Dynamic Checks of Evidence Models for Assurance Projects in Eclipse OpenCert

Jabier Martinez[1(✉)] and Ángel Jesús Varela-Vaca[2]

[1] Tecnalia, Basque Research and Technology Alliance (BRTA), Derio, Spain
jabier.martinez@tecnalia.com
[2] IDEA Research Group, University of Seville, Seville, Spain
ajvarela@us.es

Abstract. The modelling of regulatory frameworks and industry standards, including their argumentation and expected evidence, are used during assurance processes to demonstrate the compliance of systems. However, this is handled mainly in a static fashion, and using these models for dynamic evidence checking along the system life-cycle, including operation (checking the model at runtime), is not yet mainstream. This preliminary work shows a tool-supported modelling method for the automatic and dynamic evaluation of evidence. The solution is supported by an Eclipse OpenCert tool extension where the capabilities of evidence models are extended with automatic checks. The user monitoring the assurance project receives alerts when evidence are unsatisfied. It also exports a continuous log of these checks using the XES standard to enable traceability and historical creation of passing and failing checks for analysis and auditing purposes. While some evidence checks are generic, the diversity of checking processes required our solution to be extensible.

Keywords: Safety · Security · Reference frameworks · Dynamic check

1 Introduction

Reference frameworks for systems engineering, such as safety or security regulations, standards, industrial guidelines, or even in-house guidelines, represent a reasonable way to build trust in systems and have confidence in the process of their construction. To make explicit or to audit the compliance to a certain reference framework, assurance projects are conducted. An assurance case is a structured argument supported by a body of evidence, which provides a convincing and valid justification that a system meets its assurance requirements for a given application in a given operating environment [4]. Dynamic evidence checks can be useful for **processes** (e.g., organisational or engineering processes) with the need for continuous checking that the process is followed as established and the expected assets are produced and checked for validity. One specific type of

process is **systems during operation** (e.g., self-adaptive systems) that need to demonstrate or validate correct functioning at runtime.

On the left side of Fig. 1, we illustrate an overview of assurance consisting of three layers. The reference frameworks provides guidance for assurance projects which, at the same time, provide guidance for the systems engineering itself, e.g., the creation of system assets and their characteristics, the verification and validation (V&V), or concrete activities to follow in the organization. The systems engineering project assets are intended to provide traceable evidences to fulfil the assurance needs, and consequently, the compliance to reference frameworks. On the right side of the figure we illustrate the modelling solution proposed in Eclipse OpenCert [6] as a tool for assurance management. Reference frameworks are modelled through the Common Assurance and Certification Metamodel (CACM) [11] with concepts such as requirements, activities, artefacts, roles, levels, and a large set of elements expressive enough to capture diverse reference frameworks. Then, for the assurance layer, two types of models are key, namely assurance case models, and evidence models. Assurance case models uses the Goal Structuring Notation (GSN) [7] and it is similar to the elements proposed in the Structured Assurance Case Metamodel (SACM) [10] such as claims, argumentation, evidences. The evidence models is another part of the CACM to capture the life-cycle of the evidences and more technical details about the evidences than in the assurance cases models. Finally, in the project assets, given the diverse nature and technological stack of each project, we have the actual assets of the systems engineering activities.

Traceability means are possible through the mapping models provided by the tool, and for the case of evidence models and the actual project assets, Uniform Resource Identifiers (URI) are used as attributes in the evidence model elements.

In the scope of this work, we focus on the part inside the square of Fig. 1. The concrete implementation in Eclipse OpenCert of this high-level picture is considered an iterative process where the assurance cases are built and the assurance is managed across the life-cycle. We make the following contributions:

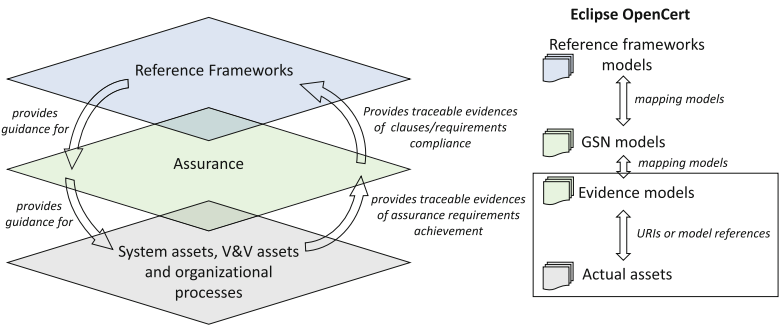

Fig. 1. Overview of reference frameworks, assurance, and project assets, with the corresponding models or expected project assets in Eclipse OpenCert.

