Recent Research in the Selection of Software Verification Techniques

Nathan Karasch
Iowa State University
nkarasch@iastate.edu

Abstract

For any given piece of safety-critical software, it is necessary to build a convincing argument showing the software meets specified safety requirements before it can be deployed. In the domain of avionics, the software must satisfy the DO-178C safety standard. Well-established techniques such as formal methods exist to verify that the software meets the requirements; however, there is currently no standardized way to argue the appropriateness of cutting-edge verification techniques. Recent research by Cârlan et al. suggests a modification to current practice to aid in selecting verification techniques to fulfill the requirements. The research extends the Structured Assurance Case Metamodel (SACM) to reason about verification technique selection and provides a pattern for arguing the appropriateness of the verification techniques selected.

Keywords: safety-critical software, verification, Structured Assurance Case Metamodel, technique selection, argument pattern

1. Introduction

How do we convince ourselves that something is safe to use? Typically, a governing agency is in charge of certifying the "safeness" of a given thing according to a set of established standards, and the end users of that thing then choose to trust the rating given to it. For software, it's no different. Safety-critical software, which is defined by Leveson as "any software that can directly or indirectly contribute to the occurrence of a hazardous system state," [4] must have attained a certification stating that the system meets established safety standards before people will trust the system with their lives. To attain this certification, an *assurance case* must be made for the software.

An assurance case, also known as a safety case, is defined by the Object Management Group (OMG) as "a collection of auditable claims, arguments, and evidence created to support the contention that a defined system/service will satisfy its assurance requirements" [6]. Just as defense attorneys must build a case to argue the innocence of their clients, so also must software producers build a case to argue the safety of their software. The systematic approach to building such a case is the Structured Assurance Case Metamodel (SACM) [6]. The SACM is communicated as a Unified Modeling Language (UML) class diagram containing model elements, their properties, and the relationships between them. Figure 1 shows the Artifact Metamodel, a sub-structure of the larger SACM framework.

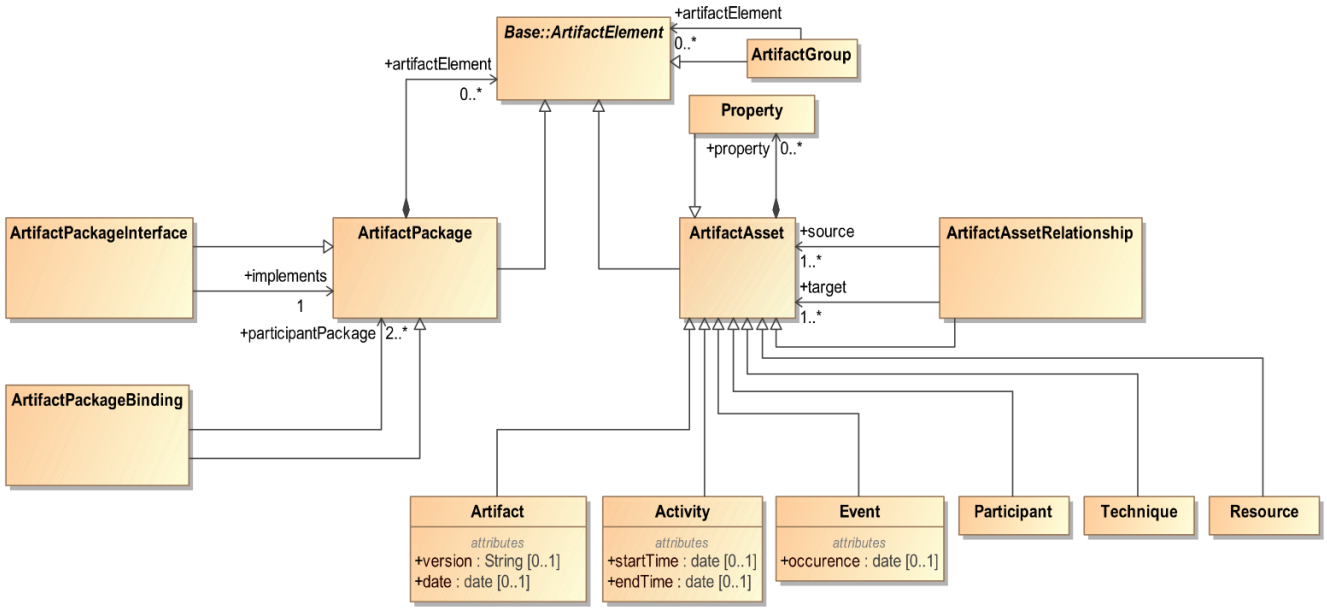

Figure 1. The SACM Artifact Metamodel.

