ArchSafe Unified Modeler: Automated ASIL, Change-Impact, and Failure Analysis of Architecture and Safety

Muhammad Asim Minhas*
KASTEL
Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131 Karlsruhe, Germany
Karlsruhe, Baden-Württemberg, Germany
muhammad.minhas@kit.edu

Erik Burger
KASTEL – Institute of Information Security and Dependability
Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131 Karlsruhe, Germany
Karlsruhe, Baden-Württemberg, Germany
burger@kit.edu

Ralf Reussner
KASTEL
Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131 Karlsruhe, Germany
KARLSRUHE, Baden-Württemberg, Germany
reussner@kit.edu

Abstract

Safety engineering and design modeling in the development of Cyber-physical Systems (CPS) involves fragmented tools, languages, and representations. Furthermore, both modeling activities are performed at different stages of the CPS development life cycle. This fragmentation necessitates manual synchronization and consistency management of these models. Furthermore, traceability is a very challenging yet vital aspect for such systems that require compliance with relevant safety standards, such as ISO 26262. To address these vital issues, we present ArchSafe, an Eclipse/Sirius modelling workbench developed on a semantically unified SysML ∪ Risk Analysis & Assessment Modeling Language (RAAML) metamodel in which safety concepts are first-class attributes of architectural elements. Using this single unified EMF model, ArchSafe derives seven views: architecture, safety, requirements, Failure Modes & Effects Analysis (FMEA), a quality metrics dashboard, an interactive failure propagation simulation, and an automatically generated traceability view. The tool offers 35 right-click menu operations across all of these views, with advanced impact analysis, ASIL propagation, model health metrics, and ISO 26262-cited validation reports. ArchSafe provides two case studies, autonomous emergency braking and a medical infusion pump. It has been shipped as open source, a P2 update site, a Docker image, and multi-platform RCP products.

CCS Concepts

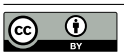
• **Software and its engineering** → **Development frameworks and environments**; **Software safety**; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Computing methodologies** → *Modeling and simulation.*

Keywords

model-based safety engineering, ISO 26262, single underlying model, multi-view modeling, FMEA, failure propagation analysis, EMF, Sirius

ACM Reference Format:
Muhammad Asim Minhas, Erik Burger, and Ralf Reussner. 2026. ArchSafe Unified Modeler: Automated ASIL, Change-Impact, and Failure Analysis of Architecture and Safety. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3837062.3838862

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
MODELS Companion 2026, Málaga, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2903-4/26/10
https://doi.org/10.1145/3837062.3838862

1 Introduction

An engineer responsible for developing a road vehicle function under ISO 26262 produces four kinds of fundamental descriptions. The first is a system architecture which defines the components and the flow of signals and control between them. The second is a hazard analysis that identifies global hazards, their impact on the system, and the severity of their outcome. During safety analysis, such as FMEA, he examines each failure mode and the risk. The fourth is a set of safety goals/requirements that are refined down to safety mechanisms to provide mitigation strategies. All of these artifacts are closely related and dependent on each other. Safety goals exist because of a specific hazard, a technical requirement exists to realise a functional requirement, and a failure model entry directly corresponds to a component in the architecture.

In common practice, each description is maintained and developed in a different tool. Architecture modeling is performed in a single editor, while hazard and failure analyses are usually maintained in tables, and requirements are stored in a requirements manager. The engineer has to maintain the links between them by hand. This has two costs. Traceability in this case is fragile because a cross-reference between all of these artifacts is text that can drift when either side changes. Validation is late because rules that span two descriptions cannot be checked until both have been exported and reconciled.

This paper presents *ArchSafe Unified Modeler*, a tool that removes the boundary between these descriptions by placing all four in one model. Placing them together also lets the tool reason across them, so that checks and analyses which span two or more concerns become single operations rather than manual work across several tools. The rest of the paper describes the model (Section 3), the coordinated views over it (Section 4), the automated analyses it performs (Section 5), a worked example on an emergency braking system (Section 6), the maturity and availability of the tool and what the demonstration shows (Section 7), and how the tool relates to existing environments (Section 8).

