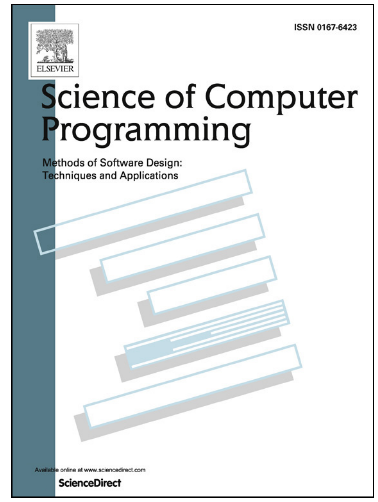
Journal Pre-proof



ACSmt: A plugin for eclipse papyrus to model systems of systems

Sean Kristian Remond Harbo, Emil Palmelund Voldby, Jonas Madsen and Michele Albano

PII: S0167-6423(23)00090-4
DOI: https://doi.org/10.1016/j.scico.2023.103008
Reference: SCICO 103008

To appear in: *Science of Computer Programming*

Received date: 9 December 2022
Revised date: 7 August 2023
Accepted date: 7 August 2023

Please cite this article as: S.K. Remond Harbo, E. Palmelund Voldby, J. Madsen et al., ACSmt: A plugin for eclipse papyrus to model systems of systems, *Science of Computer Programming*, 103008, doi: https://doi.org/10.1016/j.scico.2023.103008.

Highlights

- The Abstract Communicating Systems (ACS) System of System (SoS) engineering methodology presents potential for designing complex SoS-based platforms.
- ACS modeling tool (ACSmt) is the first tool implementing the ACS methodology.
- ACSmt is implemented as an Eclipse Papyrus plugin, which supports UML 2.5 and is well-accepted in the industry.
- To facilitate ACSmt implementation in Papyrus, ACS was mapped on UML 2.5. This also concurs to a gentler learning curve.
- ACSmt allows for verifying structural properties of the designed SoS.
- The open source code of ACSmt can be used as reference when implementing plugins for Papyrus.

ACSmt: A Plugin for Eclipse Papyrus to Model Systems of Systems

Sean Kristian Remond Harbo

Department of Computer Science, Aalborg University, Selma Lagerløfs Vej 300, 9220 Aalborg, Denmark

Emil Palmelund Voldby

Department of Computer Science, Aalborg University, Selma Lagerløfs Vej 300, 9220 Aalborg, Denmark

Jonas Madsen

Department of Computer Science, Aalborg University, Selma Lagerløfs Vej 300, 9220 Aalborg, Denmark

Michele Albano

Department of Computer Science, Aalborg University, Selma Lagerløfs Vej 300, 9220 Aalborg, Denmark

Abstract

While System of Systems (SoS) architectures for large and complex software projects are gaining momentum, the commonly used modeling and tooling approaches are still general-purpose or oriented towards single systems. Developers could benefit from methods and tools that avoid system-centric details in favor of native SoS modeling support. This paper presents a diagram-centric modeling tool with native SoS modeling support. The tool is implemented as a plugin for the Eclipse Papyrus modeling tool. The tool was showcased as a demo at MODELS'22. The code of the plugin is freely available via Github.

Keywords: Papyrus, UML Profile, Metamodel, SoS modeling, DSML

Metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0
C2	Permanent link to code/repository used for this code version	`https://github.com/acs-modeling-tool/acs-modeling-tool/tree/v_1.0`
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	GNU General Public License version 3.0 (GPL-3.0)
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Java, Eclipse Modeling Tools, Papyrus
C7	Compilation requirements, operating environments and dependencies	Support for Windows, Linux & MacOS. Dependency on Eclipse Modeling Tools 2023-03 and Papyrus
C8	If available, link to developer documentation/manual	`https://github.com/acs-modeling-tool/acs-modeling-tool/wiki`
C9	Support email for questions	`seankrh@cs.aau.dk`

Table 1: Code metadata (mandatory)

1. Motivation and Significance

In recent years the number of internet-connected devices has continued to rise [1] and, as software projects get larger and more complex, current trends aim to support applications by means of a System of Systems (SoS), a system being defined by operational independence (they can fulfill valid purposes in their own right) and by managerial independence (the systems are managed for their own purposes rather than the purposes of the whole) [2, 3].

