On the Transition from Design Time to Runtime Model-Based Assurance Cases

Ran Wei
University of York
United Kingdom
ran.wei@york.ac.uk

Jan Reich
Fraunhofer Institute for Experimental Software Engineering (IESE)
Germany
jan.reich@iese.fraunhofer.de

Tim Kelly
University of York
United Kingdom
tim.kelly@york.ac.uk

Simos Gerasimou
University of York
United Kingdom
simos.gerasimou@york.ac.uk

ABSTRACT

System assurance cases are used to demonstrate confidence in system properties of interest (e.g. safety and/or security). They are key artefacts for safety and/or security acceptance for systems before they become operational.

Cyber-Physical Systems (CPS) form a new technological frontier for their vast economic and societal potentials in various domains. CPS are often safety-critical systems. Thus, their safety and/or security need to be assured using system assurance cases. However, due to the open and adaptive nature of CPS, the need for system assurance at runtime is imperative.

Therefore, assurance cases are expected to be exchanged, integrated and verified at runtime to ensure the dependability of CPS when they intend to execute a cooperative behaviour.

In this position paper, we identify the importance of model-based system assurance, we discuss the paradigm shift of assurance cases from being manually created artefacts to (semi-)automatically created models. We discuss the application of model-based assurance cases in ensuring the dependability of CPS.

CCS CONCEPTS

• **General and reference → Reliability**;

KEYWORDS

Model Driven Engineering, Models at Runtime, Structured Assurance Case Metamodel, System Assurance

1 INTRODUCTION

Systems/services used to perform critical functions require justifications that they exhibit necessary properties (i.e. safety and/or security). *Assurance case*s provide an explicit means for justifying and assessing confidence in these critical properties. An assurance case is a document that facilitates information exchange between various system stakeholders (e.g. between operator and regulator), where the knowledge related to the safety and/or security of the system is communicated in a clear and defendable way [1]. Assurance cases are key artefacts for safety/security acceptance for systems/services before they become operational.

The physical and digital worlds are gradually merging into a largely connected globe. This is backed by the emergence of concepts such as Cyber-Physical Systems (CPS). Openness and adaptivity are core properties of CPS as constituent systems dynamically connect to each other and have to adapt to a changing context at runtime [2]. CPS harbour the potential for vast economic and societal impact in domains such as automotive, health care, and home automation due to their open and adaptive nature [3]. The majority of application domains of CPS are safety-critical, such as car2car scenarios and collaborative autonomous mobile systems. If these systems fail, they may cause harm and lead to temporary collapse of important infrastructures, with catastrophic consequences for industry and society. Therefore, it is imperative to ensure the dependability of CPS in order to realise their full potential. However, the open and adaptive nature of CPS poses significant challenges to assuring such systems, as it is nearly impossible to anticipate the concrete CPS structure, its capabilities and the environmental context sufficiently at design time.

Therefore, existing design time system assurance activities are inappropriate to enable dynamic system assurance for CPS at runtime. Thus, a paradigm shift for system assurance activities from design time to runtime is needed to assure dependability properties of CPS. One important aspect of this shift is the transition from design time assurance cases crafted from manually created documents to runtime assurance case *models* being (semi-)automatically synthesized and evaluated.

In this position paper, we discuss the notion of assurance cases and the importance of a model-based approach in system assurance, motivated by open adaptive safety-related systems. We discuss the state of practice in system assurance cases and the importance of system assurance case models@runtime for CPS. We discuss in detail our vision of runtime system assurance cases and how they help CPS to reason about the safety and/or security at runtime by themselves. We point out potential research directions towards model-based system assurance at runtime.

