ASMETA: A Comprehensive Tool Set for Formal System Engineering Based on Abstract State Machines

Andrea Bombarda[a], Silvia Bonfanti[a], Angelo Gargantini[a], Elvinia Riccobene[b], Patrizia Scandurra[a]

[a] *University of Bergamo, Bergamo, Italy*
[b] *Università degli Studi di Milano, Milano, Italy*

Abstract

ASMETA is a tool set for the formal specification, validation, and verification of discrete event systems. It is based on the Abstract State Machine (ASM) formal method, and provides an interactive environment for editing and analyzing ASM models, as well as for transforming high-level models into code. ASMETA also integrates runtime validation and verification techniques to ensure runtime assurance and enforcement of systems. This paper provides a comprehensive overview of ASMETA, detailing its software architecture, capabilities, and illustrative applications. It concludes with a discussion on its impact and emerging research directions in cutting-edge domains, including runtime safety assurance, safeguarding of autonomous systems, and the use of formal models in next-generation Digital Twins.

Keywords:
Abstract State Machines, State-based formal models, Model validation, Model verification, Code generation from models

1. Motivation and metadata

ASMETA is a tool set for the formal specification and model analysis of computational systems at discrete events. Based on Abstract State Machines

Email addresses: andrea.bombarda@unibg.it (Andrea Bombarda), silvia.bonfanti@unibg.it (Silvia Bonfanti), angelo.gargantini@unibg.it (Angelo Gargantini), elvinia.riccobene@unimi.it (Elvinia Riccobene), patrizia.scandurra@unibg.it (Patrizia Scandurra)

Nr	Code metadata description	
C1	Current code version	25.09
C2	Permanent link to code/repository used for this code version	`https://github.com/asmeta/asmeta`
C3	Permanent link to reproducible capsule	N/A
C4	Legal code license	Eclipse Public License version 2.0 (EPL-2.0)
C5	Code versioning system used	GitHub
C6	Software code languages, tools and services used	Java, Maven, Eclipse, CD/CI: `https://gitlab.com/garganti/asmeta`
C7	Compilation requirements, operating environments, and dependencies	Java and Eclipse with Eclipse Plug-in Development Environment, XText SDK, and GEF Classic Zest SDK.
C8	If available, link to developer documentation/manual	`https://asmeta.github.io/`
C9	Support email for questions	Contact the paper authors

Table 1: Code metadata

(ASMs) [1, 2, 3], it enables the application of formal methods to software and systems engineering.

Its development was motivated to address the shortage of tools available to support the practical application of the ASMs. Indeed, although this formal approach had already proven valuable for the specification and verification of diverse software systems across multiple domains (see the *survey of ASM research* in [1]), the absence of dedicated tool support was considered a significant limitation, fueling skepticism about its use by practitioners.

When the development of ASMETA began, our main objective was to create a textual notation for encoding ASM models and enabling their execution. Today, ASMETA has evolved into a comprehensive Eclipse-based tool set that provides an interactive environment for modeling systems iteratively and at the desired level of abstraction. It integrates a variety of tools that extend beyond the specification of a system's executable behavior, including support for property checking, scenario-based validation, prototype code generation, and model-based test case generation. In addition, runtime validation and verification techniques have been integrated into ASMETA to enable runtime assurance and enforcement of system safety requirements. As a result, ASMETA supports users throughout the entire assurance life-cycle of critical discrete-event systems, from design and development to deploy-

ment and operation [4]. Table 1 shows the relevant tool set metadata.

ASMETA has been used for specification and requirement analysis of different case studies in several application domains: a wide repository of examples is available online[1]. Specifically, ASMETA has been used in the context of *medical devices* (PillBox [5], hemodialysis device [6], amblyopia diagnosis [7], PHD Protocol [8], Mechanical Ventilator Milan [9]), *software control systems* (Landing Gear System [10], Automotive Software-Intensive Systems [11, 12], Hybrid European Rail Traffic Management System [13], AMAN Arrival Manager system for air traffic control [14]), *cloud-* [15] and *service*-based *systems* [16, 17], secure communication *protocols* [18], *smart contracts* in blockchain [19, 20, 21], *self-adaptive* systems [22, 23].

In many case studies [6, 9, 11, 13, 12], the analysis techniques that ASMETA supports have been used to discover faulty or erroneous requirements, as well as to guide towards requirements disambiguation and completeness. Moreover, the use of ASMETA has helped to guarantee safety and security assurance [12, 20], detect cyber security vulnerabilities [18, 21], and enforce system safety [9]. ASMETA has also been used for providing operational semantics of Domain Specific Languages (DSL) [23]. It enables the automatic transformation of DSL models into formal executable models, thereby opening up the possibility for their formal analysis, such as validation with respect to requirements and verification of properties.