The SACM, being a metamodel, is abstract by nature and doesn't help build assurance cases without being specifically implemented to target a given domain or problem. In the area of avionics, the safety standards are primarily defined in DO-178C [7]. If an airplane's software system doesn't meet those standards, the Federal Aviation Administration (FAA) is not going to approve it to fly. It is outside the scope of this paper to describe exactly how one builds a model using SACM to construct an assurance case for DO-178C. Instead, this paper will expound upon an existing problem within the context of building an assurance case for DO-178C and discuss recent research and progress towards solving the problem. Specifically, this paper will discuss a recently-proposed approach to arguing the selection of software verification techniques for safety-critical systems.

2. Software Verification Technique Selection

2.1. The Problem

Due to the growing complexity of software and the strict requirements placed on safety-critical software, there is no one-size-fits-all technique for verifying the correctness of a piece of software. This is evident in the numerous levels and types of testing and analysis available. Ammann and Offutt describe the different levels of testing activities as acceptance testing, system testing, integration testing, module testing, and unit testing [1]. Each level of testing verifies a different phase of the system's development, from verifying requirements to verifying code-level implementation details. Countless testing techniques exist to verify different quality attributes, from load testing, which verifies performance attributes, to penetration testing, which verifies security attributes. Static analysis tools, formal methods, and other techniques also exist and serve to verify aspects of the software's correctness that dynamic testing cannot.

Given the limited scope of individual verification techniques, as well as the different *types* of results from one technique to another, no one technique can satisfy the requirements by itself. To meet the requirements of a safety standard such as DO-178C, software developers must select a set of verification techniques from among the available options. An assurance case is then built using SACM to argue that the requirements have been met, and the artifacts produced by the verification techniques provide the body of evidence for the argument. The selection of verification techniques must, in a loose sense, provide a "covering set" for the requirements.

Of course, the claim that a verification technique satisfies certain requirements holds no weight without sufficient backing evidence. For well-established techniques that have had the benefit of rigorous review and research over time, this evidence isn't hard to produce. For example, when using formal methods verification in the context of DO-178C, one needs only to reference DO-333 [8], a supplement to the standard. However, for more cutting-edge, novel techniques the evidence is harder to come by. More accurately, there isn't a clearly-defined model for arguing the appropriateness of a verification technique. Cârlan et al. explain that "a technique is *appropriate* if it provides *trustworthy* and *relevant* verification results" [2]. That is, the technique's results must be correct and must be applicable for satisfying the given requirements.

This brings us to the recent research done by Cârlan et al. ("the authors") in "Arguing on Software-level Verification Techniques Appropriateness" [2] ("the paper"). They propose several advancements in the area of verification technique selection. First, they extend the SACM to help the user select a set of verification techniques to meet the safety requirements. Within their model, they construct relationships between verification results to help achieve the "covering set" necessary to adequately satisfy the requirements. Finally, they provide a step-by-step approach for "arguing the appropriateness of a certain technique" [2] based on a model constructed with the extended metamodel.

2.2. Extending the SACM Artifact Metamodel

As mentioned above, the authors extend the Artifact Metamodel section of the Software Assurance Case Metamodel, shown previously in Figure 1. The *Activity* class is extended by *VerificationActivity*, and the *Technique* class is extended by *VerificationTechnique*. These new classes have various properties described at length in the paper that won't be discussed here. An abbreviated view of the extended classes is shown in Figure 2.

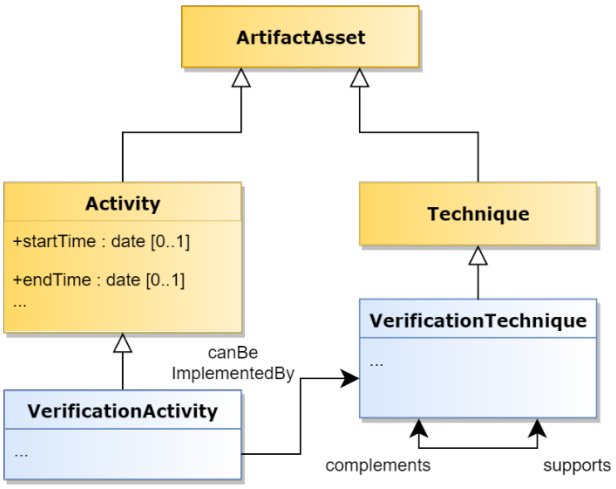

Figure 2. The authors extend two classes from the SACM Artifact Metamodel.

