Towards Automated Safety Case Generation from System Architecture[*]

Keaton Hanna, BS; Software Engineering Institute; Pittsburgh, PA, USA

Sam Procter, Ph.D.; Software Engineering Institute; Pittsburgh, PA, USA

Keywords: Architecture Analysis and Design Language (AADL), Goal Structuring Notation (GSN), Model-Based Engineering (MBE), Architecture Led Incremental System Assurance (ALISA), System Safety

Abstract

Safety-critical systems require rigorous testing before they can be certified and used. A common mechanism for communicating safety information is a safety case, which can be created using languages like Goal Structured Notation (GSN) with tools such as AdvoCATE. Safety cases are essentially structured arguments: it is important that they can be easily communicated to stakeholders that may not be familiar with the particulars of the developer's methodology. One such methodology is model-based engineering, where models are built and analyzed prior to a system's construction. The Architecture Analysis and Design Language (AADL) and the Open Source AADL Tool Environment (OSATE) can be used for the model-based development of safety-critical, embedded systems.

We propose a process where developers describe their system's architecture in AADL, use supporting analyses that come with OSATE (for, e.g., safety or timing), and then export their safety argument to GSN and AdvoCATE. We describe an extension to OSATE that generates GSN models using a mapping of elements from previously-developed requirement specification and verification languages. These languages were designed to interoperate with AADL and OSATE, but were not previously aligned with GSN-based safety argumentation. Generated safety cases can then be imported into AdvoCATE for review by certification authorities.

1. Introduction

Ensuring that safety-critical systems, which can cause harm to humans should something go wrong, are acceptably safe is increasingly important as automation becomes more common. It is also, however, increasingly challenging because those systems require additional complexity to support the new automation. Many of these systems are required to be reviewed by regulatory agencies, such as (in the United States) organizations like the Federal Aviation or Food and Drug Administrations. In order to support these reviews, structured safety arguments, termed safety cases, are sometimes used (United States Federal Aviation Administration, 2018)(AAMI Infusion Device Committee, 2019). Safety cases are often developed manually, without automated links to a system's architecture, which can make their creation and maintenance during system development particularly labor-intensive.

Safety cases generally position a system within its operating environment, which is necessary for contextualizing the hazards that are addressed. The top level goal of a safety case will typically

[*] This version of the article has been accepted for publication. The official publication is or will be available as part of the International System Safety Conference 2022 proceedings at https://system-safety.org

state that a system achieves the safety of a property deemed to be critical to a system, possibly subject to some assumptions or with some acceptable likelihood. The rest of the argument uses a body of evidence that proves this top level goal as true (Rushby, 2021). Though there are several formats for specifying safety cases, goal structuring notation (GSN) is well-established with both academic and industrial users. There are several tools for specifying safety arguments in GSN; one popular software package is NASA's AdvoCATE (Denney & Pai, 2018).

Model-based system engineering (MBSE) has emerged as a way to address the increasing complexity of modern systems. It is a technique in which, rather than building the system and evaluating it directly, system developers build models of their system and then use those models to evaluate the system's various capabilities and properties. The Architecture Analysis and Design Language (AADL) (SAE AS-2C Architecture Analysis and Design Language Committee, 2022) and its toolset, the Open Source Architecture Tool Environment (OSATE), are one MBSE technology that allows system developers to model, refine, and run analyses on their model. There are a range of analyses available, including several based on the Architecture-Led Incremental System Assurance (ALISA) framework, which extends OSATE to add support for, e.g., the specification of requirements, claims that the system meets those requirements, verification plans to establish those claims (Delange, Feiler, & Neil, 2016). These requirements express many of the same safety-related concepts that would be documented in a system's safety case, e.g., claims about safety properties the system upholds or how the enforcement of those properties is performed. What's more, they are directly traceable to the system's architecture.