2 Tool Overview and Architecture

ArchSafe has been released as a stack of Eclipse plug-ins on the Eclipse 2023-12 target platform Eclipse RCP, the EMF SDK, Sirius 7.x, and JUnit (Fig. 1). The `.unified` plug-in holds acts as a

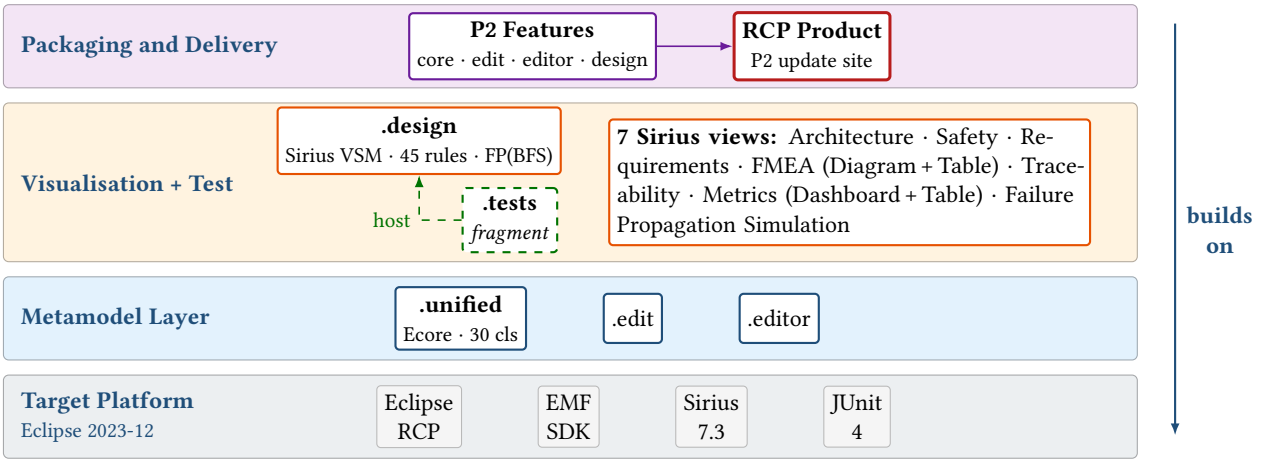


Figure 1: ArchSafe Unified Modeler architecture. The EMF `.unified` metamodel and the Sirius-based `.design` plug-in, which projects *seven* synchronized views built on the Eclipse target platform and are packaged as an RCP product via P2 features. `.tests` is an OSGi fragment of `.design`.

foundation as it holds the Ecore metamodel and its generated Java: a single `UnifiedSystemModel` root that aggregates roughly thirty classes, integrating architectural concepts (blocks, connections, associations) with safety-analysis concepts (hazards, safety goals, requirements, failure modes, FMEA items, and safety mechanisms). Two thin plug-ins build directly on it: `.edit` contributes EMF item providers, and `.editor` contributes a reflective tree editor, so the model stays usable even without diagrams. A single Sirius plug-in, `.design`, forms the visualisation layer: it carries the viewpoint specification, the AQL query services, the validation rules, and the failure-propagation engine. A JUnit fragment, `.tests`, attaches to `.design` and runs in continuous integration. Finally, four P2 *features* aggregate the plug-ins, and a Maven/Tycho reactor packages them into the multi-OS **RCP product** and a P2 update site.

One model, many views. In ArchSafe, each view type presented to the user is a direct projection from the unified metamodel. The Sirius viewpoint declared in `unified.odesign` offers **seven synchronised views**—*Architecture*, *Safety*, *Requirements*, *FMEA*, *Quality Metrics*, *Failure-Propagation Simulation*, and an auto-generated *Traceability* view realised as seven Sirius representations FMEA and Quality Metrics each provide both a diagram and a synchronised table. AQL expressions and Java services are used to resolve the Node and edge mappings; hence, the views do not store their own data; consequently, no view holds private state. Because every representation reads from and writes to the shared `UnifiedSystemModel`, an edit in one view is updated immediately in all the others; for example, adding a safety-critical block in the Architecture view results in the creation of the same block at once in the Safety and Traceability views, consistency by construction. Bidirectional traceability is achieved for free from EMF `eOpposite` references, and **45 semantic validation rules**, each tied to an ISO 26262 clause, are evaluated continuously over the model and report violations in place.

