Carnegie Mellon University
Software Engineering Institute

VANDERBILT UNIVERSITY

Federal Aviation Administration

Assurance Evidence of Continuous Evolving Real-Time Systems

Lead Authors

Dionisio de Niz
Technical Director, Assuring CPS (ACPS)/SSD/SEI/CMU

Bjorn Andersson
Principal Researcher, FVCPS/ACPS/SEI/CMU

Mark Klein
Principal Technical Advisor, SSD/SEI/CMU

Hyoseung Kim
Associate Professor, ECE/UC Riverside

John Lehoczky
Thomas Lord University Professor, CMU

Doug Schmidt
Cornelius Vanderbilt Professor, Vanderbilt University

George Romanski
Chief Scientific and Technical Advisor, Software, FAA

November 2022

Contributing Authors

James Anderson
W.R. Kenan Distinguished Professor
University of North Carolina

Sanjoy Baruah
Professor
Washington University in St. Louis

Tuan Bui
Software Airworthiness Lead
Army AvMC, SRD

Christopher Collins
Executive Director, DTE&A
Office of the Undersecretary of Defense (R&E)

Kenneth Costello
Engineering Services Lead
NASA

Floyd Fazi
Systems Engineer Principal
Lockheed Martin Corporation

Ariel Fershtut
Deputy Division Chief
Sentinel Software Division

Kyle Fox
Sentinel Chief Software Engineer (CSE)
US Air Force

Nickolas Guertin
Director
Operational Test and Evaluation (DOT&E)

Jason Kelly
Software Engineer Sr. Staff / Associate LM Fellow
Lockheed Martin Corporation

Ragunathan Rajkumar
George Westinghouse Professor ECE
Carnegie Mellon University

John Robert
Deputy Director, Software Solutions Division
Software Engineering Institute, CMU.

John Ross
Embedded Software Developer
Brockwell Technologies, Inc

David Tate
Senior Defense Analyst
Institute for Defense Analysis

Alexander Volynkin
Senior Research Scientist
Software Engineering Institute, CMU

Reginald White
Senior Military Assistant
OSD/DOT&E

Mark Wotell
Software Engineer
Intrepid LLC

Albert Cheng
Professor of Computer Science and ECE
University of Houston

Cherish Franco
Software Engineer
US Air Force

Willie Fitzpatrick
Senior Software Engineer
TriVector Services

John Goodenough
Fellow
Software Engineering Institute, CMU

Richard Kutter
Technical Advisor, Embedded Computing
US Air Force

Tom Longstaff
Chief Technology Officer
Software Engineering Institute, CMU

Kate Nguyen
Engineer
AFSEC

Travis Redfield
Computer Scientist
US Army System Readiness Directorate

Benjamin Robertson
Lead Engineer
Booz Allen Hamilton

This material is based upon work funded in part 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.

DM22-1019

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Table of Contents

Acknowledgments

This report is both the conclusion of a hard effort to organize and execute the ASERT workshop and the beginning of a workgroup effort to tackle the challenges discussed in the workshop and presented in this report. Hence, we have a long list of colleagues that made this possible and deserve many thanks.

We want to start with the people that supported our work in the workshop organization. First, we want to thank our home organizations, CMU, SEI, UC Riverside, Vanderbilt University, and the FAA for supporting our work during the workshop and the writing of the report. Secondly, we want to thank Tom Longstaff the SEI CTO that encourage and supported the SEI organizers to go after this endeavor as well as the Anita Carleton and John Robert director and deputy director of the Software Solutions Division of the SEI for their guidance and support. Thirdly, the many people that allowed us to connect to the proper audience including, Eileen Wrubel, Brigid Petrie O'Hearn, Harold Ennulat, and Hasan Yasar. Finally, we want to thank the support people that helped us host the workshop including Michele Falce, Shay Badolato, Dan Bidwa, and Linda Canon.

Finally, we want to thank the participants and collaborative authors of this report for an incredibly rich discussion and insights that create the foundation for what we anticipate can be a truly transformative endeavor.

Executive Summary

The workshop ASsurance Evidence for Continuously Evolving Real-Time Systems (ASERT) took place in Arlington, Virginia on August 16-17, 2022. The workshop was motivated by the increasingly essential need to efficiently and effectively certify (or recertify) real-time systems, which are also often safety critical. The premise of the workshop was that fostering communication between the producers of real-time research results, the developers of real-time systems, and certifiers could expose issues that are impediments to developing an improved methodology for certification. The workshop brought together members of government, industry, and academia to explore the following two topics:

1. Developing techniques to substantially reduce the ever-increasing time to certification in the face of the ever-increasing scale and complexity of modern systems, and
2. Developing methodologies to certify systems that adopt new hardware and software paradigms. Those two topics stimulated discussion along four themes.

These two topics stimulated discussion among four themes.

The three technical themes that emerged from the workshop were recertification, automation, and argumentation. The fourth theme was transition.

Re-certification. In software-reliant systems, such as defense and aerospace systems, the portion of innovations due to software is ever-increasing. The resulting continuous upgrades must be accompanied by continuous **recertification** to preserve the assurance required in safety-critical systems. This certification involves the collection of evidence along with the argumentation that describes how the evidence is combined to satisfy specific assurance claims. Unfortunately, current certification processes are poorly suited to continuously evolving systems. Therefore, we face a double challenge: (1) enabling argumentation of increasingly large systems and (2) minimizing the argumentation effort for modification of the system. Meeting these challenges implies that future certification processes will need to rely heavily on automation.