There is general agreement that modeling the systems [4] can be useful for documenting and driving the development process [5], for verifying the systems using formal methods [6], and for code generation [7]. Current approaches to SoS modeling and design inherits largely from system modeling and design.

Our work makes use of the *Abstract Communicating Systems* (ACS) [8] SoS engineering methodology, which is a novel approach that focuses on the communication activities of the systems that are part of the SoS. To ease the modeling efforts, ACS considers that all internal computations of systems are represented by time delays. Thus, the SoS designer can focus on describing the structure of the SoS and on the behavior of each system, with the goal of verifying static and dynamic properties of the SoS (its structure, communication timing, ordering of the messages, type correctness). However, several alternatives exist. Given their characteristics, we argue that ACS is a better fit for a methodology that is both useful to support SoS modeling, verification and code generation, while providing a relatively shallow learning curve.

Designing for Adaptability and evolutioN in System of systems Engineering (DANSE) [9] uses contract-driven design to support modeling, validation, and design correction related to system interactions. The official DANSE implementation integrates a dozen different tools, extensions, and frameworks into one tool-suite [10]. Even though the DANSE methodology appears effective for SoSE and long-term development, it requires the user to learn a plethora of techniques and tools, some of them being still works in progress, which can be an issue since DANSE was discontinued in 2015.

The *Comprehensive Modelling for Advanced Systems of Systems* (COMPASS) consortium focuses on supporting the modeling of complex and heterogeneous SoSs [11] using both text- and graphics-based SoS-modeling tools centered around the text-based *COMPASS Modelling Language* (CML) [11]. CML defines purpose-built semantics for specifying assumptions and guarantees (e.g. contracts) about SoSs, which allows for automated fault detection using static analysis tools such as theorem provers and model checkers. However, apart from the incomplete state of the project, CML is general-purpose and details-oriented, and models might become hard to maintain as they grow in size. On the other hand, ACS abstracts away from the internal logic of the systems, and it uses the Structure diagram to capture the hierarchy in the structure of the SoS, leading to more compact models.

The approach proposed by the AMADEOS consortium [12] makes use of SysML for SoS modeling, and it can specify very diverse aspects of a SoS. However, we argue that it suffers from the same issues as DANSE [9], since its conceptual model is articulated on seven different viewpoints, each one modeled as a different SysML profile, leading to a steep learning curve.

SosADL [13] is a language used for SoS modeling, and it bases its formalism on π-calculus. The formal specification of a SoS can then be specified with the textual SosADL language or within the Architecture Development Environment SosADE, which provides a graphical concrete syntax for

SosADL. Later on, SosADL can be compiled into UPPAAL [14] models or other artifacts for verification. This latter approach is similar to ACS, still its models are inherently very detailed. One of the strengths of ACS is in its hierarchical structure that allows to build Composites, which can contain other Composites or atomic systems, leading to a structured definition of a SoS and more abstract models.

CD++ [15] is a tool for DEVS, which is a low-level formalism for modeling complex dynamic systems using a discrete event abstraction. DEVS has a more fine grained vision than ACS of the atomic systems, called atomic models. In fact, CD++ allows atomic models to be written in C++ and linked to the simulation tool. The vision of ACS is different than CD++, in that the internal execution of the atomic systems are abstracted away, to focus on the communication capabilities of the systems. On the other hand, DEVS was used by a number of tools [16] as a simulation "assembly language" to which models in other formalisms can be mapped, and it could be interesting to map ACS over DEVS.

ModelicaML [17] combines Modelica, UML and SysML into a formalism for model-driven development of cyber physical systems. Its approach is similar to ACS, with the main difference being its reliance on Modelica's formal executable modeling for analyses and simulation, while ACS has no such requirement and it allows the models to be abstract with respect to their internal logic.