To the best of our knowledge, although it is not the only tool set developed for ASM model editing and simulation [24, 25, 26], ASMETA is the only one currently actively supported, widely applied, and offering a broad range of model analysis techniques.

2. Software description

This section outlines the software architecture of the ASMETA tool set and provides an overview of its capabilities.

ASMETA is built on the Eclipse Modeling Framework (EMF), an Eclipse-based metamodeling framework[2] with code generation facility for building tools and other applications based on a structured data model (or metamodel). We took advantage of the entire EMF ecosystem and tooling to

[1]Repository `https://github.com/asmeta/asmeta/tree/master/asm_examples`

[2]EMF is the defacto reference implementation of the OMG standard EMOF(Essential Meta-Object Facility).

develop the `AsmM` metamodel and a set of tools [27, 28] covering model editing and validation. To enable a wide range of analysis activities over ASM models, ASMETA integrates several external tools, including model checkers for property verification, and SMT solvers for validating model refinements and supporting runtime verification. To achieve this, ASMETA primarily adopts a model transformation strategy based on *semantic mapping* [29, 28].

2.1. Software installation

ASMETA is distributed as a tool set packaged for Eclipse[3]. Users may decide to install ASMETA in their Eclipse installation through the ASMETA update site[4] or by directly downloading an Eclipse version with the ASMETA environment already set up from the official GitHub repository at `https://github.com/asmeta/asmeta`.

2.2. Architecture and tools overview

The ASMETA software architecture (see the component view shown in Figure 1) is organized into three *conceptual layers*, each responsible for a specific set of tools. Each layer builds upon the previous one, ensuring a structured and modular approach to ASM-based tool engineering. The `Modeling` package (*bottom layer*) provides infrastructural components for defining and editing models. The `Verification & Validation` (V&V) package (*middle layer*) provides components for formal analysis and validation of models. The *top layer* comprises the `Development` package with tools focusing on generating executable code artifacts and test scenarios, and the `Operation` package with tools for runtime monitoring and assurance. The tools are here outlined, grouped by package, to highlight their role within the architecture:

`Modeling` *package:*

- `AsmM`: The metamodel defining the ASMETA abstract syntax.
- `AsmM` language artifacts (derived from the `AsmM` metamodel): Java Interfaces representing APIs for the creation, storage, access and manipulation of an ASMETA model in terms of Java objects; an XMI (XML Metadata Interchange) interchange format for models; `AsmetaL` (ASMETA Language), a textual notation designed (concrete syntax) to effectively edit models conforming to the `AsmM` metamodel.