In this paper, we describe preliminary efforts connecting requirements specifications in ALISA to safety cases in GSN through AdvoCATE. Specifically, we make the following contributions:

1. ALISA-GSN Mapping: We propose a mapping from the requirements and system claim specification languages of the ALISA framework to individual elements in GSN.
2. ALISA-GSN Translator: We describe an open-source translator which implements the mapping. It takes as input ALISA models in OSATE and produces as output GSN models in AdvoCATE.

It is our goal that by connecting OSATE with a tool such as AdvoCATE, safety cases with strong traceability links to a system's architecture will be easier to create and maintain.

The rest of this work is organized as follows: in Section 2 we provide a background of the technologies and concepts we build upon in this work. In Section 3 we describe the mapping itself and our translator implementation. Section 4 describes related work and Section 5 discusses future work and concludes.

2. Background

2.1 AADL

The Architecture Analysis and Design Language (AADL) is a domain-specific model-based systems engineering (MBSE) language that is standardized by the SAE (SAE AS-2C Architecture Analysis and Design Language Committee, 2022). It has two syntaxes: one graphical (see Figure 1), and one textual. It also has well-defined semantics which allow for the modeling of both an application's software, such as processes and threads, as well as the execution platform, such as

processors, memory, buses, and more (P. H. Feiler & Gluch, 2012). The well-defined timing and execution semantics allow for quantitative or qualitative analyses of attributes such as overall system latency. The analyses can be done on either partially or fully completed systems, which enables MBSE activities to be performed early, as well as late, in the system development lifecycle.

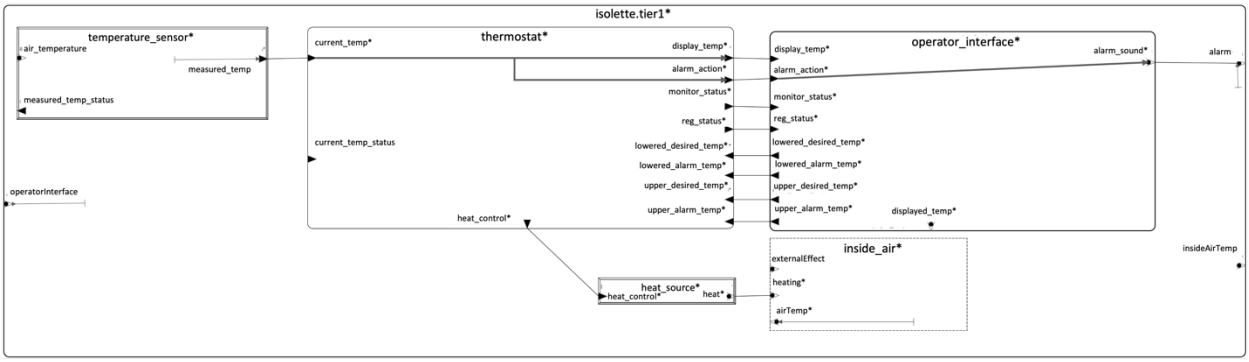

Figure 1 — A portion of a neonatal isolette system, shown in AADL's graphical syntax as displayed by OSATE. Temperatures are measured by the sensor, processed by the thermostat to the operator interface (optionally triggering an alarm) and turn on the heater, if necessary. System adapted from descriptions in Lempia & Miller and Larson et al.

2.2 OSATE

The Open Source AADL Tool Environment[1] (OSATE) is an integrated development environment for AADL maintained by the Software Engineering Institute. It supports the creation and editing of AADL models using both textual and graphical syntaxes. Models are validated against legality rules set by the AADL standard (SAE AS-2C Architecture Analysis and Design Language Committee, 2022). In addition to core AADL, the tool also supports the error modeling (in its second revision, abbreviated EMV2) annex for modeling off-nominal system behaviors (SAE AS-2C Architecture Analysis and Design Language Committee, 2015) and ALISA (see below) for system assurance. OSATE comes with a variety of analyses that may be run on a model throughout its lifetime, including fault tree analysis (P. Feiler & Delange, 2017), fault impact analysis (Delange, Feiler, Gluch, & Hudak, 2014), latency verification (P. Feiler & Hansson, 2007), as well as others. An analysis can also be created by the end user in order to support domain-specific needs.