Since these new classes are used to help argue the compliance of the techniques to DO-178C, the authors

derived some of the attributes from the standard. For other aspects that were missing from the standard, the attributes are pulled from "specialized literature depicting safety verification techniques" [2]. These kind of attributes aren't in the standard because the standard doesn't address arguing the appropriateness of a verification technique for a given activity [2].

2.3. Establishing Relationships between Techniques

Figure 2 also shows two relationships between *VerificationTechnique* classes. One is the *supports* relationship, and the other is the *complements* relationship. The *supports* relationship is used when one technique verifies the results of another technique. That is, *A supports B* when *A* is in charge of making sure the results of *B* are correct. It doesn't actually add to the "covering set" needed to fulfill the requirements; instead, it strengthens and validates the results of a technique that does.

The *complements* relationship, on the other hand, **does** add to the "covering set." We can say that *B complements C* if *B*'s results verify different aspects of the requirements than *C*'s results. Figure 3 shows a useful heuristic for thinking about the different between *supports* and *complements*.

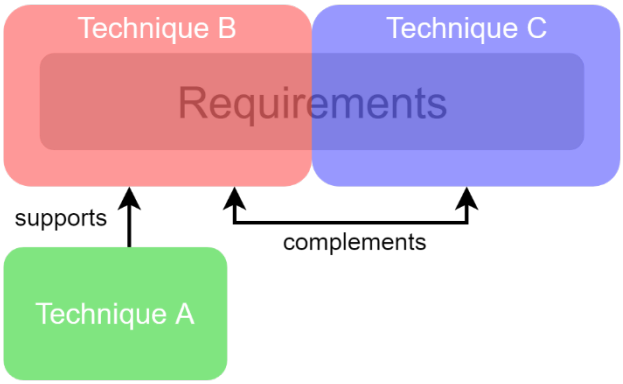

Figure 3. The difference between supports and complements relationships.

2.4. Arguing the Appropriateness of Techniques

Once a model has been constructed and relationships have been established between *VerificationTechnique* instances, the next step is to use the model to argue the appropriateness of the techniques. The authors use Goal Structuring Notation (GSN) to model the reasoning used for the argument. They call it "the *technique appropriateness* argument pattern" [2]. According to the standard, "GSN is a graphical argumentation notation that can be used to document explicitly the elements and structure of an argument and the argument's relationship to evidence" [3].

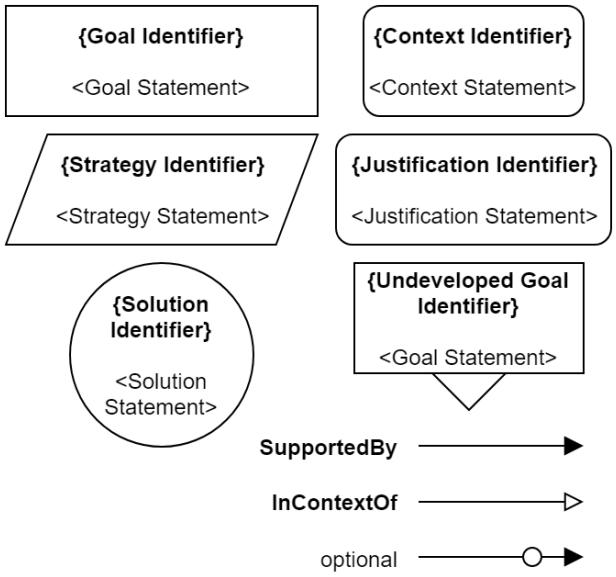

Figure 4. The subset of GSN [3] syntax used in the paper.

