# Towards Continuous Assurance Case Creation for ADS with the Evidential Tool Bus

Lev Sorokin*[1], Radouane Bouchekir*[1], Tewodros A. Beyene*[1], Brian Hsuan-Cheng Liao[2], and Adam Molin[2]

[1] fortiss GmbH, An-Institut Technische Universität München, Guerickestraße 25, 80805 München, Germany
{sorokin, bouchekir, beyene}@fortiss.org
[2] DENSO AUTOMOTIVE Deutschland GmbH, Freisinger Str. 21, 85386 Eching, Germany
{h.liao, a.molin}@eu.denso.com

**Abstract.** An assurance case has become an integral component for the certification of safety-critical systems. While manually defining assurance case patterns can be not avoided, system-specific instantiations of assurance case patterns are both costly and time-consuming. It becomes especially complex to maintain an assurance case for a system when the requirements of the System-Under-Assurance change, or an assurance claim becomes invalid due to, e.g., degradation of a systems' component, as common when deploying learning-enabled components.
In this paper, we report on our preliminary experience leveraging the tool integration framework Evidential Tool Bus (ETB) for the construction and continuous maintenance of an assurance case from a predefined assurance case pattern. Specifically, we demonstrate the assurance process on an industrial Automated Valet Parking system from the automotive domain. We present the formalization of the provided assurance case pattern in the ETB processable logical specification language of workflows. Our findings, show that ETB is able to create and maintain evidence required for the construction of an assurance case.



**Keywords:** Assurance Case Maintenance, Safety Assurance, Tool Integration, Automated Driving.

## 1 Introduction

Assurance cases are an integral component of the certification process of safety-critical systems. They are based on a goal-oriented paradigm, which places greater emphasis on explicitly stating safety claims, and supplying an argument along with assurance evidence that needs to be generated to support these claims [6] [13]. In general, such structured arguments and evidence are captured in the form of safety cases, i.e., a comprehensive, defensible, and valid justification of the safety of a system for a given application in a defined operating environment.

* The authors contributed equally to this paper.

The current practice of developing assurance cases is that safety engineers specify manually the arguments that connect higher-level safety claims to low-level assurance evidence. This practice of developing assurance cases is both highly expensive and labor-intensive, as it involves the extensive generation and meticulous maintenance of a substantial amount of evidence. In particular, assurance case development faces multiple challenges encompassing automation, tool integration, assurance distribution, and assurance maintenance. Although the manual definition of assurance case patterns may be unavoidable, the instantiation of system-specific assurance case patterns can be automated through the creation of automated assurance workflows. This automation not only reduces costs but also enhances efficiency by the automatic generation of assurance evidence. Tool integration presents another challenge, as diverse tools used in different phases of assurance case development must be integrated to generate assurance evidence. Yet, certification standards across various domains, such as DO-178C [3] in avionics and ISO 26262 [4] in automotive, encourage the use of complementary tools when a single tool is not sufficient for a given test, analysis, or verification activity. The distribution of assurance across various stakeholders, system components, and platforms adds complexity, requiring careful management to ensure consistency and alignment with overarching assurance goals. In addition, continuous maintenance is required, particularly for systems powered by learning-enabled components (LECs), as updates to assurance artifacts, e.g., data and models, occur frequently. This necessitates an efficient maintenance procedure that monitors all claims and assurance evidence affected by these changes. Moreover, such a procedure should aim to minimize maintenance costs by selectively and incrementally updating only the portions of the assurance case impacted by these changes.



In this paper, we report on our experience in the development of an assurance case for an industrial Automated Valet Parking (`AVP`) system from the automotive domain. Our focus is on utilizing `ETB`[5][9] to address the previously mentioned challenges. Specifically, the automation of evidence generation and claims maintenance. In that regard, we illustrate the steps involved in creating an assurance case for the AVP system. This encompasses the assurance case pattern creation, formalization of the predefined assurance patterns, and considerations related to assurance distribution and maintenance. Additionally, we highlight the lessons learned through our experience leveraging an automated tool-chain for assurance case generation.

The rest of the paper is structured as follows. Section 2 presents the `AVP` case study. In section 3, we describe the assurance case development for the `AVP` system and the use of the framework `ETB` for assurance case generation. In Section 4 we discuss the lessons learned from our study and present in Section 5 related work. Finally, we provide concluding remarks and pointers to future work in Section 6.