2 ASSURANCE CASES

The concept of assurance cases has been long established in the safety-related domain, where the term *safety case* is normally used. For many industries, the development, review and acceptance of a safety case are key elements of regulatory processes. This includes

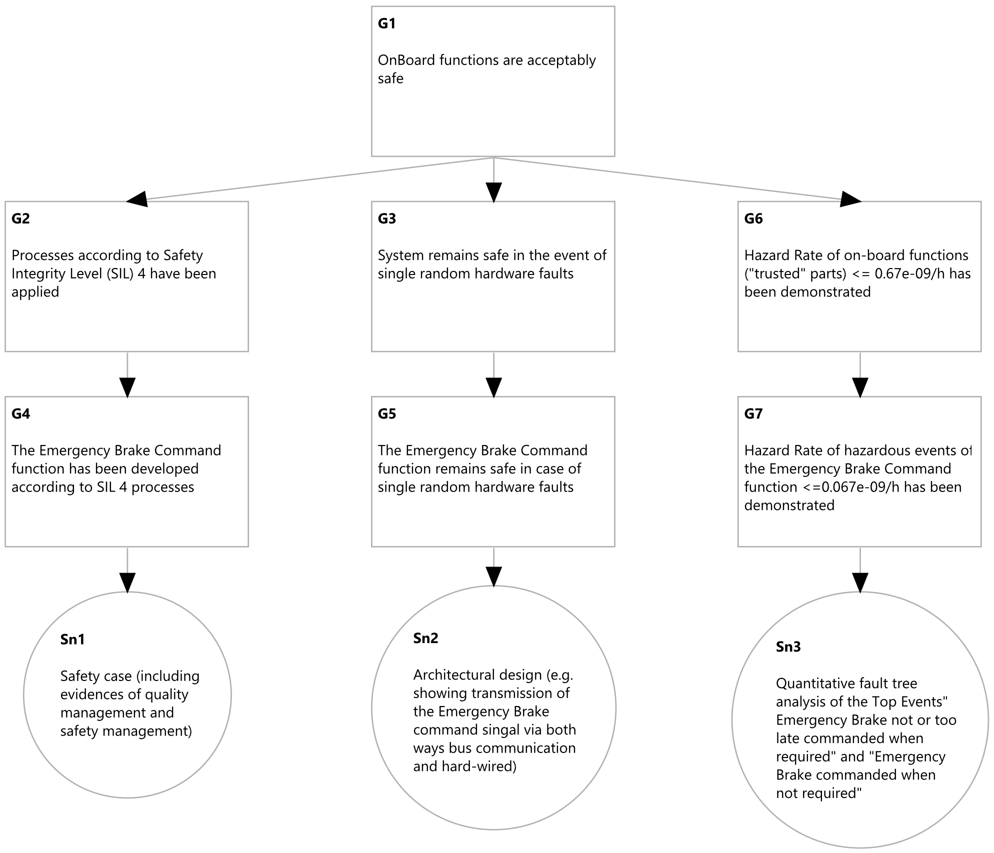
G1
OnBoard functions are acceptably safe
G2
Processes according to Safety Integrity Level (SIL) 4 have been applied
G3
System remains safe in the event of single random hardware faults
G6
Hazard Rate of on-board functions ("trusted" parts) <= 0.67e-09/h has been demonstrated
G4
The Emergency Brake Command function has been developed according to SIL 4 processes
G5
The Emergency Brake Command function remains safe in case of single random hardware faults
G7
Hazard Rate of hazardous events of the Emergency Brake Command function <=0.067e-09/h has been demonstrated
Sn1
Safety case (including evidences of quality management and safety management)
Sn2
Architectural design (e.g. showing transmission of the Emergency Brake command singal via both ways bus communication and hard-wired)
Sn3
Quantitative fault tree analysis of the Top Events" Emergency Brake not or too late commanded when required" and "Emergency Brake commanded when not required"

Figure 1: An example GSN safety case for the European Train Control Systems (ETCS5).

the nuclear [4], defence [5], civil aviation [6] and railway [7] industries. The concept of safety case is defined in [8] as follows: *A safety case communicates a clear, comprehensible and defensible argument that a system is acceptably safe to operate in a particular context.*