From clicks to model changes. Because of the strong semantic unification of architecture and safety concepts, ArchSafe transforms the analysis workflow into direct manipulation. **14 Sirius tool sections** expose right-click commands on the diagrams and tables and bind them to **35 Java `ExternalJavaActions`**, organised into six capability groups (detailed in section 5). Java actions run within EMF `RecordingCommand` transactions, so every change is atomic and undoable, and the views remain consistent; the remaining actions are read-only, producing ISO 26262 validation reports and audit-ready exports. Table 1 lists a representative slice of this wiring.

3 The Unified Model

The foundation of the tool is a single metamodel that holds every concern. The unified metamodel and its evaluation were presented in our ECMFA 2026 paper [5]; however, this paper presents the ArchSafe Unified Modeler tool that implements them. System blocks and safety-critical blocks describe the architecture, block connections describe the flow between them, and failure modes are declared on the blocks that exhibit them. Hazards describe what can go wrong and refer to the blocks that can cause them. A failure analysis groups failure analysis items, each of which refers to a component and a failure mode. Safety goals, functional safety requirements, technical safety requirements, and safety mechanisms form the requirement chain. All of these are elements of one model, built on the Eclipse Modeling Framework [6].

Because the concerns share a model, the connection between them is a typed reference rather than a string. When a safety goal records the hazard it addresses, it holds a link to that hazard object. When a technical requirement records the component that realises it, it holds a link to that block. A link of this kind cannot silently point at the wrong element or at nothing, and it moves with its target when the model changes. Figure 2 shows the model at the centre and the seven views that are generated from it.

4 Coordinated Views and Editing

ArchSafe presents seven views over the same model, built with Eclipse Sirius [7]. The architecture view edits blocks and their

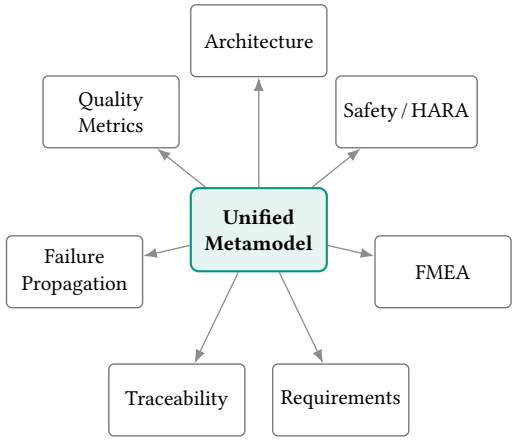


Figure 2: The unified metamodel and the seven coordinated views generated from it. Each view edits the same model, so a reference between two concerns is a model link rather than a separate export.

connections. The safety view edits hazards and their relations to blocks. The failure analysis view edits the failure analysis and its items. The requirements view edits safety goals and their refinement into functional and technical requirements and safety mechanisms. The traceability view presents the links across these concerns and reports the elements that are not yet traced. The quality metrics view and the failure propagation view are described in Section 5. Two of these concerns, the failure analysis and the quality metrics, are additionally offered in a tabular form, so the tool ships nine representations over these seven views.

Each view provides a palette for creating typed elements and context tools for linking them, for example a tool that refines a functional requirement into a technical one or allocates a safety goal to a functional requirement. The views are projections over a single underlying model. Every element shown on a view refers back to an element of the one semantic model, and the Sirius representation resource persists only graphical state, such as layout and layers, and holds no domain data of its own. An edit made in one view is therefore present in the others once they are refreshed. Consistency holds by construction rather than by synchronisation between separate models, and the engineer works on a single artefact through several lenses rather than on several artefacts that must be kept in step.

5 Automated Analyses on the Unified Model

Because every concern is held in one model, ArchSafe can offer analyses that read across all of them at once. Each analysis is a menu operation that an engineer runs on the model or on a selected element, and each reports its result with the ISO 26262 clause it supports. Table 1 lists the operations. The rest of this section describes the four that are least common in existing environments, and then the quality metrics and the export of work products.

