 2015), *Lecture Notes in Computer Science*, vol. 9033, pp. 202–217. Springer (2015). https://doi.org/10.1007/978-3-662-46675-9_14
14. Blom, S., Darabi, S., Huisman, M., Safari, M.: Correct program parallelisations. Int. J. Software Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-020-00601-z
15. Darabi, S., Blom, S.C.C., Huisman, M.: A verification technique for deterministic parallel programs. In: C.W. Barrett, M. Davies, T. Kahsai (eds.) Proceedings of the 9th NASA Formal Methods International Symposium (NFM 2017), *Lecture Notes in Computer Science*, vol. 10227, pp. 247–264. Springer (2017). https://doi.org/10.1007/978-3-319-57288-8_17
16. El Ballouli, R., Bensalem, S., Bozga, M., Sifakis, J.: Four exercises in programming dynamic reconfigurable systems: Methodology and solution in DR-BIP. In: T. Margaria, B. Steffen (eds.) Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems (ISoLA 2018), *Lecture Notes in Computer Science*, vol. 11246, pp. 304–320. Springer (2018). https://doi.org/10.1007/978-3-030-03424-5_20
17. El Ballouli, R., Bensalem, S., Bozga, M., Sifakis, J.: Programming dynamic reconfigurable systems. In: K. Bae, P.C. Ölveczky (eds.) Proceedings of the 15th International Conference Formal Aspects of Component Software (FACS 2018), *Lecture Notes in Computer Science*, vol. 11222, pp. 118–136. Springer (2018). https://doi.org/10.1007/978-3-030-02146-7_6
18. El Ballouli, R., Bensalem, S., Bozga, M., Sifakis, J.: Programming dynamic reconfigurable systems. Int. J. Software Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-020-00596-7
19. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '95, pp. 85–94. Association for Computing Machinery, New York, NY, USA (1995). https://doi.org/10.1145/199448.199468
20. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, 1st edn. Addison-Wesley Professional, Boston (1994)
21. Gouglidis, A., Grompanopoulos, C., Mavridou, A.: Formal verification of usage control models: a case study of UseCON using TLA+. In: S. Bliudze, S. Bensalem (eds.) Proceedings of the 1st International Workshop on Methods and Tools for Rigorous System Design (MeTRiD), *Electronic Proceedings in Theoretical Computer Science*, vol. 272, pp. 52–64. Open Publishing Association (2018). https://doi.org/10.4204/EPTCS.272.5
22. Grompanopoulos, C., Gouglidis, A., Mavridou, A.: Specifying and verifying usage control models and policies in TLA$^+$. Int. J. Software Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-020-00600-0
23. Han, P., Zhai, Z., Nielsen, B., Nyman, U.: A compositional approach for schedulability analysis of distributed avionics systems. In: S. Bliudze, S. Bensalem (eds.) Proceedings of the 1st International Workshop on Methods and Tools for Rigorous System Design (MeTRiD), *Electronic Proceedings in Theoretical Computer Science*, vol. 272, pp. 39–51. Open Publishing Association (2018). https://doi.org/10.4204/eptcs.272.4
24. Han, P., Zhai, Z., Nielsen, B., Nyman, U.: Model-based optimization of ARINC-653 partition scheduling. Int. J. Software Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-020-00597-6
25. Konnov, I., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In: P. Baldan, D. Gorla (eds.) Proceedings of the 25th International Conference on Concurrency Theory (CONCUR 2014), *Lecture Notes in Computer Science*, vol. 8704, pp. 125–140. Springer (2014). https://doi.org/10.1007/978-3-662-44584-6_10
26. Konnov, I.V., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: reachability. Inf. Comput. **252**, 95–109 (2017). https://doi.org/10.1016/j.ic.2016.03.006
27. Lekidis, A., Katsaros, P.: Model-based design of energy-efficient applications for iot systems. In: S. Bliudze, S. Bensalem (eds.) Proceedings of the 1st International Workshop on Methods and Tools for Rigorous System Design (MeTRiD), *Electronic Proceedings in Theoretical Computer Science*, vol. 272, pp. 24–38. Open Publishing Association (2018). https://doi.org/10.4204/eptcs.272.3
28. Lekidis, A., Katsaros, P.: Energy characterization of IoT systems through design aspect monitoring. Int. J. Software Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-020-00598-5
29. Leroy, X., Blazy, S., Kästner, D., Schommer, B., Pister, M., Ferdinand, C.: CompCert – A formally verified optimizing compiler. In: ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress. SEE, Toulouse, France (2016). https://hal.inria.fr/hal-01238879
30. Marmsoler, D.: Hierarchical specification and verification of architectural design patterns. In: A. Russo, A. Schürr (eds.) Proceedings of the 21st International Conference Fundamental Approaches to Software Engineering (FASE 2018), *Lecture Notes in Computer Science*, vol. 10802, pp. 149–168. Springer (2018). https://doi.org/10.1007/978-3-319-89363-1_9
31. Mavridou, A., Stachtiari, E., Bliudze, S., Ivanov, A., Katsaros, P., Sifakis, J.: Architecture-based design: a satellite on-board software case study. In: 13th International Conference on Formal Aspects of Component Software (FACS 2016), *Lecture Notes in Computer Science*, vol. 10231, pp. 260–279 (2016). https://doi.org/10.1007/978-3-319-57666-4_16
32. Rosen, R., von Wichert, G., Lo, G., Bettenhausen, K.D.: About the importance of autonomy and digital twins for the future of manufacturing. IFAC-PapersOnLine **48**(3), 567–572 (2015). https://doi.org/10.1016/j.ifacol.2015.06.141. 15th IFAC Symposium on Information Control Problems in Manufacturing

33. Sifakis, J. Rigorous system design. Foundations and Trends® in Electronic Design Automation **6**(4), 293–362 (2012) . https://doi.org/10.1561/1000000034
34. Stachtiari, E., Mavridou, A., Katsaros, P., Bliudze, S., Sifakis, J.: Early validation of system requirements and design through correctness-by-construction. J. Syst. Software **145**, 52–78 (2018). https://doi.org/10.1016/j.jss.2018.07.053