[3] https://eclipseide.org

[4] `https://raw.githubusercontent.com/asmeta/asmeta_update_site/master`

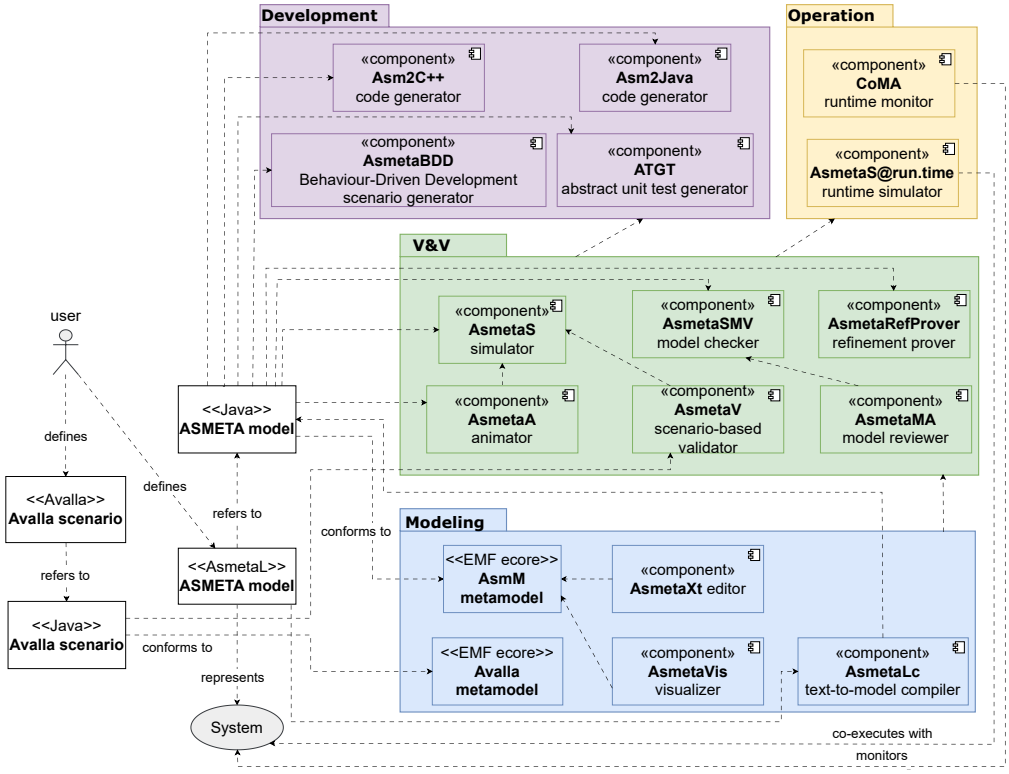

Figure 1: ASMETA software architecture.

• `AsmetaLc`: The text-to-model compiler that parses an `AsmetaL` model (the parser for the `AsmetaL` EBNF grammar is generated by the JavaCC parser generator), checks for its compliance w.r.t the OCL constraints defined over the `AsmM` metamodel, and creates an ASMETA model as an in-memory Java object representation using the `AsmM` Java Interfaces (i.e., an instance of the `AsmM` metamodel).

• `AsmetaXt` and `AsmetaVis`: `AsmetaXt` is an Eclipse-based IDE developed by using XText [30] for the editing of `AsmetaL` models, while `AsmetaVis` allows for their graphical visualization in terms of a navigable tree structure of control states.

`Verification & Validation (V&V)` *package:*

• `AsmetaS`: A simulator to execute ASMETA models (offline/design-time use) supporting random and interactive simulation. In random mode, `AsmetaS`

automatically assigns values to monitored functions that are consistent with their co-domains. In interactive mode, the user must manually provide valid values for the monitored functions.

• `AsmetaA`: An animator that offers a comprehensive, visually enhanced view of state locations and their evolution, helping users track model computation and state changes step by step. Like `AsmetaS`, `AsmetaA` supports both random and interactive modes. In interactive mode, input functions are entered via context-specific dialog boxes based on their type, while in random mode values are randomly assigned.

• `AsmetaV`: The model validator that allows users to perform scenario-based validation of ASMETA models. The user must specify a fragment of system interactions by defining a scenario in the `Avalla` notation. The scenario is then executed by `AsmetaV`. Each scenario loads the target ASMETA model and includes commands such as *set* to update input values, *step* and *step until* to simulate system reactions, and *check* to inspect properties in the current state.

• `AsmetaMA`: The model reviewer to analyze *quality* properties on the ASMETA model. It performs a static analysis and checks the presence of seven types of modeling errors by using suitable meta-properties specified in CTL and verified using the NuSmv [31] model checker.

• `AsmetaSMV`: It verifies temporal logic properties, expressed in LTL/CTL, on the ASMETA model. NuSmv [31] and NuXmv [32] are used as symbolic model-checking back-ends for LTL/CTL verification.

• `AsmRefProver`: The refinement prover that verifies correct refinements by means of an SMT Solver. It exploits the SMTLib [33] library.

`Development` *package:*

• `ATGT`: A test case generator from ASMETA models based upon the NuSmv [31] model checker.

• `AsmetaBDD`: It is a tool for the generation of *behavior-Driven Development* [34] scenarios written by exploiting the functionalities of the Catch2 framework, a C++ unit testing framework that provides a simple and expressive syntax for writing and organizing test cases.

• `Asm2C++` and `Asm2Java`: They provide facilities for generating executable C++ or Java code from a refined ASMETA model. In the case C++ is used, the generated code can be automatically adapted for its usage on Arduino.

`Operation` *package:*

• `CoMA`: It is a tool for runtime monitoring. It runs together with a Java program and assess whether the state of the Java implementation complies with that of the ASMETA specification.

• `AsmetaS@run.time`: It is a standalone tool for simulating at runtime an ASMETA model. It can be executed in tandem with a real system to use the ASMETA formal model as an enforcer or guard-rail. It supports (via CLI or via a REST API) simulation as-a-service features of the `AsmetaS` simulator, and additional features such as model execution with timeout and model roll-back to the previous safe state after a failure occurrence (e.g., invariant violations, inconsistent updates, ill-formed inputs, etc.) during model execution.

Tools of `Development` and `Operation` packages, and `AsmRefProver` are not available within Eclipse but as standalone CLI programs or via web API.

In the following section, we show how the tool set ASMETA presented above can be put into practice by means of a concrete case study.