2.3 ALISA

The Architecture Led Integrated System Assurance (ALISA) is an incremental assurance framework built on top of AADL. An overwhelming majority of hardware and software implementation errors are found at integration, with about 70% of those errors likely introduced in requirement specification (Delange et al., 2016). In ALISA, requirements are connected to system artifacts, allowing for traceability and validation throughout the development life-cycle of a system. These system artifacts include system requirements, stakeholder goals, and verification reports. By using this process, requirements—such as system specifications, quality attributes, hazards, and uncertainty in specific requirements—can be continually tested, ensuring that

[1] https://osate.org

requirements will be of better quality (Delange et al., 2016). ALISA is comprised of several notations, three of which support requirement specification and verification:

1. ReqSpec: Supports the specification of system requirements and stakeholder goals.
2. Verify: Enables creation of verification plans that assess how well various areas of the system architecture meet the system requirements.
3. Alisa: Combines individual assurance plans into an overall assurance case.

2.4 Safety Assurance

Safety assurance is the process of establishing that a system is acceptably safe. A popular way to prove a system is safe is through the use of safety cases. These arguments connect safety claims to evidence in order to support a given claim. A chain of claims that break down the top-level proposition to the point that evidence can be given to lower-level claims, which in turn justifies the truth of the overall proposition. Safety cases are the collection of safety arguments. They provide valid justification of the safety of a system. The safety case is comprehensive and defendable for a given application in a defined operating setting when every subclaim can established by evidence. All artifacts that are created throughout the life-cycle of creating a system are included to assure that it is acceptably safe (Denney & Pai, 2018).

2.5 GSN

The Goal Structuring Notation (GSN) is a graphical safety case notation. GSN provides a clearer view of the structure of a safety argument when compared to plain English text (Kelly, 2004). It consists of six different nodes: goals, strategies, contexts, assumptions, justifications, and solutions. Each node is linked to another through the use of one of two arrow types, either “in context of” or “is supported by”. The “in context of” arrow is used to denote that a node provides context to another node. These nodes refer to documents such as: requirement documents, design documents, hazard logs, safety analyses, etc. The “is supported by” arrow is used to show that the node being pointed to supports the previous node. These eight elements make GSN clear, concise, and easily understandable for the review process done by certification authorities (Kelly, 2004).

2.6 AdvoCATE

AdvoCATE is an Eclipse application that targets assurance activities. The tool allows for the creation of GSN models (Denney & Pai, 2018), hazard tables (Denney & Pai, 2018), and bow tie diagrams (Denney, Pai, & Whiteside, 2017). It provides formal methods integration and has semi-automated creation of arguments through the use of argument pattern instantiation. The ability to import external test results allows for traceability between related assurance documents (Denney & Pai, 2018).

2.7 Isolette Example

Throughout this paper, we refer to the example of an isolette’s thermostat (see Figure 1). An isolette is a safety-critical system that is often used for prematurely-born infants in Neonatal Intensive Care Units. It is an incubator that controls temperature, humidity, and sometimes oxygen. The thermostat interacts with the temperature sensor, the heating unit, and the operator’s interface.

The main goal of the thermostat is to keep the infant at a safe temperature. If the temperature becomes unsafe, the isolette triggers an alarm to alert the nurse. The nurse is then able to adjust the temperature through the operator's interface. The new temperature is sent to the thermostat, which will then turn on the heating unit and maintain a safe temperature (Lempia & Miller, 2009).

3. The ALISA to GSN Exporter

Our work provides a translation of various ALISA constructs to equivalent elements in GSN. The input is an AADL model that has been annotated with ALISA's ReqSpec and Verify notations. It will produce GSN models which can then be imported into the AdvoCATE application. The ability to translate between the two applications / languages will allow for easier readability and understanding of the safety argument, as GSN is more commonly used when submitting safety cases for review by certification authorities.