- **Evidence checks:** The traceability information of evidence models to actual assets is enriched with an extensible method for their validation. That means that traces are not just pointers anymore, but also a way to automatically check that the pointed elements are valid. As basic example, an evidence can actually point to a file, but the file can be empty or incomplete with respect to the expected content.
- **Dynamic evidence checks:** The automation of evidence checks opens the door for a continuous validation of the evidence models. Our preliminary implementation allows to schedule validations, receive alerts for the user monitoring the assurance project, and exporting a continuous log.
- **Research results availability:** The tool extension is publicly available[1] to make this research reproducible and allow others to build on top of it.

The paper is organised as follows. The approach is detailed in Sect. 2, and the related work is presented in Sect. 3. Finally, a conclusion is drawn and perspectives are outlined in Sect. 4.

2 Approach

Section 2.1 provides technical details about how we integrated automatic evidence checks in the tool, how it is extensible, and it also presents some available generic checks that can be already used. Then, Sect. 2.2 presents how we included this functionality as part of a dynamic evidence checking loop, including the generation of logs.

2.1 Extensible Evidence Checks

Figure 2 shows an illustrative example of an evidence model. An `Artefact Model` contains `Artefact Definition`s with their corresponding `Artefact`s with its version, creation date etc. An `Artefact` can have several `Resource`s where one of the attributes is the location. In this attribute, an URI is expected pointing to the actual resource. An `Artefact` can contain `Assurance Asset Evaluation`s which semantics refer to any assessment of judgement about the asset. This way, each `Assurance Asset Evaluation` has attributes for entering free plain text named `criterion`, `criterion description`, `evaluation results`, and the `rationale`. Metamodel details are in the CACM specification [11].

Instead of modifying the CACM, we decided to reuse the `Assurance Asset Evaluation` concept defining a tool-supported convention. Basically, when the `criterion` match the identifier of a registered automatic check type, the tool will be able to launch an automatic check. If further information is needed (e.g., parameters), they will be obtained from the `criterion description`.

[1] Plugin org.eclipse.opencert.evidence.check at https://gitlab.eclipse.org/eclipse/opencert/opencert/-/tree/release/2.0.

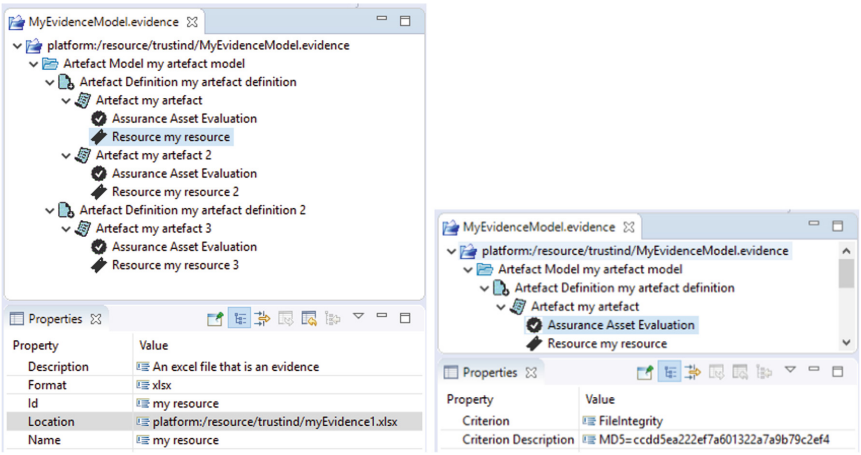

Fig. 2. Example of evidence model.

Eclipse OpenCert was enriched with an extension point to register automatic checks. The interface to be implemented is quite simple but allows complex operations. The method to implement just receive the `Assurance Asset Evaluation` element as input, and this allow to access the `Resource` and its URI, and the `criterion description` if needed. The user of the tool can, on demand, select and launch the evaluation or a set of evaluations.