Historically, safety arguments were typically communicated in safety cases through free text. However, problems are experienced when text is the only medium available for expressing complex arguments [8]. One problem of using free text is that if the language used in the text is unclear and poorly structured, there is no guarantee that system engineers would produce safety cases with clear and well-structured language. To overcome these problems, graphical argumentation notations were developed. Graphical argumentation notations are capable of explicitly representing the elements that form a safety argument (i.e. requirements, claims, evidence and context), and the relationships between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). Amongst the graphical notations, the Goal Structuring Notation (GSN) [9] has been widely accepted and adopted [10]. The key benefit experienced by companies/organisations of adopting GSN is that it improves the comprehension of the safety argument amongst all key project stakeholders, therefore improving the quality of the debate and discussion amongst the stakeholders and reducing the time taken to reach agreement on the argument approaches being adopted. An example *safety case* created using GSN is provided in Figure 1[?]. It can be seen that graphical argumentation notations make it easier to identify safety goals and evidence/artefacts that satisfy those goals.

The concept of structured argumentation is also used in other domains, particularly for demonstrating system security [11]. Thus, the term *Assurance Case* is a broader definition, that an assurance case is used to demonstrate confidence in system safety and/or security.

3 MODEL-BASED ASSURANCE CASES

In the current state of practice, assurance cases are manually created documents rather than models. This is due to the nature of the intended usage of assurance cases – that they are meant to be reviewed by safety/security experts and are used to improve the quality of discussion.

In recent years there has been a tendency for system assurance tools to adopt Model-Driven Engineering (MDE) in order to benefit from improved efficiency and consistency provided by MDE. In [12], the authors identified the need for model-based solutions for GSN pattern instantiation. GSN patterns are abstract safety case templates which capture good practice of system assurance, and

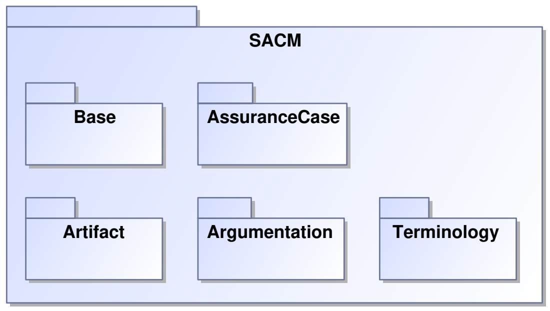

Figure 2: Packages of SACM

can be *instantiated* by linking system information (model) with the elements in the GSN pattern to create a concrete safety case. The authors proposed the use of an intermediate model called the *weaving model* to link external model elements with GSN model elements for pattern instantiation.

To promote standardisation and interoperability, the Object Management Group (OMG) specified and issued the *Structured Assurance Case Metamodel* (SACM) [13]. SACM provides a sound solution for model-based system assurance case construction, in the sense that it provides mechanisms to model evidence and related information used in a structured assurance case. As shown in Figure 2, SACM provides packages that allow practitioners to group their structured argumentations (via the Argumentation package), corresponding evidences (via the Artifact package) and controlled languages (via the Terminology) into atomic packages, and incorporates them into assurance case packages. In this sense, SACM is more powerful than GSN for its ability to reference and relate various kinds of artefacts (e.g. system design models) and its ability to use controlled vocabularies.

In summary, in the current state of practice, assurance cases are still manually created design time documents to certify the safety and/or security of systems. There are tools that adopt MDE to aid the creation of assurance cases. The Structured Assurance Case Metamodel (SACM) is still in its infancy and there is currently no tool available implementing SACM.

4 THE NEED FOR MODEL-BASED ASSURANCE CASES AT RUNTIME

It can be seen that MDE factors in approaches for system assurance mainly focusing on exploiting the benefits of MDE for improved efficiency and automation. However, the increasing complexity of Cyber-Physical Systems (CPS) boosts the need for model-based approaches for system assurance. The premise of pervasive applications of CPS to realise its economic and societal impact is that the safety and/or security of CPS are ensured, so that failures of such systems do not cause damage of different severities (ranges from moderate to catastrophic).