This paper introduces ACS modeling tool (ACSmt), which is an implementation of the ACS methodology as a plugin for Eclipse Papyrus [18], which is well accepted by the industry and academia alike as a prominent system modeling tool. An advantage of ACSmt is that, since the IDE is based on Eclipse Papyrus, experienced users should have an easier time adopting ACSmt IDE. Another contribution of this work is related to Eclipse Papyrus itself, since its developer documentation [19] is sometimes lacking, and the code of ACSmt (`https://github.com/acs-modeling-tool/acs-modeling-tool`) is open source and well documented and can be used as reference for further development of Papyrus plugins.

2. Software Description

There exist several open-source modeling editors [20, 18, 21], that might be utilized as a basis to create the ACSmt tool. These existing projects represent an opportunity to capitalize on existing code and established best practices. More than ten domain specific modeling projects (and various other projects) have some software support for their graphical notation using Eclipse Papyrus [22, 23, 24, 25, 19, 26]. Papyrus has been supported for a

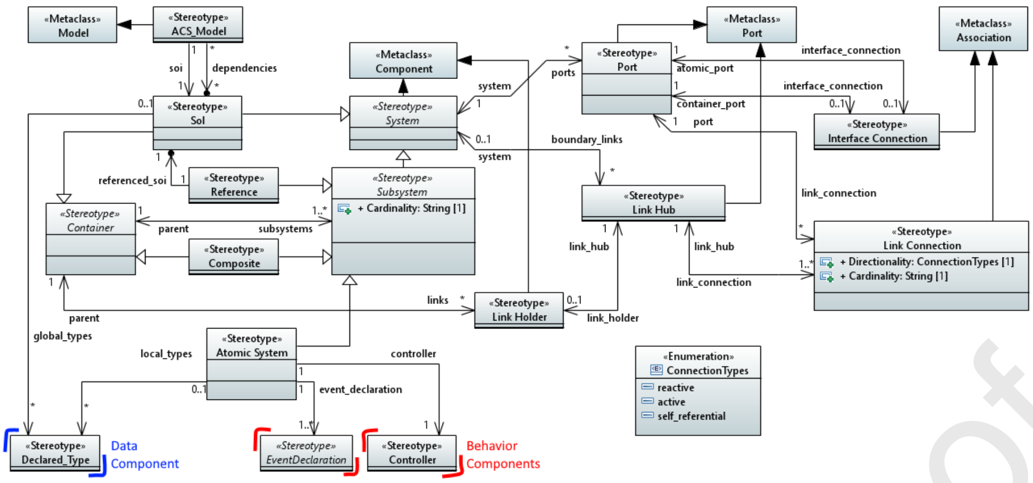


Figure 1: ACS Structural components as part of the UML profile.

long time, evident by the approximately 17.000 commits it has received since 2009 [27]. We considered Papyrus to be a mature and well-tested tool, suited to create the ACS tool. For this reason, the dedicated ACS tool and IDE is implemented using the Papyrus framework.

2.1. Software Architecture

Papyrus is a plugin for the Eclipse platform, and it supports UML 2.5 [25] natively. Thus, it was deemed useful to map the ACS methodology on the UML 2.5 metamodel, in which ACS concepts are mapped to extensions of UML concepts. The ACS UML profile was created using the software "*Eclipse Modeling Tools v. 2023-03*" [28], which provides a great deal of facilities to ease the development of plugins for "*Eclipse Papyrus v. 2023-03*" [18], such as templates, code-generation, file management, and various editors (profile, code, etc.). The ACS UML profile contains the ACS Structure (see Fig.1), Behavior, and Data layers. More information on this mapping can be accessed on [29]. The ACS UML profile is compiled by Eclipse Modeling Tools into a number of specialized plugins, which are run together and are deployed as a single, fully-functioning ACSmt executable for Windows, Mac, and Linux.

2.1.1. Plugins

Each plugin of ACSmt is dedicated to a specific function in the modeling tool, namely *UML Profile*, *Wizard*, *Architecture*, *Palette*, *Properties*, and *Validation*.