The authors use a shorthand notation for the GSN syntax and assume the readers have prior knowledge of what it means. For the sake of clarity, the subset of GSN elements used in the authors' diagram is shown in Figure 4. The argument is built from the attributes assigned to the *VerificationTechnique* being reasoned about. For example, Goal 1 (shown in the paper as **G1: Mere compliance**) reads, "Technique {*techniqueName*} is an appropriate means of compliance for verification activity {*a*}, in order to achieve {*assuranceLevel*}" [2]. The placeholder {*techniqueName*} refers to an attribute of the *VerificationTechnique* instance, {*a*} refers to the VerificationActivity the technique is being reasoned against, and {*assuranceLevel*} is the specified level of assurance for the given activity.

3. Evaluation

The final portion of the paper gives an example of the technique appropriateness argument pattern in use to help arrive at a decision about which compiler to user for a project. In the example, they've narrowed it down to two choices, each of which exhibits different verification techniques (static analysis goals) during compilation.

The example shows how the methods developed by the authors could be used in practice and presents a good starting point for research to follow. Further study would be needed on a variety of scenarios and under various conditions in which the models presented could be scrutinized and improved upon. As they work through the example, the authors make observations about different aspects of the model they may not have noticed before. It's hard to know whether those observations produced any change to the

model, but it's clear that with more usage one would expect to make more observations to improve the model and make it more robust.

One interesting aspect that the authors point out as an area for further development is the "(semi-)automatic creation of safety arguments based on the proposed metamodel" [2]. Given the systematic nature of the approach, the idea holds merit and could potentially drastically reduce the amount of effort needed to build an assurance case. However, one could also imagine the negative consequences of automating away such important work. The supplements to DO-178C, such as the formal methods supplement DO-333, came into realization only after years of rigorous effort [2]. The easier we make it to standardize verification techniques, the less likely it is that the unexpected behaviors and anomalies in the techniques will surface. The formulaic nature of the argument generation process could develop a confirmation bias in the developers. That is, once they've gathered enough information to show the software is safe, they may stop looking for faults in the system. Leveson expressed this concern with safety cases when she wrote the following:

> If the goal is to prove the system is safe, [people] will focus on the evidence that shows it is safe and create an argument for safety…The value that is added by system safety engineering is that it takes the opposite goal: to show that the system is unsafe. Otherwise, safety assurance becomes simply a paper exercise that repeats what the engineers are most likely to have already considered [5].

The effect of automatically generating an assurance case may actually push the developer's mindset in the wrong direction, resulting in an unsafe system being cleared for production before all the major hazards have been identified and mitigated.

4. Conclusion

To show that safety-critical software is compliant with the DO-178C standard, software developers must argue the appropriateness of the verification techniques used. However, the standards don't currently have a way to do this for cutting-edge technology. Cârlan et al. have made recent advances in this area, providing a technique appropriateness argument pattern for building an assurance case. The pattern and associated metamodels show promise in creating a starting point for this area of research. Further study will be needed to refine the model and prove its effectiveness, but it has the potential to drastically reduce the effort needed to build a safety case. This could be a great benefit to the software engineering community, but if developers slip into the mindset that they can easily prove a piece of software is safe, they will be taking steps in the wrong direction that could produce less-safe software.

References

1. P. Ammann and J. Offutt, *Introduction to Software Testing*. New York, NY: Cambridge University Press, 2008.
2. C. Cârlan, B. Gallina, S. Kacianka, and R. Breu, "Arguing on Software-Level Verification Techniques Appropriateness," Lecture Notes in Computer Science Computer Safety, Reliability, and Security, pp. 39–54, 2017. [Online]. Available: https://pdfs.semanticscholar.org/0984/e64fc9d717f88631a51ace7e7d195610baa2.pdf. [Accessed 20-Mar-2018].
3. GSN, "Community Standard Version 1," 2011. [Online]. Available: http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf. [Accessed 16-Apr-2018].
4. N. G. Leveson, *Safeware: System Safety and Computers*. Boston, MA: Addison-Wesley, 2001.
5. N. G. Leveson, "White Paper on the Use of Safety Cases in Certification and Regulation," 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.300.3899&rep=rep1&type=pdf. [Accessed 15-Apr-2018].
6. Object Management Group, "Structured Assurance Case Metamodel Version 2.0," 2018. [Online]. Available: https://www.omg.org/spec/SACM/2.0/PDF. [Accessed 15-Apr-2018].
7. RTCA, "DO-178C: Software Considerations in Airborne Systems and Equipment Certification," RTCA & EUROCAE, 2011.
8. RTCA, "DO-333: Formal Methods Supplement to DO-178C and DO-278A," RTCA & EUROCAE, 2011.