Automation. Assurance cases drove much of the discussion at the ASERT workshop. One inspiration for emphasizing assurance cases was the observation that general design decisions and safety design decisions (such as those considered in a safety case—an assurance case focused on safety) should be performed in parallel. Since increasing scale and rate of change are driving concerns of today's systems, workshop participants discussed the evolvability and scalability of assurance cases, which highlighted the importance of **automation**.

Automation is crucial to effectively apply assurance cases in evolving systems. Without automation it is likely that the system and its assurance case will become out of sync with one another resulting in either assurance issues that are found late in design leading to significant rework; unsafe systems due to undiscovered assurance issues and/or design decisions for which the design rationale is lost. Since large-scale systems require reasoning in many different and interacting scientific domains, automation is also important to manage the diversity of evidence and arguments in a large-scale assurance case.

Diversity in evidence and in reasoning techniques across many scientific domains has led to **argumentation** complexity. Moreover, overly restrictive assumptions and a disconnect between theoretical results and typical verification claims have led to potentially useful theoretical work being considered too impractical to incorporate into certification processes.

Moreover, the certification community is not unified, so different types of appraisers demand different types of claims. For instance, in the nuclear deterrence realm, at least three types of appraisers must be considered: safety, cybersecurity, and nuclear surety. However, similar verification techniques can potentially provide the basis for assurance argumentation across multiple claims. Assurance cases could be a helpful construct to operationalize this process. Unfortunately, assurance cases are neither well known in the certification communities nor fully standardized. Thus, more communication is needed between researchers, developers, and certifiers if techniques like assurance cases are to be transitioned successfully into practice.

Transition. Transitioning from current practice to assurance cases must undoubtedly be an incremental process. Recognizing the significant inertia in using traditional assurance methods for certification, workshop participants believed that a viable approach could be to use traditional methods side-by-side with formal assurance argumentation techniques in a way that is mutually supportive. Two other issues of practical importance in the transition of assurance cases into adopted practices include accommodating multiple levels of rigor and enabling the evolution of assurance cases in response to systems that evolve.

Workshop participants generally thought that the workshop could serve as the beginning of an effort to respond to the current and future certification challenges of continuously evolving (real-time) systems.

1 Introduction

The ASurance Evidence for Continuously Evolving Real-Time Systems (ASERT) held on August 16-17 brought together members of government, industry, and academia to address the issues related to improving the methodology for the certification of evolving real-time systems that would both increase the certainty of correct operation for expected conditions of operation and would reduce the time to achieve certification. The aim of the workshop was to explore the following foci:

1. Methods to substantially reduce the ever-increasing time to certification in the face of the ever-increasing scale and complexity of modern systems, and
2. Methodologies to certify systems that adopt new hardware and software paradigms.

The first focus dealt with structured arguments which are known as assurance cases. They are used to convince an oversight authority that the system is ready for fielding in a specific operational context Systems must and do evolve. This brings the questions of whether and how we can reuse an assurance case after a change.

The second focus dealt mainly with multicore processors, which offer the opportunity for physical concurrency; however, inter-core interference can compromise that benefit. For example, tasks on separate cores can interfere with each other at lower levels of the architecture, such as caches, memory busses, and memory banks. More generally, the introductions of new algorithms and/or new hardware technology requires the development of evidence about the new technology itself, but also its interactions with the residual (e.g., legacy) technology whose use continues in existing and planned systems.

The rest of this document presents the findings from the workshop that includes the challenges raised due to the disconnection of verification techniques from the certification processes of critical systems in general and of real-time systems (and its timing verification techniques) in particular.

2 Recertification of Continuously Evolving Systems

In software-reliant systems, such as defense and aerospace systems, the portion of innovations due to software is ever-increasing as reported in the National Agenda for Software Engineering [4]. Of course, continuous upgrades and evolution over time should be accompanied by certification processes to provide the degree of assurance sought in safety critical systems. Unfortunately, current certification processes are poorly suited to continuously evolving systems. This problem was identified by workshop participants and has also been echoed in the research literature [3]:

> *The traditional approach to certification is to prepare monolithic safety cases as bespoke developments for a specific system in a fixed configuration. … A more promising approach is to attempt to establish a modular, compositional, approach to constructing safety arguments that has a correspondence with the structure of the underlying system architecture.*

Current certification processes rely heavily on manual labor and comprehensive one-off testing procedures, thereby consuming significant amounts of time, effort, and cost. This approach might have been considered acceptable for systems that rarely---if ever---evolve because the certification of such systems happens only once at the acquisition stage. Certification is no longer a one-time event, however, yet the demand for re-certification continues to increase as technology advances and threats/requirements become more complex. To respond to the challenges of continuous evolution, certification processes, as well as system development practices, need to evolve with the following considerations:

1. The effort of recertification, particularly for generating and assessing assurance arguments, should be proportional to the change made.
2. More widespread adoption of automation tools is needed to ensure objectivity and soundness of assurance arguments.
3. Systems should be designed for upgrades with recertification in mind.