CPS are typically loosely connected systems and come together as temporary configurations of constituent systems, which dissolve and give place to other configurations. Therefore, the configurations a CPS may assume over its lifetime are unknown and potentially infinite. This makes currently available approaches for system assurance insufficient to assure the safety and/or security of CPS, due to the fact that system assurance activities are typically required at runtime for CPS.

Runtime system assurance for CPS has been identified as one of the major challenges towards the application of CPS [2, 3]. One common vision toward runtime system assurance is the exchange of models@runtime, which is the upcoming paradigm for the development of CPS. As it is not possible to anticipate the runtime context for CPS at design time, Models@Runtime follows the idea of making important models available at runtime in order to enable the system itself to reflect on its current (safety) state based on monitoring the runtime context. By generating context-awareness in this manner, potentially required adaptation strategies can be planned and executed in order to maintain safe or achieve optimized CPS behaviour.

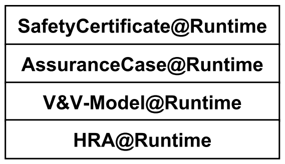

Figure 3: Models@Runtime on different abstraction levels.

In [2], the authors identified models on four abstraction levels, which can be exchanged at runtime, as shown in Figure 3. The most abstract model that CPS can exchange are safety certificate models. The idea of certification at runtime was introduced in [14]. Safety certificates at runtime describe a formal safety interface using contract-like interface specifications defining, which safety properties can be guaranteed by the system under the assumption that specific safety demands are fulfilled by the integration context. In [15], the authors introduce a concrete form of runtime safety certificates called *Conditional Safety Certificates* (ConSerts), which are modular and contract-based definitions of safety certificates factoring in variants through Boolean mappings between different sets of safety guarantees and demands.

Sometimes it is imperative to know at runtime, how a CPS reaches the conclusion that the safety properties provided at runtime are guaranteed. In such cases, it is necessary to exchange assurance case models at runtime, such that the argumentation leading to the guarantees of the safety properties can be accessed and reviewed in an automated way.

In case the system adaptations lead to invalid evidence, a revalidation of evidence can be triggered at runtime. This implies that a set of pre-defined V&V (Verification & Validation) activities need to be performed at runtime. V&V models can be used to execute V&V activities (regression testing, generation of test cases, etc.) at runtime. However, it is a significant challenge to have V&V models at runtime, typically because it is a rather difficult step for developers at development time, the difficulties to shift this step to runtime would be multi-fold. In addition, it is almost impossible to perform extensive V&V activities at runtime, provided that real time requirements may be in place for CPS.

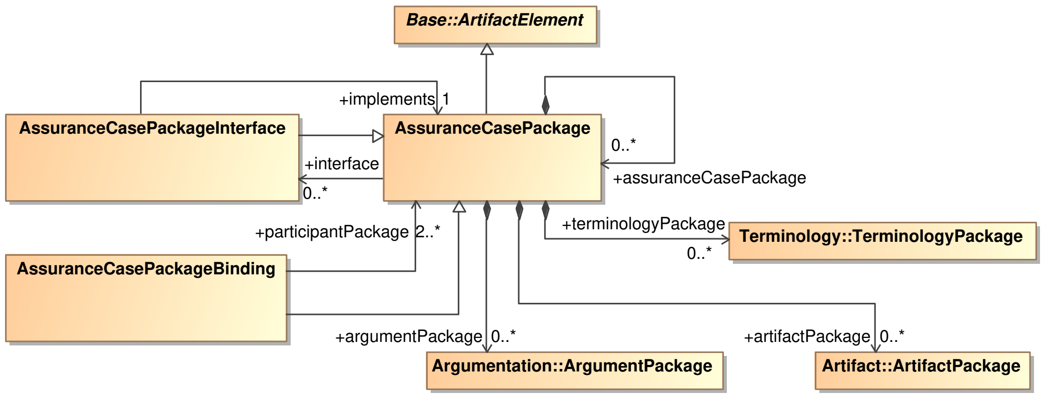

Figure 4: An overview of the structure of an assurance case.