3. Illustrative examples

As an illustrative example to show the complete system engineering process from requirements specification to executable code using ASMETA, we refer to the Pill-Box system presented in the tutorial paper [35]. The Pill-Box system is an academic case study designed to demonstrate modeling and refinement techniques. It represents a smart medicine dispenser with multiple drawers, each containing pills of a single drug type. Each drawer is equipped with a switch to detect pill retrieval and a LED to notify the user when it is time to take a pill. The system supports multiple daily deadlines per pill type, but only one LED is activated at a time, even if multiple pills are scheduled simultaneously.

We have modeled the system by applying a refinement approach: the *ground* model abstracts away time and multiple pills, modeling only one pill per drawer; the *first refinement* step introduces a timer to represent time progression, still assuming one pill per drawer; the *final model* captures the full requirements, including multiple pills and deadlines per drawer.

A video tutorial showing how to develop the Pill-Box case study, moving seamlessly from requirements to code using the ASMETA tool set, is available at `https://youtu.be/oy4NEiabZp0`, while the ASMETA models and artifacts can be found at `https://zenodo.org/records/12770854` .

4. Impact

ASMETA is an open-source tool set developed in academia. It improves the pursuit of state-based formal method for reliable software development by turning high-level, ASM models into executable specifications that can be simulated, validated via scenario-based testing, verified via model checking through temporal logic properties, and, when desired, driven toward correct-by-construction implementations through systematic model-to-artifact transformations, thereby increasing safety and trust in early defect detection across the development life-cycle.

4.1. Novel research directions with ASMETA

In response to the emerging challenges brought by the convergence of cyber-physical systems, IoT, edge-cloud infrastructures, intelligent agents, and digital twin platforms, ASMETA (and formal methods, in general) could play an important role in the design of intelligent systems and in ensuring safety in autonomous decision-making, as well as in the interdisciplinary integration of digital twin architectures and formal methods. We here briefly explore such emerging directions:

Formal models@run.time. Modern software systems are rapidly growing in complexity, scale, and autonomy, making many realistic usage scenarios difficult, if not impossible, to reproduce and validate at design time. As envisioned by the models@run.time research community [36], formal methods at runtime [37] are fundamental to address this challenge. With the `AsmetaS@run.time` tool we put this definition into the context of runtime assurance of software systems that must operate reliably and safely among humans. An ASMETA model that expresses certain correctness policies, being executable, can be deployed and run in tandem with a target system in real-life dynamic environments to ensure functional safety and endow the system with proactive self-healing abilities[5].

Runtime safety enforcement. Among the different approaches and techniques proposed in literature that exploit the concept of model@runtime, runtime enforcement [38] is a runtime verification method that focuses on steering

[5]Proactive self-healing is a strategy where systems continuously monitor for potential issues and automatically prevent failures before they occur, contrasting with reactive self-healing, which responds to actual problems.

system executions with the goal of preventing and reacting to misbehavior 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). 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 [39], 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 [40]. Additionally, mission-critical systems, such as autonomous systems that integrate black-box components (like AI agents), operate in dynamic environments where unexpected events should be managed while guaranteeing safe behavior. Ensuring the safety of these complex systems is a major open challenge and requires robust mechanisms to enforce correct behavior during runtime. Again, a runtime safety enforcement mechanism for the output sanitization of the decision made by autonomous agents is crucial. The enforcement mechanism may be based on a (formally validated and verified) ASMETA model representing the enforcement rules and used at run-time to check and eventually steer the system to behave safely [41, 12]. In the context of models@runtime, we say that the ASMETA models behaves as a *guardrail*. We do not target hard real-time systems since these systems require dedicated solutions (e.g., real-time operating systems) and pose specific challenges that are beyond the scope of ASMETA.

Formal models for Next-Generation Digital Twins. A *Digital Twin* (DT) is a digital replica of a physical system, connected to the Physical Twin (PT) through real-time data. DTs give a way to think and explore the issues of the PT in a safe way [42]. We share the view presented in [43], which defines a DT as comprising two fundamental components: the digital twin *model* and the digital twin *capability*. The *model* serves as a virtual replica or representation of a physical entity. Its format varies depending on the nature of the system being represented. In particular, for digital devices, the model captures the system behavior and can be developed using ASMETA. The *capability* of a digital twin refers to various functionalities of the digital twin, such as prediction/prescriptive abilities based on the operation of the model counterpart, or on some data model analysis algorithm, protocol, or procedure. Specifically, if the model is an ASMETA specification representing a physical entity behavior (e.g., a device controller), its primary capability

component is the ASMETA simulator that can be invoked as a runtime service to mimic the PT behavior and to predict safe/unsafe system operation, as well as any other analysis tool, depending on the nature of the capability required to reason on the specific physical system. We also envision that machine learning models can be used to predict unsafe system operations and guide the evolution of the DT model to enforce system safety.