5.1 ASIL propagation and decomposition

An automotive integrity level, from QM up to ASIL D, must be carried consistently down the requirement chain. When a safety goal is met by several requirements, its integrity level may be decomposed only in combinations permitted by ISO 26262-9. ArchSafe encodes these rules and checks them on the model. Two of the checks yield findings for a real model: no requirement may carry a higher integrity level than its parent, and no requirement may be allocated to a component classified below it. A third check compares a decomposition across several requirements against the decomposition table of ISO 26262-9 (Part 9, §5.4.6), which the report prints alongside the result; in the present metamodel a functional safety requirement inherits the integrity level of its safety goal, so this case is verified but cannot yet be violated. The operation produces a report that lists each pass and each violation, along with its clause. Figure 3 shows this report on the braking model after a technical safety requirement was deliberately raised above its parent's integrity level: the tool reports a failure, because the requirement now exceeds its parent, and two warnings, because that requirement and the radar are realised by blocks classified below their level, while the remaining chains pass. This is reasoning that engineers otherwise perform by hand and record informally.

5.2 Change impact analysis

Before changing an element, an engineer needs to know what else the change affects. For any selected element, whether a safety goal, a functional or technical requirement, a safety mechanism, a block, a hazard, a failure mode, or a failure analysis item, ArchSafe follows the model links and reports the set of elements that depend on it, across the architecture, the hazards, the requirements, and the failure analysis. Because the links are model references rather than text, the reported set is exact and does not depend on naming conventions.

5.3 Failure analysis and risk

Each failure mode declared on a block can be turned into a failure analysis item. An item records severity, occurrence, and detection on a scale from one to ten, and their product gives a risk priority number. The tool generates items from the declared failure modes and marks them for engineer review, so the analysis starts from the architecture rather than from a blank table. Items whose risk priority number exceeds one hundred are reported as requiring a mitigation action, and items above two hundred are reported as also requiring a named responsible owner.

5.4 Failure propagation

The failure propagation view lets the engineer inject a fault at a block and observe its effect. A breadth first traversal follows the connections in the architecture, in the forward, backward, or both directions, and marks the blocks that the fault reaches. The operation then reports which hazards the fault activates and identifies the most critical path from the injected fault to an activated hazard, and each step carries the failure effect and failure rate of the block it passes through. The analysis runs on the same model the engineer designed, so there is no separate fault model to build and maintain.

Table 1: Automated operations on the unified model.

Operation	Result	ISO 26262
ASIL propagation and decomposition	pass or fail per requirement link and per decomposition	Part 9 Clause 5.4
Change impact	transitive set of affected elements	Part 2 Clause 6.4.3
Failure propagation	reached blocks, activated hazards, critical path	Part 9 Clause 8.2
HARA completeness	uncovered or unmitigated hazards	Part 3 Clause 6.4.2 / Clause 6.4.3
Requirement chain	gaps from goal to mechanism	Part 8 Clause 6.4.3.2
Quality metrics	six scored metrics against thresholds	GQM plan
Work-product export	safety case fragment, reports	Part 2 Clause 6.4.8 / Clause 6.4.9

Table 2: The six quality metrics and their thresholds, fixed in a goal, question, metric plan.

Metric	Measures	Threshold
MCR	metamodel concept categories populated	≥ 0.90
HTI	hazards traced to a safety goal	$= 1.00$
RAR	technical requirements allocated	≥ 0.95
FLC	failure analysis items linked	$= 1.00$
TDS	density of cross-concern links	≥ 0.80
MVR	safety mechanisms verified	≥ 0.85