If requirements are modified at development, from a safety engineering perspective, Hazard Analysis and Risk Assessment activities (HARA) need to be performed. Hence, the idea of HARA models at runtime. However, since HARA is a very creative process and often relies on the experience of system safety experts, it is very unlikely that HARA can be performed at runtime.

Apart from the works previously mentioned on safety certificate models at runtime, no work has been done in the line of assurance case models at runtime. This is due to the fact that system engineers still perceive assurance cases as design time artifacts. However, as discussed in this section, the need for assurance case models at runtime is imperative.

5 SACM TOWARDS RUNTIME ASSURANCE CASE MODELS

In this section, we discuss the intended usage of Structured Assurance Case Metamodel (SACM) since we have been involved, to a large extent, in the specification of SACM, and it has not been sufficiently explained since the release of SACM v2.0.

In SACM, an assurance case model contains a number of packages as shown in Figure 4. SACM organises assurance cases in packages to promote modularity. An assurance case package can contain a number of argument packages, artifact packages and terminology packages. Argument packages store information about the argumentation part of an assurance case, where safety/security claims are broken down into sub claims until they are directly backed by evidence.

Evidence used in the argument packages can be modeled and organised in artifact pakcages. For example, a hazard analysis model can be recorded in an artifact package, the user may also specify when the analysis is performed, who participated in the analysis process and what techniques are used in the process.

SACM also provides the mechanisms to create controlled natural languages so that the users can establish a finer grade of reference to system models. The terminology package of SACM provides the mechanisms to create *Expressions*, *Categories* and *Terms*. An example of controlled language is shown in Figure 5. The upper part of the figure is a claim: *Hazard H1* is sufficiently mitigated.

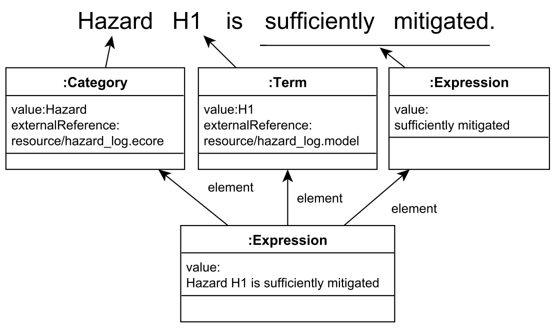

Figure 5: An example of controlled terminology

In this claim, the user can refer to expression elements in their terminology packages. For example, *Hazard* may refer to a *Category* in the terminology package, which in turn points to a hazard log meta-model through its *externalReference* property. In this way, hazard log meta-model provides a definition of what a *Hazard* is. Then, hazard *H1* can refer to a *Term* in the terminology package, which in turn refers to an instance hazard log model (that conforms to the hazard log meta-model). The hazard log model may then contain information on how *H1* is identified, its causes and consequences, etc. The *Expression sufficiently mitigated* is recorded in the terminology package so that it can be reused. The user is also free to add any explanatory information to the *Expression* so that it better explains what *sufficiently mitigated* means. Finally, an overall *Expression* which references the three previous elements is created. This expression can be referenced in the argumentation package (e.g. as a description of a *Claim*).

SACM promotes modularity, in the sense that elements are organised in different packages. To refine modularity, SACM provides three different types of packages. Figure 6 shows a segment of the meta-model for the argument package of SACM, illustrating the three types of packages in the argumentation package of SACM. *ArgumentPackage* is the main package in which structured argumentation is stored. The users can disclose part of the argumentation