4.2. Broader use of ASMETA

Outside the development group, several research groups and individuals have been working with ASMETA. Rafe et al. have used ASMETA for the validation of an event-based software architecture [44] and have expanded the Bogor tool to support ASMETA specifications for the verification of properties [45]. Bósa has used the tool set for modeling Client-to-Client interaction in a cloud environment [46]. Alkrarha and Hassine have applied mutations to `AsmetaL` specifications by defining several operators [47, 48, 49]. Al-Shareefi et al. have applied ASMETA to safety-critical systems [50] and to authentication and security systems and protocols [51, 52]. Benduhn et al have combined the use of ASMETA with Agile methods [53]. Buga and Nemes have experimented the use of ASMETA for distributed systems [54, 55, 56, 57, 58]. ASMETA has been used also for the verification of network protocols [59, 60, 61]. Recently, Del Castillo has integrated a technique for symbolic execution of ASM models [62] into ASMETA [20].

ASMETA is used not only for research purposes, but also for teaching formal methods. Indeed, it is part of the teaching material in one master course on *System Modeling and Analysis* at the University of Milan (Italy) and in one master course titled *Software Testing and Verification* at the University of Bergamo (Italy). The tools have been exploited in many bachelor's and master's theses.

5. Conclusions

In this paper, we have presented ASMETA as a comprehensive and mature tool set for system development based on Abstract State Machines. We have described its technical metadata and layered architecture, demonstrated its practical applicability through a reference case study, and discussed emerging research challenges and future directions.

Acknowledgments

The work of Andrea Bombarda is supported by PNRR - ANTHEM (AdvaNced Technologies for Human-centrEd Medicine) - Grant PNC0000003 – CUP: B53C22006700001 - Spoke 1 - Pilot 1.4. The work of Silvia Bonfanti, Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra was partially supported by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, and partially by the PRIN project SAFEST (G53D23002770006 and F53D23004230006).

References

[1] E. Börger, R. Stärk, Abstract State Machines, Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-642-18216-7.

[2] E. Börger, A. Raschke, Modeling Companion for Software Practitioners, Springer, Berlin, Heidelberg, 2018. doi:10.1007/978-3-662-56641-1.

[3] E. Börger, V. Gervasi, Structures of Computing - A Guide to Practice-Oriented Theory, Springer, 2024. doi:10.1007/978-3-031-54358-6.

[4] P. Arcaini, A. Bombarda, S. Bonfanti, A. Gargantini, E. Riccobene, P. Scandurra, The ASMETA Approach to Safety Assurance of Software Systems, Springer International Publishing, Cham, 2021, pp. 215–238. doi:10.1007/978-3-030-76020-5_13.

[5] A. Bombarda, S. Bonfanti, A. Gargantini, Developing Medical Devices from Abstract State Machines to Embedded Systems: A Smart Pill Box Case Study, in: M. Mazzara, J.-M. Bruel, B. Meyer, A. Petrenko (Eds.), Software Technology: Methods and Tools, Springer International Publishing, Cham, 2019, pp. 89–103. doi:10.1007/978-3-030-29852-4_7.

[6] P. Arcaini, S. Bonfanti, A. Gargantini, A. Mashkoor, E. Riccobene, Integrating formal methods into medical software development: The ASM approach, Science of Computer Programming 158 (2018) 148–167. doi:10.1016/j.scico.2017.07.003.

[7] P. Arcaini, S. Bonfanti, A. Gargantini, A. Mashkoor, E. Riccobene, Formal validation and verification of a medical software critical component, in: Formal Methods and Models for Codesign (MEMOCODE),

2015 ACM/IEEE Int. Conference on, IEEE, 2015, pp. 80–89. doi:10.1109/MEMCOD.2015.7340473.

[8] A. Bombarda, S. Bonfanti, A. Gargantini, Y. Lei, F. Duan, RATE: A model-based testing approach that combines model refinement and test execution, Software Testing, Verification and Reliability 33 (2023) e1835. doi:https://doi.org/10.1002/stvr.1835.

[9] S. Bonfanti, E. Riccobene, P. Scandurra, Formal specification and validation of the MVM-Adapt system using Compositional I/O Abstract State Machines, Sci. Comput. Program. 244 (2025) 103299. doi:10.1016/J.SCICO.2025.103299.

[10] P. Arcaini, A. Gargantini, E. Riccobene, Rigorous development process of a safety-critical system: from ASM models to Java code, International Journal on Software Tools for Technology Transfer 19 (2017) 247–269. doi:10.1007/s10009-015-0394-x.