The *UML profile* plugin contains the UML profile for ACS (see for example the ACS Structural components in the profile in Fig.1) and in this sense it

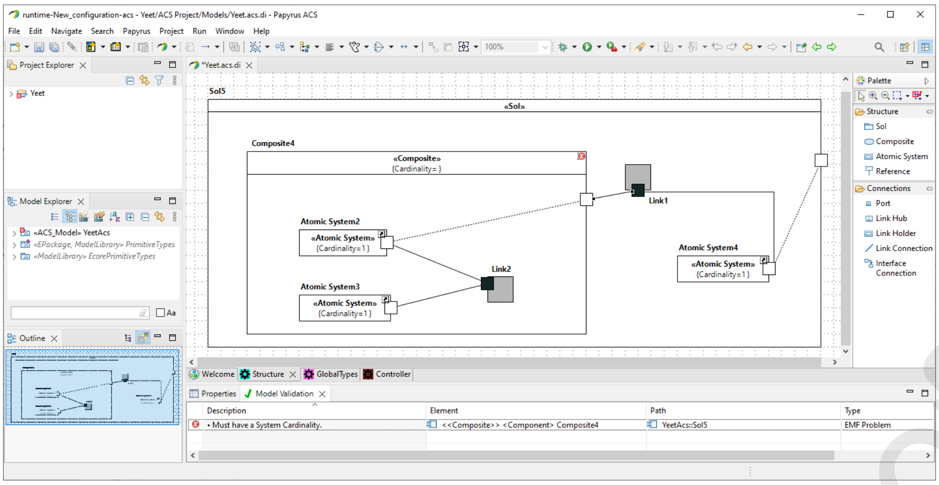


Figure 2: ACSmt: Structure example [30]

implements the modeling constructs of the ACS methodology, and the Java classes that Papyrus generates from it. The *Wizard* plugin is accessible from the IDE menu, and it is used to create an empty ACS Project containing a default empty Structure Diagram (see Section 2.2 and [8] for information regarding the diagrams composing the ACS methodology). The *Architecture* plugin adds the ACS look & feel such that diagrams, colors and styles elements are the ones expected by the ACS methodology, instead of the UML elements they extend. This is achieved for example with a Cascading Style Sheet type file. The *Palette* plugin presents a toolbar specific to ACS, which contains the ACS elements that are usable in the new ACS diagram types. The toolbar is context-specific and for example only Controller-related elements are visible when editing a Controller Diagram. The *Properties* plugin adds an ACS menu to the Properties view of Papyrus, which contains ACS-specific properties of the elements added by the *UML Profile* plugin. The *Validation* plugin is the entry point to the model-wide validation capabilities of ACS, it ensures that the model is structurally correct, and provides error messages that hint to a solution to a bad model. As of writing, ACSmt can validate the Structure diagram and some of the other diagrams. Still, the tool is a work in progress and it lacks the majority of validation functionality related to the Behavior and Data layers.

2.1.2. Software Setup

The ACSmt github wiki page contains a guide for setting up the software.

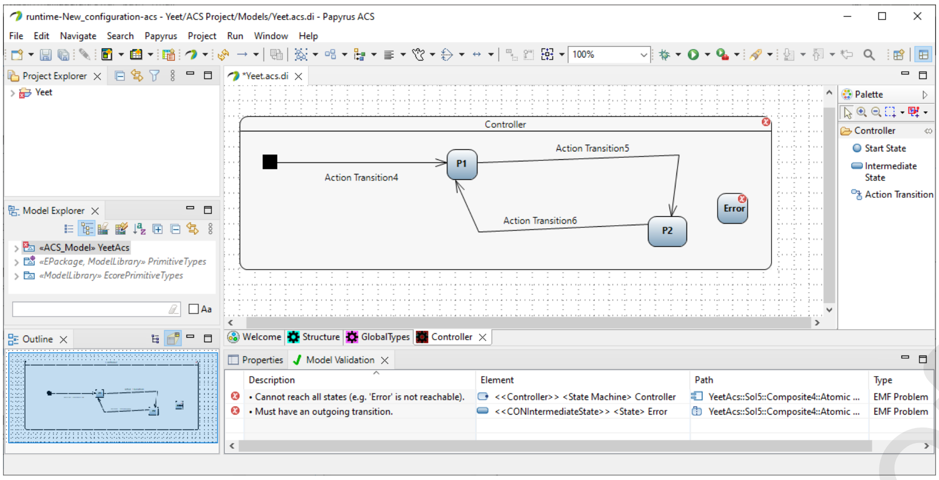


Figure 3: ACSmt: Controller example [30]