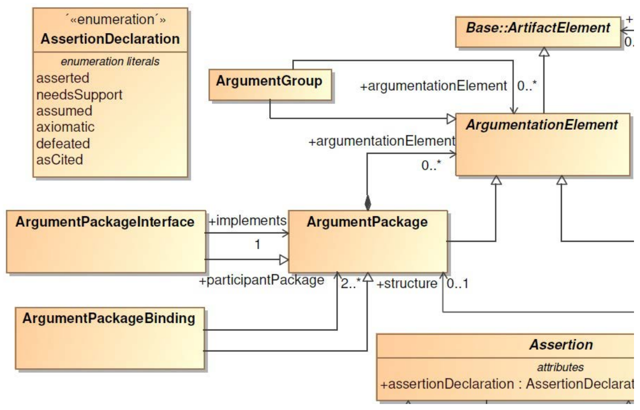

Figure 6: Argumentation packages.

externally with the use of *ArgumentPackageInterface*s. To do this, in *ArgumentPackageInterface*s, citation elements need to be created which *cite* to original elements in *ArgumentPackage*s. Figure 7 shows a segment of SACM for the citation mechanism. All *SACMElement*s have the capability of citing other *SACMElement*s via the *+citedElement* reference. If an element cites another, it automatically becomes abstract and citation via its *+isAbstract* and *+isCitation* features. *ArgumentPackageInterface* only contains citation elements, it should be enforced by constraints on the meta-model.

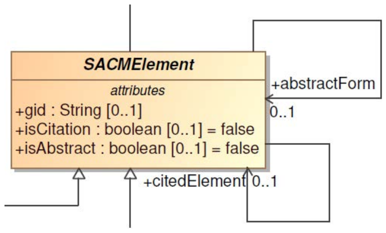

Figure 7: Citation mechanism of SACM.

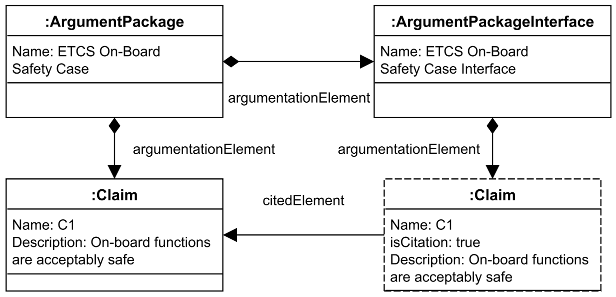
:ArgumentPackage
Name: ETCS On-Board Safety Case
:ArgumentPackageInterface
Name: ETCS On-Board Safety Case Interface
argumentationElement
argumentationElement
argumentationElement
:Claim
Name: C1
Description: On-board functions are acceptably safe
citedElement
:Claim
Name: C1
isCitation: true
Description: On-board functions are acceptably safe

Figure 8: An example ArgumentPackageInterface.

An example use of *ArgumentPackageInterface* is illustrated in Figure 8, this example is taken from the DEIS project [3] for the ETCS (European Train Control System) use case. The ETCS use case contains a CPS examined in the DEIS project, which consists of two major constituent systems at runtime: the on-board system which is installed on trains and the trackside system, which is installed on important nodes at the track side (such as stations and junctions). The left-hand side of the figure shows the *ArgumentPackage* of the ETCS on-board system (only top level *Claim* is shown here). The right-hand side of the figure shows the *ArgumentPackageInterface*, where a citation *Claim* is created to cite *Claim* C1 in the *ArgumentPackage*.

ArgumentPackageBinding is used to *bind ArgumentPackage*s together by referencing elements stored in their *ArgumentPackageInterface*s. To integrate *ArgumentPackage*s, integration engineer needs to create an *ArgumentPackageBinding*, and create again citation elements in the *ArgumentPackageBinding*, which in turn cite the elements in *ArgumentPackageInterface*s from the *ArgumentPackage*s. An example of the use of *ArgumentPackageBinding* is shown in Figure 9.