[11] P. Arcaini, S. Bonfanti, A. Gargantini, E. Riccobene, P. Scandurra, Modelling an automotive software-intensive system with adaptive features using ASMETA, in: A. Raschke, D. Méry, F. Houdek (Eds.), Rigorous State-Based Methods, Springer International Publishing, Cham, 2020, pp. 302–317. doi:10.1007/978-3-030-48077-6_25.

[12] A. Bombarda, S. Bonfanti, A. Gargantini, N. Pellegrinelli, P. Scandurra, Safety enforcement for autonomous driving on a simulated highway using ASMETA models@run.time, in: M. Leuschel, F. Ishikawa (Eds.), Rigorous State-Based Methods - 11th International Conference, ABZ 2025, Proceedings, volume 15728 LNCS of *Lecture Notes in Computer Science*, Springer, 2026, p. 212 – 230. doi:10.1007/978-3-031-94533-5_13.

[13] P. Gaspari, E. Riccobene, A. Gargantini, A formal design of the Hybrid European Rail Traffic Management System, in: Proceedings of the 13th European Conference on Software Architecture - Volume 2, ECSA '19, ACM, 2019, pp. 156–162. doi:10.1145/3344948.3344993.

[14] A. Bombarda, S. Bonfanti, A. Gargantini, Integrating formal specifications in the development and testing of UIs by formal model–view–controller pattern, International Journal on Software Tools for Technology Transfer (2025). doi:10.1007/s10009-025-00812-2.

[15] P. Arcaini, R.-M. Holom, E. Riccobene, ASM-based formal design of an adaptivity component for a cloud system, Formal Aspects of Computing 28 (2016) 567–595. doi:10.1007/s00165-016-0371-5.

[16] E. Riccobene, P. Scandurra, A formal framework for service modeling and prototyping, Formal Aspects Comput. 26 (2014) 1077–1113. doi:10.1007/s00165-013-0289-0.

[17] R. Mirandola, P. Potena, E. Riccobene, P. Scandurra, A reliability model for service component architectures, Journal of Systems and Software 89 (2014) 109–127. doi:10.1016/j.jss.2013.11.002.

[18] C. Braghin, M. Lilli, E. Riccobene, A model-based approach for vulnerability analysis of IoT security protocols: The Z-Wave case study, Comput. Secur. 127 (2023) 103037. doi:10.1016/J.COSE.2022.103037.

[19] C. Braghin, E. Riccobene, S. Valentini, Modeling and verification of smart contracts with Abstract State Machines, in: J. Hong, J. W. Park (Eds.), Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC 2024, ACM, 2024, pp. 1425–1432. doi:10.1145/3605098.3636040.

[20] C. Braghin, G. D. Castillo, E. Riccobene, S. Valentini, Using symbolic model execution to detect vulnerabilities of smart contracts, in: M. Leuschel, F. Ishikawa (Eds.), Rigorous State-Based Methods - 11th International Conference, ABZ 2025, Proceedings, volume 15728 of *Lecture Notes in Computer Science*, Springer, 2025, pp. 31–51. doi:10.1007/978-3-031-94533-5_3.

[21] C. Braghin, E. Riccobene, S. Valentini, An ASM-Based Approach for Security Assessment of Ethereum Smart Contracts, in: S. D. C. di Vimercati, P. Samarati (Eds.), Proceedings of the 21st International Conference on Security and Cryptography, SECRYPT 2024, SCITEPRESS, 2024, pp. 334–344. doi:10.5220/0012858000003767.

[22] P. Arcaini, E. Riccobene, P. Scandurra, Formal design and verification of self-adaptive systems with decentralized control, ACM Trans. Auton. Adapt. Syst. 11 (2017) 25:1–25:35. doi:10.1145/3019598.

[23] P. Arcaini, R. Mirandola, E. Riccobene, P. Scandurra, MSL: A pattern language for engineering self-adaptive systems, J. Syst. Softw. 164 (2020) 110558. doi:10.1016/J.JSS.2020.110558.

[24] R. Farahbod, V. Gervasi, U. Glässer, CoreASM: An Extensible ASM Execution Engine, Fundam. Inf. 77 (2007) 71–103.

[25] R. Lezuo, P. Paulweber, A. Krall, CASM: optimized compilation of Abstract State Machines, SIGPLAN Not. 49 (2014) 13–22. doi:10.1145/2666357.2597813.

[26] J. Schmid, Introduction to AsmGofer, 2001. URL: https://tydo.de/files/AsmGofer/AsmGoferIntro.pdf.