[3] Software Considerations in Airborne Systems and Equipment Certification

[4] ISO 26262 Road vehicles Functional safety.

[5] `https://git.fortiss.org/etb2/etb2`

## 2 The AVP Case Study

This section presents our AVP case study, which is based on an AVP System developed in the FOCETA project[6]. We use the AVP system to demonstrate the usage of ETB for the construction and maintenance of an assurance case.

**System Description.** The AVP System is a feature added to a car that allows to autonomously park the car in an empty parking spot [7]. In particular, the car is dropped off by the driver in a designated zone, where AVP takes over the car, computes the trajectory to a free parking spot and parks the vehicle in the spot.

The architecture of the system, which consists of several components, is shown in Fig. 1. We give a brief description of the components: The *Planning Component* (PC) calculates a feasible and collision-free path for the automated vehicle given the locations of the drop-off zone and a designated parking spot. The *Path Follower* (PF) controls at every time stamp whether the vehicle follows a pre-calculated static path. During this operation, the *emergency brake* is triggered whenever there is an obstacle being recognized by the *Object Detector* (OD) within a safety distance from the vehicle. The path follower and the emergency brake together form the *Control Component* (CC) of the AVP system. To allow the save operation of AVP system, the following main safety requirement should be fulfilled [11]: *"REQ: The ego vehicle shall not collide with pedestrians, unless its velocity is zero."*.

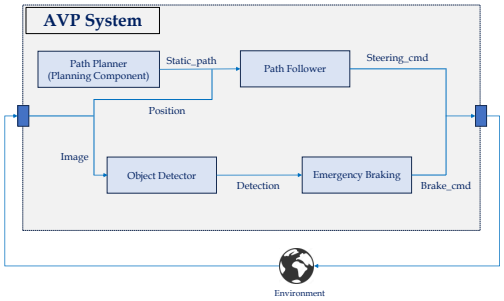



Fig. 1: Overview of the architecture of the automated valet parking system.

**Safety Assurance Challenges.** However, maintaining an assurance case for AVP system, which claims that *REQ* is satisfied faces the following challenges:

(i) *Dynamic Assurance*: The AVP system contains a LEC, i.e. the OD component for the perception of other actors and objects. LECs are in general complex systems with large and multidimensional input spaces, whose correct behavior is difficult to be verified. In particular, it is not possible to know how these systems behave for any possible input when they are deployed on the street. Therefore, assuring the safety of a system whose decisions are based on LECs is complex. For that reason, a continuous engineering process is a common procedure in the automotive domain [5], where the system gets updated even after deployment, when for instance more operational data is collected to retrain the LEC model of the OD to improve the models' performance. However, when systems components are updated, such as the OD, an updated assurance case has to be recreated.

[6] https://www.foceta-project.eu/

This recreation can be time and cost-intensive and requires a validation that all claims are considered for the assurance of *REQ*.

(ii) *Tool Integration:* Construction of an assurance case for AVP involves the usage of different tools, encompassing formalization tools [4], verification tools [1, 2, 15], or testing-related tools [3, 20]. In particular, the certification standard ISO 26262 in automotive, encourages the use of complementary tools when a single tool is not sufficient for a given test, analysis, or verification activity. The correct orchestration of these tools is manually possible but difficult due to the following reasons: 1) artifacts which that tools generate have to be manually maintained/tracked, 2) the exchange of data between tools is hard coded and have to be reimplemented from scratch, when tools are replaced by other tools.

(iii) *Distribution*: Cyber-physical systems such as AVP are in general based on different components that are developed by distinct organizational entities. Maintaining an assurance case, requires the collection of evidence from different sources to construct an assurance case for the complete system. Also, employed tools can have different requirements on the infrastructure, they are deployed on which necessitates a distributed setup. For instance, testing tools require resource-intensive simulation environments, while verification tools may not.

Given theses challenges, an automated support for the continuous maintenance of the safety assurance throughout the lifecycle of AVP is important to save operational costs and guarantee that all envisioned claims have been collected.



## 3 Assurance Case Development Using ETB

In this section, we illustrate the development of the assurance case for the AVP system using the framework ETB. ETB is developed for the execution of distributed transactions and has been already applied to resolve the challenge of automating software certification workflows for the creation of assurance cases [9, 17, 19]. It enables an end-to-end, decentralized, and continuous safety assurance process where multiple entities are involved to establish safety claims supported by evidence. We outline how to use the framework ETB to establish the assurance case for the AVP system. Specifically, we commence by introducing our assurance case pattern developed for the AVP system. Then, we describe the formalization of the assurance case pattern in the language supported by ETB, followed by the tool integration. Finally, we sketch the distributed creation and incremental maintenance of assurance cases.