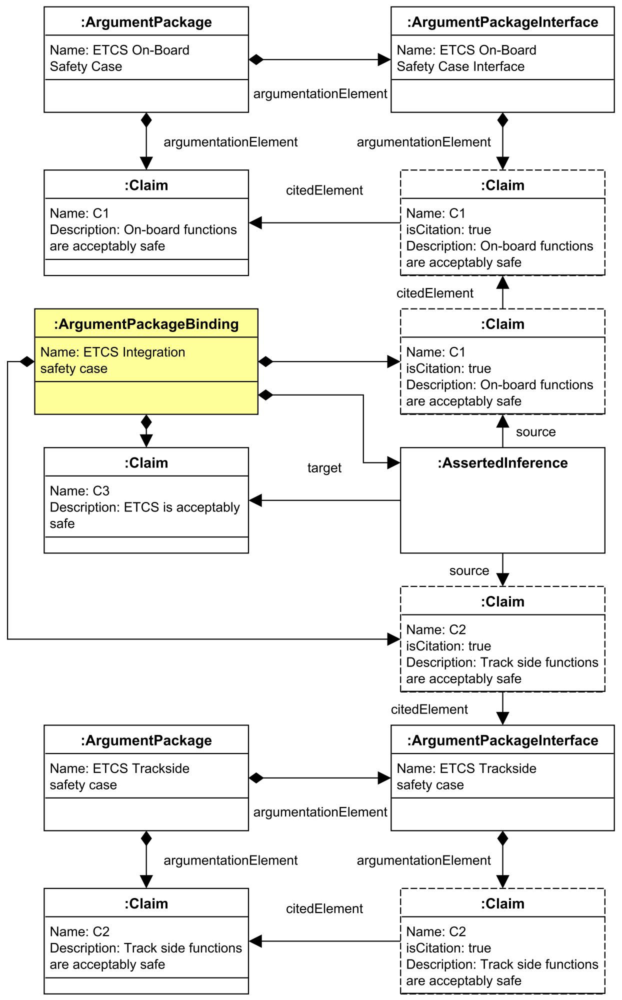
:ArgumentPackage
Name: ETCS On-Board Safety Case
:ArgumentPackageInterface
Name: ETCS On-Board Safety Case Interface
argumentationElement
argumentationElement
argumentationElement
:Claim
Name: C1
Description: On-board functions are acceptably safe
citedElement
:Claim
Name: C1
isCitation: true
Description: On-board functions are acceptably safe
citedElement
:ArgumentPackageBinding
Name: ETCS Integration safety case
:Claim
Name: C1
isCitation: true
Description: On-board functions are acceptably safe
source
:Claim
Name: C3
Description: ETCS is acceptably safe
target
:AssertedInference
source
:Claim
Name: C2
isCitation: true
Description: Track side functions are acceptably safe
citedElement
:ArgumentPackage
Name: ETCS Trackside safety case
:ArgumentPackageInterface
Name: ETCS Trackside safety case
argumentationElement
argumentationElement
argumentationElement
:Claim
Name: C2
Description: Track side functions are acceptably safe
citedElement
:Claim
Name: C2
isCitation: true
Description: Track side functions are acceptably safe

Figure 9: An example ArgumentPackageBinding.

The upper part of Figure 9 is the *ArgumentPackage* of the ETCS on-board system, and the lower part of Figure 9 is the *ArgumentPackage* of the ETCS Trackside system. The *ArgumentPackageBinding* (rendered in yellow) is created to bind the *ArgumentPackage*s of the ETCS parts. Within the *ArgumentPackageBinding*, an overall Claim C3 is created, which are backed by the citation Claims C1 and C2, which in turn cites the citation Claims in ETCS On-board safety case and ETCS Trackside safety case, respectively.

The use of packages, interfaces and bindings are the key mechanisms for the integration of SACM packages. All packages (artifact, terminology, argument and assurance case) can be integrated using this approach.

For assurance case models created using SACM to be exchanged at runtime, there is also a need for automated system integration. At the moment, arguments in assurance cases are described using natural language. To enable automation, machine-processable languages needs to be incorporated in the argument in assurance cases. With regards to this need, in SACM v2.0 the notion of *MultiLangString* is introduced. Using *MultiLangString* enables the user to describe one claim using different languages, including computer languages. This gives the possibility of automated reasoning for safety cases.