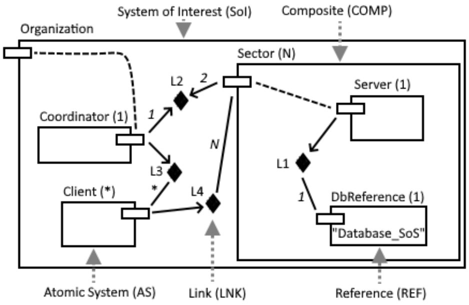
2.1.3. *Finished Result*

The ACS IDE is shown in Figures 2 (Structure diagram) and 3 (Controller diagram). Given the limitations of Papyrus, the presented Structure Diagram (Fig. 2) does not support Link Connection Directionality & Cardinality and Link Hub shape. However, overall the modeling tool facilitates modeling and structural validation of ACS models in a style similar to existing Papyrus projects, which is an advantage to ease ACS adoption by the Papyrus community. The complete source code and a compiled release build is freely available on GitHub [30].

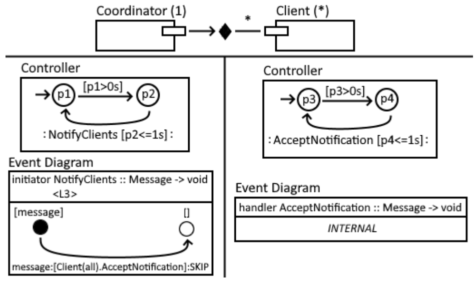
2.2. *Software Functionalities*

Broadly speaking, ACSmt currently supports creating ACS projects, as well as building and structurally verifying ACS models in custom made diagrams to look the part. An ACS model consists of the "Structural", "Behavioral", and "Data" layers (see Figures 4a and 4b, and Listing 1, respectively). ACSmt implements a *Structure Diagram* for the structure layer, which models the structure of the SoS and the connections between the systems within that communicate. Next, ACSmt implements the *Controller Diagram* and *Event Diagram* for the behavior layer, which models the timing and communication characteristics of the aforementioned systems by means of state machines that exchange messages with each other. Lastly, ACSmt implements a *Type Diagram* for the otherwise textual data layer, which models the structure of the types of data/messages communicated between systems. For now, this approximation is the only way to include ACS types in ACSmt.

To validate an ACS model, right-click anywhere in any ACS diagram and press "Model validation > Validate model".



(a) Structure Diagram



(b) A representation of the behavior of and interaction between the Client and Coordinator systems from Figure 4a

Figure 4: Examples of a Structure diagram, and Controller and Event diagrams

```
Message = (          // A record
  String message,  // A field with type "String"
  Urgency urgency, // A field with type "Urgency"
  num? deadline    // A field with type "nullable number"
) {
  .message._elem."a-zA-Z ", // Constraint array element values
  .message.[[1;100]],       // Constraint array size
  .deadline.[0;+inf],       // Constraint number-value
  .deadline.(0)             // Decimal precision = '0' (integer)
}
```

Listing 1: Definition of "Message" type mentioned in the Event Diagrams of Figure 4b.

3. Illustrative Examples

ACSmt was demonstrated at the MODELS'22 conference [31], and a video of it is available at `https://youtu.be/RT_9SuwAaRA`.

4. Impact

The implemented ACSmt is, to the best of our knowledge, the only tool implementing the ACS methodology [8], and it does that as an extension of Eclipse Papyrus [27], which is well-accepted in the industry. Moreover, the ACS methodology is mapped on UML 2.5 through a UML Profile. Together, these characteristics allow for a quick learning for the practitioner. Apart from demonstrating ACSmt at a flagship conference [31], we have performed usability testing (reported on in [29]) with a group of professional designers and developers from a large industry from France, and a group from a SME from Hungary. The feedback has been quite positive and it was used to improve the tool further, and in particular to improve the styling of the user interface.