Implemented checks: The checks can be extremely diverse. That is way extensibility was mandatory. However, a couple of predefined check are included by default: *FileExists*: Used to check that an asset that was considered finished actually exists. In a runtime context, it can be used to check that certain file did not disappear. The URI of the `Resource` is resolved (e.g., URIs using the protocols `file:/` or `platform:/resource/`) and it checks that the file exists. *FileIntegrity*: The monitoring of file integrity is a typical process in security management where certain relevant files (e.g., access control information files, security or system configuration files) are checked against malicious or unintentional modifications. Different hashing algorithms can be used with the content of a file. Java includes MD5, SHA-1, SHA-256 and many others depending on the Java version. There are three ways to use this check type. 1) Using only the id without anything on the description will calculate the hash in the first check, and this hash will be used in all the subsequent checks. Default algorithm is SHA-256. 2) Specifying the hashing algorithm in the description (e.g., MD5, SHA-384). 3) Specifying both the hashing algorithm and the concrete expected hash (e.g., MD5=ccdd5ea222ef7a601322a7a9b79c2ef4). This is preferred to avoid that the integrity of the file was not respected in the first check.

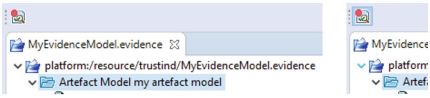

Fig. 3. Artefact model with the toggle button (icon in the upper left corner) for dynamic evidence checking.

2.2 Dynamic Evidence Checks and Log Streaming

Figure 3 shows the toggle button available in the tool to activate or deactivate the dynamic evidence checks of the evidence model. Certain parameters are requested such as the periodicity of the checking and if a continuous log wants to be exported. The background process is launched while allowing to continue using the tool. However, the user is notified when any evidence check fails. Failing a check does not stop the process as the failing messages are included and accumulated in a status window.

The logs are exported in XES (1849-2016 IEEE Standard for eXtensible Event Stream)[2] using the OpenXES library. An example is shown in Listing 1.1. Each check is streamed as an `<event>` with information such as the id of the evaluation, whether it was satisfied, and the timestamp. Each complete validation of the evidence model is contained within a `<trace>`.

Listing 1.1. Example of XES from dynamic evidence checks

```
<?xml version="1.0" encoding="utf-8"?>
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7">
       <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
       <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
       <string key="concept:name" value="Evidence model ID: null Name: my artefact model"/>
       <trace>
              <string key="concept:name" value="instance_0"/>
              <event>
                     <string key="criterion" value="FileIntegrity"/>
                     <boolean key="check" value="true"/>
                     <string key="concept:name" value="Evaluation ID: integrity Name: integrity"/>
                     <date key="time:timestamp" value="2021-09-28T16:04:17.933+02:00"/>
              </event>
              <event>
                  ... more individual checks
              </event>
       </trace>
       <trace>
          ... more complete set of checks of the evidence model
       </trace>
</log>
```

3 Related Work

The system assurance is demonstrated as a crucial discipline for industrial sectors [13] specially in certification for safety-critical systems. Dynamic assurance cases [3,5] were described as one way to enable proactive assurance management. Dynamic safety assurance have been applied in multiple industrial contexts such

[2] https://xes-standard.org/ and https://www.xes-standard.org/openxes/start.

as autonomous systems [1,8], software-intensive systems [14], and embedded systems [9]. These approaches focused on theoretical or conceptual approaches by incorporating components to feed (back), improve or empower the assurance cases, e.g., machine learning components to analyse the operation of real systems. However, other approaches deal with dynamic assurance by continuous assessment, e.g., in software-intensive systems [14]. Most approaches contributed on the level of assurance and evidence models, i.e., GSN and evidence models (e.g., [1,3–5]), while our focus is on the validation of the evidence models against the actual evidence artefacts. Contrary to conceptual frameworks or industrial tools that might cover to some extent what we propose, we focused on the technical aspects to provide a publicly available tool focused on automatic and dynamic checks.

4 Conclusion and Perspectives

Eclipse OpenCert is a tool for assurance management. In this work, the tool has been extended to automate the validation of evidences, and to enable a dynamic validation at runtime. This preliminary study does not intend to demonstrate high conclusions but some goals and achievements in supporting some aspects. In this respect, the tool presented focused on demonstrating the integration of dynamic evidence checks on the Eclipse OpenCert tool following reference frameworks supporting the metamodels of CACM [11].

The future work and perspectives opened by this preliminary work include:

Correctness and validation: The preliminary implementation is functional and available but its design and implementation could be refined, i.e., through automatic tests simulating runtime processes beyond the manual testing we conducted so far, and a real system assurance case with configuration management problematic of many evidences coming from several components.

Further support for context-awareness: We mitigated the complexity in the diversity of systems engineering projects by making the approach extensible with regard to check types. However, the tool only allows to schedule the checks with a user-defined frequency. It can be envisioned that the tool should also be extensible for allowing another kind of context events (e.g., changes in specific assets) to automatically trigger the checks. Regarding the logs, the XES information can end up in large files after a large amount of time. For the moment, it is expected that other tools consume the XES and handle the stream.

Handling the implications of evidence checks: The tool support does not cover important aspects, such as how to respond to the failing checks (e.g., corrective actions) or which are the implications at higher levels. For instance, impact analysis could be performed to identify which argumentations at the assurance case level do not hold any more. Notably, our approach can help to identify rebutting defeaters (an evidence contradicting a claim), thus potentially contributing in the process of dialectic argumentation [2,12]. In this process, the objective will be to defeat the defeater through further argumentation so as to maintain the

assurance case in a valid state. At even higher level, impact analysis can be performed at reference framework level, i.e., which parts of the reference framework is respected, and which are not because of the missing or incorrect evidence. In this sense, the approach can be extended to contribute to checkpointed safety cases for submission to regulators. Currently, the XES logs work directly at evidence model level, but not at higher levels.

Handling valid checks that are actually invalid: Our current approach does not consider evidences that might appear valid through the automatic check, but they are not. For instance, the environment might have changed causing the case to become invalid, or malicious actors can change an actual evidence artefact to promote false confidence that the system is safe when it actually is not. Dialectic argumentation to refine the assurance case (and the evidence model) seems to be a possible solution to alleviate these problems.

Acknowledgment. Jabier Martinez was supported by the TRUSTIND project (Creating Trust in the Industrial Digital Transformation), an Elkartek project funded by the Basque Government. The work of Angel J. Varela-Vaca has been funded by the projects COPERNICA (P20_01224), AETHER-US (PID2020-112540RB-C44/AEI/ 10.13039 / 501100011033), and METAMORFOSIS (US 1381375). We would like to thank the anonymous reviewers for their valuable feedback.

References

1. Asaadi, E., Denney, E., Menzies, J., Pai, G.J., Petroff, D.: Dynamic assurance cases: a pathway to trusted autonomy. Computer **53**(12), 35–46 (2020)
2. Bloomfield, R., Rushby, J.: Assurance 2.0: a manifesto. In: Parsons, M., Nicholson, M. (eds.) Systems and Covid-19: Proceedings of the 29th Safety-Critical Systems Symposium (SSS 2021), pp. 85–108. Safety-Critical Systems Club, York, UK, final draft available as arXiv preprint arXiv:2004.10474 (2021)
3. Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I., Kelly, T.: Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Trans. Softw. Eng. **44**(11), 1039–1069 (2018)
4. Denney, E., Pai, G., Whiteside, I.: Hierarchical safety cases. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 478–483. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_37
5. Denney, E., Pai, G.J., Habli, I.: Dynamic safety cases for through-life safety assurance. In: ICSE 2015, pp. 587–590. IEEE Computer Society (2015)
6. Eclipse: Opencert platform (2022). https://www.eclipse.org/opencert
7. Goal Structuring Notation Standard Working Group: Goal Structuring Notation (2021). https://scsc.uk/gsn
8. McDermid, J.A., Jia, Y., Habli, I.: Towards a framework for safety assurance of autonomous systems. In: Artificial Intelligence Safety (2019)
9. Moncada, D.S.V., et al.: Dynamic safety certification for collaborative embedded systems at runtime. In: Model-Based Engineering of Collaborative Embedded Systems, pp. 171–196. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-62136-0_8
10. OMG: Structured Assurance Case Metamodel (2017). https://www.omg.org/spec/SACM

11. Ruiz, A., et al.: AMASS platform validation, D2.9 (2019). https://www.amass-ecsel.eu/sites/amass.drupal.pulsartecnalia.com/files/D2.9_AMASS-platform-validation_AMASS_Final.pdf
12. The Assurance Case Working Group (ACWG): Assurance case guidance: challenges, common issues and good practice v1 (2021). https://scsc.uk/r159:1
13. Virvou, M., Matsuura, S.: Toward dynamic assurance cases. In: Knowledge-Based Software Engineering: Proceedings of the Tenth Joint Conference on Knowledge-Based Software Engineering, vol. 240 (2012)
14. Zeller, M.: Towards continuous safety assessment in context of DevOps. In: Habli, I., Sujan, M., Gerasimou, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2021. LNCS, vol. 12853, pp. 145–157. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-83906-2_11