The idea behind creating a mapping between ALISA and GSN is to give developers the freedom to use the modeling language of AADL, use supported analyses from OSATE, and utilize the rigorous assurance verification of ALISA, while not requiring stakeholders or reviewing authorities to learn a new modeling language to understand the final assurance argument. The mapping that is described in the following subsections allows for the full use of ALISA and its different notations, while extracting the required information to generate a clear and concise GSN argument.

3.1 Mapping

The mapping from ALISA to GSN provides the use of six elements within GSN, see Figure 2. The ReqSpec file contains elements which map to goal, strategy, and context nodes. The Verify file connects solution nodes to their given requirement / goal nodes. An in-depth breakdown of each file type and their mappings is given below.

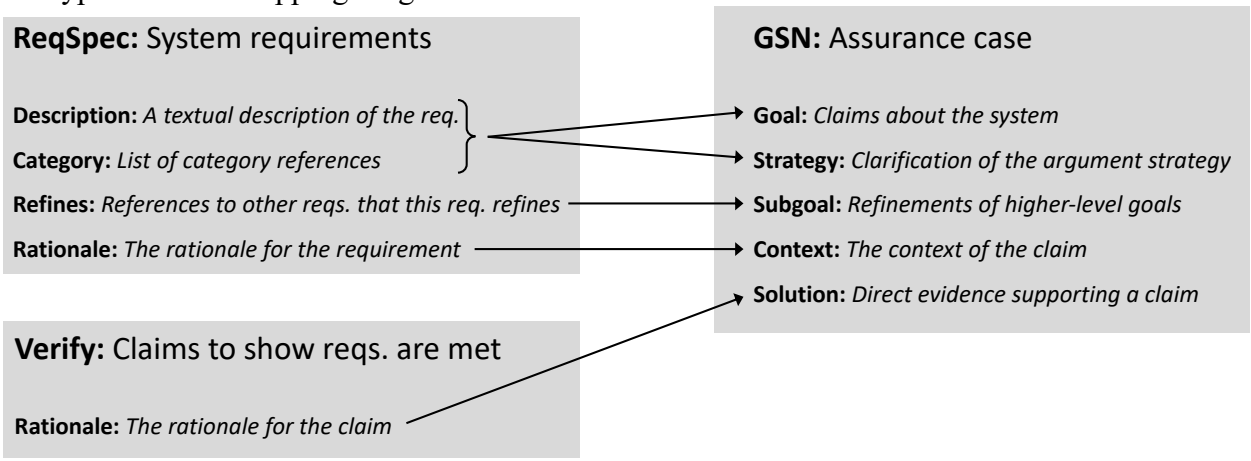

Figure 2 — The mapping from ALISA's ReqSpec and Verify notations to GSN. Note that a requirement's *description* maps to either a GSN *goal* or *strategy* node, depending on the requirement's *category*. ALISA descriptions adapted from the ALISA User Guide[2], GSN from (Kelly, 2004).

[2] https://rawgit.com/osate/alisa/develop/org.osate.alisa.help/contents/00-Main.html

3.2 ReqSpec

ReqSpec allows for the management of stakeholder and system requirements to be tracked. The overall goal of the ReqSpec notation "is to support the elicitation, definition, and modeling of requirements for real-time embedded systems in an iterative process" (P. H. Feiler, Delange, & Wrage, 2016). The translator utilizes this function to generate three different GSN elements: goals, strategies, and context nodes. All aspects of the ReqSpec notation can be utilized by ALISA, but only information following the keywords **description**, **rationale**, **category**, and **refines** are used in the generation of a GSN argument. The **requirement** element within a ReqSpec file can be either a goal or a strategy, depending on the category assigned to the requirement. To denote that a requirement is a goal, system designers should use the category **GSN.Goal**. Subsequently, system designers can specify a strategy with the category **GSN.Strategy**. Goals and strategies are linked together using **is supported by** arrows through the use of the requirement's **refine** feature. This feature allows each requirement to be broken down into more verifiable requirements, similar to the structure of GSN models. Context nodes are generated using the keyword **rationale** within a requirement element and are connected using the **in context of** arrows. If a given requirement that is categorized as **GSN.Goal** is not refined, then the exporter will not generate a solution node in the GSN model.