ACSmt poses an opportunity to introduce the ACS methodology's principles and ideas to industry experts and to explore the efficacy and soundness thereof, where ACS itself explores a new approach to view, model, and verify SoSs, where all diagrams and model-components are interconnected.

It can be argued that the existence of an IDE heavily affects the reception of the formalism it implements. Moreover, ACSmt also explores ACS's compatibility with traditional diagrammatic editors, and whether new approaches to such editors are required. This might be the case since the tight-woven relationships between an ACS model's diagrams and types are difficult to capture using the traditional approach of describing each one with its own diagram, or splitting a bigger model into a number of smaller diagrams. One such new approach could be to embed the different Behavior and Data diagrams of an ACS model into its Structure diagram, so that the entire model seamlessly fits together while respecting the inherent hierarchy typical of a SoS.

5. Future Plans

We are in the process of integrating ACSmt with the UPPAAL model checker [14], which is an industry leader for verification of timed systems. By means of a novel plugin, ACSmt will create an XML representation of a SoS, which will be fed to UPPAAL to check the timing and ordering properties defined in the Behaviour layer of the model.

We are interacting with the Eclipse Arrowhead project [32], which is a framework for the support of industrial SoSs. The aim of this joint work is to allow ACS to be used at design time for Arrowhead-compliant systems, and to extend ACSmt to generate a configuration for the systems participating in a Arrowhead-compliant SoS.

Acknowledgements

Research funded in part by the ERC Advanced Grant LASSO; by the S4OS Villum Investigator Grant (37819) from Villum Fonden.

We thank Saadia Dhouib (CEA List, France) and Géza Kulcsár (IncQuery Labs, Hungary) for their support while designing and implementing our tool as a plugin for the Eclipse Framework, and for participating in usability testing of ACSmt, which is reported in [29].

References

[1] S. Al-Sarawi, M. Anbar, R. Abdullah, A. B. Al Hawari, Internet of things market analysis forecasts, 2020–2030, in: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), 2020, pp. 449–453. doi:10.1109/WorldS450073.2020.9210375.

[2] M. W. Maier, Architecting principles for systems-of-systems, Systems Engineering: The Journal of the International Council on Systems Engineering 1 (4) (1998) 267–284.

[3] J. Axelsson, A refined terminology on system-of-systems substructure and constituent system states, in: 2019 14th Annual System of Systems Engineering Conference (SoSE), IEEE, Tampere, Finland, 2019, pp. 31–36.

[4] A. M. Madni, M. Sievers, Model-based systems engineering: Motivation, current status, and research opportunities, Systems Engineering 21 (3) (2018) 172–190.

[5] M. A. Mohamed, M. Challenger, G. Kardas, Applications of model-driven engineering in cyber-physical systems: A systematic mapping study, Journal of Computer Languages 59 (2020) 100972.

[6] P. L. Laursen, V. A. T. Trinh, A. E. Haxthausen, Formal modelling and verification of a distributed railway interlocking system using uppaal, in: International Symposium on Leveraging Applications of Formal Methods, Springer, Rhodes , Greece, 2020, pp. 415–433.

[7] M. Albano, B. Nielsen, Interoperability by construction: code generation for arrowhead clients, in: 2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS), Vol. 1, IEEE, Tampere, Finland, 2020, pp. 429–432. doi:10.1109/ICPS48405.2020.9274746.

[8] S. K. R. Harbo, M. K. Kristensen, E. P. Voldby, S. V. Andersen, F. C. Petersen, M. Albano, Communication oriented modeling of evolving systems of systems, in: 2021 16th Annual System of Systems Engineering Conference (SoSE), IEEE, Västerås, Sweden, 2021, pp. 88–94.

[9] DANSE Consortium, Danse - designing for adaptability and evolution in system of systems engineering., https://web.archive.org/web/20210424073231/http://www.danseip.eu/home/index.html (2015).

[10] DANSE Consortium, Configure danse tool-net environment., https://web.archive.org/web/20210424073233/http://www.danseip.eu/home/configure-danse-tool-net-environment.html (2015).