### 3.1 Assurance Pattern Creation

Let us consider a top-level assurance goal for the AVP system related to the safety requirement *REQ*. Although the primary goal here is not to develop a complete assurance case for the entire AVP system, in Fig.2, we highlight three fragments of the argument pattern corresponding to various abstraction levels of the system with respect to the specified safety goal.

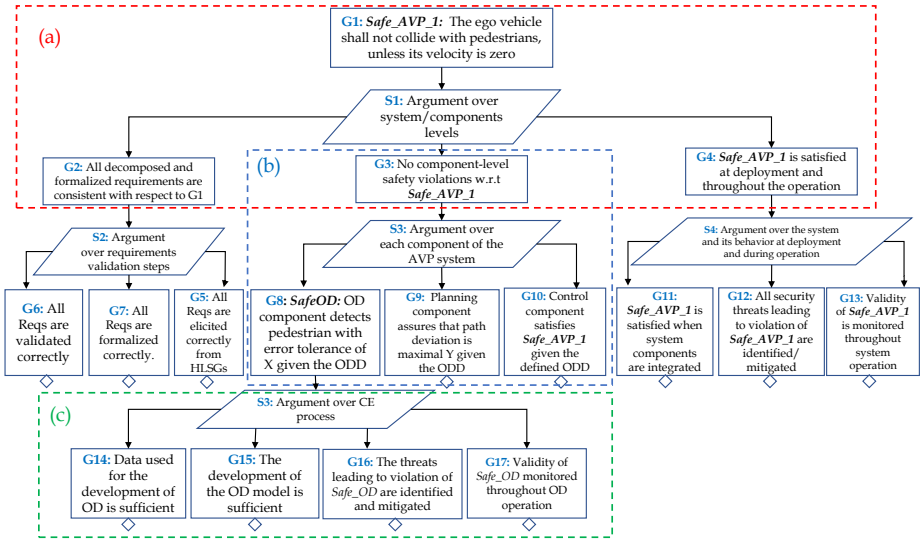



Fig. 2: Top-level of GSN-based pattern for safety case construction for AVP system.



The first argument pattern, which is shown in GSN-like [7] notation in Fig. 2(a), divides the top-level goal, which is labeled as $G_1$, into three sub-goals, $G_2$, $G_3$ and $G_4$, where $G_2$ targets the validation of the input requirements, and, $G_3$ and $G_4$ target the safety of the individual components and the overall system, respectively. The second argument pattern is dedicated to composing assurance arguments from each component of the AVP system. As shown in Fig. 2(b), its three sub-goals, $G_8$, $G_9$ and $G_{10}$, target OD, PC, and CC of the AVP system. The third argument pattern, shown in Fig. 2(c), targets the OD component. Various recent approaches for the safety assurance of LECs propose safety assurance patterns over the LECs life-cycle [13, 18, 23]. Building on such approaches, our argument pattern for the OD component targets stages such as data assurance, model assurance, verification assurance, and run-time assurance. As shown in Fig. 2(c), its sub-goals, $G_{14}$, $G_{15}$, $G_{16}$ and $G_{17}$, target design-time (data, model and verification) and run-time assurance methods and evidence for the OD component. The three argument patterns illustrate how assurance patterns can be constructed for the AVP system at different granularity levels by applying appropriate strategies.

### 3.2 Formalisation of Assurance Argument Patterns

As next, developed argument patterns should be specified as so called *workflows* and formalized in Datalog, which is the scripting language supported by ETB [8]. For example, the top-level argument pattern (see Fig. 2) is formalized using the

[7] In this work, we have only considered the Goal and Strategy elements as well as the Supported-By relation of GSN in our assurance case fragments.