The requirement in Figure 3 with the name **EA_TS_4** will be used to take a deeper look at what will happen to a requirement when being translated to GSN. The requirement's name will be taken and used as the GSN node's name. This requirement has been denoted as a goal node, because its category is **GSN.Goal**. The text following the keyword **description** is inserted into the goal node's description area. The information following the **rationale** keyword will be imported into the GSN argument as a context node and connected to the goal node using an **in context of** arrow. Since this particular requirement refines requirement **OE_IS_SR2_**, an **is supported by** arrow will link the two nodes together. Requirement **OE_IS_SR2_** does not refine any other requirement, which means it is the top-level goal of the argument.

```
requirement OE_IS_SR2_ for alarmResponse [
                description "The Infant should be removed
                    from the Isolette within 15 seconds after
                     the Current Temperature falls below or
                    rises above the Alarm Temperature Range."
                category GSN.Goal
                see goal TempGoals.GSN1
]

requirement EA_TS_4 for current_temperature_status [
                category GSN.Strategy
                description "The Current Temperature will be
                     provided to the Thermostat with a
                    validity indication."
                rationale "Validity allows thermostat to
                    assess malfunction of temp sensor."
                refines OE_IS_SR2_
]

requirement EA_TS_3 for current_temperature [
                description "The Current Temperature will
                    cover the range of at least "
                    SensedTemperatureRange " deg F."
                refines EA_TS_4
                category GSN.Strategy
]

requirement EA_OI_5 for lower_desired_temperature [
                description "The Lower Desired Temperature
                    will always be greater or equals "
                    LowerDesiredBound " deg F."
                refines EA_TS_3
                category GSN.Goal
]
```

Figure 3 — Example Requirements, adapted from a public example[3]

3.3 Verify

The information contained within a solution node is retrieved from individual claims within the Verify file. The Verify notation enforces that each claim is related to a requirement within the ReqSpec file (Delange et al., 2016), making the connection between the two easy to understand. Each claim's name directly refers to the requirement it is verifying. As with ReqSpec, the full capabilities of the Verify notation can be used, however only information following the keyword **rationale** is used to generate the solution nodes description.

Each requirement needs to have a claim within the Verify file. Figure 4 shows the claim for **OE_OI_10_**. The text following the keyword **rationale** will be used for its solution node.

```
claim EA_OI_5 [
    rationale "Alarm will sound to notify nurse if
        temperature drops below 97 deg F."
]
```

Figure 4 — Example Claim

[3] https://github.com/osate/examples/tree/master/isolette/project/isolette

3.4 Usage

While our tool aims to flexibly support different system development processes, we generally advise the following steps when using the translator (see Figure 5):

1) Create an AADL component that requires an assurance case. The example we used focuses on a system element in AADL.
2) An additional three files are required for the exporter to generate the XML file that will be imported into AdvoCATE.
   a) A category file is used to denote whether a given requirement is a **Goal** or **Strategy** element in GSN. The file needs to have a category type "GSN" and two labels, "Goal" and "Strategy".
   b) The requirement file (ReqSpec) is the main source of information for the exporter. A requirement must be categorized as either a **Goal** or **Strategy** node by using the category keyword in the requirement. See Section 3.2.
   c) The verify file is connected to the requirement set containing all of the goal, strategy, and context nodes. Only goals that are not refined will generate a solution node. See Section 3.3.
3) After creating the required files, select the verify file in the AADL Navigator that will generate the GSN argument. Under "Analyses > Model Maintenance" select "GSN Exporter". A folder will be created titled "Arguments" and will contain the XML file.

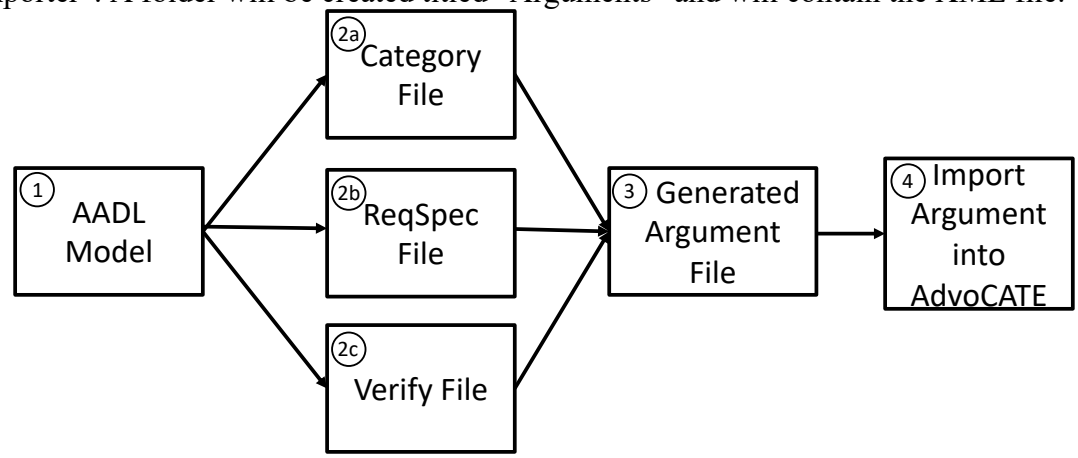

Figure 5 — Typical workflow for converting an AADL Model into GSN

3.5 Output to GSN

Figure 6 shows the output given within AdvoCATE after translation has been completed. Each requirement from Figure 3 is shown connected together with the claim from Figure 4 shown as a solution node for **OE_OI_10_**. If the imported argument is not properly laid out, AdvoCATE provides a button in the upper left of the workspace to automatically arrange all nodes.

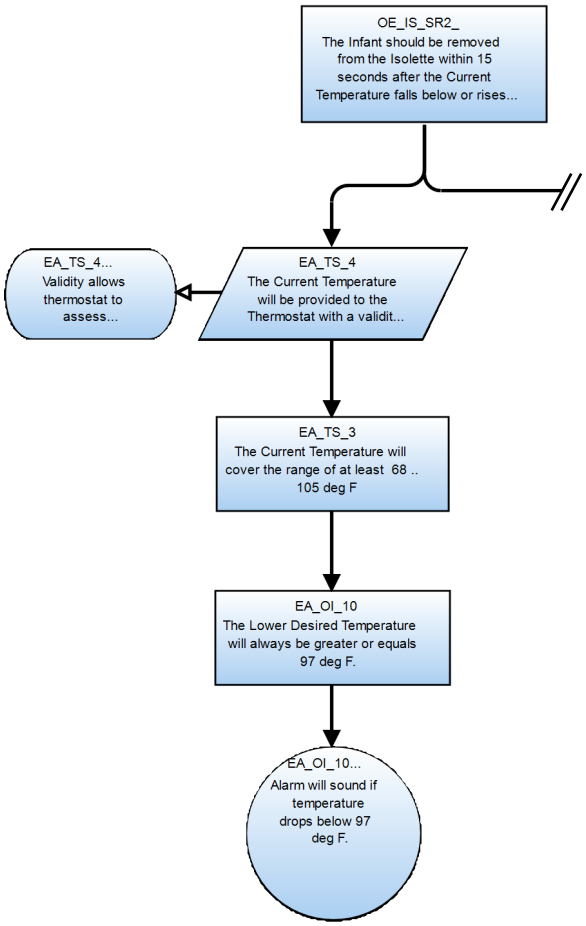

Figure 6 — Example Output to GSN from AdvoCATE

4. Related Work

In the world of safety-critical systems, safety assurance is an important topic. Aligning our work with trends within the community of safety assurance will open the door for more users, developers, and system architects to use OSATE, while gaining the powerful system assurance tools within AdvoCATE. A few other tools have similar capabilities as the two being used in this paper.

4.1 safeTbox

safeTbox stands for Safety Toolbox and has many of the same goals as OSATE. It supports the SysML modeling language for the modeling of hardware, in depth hazard and risk analysis, fault analysis, and safety argumentation with GSN support. Their idea of system modeling and assurance are similar to those of AADL and ALISA, just with the use of different languages and formats. safeTbox supports the modular extension of GSN, while AdvoCATE supports the standard GSN argument (Antonino, Moncada, Schneider, Trapp, & Reich, 2015).

4.2 Other Tools Similar to AdvoCATE

A list of assurance and safety case tools that support the development of GSN arguments has been compiled by the Safety-Critical Systems Club. Besides AdvoCATE, other tools on the list include ASCE (Assurance and Safety Case Environment) by Adelard, DSM (Diametric Safety Case Manager) by Diametric Software, NOR-STA by Argevide, and Socrates' Assurance Case Editor. Each of these tools support GSN or an extension of GSN (Goal Structuring Notation, n.d.). ASCE supports both CAE (claim, argument, evidence) and the modular extension of GSN, similar to safeTbox (Netkachova, Netkachov, & Bloomfield, 2014). On the other hand, DSM supports the GSN Argument Pattern extension, which supports structural and element abstraction (Assurance Case Working Group, 2021). NOR-STA has based the underlying principles of their TRUST-IT metamodel around GSN and CAE (Górski, et al., 2012). Socrates' Assurance Case Editor supports the Dialectic notation of GSN (Goal Structuring Notation, n.d.), which allows for system developers to add strength to assurance arguments through constructive criticism and comparing options within the argument (Assurance Case Working Group, 2021).

5. Future Work

This work's value lies partially in facilitating easier interoperability between requirements specification / verification and assurance argumentation. We want to further develop that aspect by exploring three additional potential integrations:

5.1 Architecture-Supported Architecture Processor

Hazard analysis techniques are complementary methods of ensuring systems, especially safety-critical systems, are acceptably safe. The Architecture-Supported Audit Processor, or ASAP, is a tool that has been created for OSATE and AADL with system auditors in mind (Procter & Hugues, 2022). Using the Error Modeling Annex, ASAP generates views that allow developers and auditors to understand the effects each component of a system has on connected components. Being able to connect ASAP with the ability to generate GSN arguments will allow for hazard analysis to play a major role in the safety argument of a safety-critical system.

5.2 AdvoCATE

We have explored, in informal conversations with the developers of AdvoCATE, a deeper integration between the two applications. This would allow for AADL components to be directly referenced in AdvoCATE, more clearly showing the underlying architecture of each component within OSATE. Being able to quickly switch between both applications will allow for users to model the architecture of the system using AADL, while also creating deeper traceability to that architecture in the assurance cases created in AdvoCATE.

5.3 Risk Analysis and Assessment Modeling Language

The Risk Analysis and Assessment Modeling Language, or RAAML, is a specification created by the Object Management Group. Its goal is to define extensions to SysML that support system and reliability analyses. GSN is one of the system safety domains that is supported within the specification (Risk Analysis and Assessment Modeling Language, 2022). Aligning our GSN arguments with a standardized serialized form of GSN will ensure that our arguments can be imported into a variety of applications that support GSN, rather than just AdvoCATE.

Acknowledgements

Copyright 2022 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0646

References

AAMI Infusion Device Committee. (2019). AAMI TIR38:2019 Medical Device Safety Assurance Case Guidance (Tech. Rep.). Association for the Advancement of Medical Instrumentation.

Antonino, P. O., Velasco Moncada, D. S., Schneider, D., Trapp, M., & Reich, J. (2015). I-safe: An integrated safety engineering tool. IFAC-PapersOnLine, 48 (7), 23-28. Retrieved from https://www.sciencedirect.com/science/article/pii/S2405896315007077 (5th IFAC International Workshop on Dependable Control of Discrete Systems) doi: https://doi.org/10.1016/j.ifacol.2015.06.468

Safety-Critical System Club (n.d.). Goal Structuring Notation. https://scsc.uk/gsn?page=gsn6tools. (Accessed: 2022-05-09)

Delange, J., Feiler, P., Gluch, D. P., & Hudak, J. (2014). AADL Fault Modeling and Analysis within an ARP4761 Safety Assessment (Tech. Rep.).CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Delange, J., Feiler, P., & Neil, E. (2016). Incremental Life Cycle Assurance of Safety-Critical Systems. In 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016).

Denney, E., & Pai, G. (2018, sep). Tool Support for Assurance Case Development. Automated Software Engineering, 25 (3), 435–499. Retrieved from https://doi.org/10.1007/s10515-017-0230-5 doi: 10.1007/s10515-017-0230-5

Denney, E., Pai, G., & Whiteside, I. (2017). Modeling the Safety Architecture of UAS Flight Operations. In International Conference on Computer Safety, Reliability, and Security (pp. 162–178).

Feiler, P., & Delange, J. (2017). Automated Fault Tree Analysis from AADL Models. ACM SIGAda Ada Letters, 36 (2), 39–46.

Feiler, P., & Hansson, J. (2007). Flow Latency Analysis with the Architecture Analysis and Design Language (AADL) (Tech. Rep.). Carnegie-Mellon University Pittsburgh PA Software Engineering Inst.

Feiler, P. H., Delange, J., & Wrage, L. (2016). A Requirement Specification Language for AADL (Tech. Rep.). CARNEGIE-MELLON UNIV PITTSBURGH, PA United States.

Feiler, P. H., & Gluch, D. P. (2012). Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis and Design Language. Addison-Wesley.

Górski, J., Jarzębowicz, A., Miler, J., Witkowicz, M., Czyżnikiewicz, J., & Jar, P. (2012, September). Supporting Assurance by Evidence-based Argument Services. In *International Conference on Computer Safety, Reliability, and Security* (pp. 417-426). Springer, Berlin, Heidelberg.

Assurance Case Working Group, et al. (2021). Goal Structuring Notation Community Standard (Version 3). May 2021.

Kelly, T. (2004). A Systematic Approach to Safety Case Management. SAE Transactions, 113, 257–266. Retrieved 2022-05-04, from http://www.jstor.org/stable/44699541

Lempia, D. L., & Miller, S. P. (2009). Dot/faa/ar-08/32. Requirements Engineering Management Handbook (Tech. Rep.). Federal Aviation Administration.

Netkachova, K., Netkachov, O., & Bloomfield, R. (2014). Tool Support for Assurance Case Building Blocks. In International Conference on Computer Safety, Reliability, and Security (pp. 62–71).

Procter, S., & Hugues, J. (2022). Architecture-Supported Audit Processor: Interactive, Query-Driven Assurance. In Proceedings of the eleventh European Congress on Embedded Real-Time Systems (ERTS) 2022.

Risk Analysis and Assessment Modeling Language (Tech. Rep.). (2022). Object Modeling Group.

Rushby, J. (2021). The indefeasibility criterion for assurance cases. In Implicit and explicit semantics integration in proof-based developments of discrete systems (pp. 259–279). Springer.

SAE AS-2C Architecture Analysis and Design Language Committee. (2015, September). AS5506/1A Annex E: Architecture Analysis and Design Language (AADL) Error Model Annex (Tech. Rep.). SAE International.

SAE AS-2C Architecture Analysis and Design Language Committee. (2022, April). AS5506D: Architecture Analysis and Design Language (AADL) (Tech. Rep.). SAE International.

United States Federal Aviation Administration. (2018). Volume 16: Unmanned Aircraft Systems, Chapter 4: Operational Requirements and Approval, Section 8: Safety risk management (Vol. 16). (https://fsims.faa.gov/wdocs/8900.1/v16unmannedaircraftsystems/chapter04/16 004 008.pdf)