[11] COMPASS Consortium, Comprehensive modelling for advanced systems of systems., http://www.compass-research.eu/resources/COMPASSdatasheet.pdf (2014).

[12] M. Mori, A. Ceccarelli, P. Lollini, B. Frömel, F. Brancati, A. Bondavalli, Systems-of-systems modeling using a comprehensive viewpoint-based sysml profile, Journal of Software: Evolution and Process 30 (3) (2018) e1878.

[13] F. Oquendo, Formally describing the software architecture of systems-of-systems with sosadl, in: 2016 11th Annual System of Systems Engineering Conference (SoSE), IEEE, 2016, pp. 1–6.

[14] UPPAAL developers, Features of the uppaal model checking tool, https://uppaal.org/features/ (2023).

[15] G. Wainer, Cd++: a toolkit to develop devs models, Software: Practice and Experience 32 (13) (2002) 1261–1306.

[16] Y. Van Tendeloo, H. Vangheluwe, An evaluation of devs simulation tools, Simulation 93 (2) (2017) 103–121.

[17] L. Zhang, Modeling large scale complex cyber physical control systems based on system of systems engineering approach, in: 2014 20th International Conference on Automation and Computing, IEEE, 2014, pp. 55–60.

[18] The Eclipse Foundation, Eclipse papyrus™ modeling environment, https://www.eclipse.org/papyrus/ (2022).

[19] The Eclipse Foundation, Papyrus sysml git, https://git.eclipse.org/c/papyrus/org.eclipse.papyrus-sysml16.git (2022).

[20] T. C. Lethbridge, A. Forward, O. Badreddin, D. Brestovansky, M. Garzon, H. Aljamaan, S. Eid, A. Husseini Orabi, M. Husseini Orabi, V. Abdelzad, O. Adesina, A. Alghamdi, A. Algablan, A. Zakariapour, Umple: Model-driven development for open source and education, Science of Computer Programming 208 (2021) 102665. doi:https://doi.org/10.1016/j.scico.2021.102665.

[21] Various Authors, Modelio the open source modeling environment, https://github.com/ModelioOpenSource/Modelio (2023).

[22] B. Maggi, Papyrus-list, https://github.com/bmaggi/Papyrus-List (2020).

[23] The Eclipse Foundation, Papyrus robotics git, https://git.eclipse.org/c/papyrus/org.eclipse.papyrus-robotics.git (2022).

[24] The Eclipse Foundation, Papyrus rt git, https://git.eclipse.org/c/papyrus-rt/org.eclipse.papyrus-rt.git (2017).

[25] The Eclipse Foundation, Papyrus uml light git, https://github.com/eclipsesource/papyrus-umllight (2019).

[26] The Eclipse Foundation, Papyrus opcua git, https://github.com/model-UA/papyrus-opcua-plugin (2021).

[27] Various Authors, index-org.eclipse.papyrus.git, https://git.eclipse.org/c/papyrus/org.eclipse.papyrus.git/ (2023).

[28] The Eclipse Foundation, Eclipse modeling tools, https://www.eclipse.org/downloads/packages/release/2022-03/r/eclipse-modeling-tools (2022).

[29] E. P. Voldby, J. Madsen, S. K. R. Harbo, A modeling tool for system of systems, https://projekter.aau.dk/projekter/en/studentthesis/a-modeling-tool-for-system-of-systems(b51142c3-e698-41d2-bea6-716b8cc79461).html (2022).

[30] S. K. Remond Harbo, E. Palmelund Voldby, J. Madsen, M. Albano, The acs modeling tool, https://github.com/acs-modeling-tool/acs-modeling-tool (2022).

[31] S. K. R. Harbo, E. P. Voldby, J. Madsen, M. Albano, A diagram-centric modeling tool for systems of systems, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS'22), ACM, New York, NY, USA, 2022, p. 51–55. doi:10.1145/3550356.3559093.

[32] J. Delsing, P. Varga, L. Ferreira, M. Albano, P. P. Pereira, J. Eliasson, O. Carlsson, H. Derhamy, The arrowhead framework architecture, in: IoT Automation, CRC Press, Boca Raton, USA, 2017, pp. 79–124.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: