The Journal of Open Source Software

CommaSuite: Monitoring and Testing of OpenAPI and AsyncAPI Software Interfaces

Ivan Kurtev [1], **Daan van der Munnik**[2], **and Mathijs Schuts** [3]

1 TU/e, The Netherlands **2** Philips, The Netherlands **3** TNO-ESI, The Netherlands

DOI: 10.21105/joss.09069

Software

- Review
- Repository
- Archive

Editor: Richard Littauer

Reviewers:

- @rohanrasane
- @kkmurthyt21
- @abhishektiwari

Submitted: 08 September 2025
Published: 08 October 2025

Summary

A broad spectrum of scientific software engineering research can benefit from the use of precise component descriptions. `CommaSuite`[1] is an open-source research tool designed to support the specification and design of software interfaces and components (Kurtev et al., 2024). It utilizes DSLs (Domain Specific Languages) to model interface and component behavior in an implementation technology agnostic notation. `CommaSuite`'s main capability is to generate a runtime monitor (Falcone et al., 2021) from these models. This monitor enables verification of whether the component interactions conform to the specified behavior. A recent addition to `CommaSuite` includes support for MBT (Model-Based Testing) (Utting et al., 2012) of component implementations. With this capability, test applications are automatically generated from models.

Since 2015, `CommaSuite` has been used in applied research projects within the Dutch high-tech industry. Initially, only proprietary, company-specific interface technologies that are closed-source were supported. In this paper, we present a major new release of `CommaSuite` that includes support for widely used, open-source software interface technologies.

Statement of need

Verification of high-tech systems remains a dynamic and evolving area of research. Key solution approaches are being explored across several research directions such as interface definition languages, runtime monitoring, and model-based testing.

CPSs (Cyber-Physical Systems)–such as autonomous vehicles (Araujo et al., 2023), robots (Caldas et al., 2024), and drones (Islam et al., 2024)–are increasingly driven by AI (Artificial Intelligence) (Rahman et al., 2021). To ensure the safe operation of these AI-enabled systems, runtime monitoring can be employed to detect and respond to deviations from the specified behavior. For systems that do not rely on AI, MBT offers a promising approach for pre-release acquiring confidence in the correctness.

Although `CommaSuite` is based on technology agnostic specification languages, the generated artefacts have to interact or consume data via concrete interface technologies. This requires development of converters and adaptors. For example, the observed messages to and from components are usually interface technology specific and need to be converted to a format understandable by the generated runtime monitor. Similarly, the generated test applications need to use a concrete interface technology to access the intended component under test.

In the 3.0.0 release[2] of `CommaSuite`, support has been added for two widely adopted open-source software interface technologies: OpenAPI and AsyncAPI. OpenAPI (Karavisileiou et al., 2020) is extensively used in both web development and CPSs, and is supported by a broad ecosystem of

[1] https://eclipse.dev/comma/
[2] https://gitlab.eclipse.org/eclipse/comma/comma/-/releases/v3.0.0

tools[3]. AsyncAPI (Gómez et al., 2020) is commonly used in the development of IoT (Internet-of-Things) applications and CPSs, and also benefits from a rich set of supporting tools[4]. By integrating these technologies into the open-source version of CommaSuite, researchers and practitioners can benefit from CommaSuite across a wider range of active research and application domains, and can experiment with the verification of applications based on these technologies.

Functionality

To tailor CommaSuite to specific interface technologies, dedicated translators and generators are required. Figure 1 illustrates the architecture of CommaSuite, which includes DSLs at its core along with the translators and generators that handle technology-specific artifacts. In the following sections, we describe the generation of interface technology-specific specifications, and the translation of Wireshark captures into CommaSuite's input format for runtime monitoring. Additionally, we explain the generation of adapters for CommaSuite's model-based testing application.

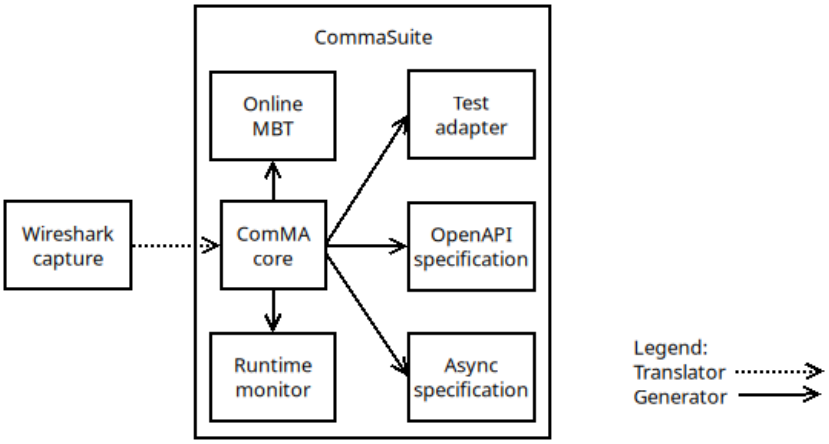

Figure 1: Overview of CommaSuite's translators and generators

Generating Specifications

As illustrated in Figure 1, CommaSuite can generate OpenAPI and AsyncAPI interface specifications directly from CommaSuite models. These specifications are produced in YAML (YAML Ain't Markup Language) format (Evans et al., 2017), which serves as input for the various supporting tools referenced earlier. Since OpenAPI and AsyncAPI share a common format for defining data types, the CommaSuite generator avoids duplication by creating a separate YAML file containing all data type definitions. This shared data types file is then imported into the OpenAPI and AsyncAPI interface specifications.

Runtime Monitoring

Runtime Monitoring (Kurtev et al., 2017; Kurtev & Hooman, 2022) is the main capability of CommaSuite. It includes the interface technology agnostic Events DSL, a core part of CommaSuite, which serves as an input language for the runtime monitor. The Events DSL is used to encode the observed component interactions in terms of exchanged messages. To enable runtime monitoring for OpenAPI and AsyncAPI software interfaces, a dedicated translator has been developed. This allows interactions between components–observable at the

[3] https://tools.openapis.org/
[4] https://www.asyncapi.com/tools

software interface level such as using Wireshark (Jain, 2022)–to be captured and translated into the CommaSuite's Events DSL format.

Model-Based Testing

MBT (Schuts et al., 2025) is another capability of CommaSuite. From CommaSuite models, a test application can be automatically generated. This test application is interface technology agnostic by design. However, to test a SUT (System Under Test) that uses OpenAPI and/or AsyncAPI interfaces, a dedicated test adapter is required. This adapter, which bridges the test application and the SUT, can also be generated from CommaSuite models.

Application at Philips IGT

Earlier versions of CommaSuite have been successfully applied in industrial research projects at Philips IGT. In these projects, both runtime monitoring (Kurtev et al., 2017) and MBT (Schuts et al., 2025) were employed using a proprietary, Philips-specific interface technology.

More recently, the newly added support for OpenAPI and AsyncAPI has been utilized at Philips IGT for the definition, design, implementation, and verification of new software interfaces. This demonstrates CommaSuite's applicability to open-source interface standards in a large, industrial application.

State of the field

There exist a number of tools that support development of APIs such as Postman[5], SwaggerHub[6], AsyncAPI Studio[7], among others. These tools usually support technology-specific notations for specifying the APIs and data schemas, for example, YAML files.

CommaSuite provides interface technology agnostic modeling notation at a higher level of abstraction where the models can be used for multiple purposes. Furthermore, CommaSuite supports the specification of the allowed order of API calls in the form of state machines and the expected timing behavior. While some of the existing tools provide a degree of test automation, in most of the cases the developers still need to develop the test cases manually. CommaSuite via its MBT support enables automatic generation of test cases and test adapters.

It should be noted that our approach does not aim at generating implementation of the OpenAPI and AsyncAPI specifications. Some of the mentioned existing tools provide support for this task.

Documentation

Comprehensive documentation for the newly added OpenAPI and AsyncAPI support is available at https://eclipse.dev/comma/generators/generators.html; navigate to "OpenAPI and AsyncAPI Support" from the left-hand side menu.

The CommaSuite version 3.0.0 release includes an example project that demonstrates how to use CommaSuite with OpenAPI and AsyncAPI software interfaces. The example project provides users with a concrete starting point to explore and apply the described capabilities[8].

Note that the limitations of the described capabilities are part of the referenced documentation.

[5] https://www.postman.com/

[6] https://tools.openapis.org/

[7] https://www.asyncapi.com/tools

[8] See https://eclipse.dev/comma/site/download.html; the example can be obtained via File → New → Example... → Vending Machine Test Application REST Example; the README file of this example contains more information, e.g., on the execution of the SUT and test application, and about creating a Wireshark capture.

Acknowledgements

The research is carried out as part of the "Model-Based Testing with ComMA" program under the responsibility of TNO-ESI in cooperation with Philips. Model-Based Testing with ComMA is funded by Holland High Tech | TKI HSTM via the PPP Innovation Scheme (PPP-I) for public-private partnerships.

We would like to thank Jordi Betting and Dheeraj Kulkarni from Philips for their valuable discussions and feedback on the prototype implementations.

References

Araujo, H., Mousavi, M. R., & Varshosaz, M. (2023). Testing, validation, and verification of robotic and autonomous systems: A systematic review. *ACM Transactions on Software Engineering and Methodology*, *32*(2), 1–61. https://doi.org/10.1145/3542945

Caldas, R., García, J. A. P., Schiopu, M., Pelliccione, P., Rodrigues, G., & Berger, T. (2024). Runtime verification and field-based testing for ROS-based robotic systems. *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/tse.2024.3444697

Evans, C., Ben-Kiki, O., & Net, I. döt. (2017). *YAML ain't markup language (YAML™) version 1.2.*

Falcone, Y., Krstić, S., Reger, G., & Traytel, D. (2021). A taxonomy for classifying runtime verification tools. *International Journal on Software Tools for Technology Transfer*, *23*(2), 255–284. https://doi.org/10.1007/s10009-021-00609-z

Gómez, A., Iglesias-Urkia, M., Urbieta, A., & Cabot, J. (2020). A model-based approach for developing event-driven architectures with AsyncAPI. *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 121–131. https://doi.org/10.1145/3365438.3410948

Islam, M. N. A., Cleland-Huang, J., & Vierhauser, M. (2024). Adam: Adaptive monitoring of runtime anomalies in small uncrewed aerial systems. *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 44–55. https://doi.org/10.1145/3643915.3644092

Jain, V. (2022). *Introduction to Wireshark* (pp. 1–34). Apress. https://doi.org/10.1007/978-1-4842-8002-7_1

Karavisileiou, A., Mainas, N., & Petrakis, E. G. (2020). Ontology for OpenAPI REST services descriptions. *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 35–40. https://doi.org/10.1109/ictai50040.2020.00016

Kurtev, I., & Hooman, J. (2022). Runtime verification of compound components with ComMA. In *A journey from process algebra via timed automata to model learning: Essays dedicated to Frits Vaandrager on the occasion of his 60th birthday* (pp. 382–402). Springer. https://doi.org/10.1007/978-3-031-15629-8_21

Kurtev, I., Hooman, J., & Schuts, M. (2017). *Runtime monitoring based on interface specifications*. Springer. https://doi.org/10.1007/978-3-319-68270-9_17

Kurtev, I., Hooman, J., Schuts, M., & Munnik, D. van der. (2024). Model based component development and analysis with ComMA. *Science of Computer Programming*, *233*, 103067. https://doi.org/10.1016/j.scico.2023.103067

Rahman, Q. M., Corke, P., & Dayoub, F. (2021). Run-time monitoring of machine learning for robotic perception: A survey of emerging trends. *IEEE Access*, *9*, 20067–20075. https://doi.org/10.1109/access.2021.3055015

Schuts, M., Hooman, J., Kurtev, I., Tlili, I., & Oerlemans, E. (2025). Online model-based testing reusing multiple design models in an industrial setting. *Journal of Object Technology*, *24*(2), 2:1–14. https://doi.org/10.5381/jot.2025.24.2.a6

Utting, M., Pretschner, A., & Legeard, B. (2012). A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, *22*(5), 297–312. https://doi.org/10.1002/stvr.456