2.1 Argument Evolvability

Systems continue to grow in complexity and as mentioned earlier, they must now enable continuous change. As a result, we face a double challenge: (1) enabling argumentation of increasingly large systems and (2) minimizing the argumentation effort for modification of the system.

To enable argumentation of increasingly large systems, we need to take an incremental approach to develop arguments and collect evidence as the system evolves. Incremental approaches are not new in existing certification processes since safety standards such as DO 00-56 already emphasize that “safety cases should be started at the earliest possible stage”. However, the importance of incremental argumentation is substantially higher in continuously evolving systems since otherwise, recertification becomes an intractable problem for both certifiers and developers.

An issue closely related to incremental argumentation is modularization. To minimize the impact of any changes made in the system, arguments should be modularized with a clear description of their scope and supporting evidence and materials. This approach necessitates the investigation of more comprehensive and well-organized ways to check the range of changes that occurred and identify change propagation and impact on other subsystems and the corresponding arguments. In this context the meaning of changes includes not only functional changes in the system, but also non-functional changes such as code refactoring, operational usage, and performance optimization. All these types of changes could potentially affect the timing correctness and safety of the system. Regardless of the type of changes, therefore, we need to identify potentially affected subsystems, and carefully refactor respective arguments.

2.2 Tool Support

Rapid changes in continuously evolving systems are driving the need for alternative methods for assurance cases. In particular, the certification processes need to rely heavily on the automation of assurance arguments to cope with the ever-increasing scale of these systems and the pace at which new arguments must be created and processed. Given that the amount of evidence and arguments the certifiers must address is already huge, automation tools with support for modularization are a prerequisite to making incremental argumentation development a viable approach. Such tools will contribute to reducing the burden of argumentation management and recertification processes for certifiers. There is also an opinion that automation tools will help accelerate development processes by pacing up certification.

2.3 Design for Upgrade

Besides certification processes, today's system development practices could be reevaluated and enhanced to minimize the impact of recertification while ensuring performance efficiency in continuously evolving systems. Some ideas discussed at the workshop are that software components should be modularized not just based on their functionality, but on their criticality and/or trustworthiness. In particular, the latter characteristics affect the likelihood of system lifetime and update periods.

The use of space and time partitioning techniques, such as ARINC 653 in Integrated Modular Avionics (IMA), for performance isolation among software components has been a commendable effort. However, researchers have identified their inherent performance loss and inflexibility for adaptation, which may be major obstacles in evolving systems in the future. Multicore processors, which will be discussed in Section 6, also brings new challenges to the partitioning techniques.

Some presentations at the workshop focused on priority-based, hierarchical, and mixed-criticality real-time scheduling. These were not presented as a means for argument evolvability, but they can support evolvability by innovating development processes. For example, a hierarchical real-time scheduler provides a predictable supply of processing time to each sub-system. This approach can be seen conceptually similar to the time partitioning of ARINC 653, but it brings about much more flexibility and efficiency by enabling predictable interleaving of time slots with provable guarantees.

A hierarchical scheduler is able to guarantee that a module receives a particular amount of CPU time within a specific interval independently of other guarantees provided to other modules. If this

amount of CPU time is enough for the module to meet its deadlines even when the module changes, then the impact of change of one software module on another may be decreased. Chapter 4 in [1] gives a method for detecting that one part of an assurance case has been "damaged" and assesses how fixing it can create additional damage and identify limits to this additional damage. Such a real-time scheduler requires the following:

- support from a run-time system to ensure that execution in one sub-system does not influence timing of execution in another sub-system
- clear semantics, for example, what does it mean to provide a guarantee of supplied processing time in a time interval
- offline tools that can provide a proof that all timing requirements of software within a subsystem are satisfied, as well as offline tools that can dimension the underlying schedule that provides lower bounds on processing time to each subsystem.

3 Automation

The ASERT workshop discussed assurance cases with respect to the need for evolvability, scalability, and objectivity, and there was a large fraction of the participants who viewed it as the way forward. A keen observation from a workshop participant was that we need "design and safety in parallel." However, to achieve these properties (incremental safety case), we need to automate it (at least part of it). Automation is needed for specific claims (e.g., worst-case execution time analysis and schedulability analysis for claims about real-time requirements), but automation is also needed for managing assurance cases. An example of the latter is Safety Argument Manager (SAM) that was developed in the 1990s for Goal Structuring Notation (see Chapter 6 in [1]).

The following are relevant context for future research/development along the direction of creating automation/tools for assurance case:

- We need quantitative metrics on how much formal/analytical methods and tools can expand our capabilities on assurance cases.
- There are some ongoing efforts: DARPA Automated Rapid Certification Of Software (ARCOS) program (https://www.darpa.mil/program/automated-rapid-certification-of-software). It is desirable to create a means of curating evidence to generate assurance cases.

3.1 Automation for Evolvability

The case for evolvability of assurance cases has been made several times. This report mentions it in Section 2 and it has also been mentioned in [1]. Chapter 3 in [1] notes an old style that includes designing a system first and then developing an assurance case, followed by checking. Chapter 3 in [1] argues that this old style has the following drawbacks: (1) if a design is deemed to be not safe enough, then in some cases, it is necessary to go back and redesign the entire system, which can be very costly, (2) since assurance case is not used to drive the design, a system often ends up with less robust safety arguments, and (3) sometimes a system is designed to be safe but the design rationale tends to get lost in the design process. It also argues that we need incremental safety-cases and a new lifecycle. Chapter 3 in [1] proposes an example of the latter where one produces three different safety cases: preliminary safety case (in the requirements phase), interim safety case (after initial design), and operational safety case (before the system is put in service).

Another perspective raised at the workshop was about version control at the evidence argumentation. A possible direction is to manage evidence in the evidence repository.

It was pointed out during the workshop that not everything can be automated. Echoing this, Section 6.3.2.1 in [1] mentions that if an assurance case has been damaged, then an automatic tool can identify other nodes in the assurance case that may have been damaged, but this assessment is pessimistic; it may be that some of these nodes actually have not been damaged.

3.2 Automation for Scalability

Scalability in assurance cases can be a challenge because of (as mentioned in Section 1.2.1 and Section 1.2.3 in [1]) the increasing volume and diversity of evidence and argumentation to discharge the top-level claim. Even if an assurance case is presented in a hierarchical manner with clear dependencies between claim and sub-claims/evidence, we still have the challenge of having a human processing such a large amount of information. Automation is required to design, validate, and execute the arguments at scale. Likewise, an individual step of reasoning that draws a conclusion from the premise in an assurance case can be highly complex because the underlying design of the system one wants to certify is also highly complex.

For these---and other---reasons, we need software tools (automation), and they should be scalable. In particular, we need tools to manage the assurance case (a large network of claims) and to draw conclusions from premises. Scalability can also be improved through certain designs (see for example, hierarchical real-time scheduling earlier in this document).

3.3 Automation for Objectivity

It is sometimes argued that assurance cases are subjective and hence can be unfair to some applicants. If we can automate the processing of assurance cases, however, we move the interpretation of the argument to a computer, thereby removing the potential ambiguity of human interpretation.

Automation can also help certification authorities in assessing evidence. An approach that could be quite beneficial to hasten the adoption of automation, but is currently underexplored, is to have verification procedures that not only produce an output to a question but also produce an explanation for this answer. For example, if the question is “will all real-time deadlines be met at runtime,” instead of just outputting only the answer: yes/no, the verification procedure will also output an explanation for why it gave the answer yes to this question (analogous for “no”).

Such an explanation can be helpful because a certification authority can inspect it. Ideally, this capability allows a certification authority to inspect this explanation and gain trust, even if the certification authority does not understand the underlying theory that the verification tool is based on and does not necessarily trust that the tool is correctly implemented.

In addition, automation can help to prioritize critical cases (this capability applies to both the applicant and the certification authority). This is useful because in practice, there is rarely sufficient time and money to explore all cases.

4 Argumentation

Safety and mission-critical Cyber-Physical Systems (CPS) rely on certification processes to reach an acceptable level of assurance that the system performs its intended functions in a safe manner. For CPS that rely on software that interacts with physical processes, safety must appropriately consider timing, since logically correct answers delivered at the wrong time can cause incorrect behavior in the physical environment, leading to a safety violation. Unfortunately, the verification techniques developed over the years by the timing verification research community have not influenced the certification processes due to a number of factors including the following:

- complexity of the verification argumentation
- restrictive assumptions of the verification techniques
- lack of connection between verified properties and certification claims
- lack of communication between researchers, developers, and certifiers

The reminder of this section summarizes the discussion that occurred during the workshop with respect to this problem and the ideas and challenges related to it.

4.1 Multiple Appraisal Dimension

Certification authorities must appraise evidence to make sure that sufficient confidence for specific assurance goals or claims has been reached. Unfortunately, the certification community is not unified, and different types of assurance goals have their own class of appraisers. For instance, in the nuclear deterrence realm at least three types of appraisers must be considered: safety, cybersecurity, and nuclear surety.

Different types of appraisers demand different types of claims. However, similar verification techniques can potentially provide evidence for the assurance argumentation of these different types of claims. To provide this evidence, it is necessary to develop the proper connections between verification evidence and each of the assurance claims of interest.

Assurance cases is a general framework for the connection of specific evidence for a particular type of property in a particular part of the system to system-level claims that has been developed over the years. Assurance cases allow the decomposition of assurance claims into subclaims connecting them to evidence through argumentation to capture the rationale of assurance confidence into a cohesive body of evidence. Unfortunately, assurance cases are neither well known in the certification communities nor are they fully standardized. Moreover, assurance cases today are mostly processed by hand, which both hinders scalability and weakens the soundness of the argumentation.

An assurance case is a powerful artifact that may be the key to reach the scalability and agility demanded by critical continuously evolving CPS. To take full advantage of assurance cases, however, it is necessary to connect them to the claims for the different types of appraisers, such as appraisers in cybersecurity, safety, and nuclear surety that have their own types of claims and argumentation. There is a large portion of these arguments that are common that can be exploited.

Moreover, the use of assurance cases may not become prevalent until they are included in the acquisition process.

Timing verification evidence can only become usable in certification claims if the proper arguments are created to connect this evidence to the claims. While informal examples are often presented in scientific papers, a concerted effort to develop appraiser-acceptable arguments connecting verification results to assurance claims must be developed. This can take two forms:

- the development of new arguments that connect old verification techniques to assurance claims
- the development of new verification techniques specifically connected to new and old assurance claims and arguments

In both cases new assurance claims and argumentation will likely come from the challenges of scale and continuous evolutions.

The verification evidence produced by techniques developed by the research community are not only disconnected from the claims the appraisers care and understand but also from the abstractions and practices that the producers of evidence (e.g., Independent Test and Evaluation organizations) and developers use. This critical gap must be addressed. In addition, there is an issue with tool qualification that can be understood as follows. Consider a tool that is used to support a claim that takes a model as input (which could be sub-claims) and produces an output (e.g., schedulable/unschedulable). To apply such a tool in a current certification process, one has to go through tool qualification. If the tool is very complex, then it may be hard to go through tool qualification. One effort in this direction is the ERSA 2022 workshop on explainability of real-time systems and their analysis in the IEEE Real-Time Systems Symposium 2022.

4.2 Argumentation for Evidence Appraisers

Evidence appraisers need to process argumentation that includes evidence that addresses concerns coming from multiple scientific disciplines. As a result, low-level arguments only accessible to experts in the domain must be lifted to high-level arguments that connect to system-level claims. This demands a translation of the argumentation of a particular scientific discipline to a common language (explanation) where part of the soundness argumentation can be hidden, but with tool support to preserve the trust in the evidence.

4.3 Argumentation for Evidence Producers

The producers of evidence can potentially specialize in different scientific disciplines. However, putting together the evidence and argumentation from multiple disciplines requires explanations across disciplines. These explanations must identify and resolve conflicts resulting from interaction between behaviors ignored by abstractions in one discipline (e.g., computations that take zero time in continuous time models) but described and manipulated by abstractions in another (periodic tasks with worst-case execution time in real-time theory).

4.4 Argumentation for Developers

Developers receive verification feedback from evidence producers (e.g., testers) and need to understand such evidence in order to identify and correct bugs. As a result, when complex verification methods are used, evidence of bugs must be translated into explanations that can guide the correction of the bugs.

4.5 Making Argumentation Accessible to All Assurance Actors

The connections of the verification arguments to the different actors in the assurance pipeline (appraisers, producers, developers, researchers) would need to involve new ways to present arguments but also an explicit effort to learn in part argumentation techniques from other actors.

Assurance argumentation is, ultimately, a communication challenge between the actors involved in assurance. Importantly, the message in this communication must preserve a single interpretation for all the actors involved. Tools that automate the argumentation using a single interpretation play a critical role in the preservation of the message.

Clearly, evolving from today's certification practices to a future one that takes full advantage of verification techniques will require a continuous dialogue between the assurance actors.

4.6 Integrating Timing Evidence To Assurance Arguments

Timing verification, as has been the case of cybersecurity, tends to be an afterthought. Unfortunately, while cybersecurity faults (e.g., vulnerabilities) can be directly connected to design flaws and the lack of the corresponding argumentation for assurance, timing faults are frequently more difficult to connect to a system failure. Moreover, design flaws in a new system can lead to modifications where timing uncertainties accumulate and become manifested as either

- timing errors (e.g., missing deadlines, long delays) that are hard to trace back due to unknown influences (e.g., hidden scheduling) and
- uncontrollable tangled timing interference (e.g. due to the use of a general purpose scheduler that is not analyzable instead of fixed-priority scheduler that is analyzable).

4.6.1 Pitfalls of Timing Argumentation

Safety standards (e.g., DO-178C and AFMAN 91119) recognize the importance of timing but also fail to connect it to the system claims and assurance goals. Instead, they present a number of best practices for which it is not clear whether the implementation of such practices is either necessary and/or sufficient. Moreover, today’s standards do not properly address continuous evolution.

Timing is ignored in safety claims. However, we know claims about a system that interacts with the physical world must consider the timing of the physical world. Due to this omission, failures connected to timing errors tend to show up late in already fielded systems.

The hidden consequences of timing errors and the difficulty to spot them demand assurance arguments that can soundly establish the sufficient conditions to discharge the assurance claims.

Clearly, these arguments can–and must–be adapted to different levels of rigor required by the system.

5 Transition to the Use of Assurance Arguments

5.1 Incremental Transition

Transitioning from current certification approaches to an approach based on assurance arguments will undoubtedly be a gradual and incremental process. This process will likely occur this way because current approaches are often human-intensive and embedded in the culture and processes of certification organizations that are constrained to follow certification standards. Recognizing the significant inertia in using traditional assurance methods for certification, the workshop participants believe that a viable approach could be to use traditional methods side-by-side with formal assurance argumentation techniques to mutually support each other.

5.2 Standards

Regarding standards, workshop participants believed an abstraction framework that (1) is specified at a higher level than DO-178 and (2) enables mapping to other standards like ISO 26262 would be useful in allowing reuse of artifacts available from commercial components, including batteries, UAVs, and processors. Regarding DO-178C and its relationship to assurance arguments, it was unclear to what extent following DO-178C would aid in the development of an assurance argument. Participants thought it would be useful to examine this relationship more closely. Moreover, it was noted that DO-178C includes “plans (or policies)” that are not directly related to evidence. While this information might enable appraisers to have greater confidence in a system if the developer followed the specified plans, doing so does not necessarily provide sufficient evidence for assurance cases.

5.3 Levels of Rigor

Integrating verification technologies that (1) vary in level of rigor to reduce the verification complexity of less rigorous claims and (2) ensure that the soundness of rigorous claims is not compromised is critical to the practical development of acceptable argumentation. This integration will enhance the scalability, practicality and transitionability of formal assurance arguments.

5.4 System and Argument Continuous Evolution

It will be critical for assurance argumentation to support continuous system evolution from the start. When a system changes, much of an existing assurance argument should be reusable. Moreover, the argumentation approach must keep evolving as systems evolve to incorporate new technologies (e.g., AI and ML), claims, and verification technologies. This evolution has implications for the structure of the assurance arguments and hence for the structure of systems that must preserve modifiability and reusability properties. Such modifiability and reuse of assurance arguments could be enabled by using proven assurance structures such as templates or patterns.

We anticipate that developing an acceptable assurance argumentation will be an incremental process. Moreover, the argumentation approach needs to continuously evolve as the systems we

certify keep incorporating (1) new technologies (e.g., AI and ML) and (2) new claims and verification technologies.

5.5 Concurrent Traditional Assurance and Assurance Cases

The large infrastructure currently operating to support assurance with traditional assurance methods creates an inertia that must be considered. As a result, we believe the best approach is to allow traditional methods to exist side-by-side with formal assurance argumentation techniques and to mutually support each other.

5.6 Arguments for Different Certification Standards

Assurance arguments must be able to satisfy multiple certification standards such as DO-178C, ISO 26262, etc. Some standards can be more adaptable to different forms of certification evidence. In particular, DO-178C allows applicants to define their own verification plan, which should facilitate the use of different forms of assurance argumentations in the style of assurance case as part of such a plan.

In addition, it is important to enable the use and reuse of certification argument "modules" for commercial components such as batteries, processors or even UAVs.

6 Timing

An important part of the discussion at the ASERT workshop was real-time requirements, e.g., how to express them, where they matter the most, how to verify them, and how they can fit in an assurance case. The following is a summary of the discussion in general:

- Several talks were given by academic participants on real-time scheduling theory and the usefulness of it to certification authorities. Examples of topics covered included multicore memory interference (analysis [5-7] and mitigation [8-13]), mixed-criticality scheduling [20] for both single-core [14-19] and multicore [21-25], time-triggered scheduling that simplifies integration of subsystems [26-29], and priority-based scheduling and its history ("Rate-Monotonic") [30-32]. Discussions revealed a gap between academia and industry about techniques for timing verification. For example, industry generally likes Time-Division-Multiple-Access (TDMA) scheduling whereas academia focuses primarily on priority-based scheduling due to its ability to offer better flexibility, resource efficiency, schedulability, and end-to-end latency [30]. Techniques from academia to reduce timing interference (e.g., cache coloring [8, 11, 12, 24]) are often not used by industry. A general concern is how to mature academic output for transition into industry. It was also noted that academia should explore the use case for TDMA since there is an opportunity to bridge the gap between industry and academia.
- Timing becomes critical in mode switching [25] which can lead to important inconsistencies directly related to safety. Specifically, data intended for the old mode can be delivered in the new mode due to deadline misses. Mode change can also change memory footprint and the need to perform additional execution to initialize data structures in the new mode; these can lead to extra cache misses.
- Collection of "evidence" data even at runtime and/or in operation can be helpful for certification [36-38]. For example, one can collect data on memory behavior and check if it is the same as the behavior in the lab (before deployment) and if it is compatible with the model used in schedulability analysis. Such collection can also help to identify stress on a resource (e.g., memory bus) to see if it is below saturation point (if the stress is above the saturation point, then queuing delays start to build up).
- Temperature can influence timing. It is an important question how to account for that in schedulability analysis, e.g., [33-35]. If schedulability analysis is overly pessimistic, then processors have to run faster than necessary, which generates more heat and thus requires a larger and heavier cooling system. The cooling system can therefore be considered the "weight" of software.
- Critical sections [39-41] influence timing, but they are not always visible to the application developer, which represents a hidden danger. For example, operating systems often have critical sections internally to protect its data structures from concurrent updates. If the application developers buy a commercial real-time operating system (RTOS), however, they may not be aware of those critical sections.

- Interaction between different layers (e.g., RTOS and applications) requires rigorous verification and testing.

Satisfying real-time requirements was viewed as a big obstacle by participants. The following is a summary of the discussion on this topic:

- Multicore is challenging and complex; there has not been a specific and standardized way to get around that.
- We want a deterministic system using hardware that is non-deterministic because multicore has shared resources.
- Evidence for multicore is extremely hard to get; most developers do not know what it is and how to get it.
- We need assurance arguments for multicore that are easy to understand and compelling. A major gap exists in understanding the arguments, especially for multicore interference issues.
- What are the interferences? Spell them out. Can a family or class or processors have similar patterns of interference? Can't use processors that do not allow for evidence production.
- Currently guidelines like CAST-32A require developers to identify all interference channels and analyze them. How does the certifier know if this is sufficient?
- We should start by defining the evidence that we need and then look at how to produce that.
- Perhaps the best approach as the challenge is that the certifiers have a hard time understanding multicore interference.
- List the interferences and characterize them.
- Some multicore issues are also related to single core; example: cache locking bug.

7 Next Steps

The ASERT workshop participants agreed that the problems considered in this report are very important. In particular, workshop participants felt strongly about the need to solve these problems, which are key to achieving rapid fielding of new capabilities in DoD. The participants expressed a commitment to take the next steps to address the challenges discussed in the workshop. We describe these steps in this section.

7.1 Working Group

A number of ideas were presented headed toward the formation of a working group. While the specific purpose of the working group was still to be defined, a few areas of work are discussed below.

7.1.1 Use of Multicore Verification Techniques in Assurance Claims

This topic is of special interest to the FAA as well as to multiple DoD areas including nuclear surety. One of the key points of interest is the connection of verification techniques to high-level assurance claims.

The connection of timing verification results to high-level assurance claims is of particular interest, given the level of uncertainty in the practitioner community about the hidden effects in multicore systems and their impact on assurance. This connection is frequently ignored by both practitioners and researchers. Practitioners ignore it because timing is typically hidden from the abstractions they use, and the timing misbehaviors are rare and hard to reproduce. Conversely, researchers ignore it because they tend not to explore how timing faults affect other properties such as communication integrity, coordination, replication integrity, security, etc.

7.1.2 Interaction with other working groups and certification bodies

This topic includes groups like the working group MIL-STD 882 that work on the safety implications of complex software and AI/Autonomy, airworthiness certification groups (516 related), groups working on DevOps, the international system safety society, and the group in ASTM on safety standard for software in space vehicles.

Similarly, related standards and their standardization bodies such as the DO-254, DO-178C, Airworthiness 516 should be considered.

7.1.3 Formal Assurance Case Standardization

One of the most important challenges identified in the workshop was the standardization of assurance cases. Moreover, it was identified that to make it practical, we need **to automate the assurance case processing**, hence their formalization is paramount as discussed in previous sections.

7.1.4 Regular Interactions

A motion to meet on a monthly basis was suggested. The initial agreement was to use the workshop report as our first interaction. The report itself will contain the interaction plan for the working group.

7.1.5 Information-Sharing Infrastructure

As part of the infrastructure for the working group we will define the information-sharing infrastructure to be used. We will favor agility and simplicity, taking advantage of the desire to keep the information public.

7.2 Model Problems

Model problems can be a critical artifact that allows us to refine our ideas and test their effectiveness. It would be important to involve people from the different roles involved in the certification pipeline to increase the richness and effectiveness of the model problems.

7.3 Grand Challenge

We believe that one vehicle to increase the energy and pace of results in this area is to formulate a grand challenge that can drive the definition of model problems, argumentation standardization and its automation, and development of a funding structure. This grand challenge should clearly define problems for which the state of the art and practice are lacking, clearly defines the value to the funding agencies and users and demonstrates how solving this challenge breaks the current barriers of certification of continuously evolving critical systems.

References/Bibliography

URLs are valid as of the publication date of this report.

[1] Timothy Patrick Kelly, “Arguing Safety – A Systematic Approach to Managing Safety Cases,” PhD thesis, University of York, 1998.

[2] J. Michael McQuade and Richard M. Murray (co-chairs) Gilman Louie, Milo Medin, Jennifer Pahlka, Trae' Stephens, "Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage Defense Innovation Board," May 3, 2019.

[3] I. Bate and T. Kelly, "Architectural Considerations in the Certification of Modular Systems," Reliability Engineering and System Safety, 2003.

[4] Anita Carleton,, John Robert, Jeff Boleng, Mark Klein, Forrest Shull, Greg Shannon, Doug Schmidt, John Foreman, Charlie Holland, and Erin Harper. “National Agenda for Software Engineering Research and Development: Architecting the Systems of the Future.” Carnegie-Mellon University, Pittsburgh, PA, 2020.

[5] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, “Bounding memory interference delay in COTS-based multi-core systems,” RTAS, 2014.

[6] B. Andersson, H. Kim, D. De Niz, M. Klein, R. Rajkumar, and J. Lehoczky, “Schedulability Analysis of Tasks with Corunner-Dependent Execution Times,” ACM TECS, 2018.

[7] H. Kim, D. de Niz, B. Andersson, M. Klein and J. Lehoczky, "Addressing Multi-core Timing Interference using Co-Runner Locking," IEEE Real-Time Systems Symposium (RTSS), 2021.

[8] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013.

[9] H. Yun, R. Mancuso, Z. -P. Wu and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014.

[10] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, “Bounding and reducing memory interference in COTS-based multi-core systems,” Real-Time Systems 52, 356–395 (2016).

[11] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, R. Rajkumar, “Coordinated bank and cache coloring for temporal protection of memory accesses,” IEEE 16th International Conference on Computational Science and Engineering, 2013.

[12] H. Kim, A. Kandhalu, and R. Rajkumar. “A coordinated approach for practical OS-level cache management in multi-core real-time systems,” Euromicro Conference on Real-Time Systems (ECRTS), 2013.

[13] H. Kim, and R. Rajkumar. “Real-time cache management for multi-core virtualization,” International Conference on Embedded Software (EMSOFT), 2016.

[14] D. de Niz et al. “Mixed-trust computing for real-time systems,” IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2019.