5.5 Quality metrics, guidance, and work products

ArchSafe scores a model against six metrics whose thresholds were fixed in a goal, question, metric plan, shown in Table 2. They measure how much of the metamodel a model populates, how many hazards are traced to a safety goal, how many technical requirements are allocated, how completely failure analysis items are linked, the density of traceability links, and how many safety mechanisms are verified. Each metric is reported as a pass or a fail against its threshold, so the dashboard gives an at-a-glance view of model health. A companion operation turns weaknesses into concrete guidance, for example, that a catastrophic hazard lacks a mitigation or that a requirement is not yet traced. Alongside these, forty-five semantic validation rules, of which eleven check the architecture, nine the safety analysis, seven the failure analysis, and eighteen the requirements, run on the model and each reports the clause it enforces. Findings, metrics, and the requirement chain can be exported as ISO 26262 work products, including a safety case fragment and a safety requirement review report.

```
ISO 26262 ASIL PROPAGATION & DECOMPOSITION VALIDATION
Primary Ref: ISO 26262-9:2018 Sec 5 (ASIL decomposition)
Table: ISO 26262-9:2018 Sec 5.4.6 Table 5

ASIL Decomposition Table 5 (Part 9 Sec 5.4.6):
  ASIL D -> B(D) + B(D) | min child ASIL = B
  ASIL C -> A(C) + B(C) | min child ASIL = A
  ASIL B -> A(B) + A(B) | min child ASIL = A

-- FAILURES (1) ----------------------------------
✗ FAIL TSR 'TSR6: Deceleration Feedback Loop'
       ASIL=ASIL_D exceeds parent FSR 'FSR4: Brake
       Actuation Monitoring' ASIL=ASIL_C.
       Ref: ISO 26262-9:2018 Sec 5.4.3

-- WARNINGS (2) ----------------------------------
⚠ WARN TSR 'TSR1: Radar ASIL-D Hardware' [ASIL_D]
       realized by 'Front Radar Sensor' [ASIL_B].
       Ref: ISO 26262-4:2018 Sec 6.4.5.g
⚠ WARN TSR 'TSR6: Deceleration Feedback Loop'
       [ASIL_D] realized by 'Hydraulic Brake
       Actuator' [ASIL_C].
       Ref: ISO 26262-4:2018 Sec 6.4.5.g

-- PASSED (9) ------------------------------------
✓ PASS SG 'SG1: AEB shall activate when collision
       is imminent' [ASIL_D] -> FSR 'FSR1: Obstacle
       Detection Function' [ASIL_D] -- OK.
✓ PASS SG 'SG3: AEB shall apply full braking when
       commanded' [ASIL_C] -> FSR 'FSR4: Brake
       Actuation Monitoring' [ASIL_C] -- OK.
       ... (7 further passes omitted)

-- SUMMARY ---------------------------------------
1 failure, 2 warnings, 9 passes.
```

Figure 3: The automated ASIL propagation and decomposition report on the braking model, showing a deliberately induced violation. Each result is reported with its ISO 26262 clause; passing entries are abbreviated for space.

6 Worked Example: Emergency Braking

We illustrate the tool with the automotive case study that ships with it, a model of an autonomous emergency braking system. The model contains the sensing chain of a front radar and a front camera, a sensor fusion unit, a control unit, and a hydraulic brake actuator, together with their failure modes.

One thread runs through the whole model. The hazard of a collision caused by the braking function failing to activate is rated catastrophic. From this hazard, the model derives a safety goal that the system shall begin braking within a bounded time of a confirmed threat, at the highest integrity level. The goal is allocated to a functional requirement for obstacle detection, which is refined into a technical requirement that the radar hardware meet the highest integrity level with redundant processing. That technical requirement is implemented by a radar built-in self-test with a stated diagnostic coverage, and it is verified by a failure analysis

item for the radar. Every step in this thread is a model link, so the chain from the hazard to the mechanism can be read and checked without leaving the model. Running the change impact operation on the safety goal lists the functional requirement, its technical refinement, the radar block, the safety mechanism, and the verifying failure analysis item, and the ASIL operation confirms that the highest integrity level is carried down the chain without a break. When the radar fault is then injected in the failure propagation view, the same catastrophic hazard is reported as activated, which closes the loop between the requirement chain and the analysis.

7 Maturity, Availability, and Demonstration

ArchSafe is distributed in three forms, accessible via the GitHub Repo [1]. It is an Eclipse product for Linux, Windows, and both Intel and Apple silicon macOS. It is a P2 update site that installs into an existing Eclipse modelling installation. It is a container image that runs the tool in a browser with the case studies already loaded, which allows a reviewer to try it without a local installation. The tool is released under the Eclipse Public License. Two case studies ship with it, the automotive emergency braking system used above under ISO 26262, and a medical infusion pump under IEC 62304, which shows that the model is not tied to a single domain.

The demonstration video [2] is a walkthrough of the emergency braking model. It opens the architecture, moves through the hazard analysis, the failure analysis, and the requirement chain, runs the validation and reads a finding with its clause, and ends with a failure propagation run that activates the collision hazard. The walkthrough shows that the seven views describe one model and that the analyses operate on that model directly.

8 Related Tools

Several environments address parts of the same task (Table 3). Capella, with its safety viewpoints, supports architecture and analysis with the Arcadia method [8]. Papyrus provides SysML modelling with safety profiles [3]. CHESS supports component-based development with dependability analysis [1]. EAST-ADL describes automotive architecture with error models that support safety analysis [2]. ANSYS Medini offers hazard analysis, failure analysis, and fault trees for ISO 26262.

ArchSafe sits at a different point in this space. Its emphasis is a single shared model for architecture, hazards, failure analysis, and requirements, with validation that reports the standard clause it enforces, and open availability as an Eclipse product and a container image. Where the tools above often connect separate models through exchange formats, ArchSafe keeps the concerns in one model so that the links between them are part of that model. The automated checking of ASIL decomposition against the Part 9 table, the computation of change impact across concerns, and the executable failure propagation on the design model are, to our knowledge, not offered together by these environments. We do not claim that this is superior in every respect. We claim that it is a useful and openly available alternative for teaching and for early stage analysis.

[1] https://masimminhas.github.io/ArchSafeUnifiedModelerUpdateSite/
[2] https://youtu.be/ikc08I0tmFc

Table 3: Head-feature comparison (excerpt; full table in the artefact)

Capability	Capella	Papyrus	Medini	ArchSafe
Unified semantic meta-model (not a profile)	✓	✗	✓	✓
FMEA auto-seeding from architecture	✗	✗	✓	✓
Interactive failure propagation	✗	✗	○	✓
ISO 26262 clause-cited validators	✗	✗	✓	✓
Open source + reproducible artefact	✓	✓	✗	✓

✓ full support ○ partial ✗ none

9 Conclusion

ArchSafe Unified Modeler represents architecture, hazards, failure analysis, and safety requirements in a single model, presents seven coordinated views of it, and performs analyses spanning these concerns, including ASIL decomposition checking, change impact, and failure propagation, with 45 validation rules that cite the ISO 26262 clauses they enforce. The tool has not yet been evaluated with engineers outside the authors' group, and such a study is the clearest next step. Further work will broaden the set of standards covered by the validation and extend the analysis with quantitative hardware metrics.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – CRC 1608 – 501798263.

References

[1] Antonio Cicchetti, Federico Ciccozzi, Silvia Mazzini, Stefano Puri, Marco Panunzio, Alessandro Zovi, and Tullio Vardanega. 2012. CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 362–365. doi:10.1145/2351676.2351748

[2] Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Yiannis Papadopoulos, Mark-Oliver Reiser, Anders Sandberg, David Servat, Ramin Tavakoli Kolagari, Martin Törngren, and Matthias Weber. 2010. The EAST-ADL Architecture Description Language for Automotive Embedded Software. See [4], 297–307. doi:10.1007/978-3-642-16277-0_11

[3] Sébastien Gérard, Cédric Dumoulin, Patrick Tessier, and Bran Selic. 2010. Papyrus: A UML2 Tool for Domain-Specific Language Modeling. See [4], 361–368. doi:10.1007/978-3-642-16277-0_19

[4] Holger Giese, Gabor Karsai, Edward Lee, Bernhard Rumpe, and Bernhard Schätz (Eds.). 2010. *Model-Based Engineering of Embedded Real-Time Systems: International Dagstuhl Workshop, November 4-9, 2007. Revised Selected Papers.* Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-16277-0

[5] Muhammad Asim Minhas, Erik Burger, Ralf Reussner, and Tianhai Liu. 2026. A Unified Model-Driven Framework for Co-Designing System Architecture and Safety Analysis in Cyber-Physical Systems. *The Journal of Object Technology* 25, 3 (2026), 3:211. doi:10.5381/jot.2026.25.3.a17

[6] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2008. *EMF: Eclipse Modeling Framework* (2nd ed.). Addison-Wesley.

[7] Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. 2014. Sirius: A rapid development of DSM graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*. IEEE, 233–238. doi:10.1109/INES.2014.6909375

[8] Jean-Luc Voirin. 2018. *Model-based System and Architecture Engineering with the Arcadia Method.* Elsevier. doi:10.1016/C2016-0-00862-8