Datalog program that is given in Fig. 3. The head predicate is the top-level goal $G_1$, and its body contains each sub-goal of the argument pattern as a conjunct.

```
g1_safe_AVP(SUA, Reqs, ODD, Datasets, Specs,  RepSafOD, RepSafPC,
            RepSafCC, ScenariosSBT, CrTests, RepCSM, RepME) :-
    subcomponents(SUA, [OD, PC, CC]),
    g2_reqs(Reqs, Specs),
    g3_safe_components([OD, PC, CC], Specs, ODD, Datasets, RepSafOD, RepSafPC, RepSafCC),
    g4_safe_system(SUA, Specs, ODD, ScenariosSBT, CrTests, RepCSM, RepME).
g2_reqs(Reqs, Specs) :-
    g6_req_validation(Reqs),
    g7_req_formalisation(Reqs, Specs).
g3_safe_components([OD, PC, CC], Specs, ODD, Datasets, RepSafOD, RepSafPC, RepSafCC) :-
    g8_safe_perception(OD, Specs, ODD, Datasets, RepSafOD),
    g9_safe_planning(PC, Specs, RepSafPC),
    g10_safe_steering(CC, Specs, RepSafCC).
g4_safe_system(SUA,Specs,ODD,ScenariosSBT,CrTests,RepCSM,RepME):-
    g11_scenario_based_testing(SUA,Specs,ODD,ScenariosSBT,CrTests),
    g12_cyber_security_monitoring(SUA, Specs, RepCSM),
    g13_monitoring_and_enforcement(SUA, Specs, RepME).
```

Fig. 3: Datalog formalization of the complete assurance pattern.



Note, that the goal predicates in the Datalog rule contain all the variables the goal depends on and a descriptive identifier, and not just a goal index like the case for the assurance pattern. For instance, $G_1$ in the assurance pattern is formalized as the following Datalog predicate:

```
g1_safe_AVP(SUA, Reqs, ODD, Datasets, Specs,RepoSafOD, RepoSafPC,
        RepoSafCC, ScenariosSBT, CrTests, RepCSM, RepME)
```

In this predicate, the string `g1_safe_AVP` is a descriptive post-fix of the goal name, and the parameters consist of the inputs such as `SUA`, `Reqs`, `ODD`, `Datasets` as well as resulting evidence artefacts like `CrTests`, which is a set of considered to be critical test cases identified by the system-level testing method. We illustrate the description of the parameters used in the datalog formalization in table 1.

### 3.3 Tool Integration

In this step, tools that provide evidence artefacts during the actual creation of assurance case are integrated into `ETB`. Following the common practice in tool integration frameworks, `ETB` provides a wrappers API that automatically generates wrapper templates that can be customized by end-users for each tool. As an illustration, the tool `OpenSBT`, designed for the search-based testing (SBT) of automated driving systems [20], is offered as a service through the Datalog predicate `g11_scenario_based_testing(SUA, Specs, ODD, ScenariosSBT, CrTests)`. In this predicate, `ScenariosSBT` and `CrTests` are artefacts generated by `OpenSBT` and represent respectively failing test inputs and corresponding simulation traces, which contain positions, the velocity of actors over time.

Table 1: Description of the parameters used in the Datalog formalization. `I` refers to input variables, `O` refers to output variables.

| Parameter | Description | I/O | Parameter | Description | I/O |
|---|---|---|---|---|---|
| SUA | System Under Assurance | I | RepSafPC | Report on safe PC | O |
| Reqs | Requirements | I | RepSafCC | Report on Safe CC | O |
| ODD | Operational Design Domain | I | ScenariosSBT | ScenariosSBT generated by system based testing | O |
| Datasets | Datasets used for training, testing, and validating OD | I | CrTests | Critical test cases generated by simulation-based testing | O |
| Specs | Formal specifications of Reqs | O | RepCSM | Report on cyber security monitoring | O |
| RepSafOD | Report on Safe OD | O | RepME | Report on AVP monitoring and enforcement | O |

In ETB each tool can be evaluated under so-called one or more *modes*. A mode specifies which variables serve as input or output of the corresponding tool. For example, the mode used for the predicate `g11_scenario_based_testing(SUA, Specs, ODD, ScenariosSBT, CrTests)` is `(+,+,+,-,-)`. That means that the first input argument, which is the `SUA`, is of type string and holds the path to the system-under-assurance to be tested in the SBT tool, the second and third input arguments are files that capture the specifications and ODD constraints, while the two last arguments are files generated by `OpenSBT`. An example of a wrapper used to invoke `OpenSBT` is shown in Fig. 4.



```
public class OpenSBTWRP extends OpenSBTETBWRP {
  @Override
  public void run() {
    if (mode.equals("+++--")) {
      //1. Invoke OpenSBT
      String suaPath = arg1;
      File specs = new File(arg2);
      File oddFile = new File(arg2);
      String odd = getOddFromFile(oddFile);
      String openSBT_CMD ="bash interface.sh "+ suaPath + " "+ specs + " "+ odd;
      String openSBT_Outputs = this.runCMD0(openSBT_CMD);
      //2. Evidence generation
      String[] paths = openSBT_Outputs.split("\n");
      String criticalTcFilePath = this.workSpaceDirPath+"CriticalTC";
      this.createLocalFile(paths[0], criticalTcFilePath);
      this.arg3 = criticalTcFilePath;
      arg4 = new ArrayList<String>();
      this.createLocalFiles(paths, arg4);
      //Add claim to claimDB
      this.addClaimPredicate();
    }
    ...
}
```

Fig. 4: Implementation of a wrapper for the integration of `OpenSBT` into `ETB`.

### 3.4 Distributed Assurance

The usage of ETB enables the creation of both assurance cases and evidence artefacts by executing the Datalog workflows in a distributed setting. Practically, a network of so called ETB nodes can be defined where each node can create assurance cases or evidence artefacts depending on the type of the workflow it contains. In particular, each entity that wants to contribute to the assurance case - by providing a tool - has to deploy an ETB node. When a workflow is given to an ETB node, it automatically identifies tools of other nodes which can contribute with their evidence to the overall assurance case.

As an example, consider the usage of a tool that helps to adapt the ODD at the runtime of the AVP system. In particular, this tool derives at runtime of AVP constraints from execution traces where the system behaves critically or not. Such tool can be for instance HyTeM [4]. The derived constraints, i.e., the updated ODD, serve as an input for the system-level-testing tool OpenSBT (s. described in the previous section). While OpenSBT, is a testing tool that needs a comprehensive simulation environment with high resource usage, HyTeM requires considerably less resources and can be deployed in a different environment. The collection of distributed evidence provided by HyTeM and OpenSBT can be managed by ETB by the deployment of two ETB nodes: where one ETB node provides evidence from OpenSBT, while another ETB node is deployed on a different platform.



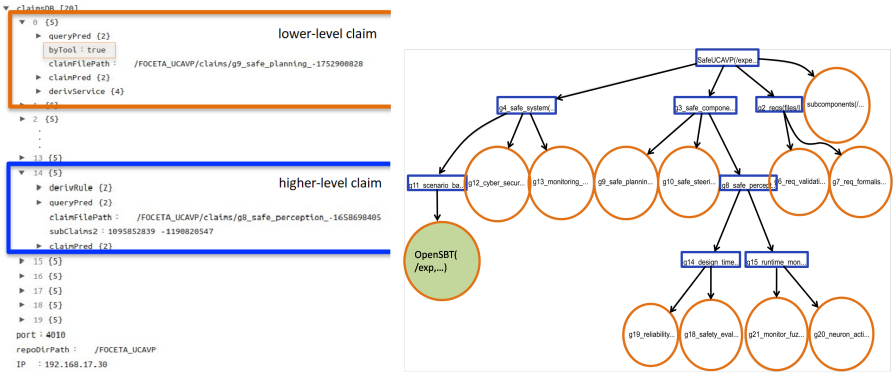
### 3.5 Automated Safety Case Creation

ETB provides a top-down left-to-right Datalog engine that automatically executes the assurance workflow specified in Datalog. A datalog claim that can be instantiated, corresponds to an assurance claim with sufficient evidence. By the end of the execution of the assurance workflow, ETB returns either the list of claims, i.e., a complete assurance case or a counter-example that points to the failed step (sub-goal or tool execution) of the assurance workflow. The established claims, sub-claims, and evidence, which compose the assurance case for the AVP system, are given in Fig. 5. Fig. 5 a) shows the claims in the database in ETB and Fig. 5 (b) shows the corresponding GSN of the safety case.

As visualized in Fig. 5(a), the created claims can be further categorized into two classes, namely *high-level claims* and *low-level claims*. While high-level claims correspond to the goals in the argument patterns that are supported by a set of sub-claims and a connecting argument, low-level claims are directly supported by evidence artifacts. For instance, the highlighted claim in green in Fig. 5(b) corresponding to the goal $\mathtt{G}_{11}$, is supported by the evidence generated by OpenSBT, as depicted in Fig. 6. ETB stores all these claims in a claims table and keeps track of all relevant and integrated tools and workflows w.r.t a given claim.

### 3.6 Assurance Case Maintenance

One defining feature of the proposed process is continuous assurance case maintenance, where updates to the SUA, its requirements, or verification plans are




(a) (b)

Fig. 5: a) shows an excerpt of the claim database of ETB in JSON format and b) shows an overview of corresponding claims in GSN-based format.

```
⋯   => query: openSBT(/experiments/PedestrianCrossing/Leuven_AVP_ori/Demo_AVP.pb, odd.txt-null,
    CriticalTc, Traces)++--
    -> no matching workflow
    -> [WARNING] processing via workflows not successful
             -> invoking a local service
             -> invoking a user service
    -> [claim] : openSBT(/experiments/PedestrianCrossing/Leuven_AVP_ori/Demo_AVP.pb, odd.txt-
    8eded3cc64b062d93303afdf97e6400b1e9159d0, claimFiles/openSBT_1924485658CriticalTC-
    2322744390fc63916ef12096e91797145ea8c809, [claimFiles/openSBT_1924485658/traces_1-
    315f778a226d8483224ecb5bb94922c0d6604ce0, claimFiles/openSBT_1924485658/traces_2-
    018afaa76b143846436f9852bdff85557b7ad5af, claimFiles/openSBT_1924485658/traces_3-
    315f778a226d8483224ecb5bb94922c0d6604ce0, claimFiles/openSBT_1924485658/traces_4-
    f1d3583b0d650b5564294330fa33086f02f85ee9, claimFiles/openSBT_1924485658/traces_5-
    315f778a226d8483224ecb5bb94922c0d6604ce0, claimFiles/openSBT_1924485658/traces_6-
    f8d4c22c0cb7591afc4b0253350cfde8afb7c491, claimFiles/openSBT_1924485658/traces_7-
    315f778a226d8483224ecb5bb94922c0d6604ce0, claimFiles/openSBT_1924485658/traces_8-
    f8d4c22c0cb7591afc4b0253350cfde8afb7c491, claimFiles/openSBT_1924485658/traces_9-
    f8d4c22c0cb7591afc4b0253350cfde8afb7c491, claimFiles/openSBT_1924485658/traces_10-
    b431b9a1e96c96dba5f5a96b229fe7c81a579686])++++
    -> [✓] claim added
```

Fig. 6: Console output of ETB after triggering OpenSBT. OpenSBT identifies critical test inputs with corresponding traces whose path is given on the right half of the Figure. These artefacts serve as evidence and are managed internally by ETB using a hash identifier as shown on the left part of the Figure.

anticipated. In the event of such updates, outdated assurance artefacts including assurance cases and evidence artefacts are incrementally maintained, i.e., only a minimal set of maintenance actions are identified and executed. The `ETB` framework enables such assurance process by continuously executing relevant Datalog programs for outdated assurance cases and by continuously invoking verification tools for outdated assurance artefacts.

Let us consider the top-level claim `g1_safeAVP` created for the `AVP` system. A common practice involves leveraging operational data to refine LECs, facilitating adjustments/improvements based on the evolving operational environment. Specifically, operational data plays a crucial role in refining the requirements and ODD [21]. Yet, any updates to the requirements render the existing safety case

```
g1_safe_AVP(SUA, Reqs, ODD, Datasets, Specs,  RepSafOD, RepSafPC,
            RepSafCC, ScenariosSBT, CrTests, RepCSM, RepME) :-
    subcomponents(SUA, [OD, PC, CC]),
    g2_reqs(Reqs, Specs),
    g3_safe_components([OD, PC, CC], Specs, ODD, Datasets, RepSafOD,
    RepSafPC, RepSafCC),
    g4_safe_system(SUA, Specs, ODD, ScenariosSBT, CrTests, RepCSM,
    RepME).
```

Fig. 7: Example illustrating the incremental maintenance procedure after changing the `Reqs` variable. Directly impacted goals and indirectly impacted goals are respectively marked in red and yellow.

obsolete, and the generation of new claims and evidence becomes imperative. `ETB` applies its lightweight static dependency computation procedure to compute a sub-tree of the assurance case impacted by such update. The procedure is applied to the formalized assurance pattern that was executed during the establishment of the top-level claim.

In particular, the maintenance procedure distinguishes between two types of goals: *directly impacted goals* and *indirectly impacted goals* (s. Fig. 7). Directly impacted goals, such as $\mathtt{G}_1$, $\mathtt{G}_2$, $\mathtt{G}_6$ and $\mathtt{G}_7$ are always re-run to re-establish their corresponding claims. However, indirectly impacted goals, which are $\mathtt{G}_3$, $\mathtt{G}_4$, $\mathtt{G}_8$, $\mathtt{G}_9$, and $\mathtt{G}_{10}$, may not need re-running if the re-running of the directly impacted goals did not result in an update to their assurance artefact.



## 4 Discussion

In this section, we discuss on the experience of using the Evidential Tool Bus for assurance case maintenance and the limitations of `ETB`, when generating assurance cases for automated driving systems. The case study reveals that the `ETB` framework provides computer-aided support for developing assurance cases, where it enables the automated execution of assurance workflows, fostering scalability in assurance evidence generation and maintenance. Additionally, `ETB` facilitates the distributed execution of assurance workflows, along with the orchestration and integration of various tools utilized in evidence generation.

However, we note that the specification, and validation of these Datalog-based workflows have to be done manually by the user. From our experience, the concrete manual specification of assurance workflows is manageable for simple workflows, as Datalog is a declarative specification language with a simple syntax. However, a comprehensive assurance case may involve a significant number of nodes to cover further aspects of the system's functionality such as fault tolerance, and to provide a sufficient analysis. Further, in our example, we did not prove the validity of single claims, such as, e.g., how confident the system-level testing tool `OpenSBT` is that a test case is critical or that no further failing tests exist. However, confidence scores can be similarly incorporated into the workflow specification and modeled as an argument in a Datalog predicate of the corresponding tool.

## 5 Related Work

This section provides a brief overview on research works on assurance cases and dynamic assurance case maintenance [22]. Recently, there has been work on assurance case pattern selection [13, 18], and assurance case pattern instantiation [10, 14, 16, 23]. We focus on the latter one, as this is the main challenge considered in our paper.

Hawkins et al. [13] have presented a methodology for the instantiation of assurance cases for autonomous systems containing ML components. The methodology comprises a set of safety case patterns and a process for (1) systematically integrating safety assurance into the ML component life cycle, as well as (2) generating the evidence base for explicitly justifying the acceptable safety of integrated ML components. While AMLAS covers several stages of the ML lifecycle, computer-aided support to automate the assurance case construction as well as its maintenance in case of requirements or system changes, as supported by `ETB`, is not discussed. In contrast, a recent work [19] proposed to "automate" the evaluation of software assurance evidence to enable certifiers to determine rapidly that system risks are acceptable. They adopted a tool-based approach to the construction of software artifacts that are supported by rigorous evidence. This concept is very important, especially for certification, where it is desirable that arguments representing safety assurance can be re-playable.

Ramakrishna et al. [16] developed the tool ACG for the automated assurance case generation, given a manually curated evidence store. The evidence store is populated with evidence artifacts, which are automatically generated from the system model architecture. While ACG automatizes the safety case instantiation, it lacks a mechanism to support dynamic safety assurance as enabled by `ETB`. The toolset AdvoCATE [10] supports the development of assurance cases and has been applied to a use case for swift unmanned aircraft system but also here no assurance case maintenance is possible.



## 6 Conclusion

In this paper, we presented the utilization of the `ETB` framework as computer-aided support for the development of an assurance case for an automated driving system. We illustrated the formalization of the assurance case as a Datalog program and have shown the integration of a testing tool to provide evidence for the argumentation of the safety of the system at the system level. In addition, we outlined the distributed execution of tools integrated with `ETB` to cope with different levels of interoperability of tools and heterogeneity of providers, as well as described how the incremental assurance is supported by `ETB`.

In our future work, we plan to extend our study incorporating all tools required to instantiate a complete assurance case for the `AVP` system. Further, we are working on an approach that enables to generate ETB workflows, i.e., Datalog specifications, from assurance case patterns represented as GSN to facilitate the application of ETB for the assurance case creation. Furthermore, we plan to extend `ETB` to support a comprehensive confidence argumentation [12].

## Acknowledgments

This work is part of FOCETA project that has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement N. 956123.