[27] E. Riccobene, P. Scandurra, Metamodelling a formal method: applying MDE to Abstract State Machines (2006). URL: https://air.unimi.it/handle/2434/29854.

[28] A. Gargantini, E. Riccobene, P. Scandurra, A semantic framework for metamodel-based languages, Autom. Softw. Eng. 16 (2009) 415–454. doi:10.1007/s10515-009-0053-0.

[29] D. Harel, B. Rumpe, Meaningful modeling: What's the semantics of "semantics"?, Computer 37 (2004) 64–72. doi:10.1109/MC.2004.172.

[30] L. Bettini, Implementing Domain-Specific Languages with Xtext and Xtend, Packt Publishing, 2013.

[31] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: An opensource tool for symbolic model checking, in: E. Brinksma, K. G. Larsen (Eds.), Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 359–364. doi:10.1007/3-540-45657-0_29.

[32] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta, The NUXMV symbolic model checker, in: A. Biere, R. Bloem (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2014, pp. 334–342. doi:10.1007/978-3-319-08867-9_22.

[33] E. G. Karpenkov, K. Friedberger, D. Beyer, JavaSMT: A Unified Interface for SMT Solvers in Java, Springer International Publishing, 2016, p. 139–148. doi:10.1007/978-3-319-48869-1_11.

[34] M. S. Farooq, U. Omer, A. Ramzan, M. A. Rasheed, Z. Atal, Behavior Driven Development: A Systematic Literature Review, IEEE Access 11 (2023) 88008–88024. doi:10.1109/ACCESS.2023.3302356.

[35] A. Bombarda, S. Bonfanti, A. Gargantini, E. Riccobene, P. Scandurra, ASMETA tool set for rigorous system design, in: Formal Methods, Springer Nature Switzerland, 2024, p. 492–517. doi:10.1007/978-3-031-71177-0_28.

[36] N. Bencomo, S. Götz, H. Song, Models@run.time: a guided tour of the state of the art and research challenges, Softw. Syst. Model. 18 (2019) 3049–3082. doi:10.1007/s10270-018-00712-x.

[37] R. Calinescu, S. Kikuchi, Formal methods@runtime, in: R. Calinescu, E. Jackson (Eds.), Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems, Springer Berlin Heidelberg, 2011, pp. 122–135.

[38] Y. Falcone, L. Mariani, A. Rollet, S. Saha, Runtime Failure Prevention and Reaction, Springer International Publishing, Cham, 2018, pp. 103–134. doi:10.1007/978-3-319-75632-5_4.

[39] S. Pinisetty, P. S. Roop, S. Smyth, N. Allen, S. Tripakis, R. V. Hanxleden, Runtime Enforcement of Cyber-Physical Systems, ACM Trans. Embed. Comput. Syst. 16 (2017). doi:10.1145/3126500.

[40] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, J. Raman, Analysis of Safety-Critical Computer Failures in Medical Devices, IEEE Security & Privacy 11 (2013) 14–26. doi:10.1109/MSP.2013.49.

[41] S. Bonfanti, E. Riccobene, P. Scandurra, A component framework for the runtime enforcement of safety properties, J. Syst. Softw. 198 (2023) 111605. doi:10.1016/J.JSS.2022.111605.

[42] M. W. Grieves, Product lifecycle management: the new paradigm for enterprises, International Journal of Product Development 2 (2005) 71–84.

[43] S. Ali, P. Arcaini, A. Arrieta, Foundation Models for the Digital Twins Creation of Cyber-Physical Systems, in: Leveraging Applications of Formal Methods, Verification and Validation. Application Areas: 12th Int. Symposium, ISoLA 2024, Proceedings, Part V, 2024, p. 9–26. doi:10.1007/978-3-031-75390-9_2.

[44] Y. Asadollahi, V. Rafe, S. Asadollahi, A formal framework to model and validate event-based software architecture, Procedia Computer Science 3 (2011) 961–966. doi:10.1016/j.procs.2010.12.157.

[45] V. Rafe, S. Doostali, ASM2Bogor: An approach for verification of models specified through Asmeta language, Journal of Visual Languages & Computing 23 (2012) 287–298. doi:10.1016/j.jvlc.2012.05.002.

[46] K. Bósa, An Ambient ASM Model of Client-to-Client Interaction via Cloud Computing and an Anonymously Accessible Docking Service, in: J. Cordeiro, M. van Sinderen (Eds.), Software Technologies, Springer Berlin Heidelberg, 2014, pp. 235–255. doi:10.1007/978-3-662-44920-2_15.