[15] D. de Niz, B. Andersson, H. Kim, M. Klein, L.T.X. Phan, and R. Rajkumar, “Mixed-criticality processing pipelines,” Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017.

[16] D. de Niz, K. Lakshmanan, and R. Rajkumar, “On the scheduling of mixed-criticality real-time task sets,” IEEE Real-Time Systems Symposium (RTSS), 2009.

[17] K. Lakshmanan, D. de Niz, R. Rajkumar, and G. Moreno, “Resource allocation in distributed mixed-criticality cyber-physical systems,” IEEE International Conference on Distributed Computing Systems (ICDCS), 2010.

[18] K. Lakshmanan, D. de Niz, and R. Rajkumar, “Mixed-Criticality Task Synchronization in Zero-Slack Scheduling,” IEEE RTAS, 2011.

[19] K. Lakshmanan, D. de Niz, R. Rajkumar, and G. Moreno, “Overload provisioning in mixed-criticality cyber-physical systems,” ACM Transactions on Embedded Computing Systems (TECS) 11.4 (2013): 1-24.

[20] Alan Burns and Robert I. Davis. "A survey of research into mixed criticality systems." ACM Computing Surveys (CSUR) 50.6 (2017): 1-37.

[21] J. Anderson, S. Baruah, and B. Brandenburg, “Multicore operating-system support for mixed criticality,” Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, 2009.

[22] M. Mollison , J. Erickson, J. Anderson, S. Baruah , and J. Scoredos, “Mixed Criticality Real-Time Scheduling for Multicore Systems,” IEEE International Conf. on Embedded Software and Systems, 2010.

[23] J. Herman, C. Kenna , M. Mollison , J. Anderson, and D. Johnson, “RTOS Support for Multicore Mixed-Criticality Systems,” IEEE RTAS, 2012.

[24] N. Kim, B. Ward, M. Chisholm, C. Y. Fu, J. Anderson, and F.D. Smith, “Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning,” IEEE RTAS, 2016.

[25] M. Chisholm, N. Kim, S. Tang, N. Otterness , J. Anderson, F.D. Smith, and D. Porter, “Supporting Mode Changes while Providing Hardware Isolation in Mixed-Criticality Multicore Systems,” RTNS, 2017.

[26] Yu Li and Albert M. K. Cheng, “Toward a Practical Regularity-based Model: The Impact of Evenly Distributed Temporal Resource Partitions,’” ACM Transactions on Embedded Computing Systems (TECS), Volume 16, Issue 4, Article No. 111, August 2017.

[27] G. Dai, P. Paluri, A. Cheng and B. Liu, "Regularity-Based Virtualization Under the ARINC 653 Standard for Embedded Systems," IEEE Transactions on Computers, vol. 71, no. 10, pp. 2592-2605, 2022.

[28] P. K. Paluri, G. Dai, and A. M. K. Cheng, “ARINC 653-Inspired Regularity-Based Resource Partitioning on Xen,” ACM International Conference on Languages Compilers, Tools and Theory of Embedded Systems (LCTES), 2021.

[29] Elena Torre and Albert M. K. Cheng, “Fault Tolerance in a Two-State Checkpointing Regularity-Based System,” IEEE Real-Time Systems Symposium (RTSS), 2020.

[30] M. Klein, T. Ralya, N. Pollak, R. Obenza, and M. G. Harbour. A practitioner’s handbook for real-time analysis: guide to rate monotonic analysis for real-time systems. Springer Science & Business Media, 1993.

[31] L. Liu and J. W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment." Journal of the ACM (JACM) 20.1 (1973): 46-61.

[33] S. Hosseinimotlagh and H. Kim. "Thermal-aware servers for real-time tasks on multi-core GPU-integrated embedded systems." IEEE RTAS, 2019.

[34] S. Hosseinimotlagh, A. Ghahremannezhad, and H. Kim. "On dynamic thermal conditions in mixed-criticality systems." IEEE RTAS 2020.

[35] Y. Lee, H. S. Chwa, K. G. Shin, S. Wang, “Thermal-aware resource management for embedded real-time systems,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11), 2857-2868.

[36] F. Jahanian, R. Ragunathan, and S. Raju. "Runtime monitoring of timing constraints in distributed real-time systems." Real-Time Systems 7, no. 3 (1994): 247-273.

[37] H. Kim, S. Yi, W. Jung, H. Cha, “A decentralized approach for monitoring timing constraints of event flows,” IEEE Real-Time Systems Symposium (RTSS), 2010.

[38] T. Amert, Z. Tong, S. Voronov, J. Bakita, F.D. Smith, and J.H. Anderson, “Timewall: Enabling time partitioning for real-time multicore+ accelerator platforms,” IEEE Real-Time Systems Symposium (RTSS), 2021.

[39] L. Sha, R. Ragunathan Rajkumar, and J. P. Lehoczky. "Priority inheritance protocols: An approach to real-time synchronization." IEEE Transactions on computers 39.9 (1990): 1175-1185.

[40] R. Rajkumar, L. Sha, and J. P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors." RTSS. 1988.

[41] A. Block, H. Leontyev, B. Brandenburg, and J. H. Anderson. "A flexible real-time locking protocol for multiprocessors." IEEE international conference on embedded and real-time computing systems and applications (RTCSA), 2007.