## Bibliography

[1] CppCheck, https://github.com/danmar/cppcheck
[2] Infer, https://fbinfer.com/
[3] Prescan, plm.sw.siemens.com/en-US/simcenter/autonomous-vehicle-solutions/prescan/
[4] Bartocci, E., Mateis, C., Nesterini, E., Ničković, D.: Mining hyperproperties using temporal logics. ACM Trans. Embed. Comput. Syst. **22**(5s) (sep 2023). https://doi.org/10.1145/3609394, https://doi.org/10.1145/3609394
[5] Bensalem, S., Katsaros, P., Ničković, D., Liao, B.H.C., Nolasco, R.R., Ahmed, M.A.E.S., Beyene, T.A., Cano, F., Delacourt, A., Esen, H., Forrai, A., He, W., Huang, X., Kekatos, N., Könighofer, B., Paulitsch, M., Peled, D., Ponchant, M., Sorokin, L., Tong, S., Wu, C.: Continuous engineering for trustworthy learning-enabled autonomous systems. In: Steffen, B. (ed.) Bridging the Gap Between AI and Reality. pp. 256–278. Springer Nature Switzerland, Cham (2024)
[6] Bishop, P., Bloomfield, R.: A methodology for safety case development. In: Safety and Reliability. vol. 20, pp. 34–42. Taylor & Francis (2000)
[7] Bosch: Automated valet parking, https://www.bosch-mobility.com/de/loesungen/parken/automated-valet-parking/
[8] Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. **1**, 146–166 (1989)
[9] Cruanes, S., Hamon, G., Owre, S., Shankar, N.: Tool integration with the evidential tool bus. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 275–294. Springer (2013)
[10] Denney, E., Pai, G.: Tool support for assurance case development. Automated Software Engineering **25**(3), 435–499 (2018)
[11] Esen, H., Liao, B.H.C.: Simulation-based safety assurance for an avp system incorporating learning-enabled components (2023)
[12] Hawkins, R., Kelly, T., Knight, J., Graydon, P.: A new approach to creating clear safety arguments. In: Advances in Systems Safety: Proceedings of the Nineteenth Safety-Critical Systems Symposium, Southampton, UK, 8-10th February 2011. pp. 3–23. Springer (2011)
[13] Hawkins, R., Paterson, C., Picardi, C., Jia, Y., Calinescu, R., Habli, I.: Guidance on the assurance of machine learning in autonomous systems (amlas) (2021)
[14] Kaur, R., Ivanov, R., Cleaveland, M., Sokolsky, O., Lee, I.: Assurance case patterns for cyber-physical systems with deep neural networks. In: International Conference on Computer Safety, Reliability, and Security. pp. 82–97. Springer (2020)
[15] Liao, B.H., Cheng, C., Esen, H., Knoll, A.: Are transformers more robust? towards exact robustness verification for transformers. In: SAFECOMP 2023. vol. 14181, pp. 89–103 (2023)
[16] Ramakrishna, S., Hartsell, C., Dubey, A., Pal, P.P., Karsai, G.: A methodology for automating assurance case generation. CoRR **abs/2003.05388** (2020), https://arxiv.org/abs/2003.05388
[17] Ruess, H., Shankar, N.: Evidential transactions with cyberlogic (2023)
[18] Schwalbe, G., Knie, B., Sämann, T., Dobberphul, T., Gauerhof, L., Raafatnia, S., Rocco, V.: Structuring the safety argumentation for deep neural network based perception in automotive applications. In: Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops. pp. 383–394. Springer (2020)
[19] Shankar, N., Bhatt, D., Ernst, M., Kim, M., Varadarajan, S., Millstein, S., Navas, J., Biatek, J., Sanchez, H., Murugesan, A., et al.: Descert: Design for certification (2022)
[20] Sorokin, L., Munaro, T., Safin, D., Liao, B.H.C., Molin, A.: OpenSBT: A modular framework for search-based testing of automated driving systems. In: Tool Demonstration Track ICSE'24
[21] Tonk, A., Boussif, A., Beugin, J., Collart-Dutilleul, S.: Towards a specified operational design domain for a safe remote driving of trains. In: Proceedings of the 31st European Safety and Reliability Conference, Angers, France. pp. 19–23 (2021)
[22] Warg, F., Blom, H., Borg, J., Johansson, R.: Continuous deployment for dependable systems with continuous assurance cases. In: 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 318–325 (2019). https://doi.org/10.1109/ISSREW.2019.00091
[23] Wozniak, E., Cârlan, C., Acar-Celik, E., Putzer, H.J.: A safety case pattern for systems with machine learning components. In: Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops. pp. 370–382. Springer (2020)