[47] J. Hassine, O. Alkrarha, A Mutation-Based Approach for Testing AsmetaL Specifications, Arabian Journal for Science and Engineering 40 (2015) 3523–3544. doi:10.1007/s13369-015-1832-5.

[48] O. Alkrarha, J. Hassine, MuAsmetaL: An experimental mutation system for AsmetaL, in: 2015 12th International Conference on Information Technology - New Generations, IEEE, 2015, pp. 421–426. doi:10.1109/itng.2015.74.

[49] O. Alkrarha, J. Hassine, Applying selective mutation strategies to the AsmetaL language, IET Software 11 (2017) 292–300. doi:10.1049/iet-sen.2015.0030.

[50] F. Al-Shareefi, A. Lisitsa, C. Dixon, Abstract State Machines and System Theoretic Process Analysis for Safety-Critical Systems, in: S. Cavalheiro, J. Fiadeiro (Eds.), Formal Methods: Foundations and Applications, Springer International Publishing, Cham, 2017, pp. 15–32. doi:10.1007/978-3-319-70848-5_3.

[51] F. Al-Shareefi, A. Lisitsa, C. Dixon, Clarification of Ambiguity for the Simple Authentication and Security Layer, in: M. Butler, A. Raschke,

T. S. Hoang, K. Reichl (Eds.), Abstract State Machines, Alloy, B, TLA, VDM, and Z, Springer International Publishing, Cham, 2018, pp. 189–203. doi:10.1007/978-3-319-91271-4_13.

[52] F. Al-Shareefi, A. Lisitsa, C. Dixon, Analysing Security Protocols Using Scenario Based Simulation, in: P. Ganty, M. Kaâniche (Eds.), Verification and Evaluation of Computer and Communication Systems, Springer International Publishing, Cham, 2019, pp. 47–62. doi:10.1007/978-3-030-35092-5_4.

[53] F. Benduhn, T. Thüm, I. Schaefer, G. Saake, Modularization of Refinement Steps for Agile Formal Methods, in: Formal Methods and Software Engineering, Springer International Publishing, 2017, pp. 19–35. doi:10.1007/978-3-319-68690-5_2.

[54] A. Buga, S. T. Nemes, Towards an ASM Specification for Monitoring and Adaptation Services of Large-Scale Distributed Systems, in: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), IEEE, 2017, pp. 181–186. doi:10.1109/compsac.2017.247.

[55] A. Buga, S. T. Nemeș, A Formal Approach for Failure Detection in Large-Scale Distributed Systems Using Abstract State Machines, in: Database and Expert Systems Applications, Springer International Publishing, 2017, pp. 505–513. doi:10.1007/978-3-319-64468-4_38.

[56] S. T. Nemes, A. Buga, Adopting formal approaches for monitoring and adaptation for large-scale distributed systems, in: 2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA), IEEE, 2017, pp. 1–6. doi:10.1109/iisa.2017.8316416.

[57] S. T. Nemes, A. Buga, Towards Modeling Adaptation Services for Large-Scale Distributed Systems with Abstract State Machines, in: Proceedings of the Seventh International Symposium on Business Modeling and Software Design, SCITEPRESS - Science and Technology Publications, 2017, pp. 193–198. doi:10.5220/0006528901930198.

[58] A. Buga, S. T. Nemeș, Formalizing Monitoring Processes for Large-Scale Distributed Systems Using Abstract State Machines, in: Software

Engineering and Formal Methods, Springer International Publishing, 2018, pp. 153–167. doi:10.1007/978-3-319-74781-1_11.

[59] G. G. Bevilacqua, A. Bianchi, MOTION: an application of ASMETA to mobile ad-hoc networks domain, in: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA '18, ACM, 2018, pp. 1–4. doi:10.1145/3241403.3241444.

[60] P. Campanella, Modeling Routing Protocols in AsmetaL, in: 2021 19th International Conference on Emerging eLearning Technologies and Applications (ICETA), IEEE, 2021, pp. 48–57. doi:10.1109/ICETA54173.2021.9726565.

[61] E. Covino, G. Pani, Analysis of Mobile Networks' Protocols Based on Abstract State Machine, in: Logic, Computation and Rigorous Methods, Springer International Publishing, 2021, pp. 187–198. doi:10.1007/978-3-030-76020-5_11.

[62] G. D. Castillo, Using Symbolic Execution to Transform Turbo Abstract State Machines into Basic Abstract State Machines, in: S. Bonfanti, A. Gargantini, M. Leuschel, E. Riccobene, P. Scandurra (Eds.), Rigorous State-Based Methods - 10th International Conference, ABZ 2024, Proceedings, volume 14759 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 215–222. doi:10.1007/978-3-031-63790-2_15.