In addition, for CPS to integrate, there is also a need to express supply and demand of services for CPS. The supply of service is provided by default in assurance cases, as assurance cases created using SACM are able to relate to system models. To express demand of services, SACM enables the users to create *Claim*s with declarations. For example, to express demand, a *Claim* can be declared as *needsSupport*. To quantify demand and supply, SACM enables the user to also create *ImplementationConstraint*s and associate them to elements in the assurance case. By using *ImplementationConstraint*s, system assurance engineers are able to express what kind of guarantees are provided/needed.

The DEIS project, which aims at assuring the dependability of CPS, uses SACM as the backbone for its core concept - the Open Dependability Exchange (ODE) meta-model. The ODE is a versatile meta-model, which enables CPS developers to capture various aspects of CPS, including architecture models, HARA models, failure logic models as well as assurance case models (via usage of SACM).

In our vision, the runtime integration of two CPS (A and B) includes the following steps:

(1) Exchange of assurance case interfaces for CPS A and CPS B, which includes the demand/supply of services, as well as the guarantees needed/provided for these services;
(2) If CPS A questions the soundness of the guarantees provided by CPS B, CPS B would provide its assurance case;
(3) CPS A performs reasoning of the assurance case of CPS B;
(4) If the assurance case of CPS B is sound, connection between CPS A and CPS B will be established. An assurance case binding will be created and maintained locally for both CPS A and CPS B until they decide to disconnect from each other;
(5) If CPS A continues to question the soundness of the assurance case of CPS B, CPS B can choose to send further evidence, such as related architecture or failure logic models
(6) If the evidence provided by CPS B is sound, step (4) will be performed. Otherwise, the adaptation process ends.

6 CONCLUSION

In this paper, we talked about the need to shift assurance cases from conventionally design time documents to automatically exchanged and integrated runtime models for CPS. We discussed and presented our knowledge on the Structure Assurance Case Metamodel and how it can be used to create model-based assurance cases.

SACM lays a foundation for system assurance of CPS at runtime. Assurance case models for CPS are living models in the sense that monitoring devices provide essential evidence for assurance cases and the validity of the assurance cases are constantly verified.

In case an assurance case is invalidated, the CPS carrying it should make note of it. Therefore, it is anticipated that a CPS would carry a repository of assurance cases, which helps CPS developers to better understand what goes wrong at runtime.

CPS system assurance is a new research frontier and has gained increasing popularity in recent years. Runtime assurance cases provide a promising solution for assuring safety-related CPS. The MRT community should be aware of the complexity of CPS and the need for assurance case models at runtime.

Acknowledgements This work is supported by the European Union's Horizon 2020 research and innovation programme through the DEIS project (grant agreement No 732242).

REFERENCES

[1] Richard Hawkins, Ibrahim Habli, Tim Kelly, and John McDermid. Assurance cases and prescriptive software safety certification: A comparative study. *Safety science*, 59:55–71, 2013.
[2] Mario Trapp, Daniel Schneider, and Peter Liggesmeyer. A safety roadmap to cyber-physical systems. In *Perspectives on the future of software engineering*, pages 81–94. Springer, 2013.
[3] Ran Wei, Tim P Kelly, Richard Hawkins, and Eric Armengaud. Deis: Dependability engineering innovation for cyber-physical systems. In *Federation of International Conferences on Software Technologies: Applications and Foundations*, pages 409–416. Springer, 2017.
[4] Health and Safety Executive (HSE). *Safety Assessment Principles for Nuclear Facilities.* 2006.
[5] UK Ministry of Defence (MoD). *Defence Standard 00-56 Issue 4: Safety Management Requirements for Defence Systems.* 2007.
[6] Safety Regulation Group Civil Aviation Authority (CAA). *CAP 670 - Air Traffic Services Safety Requirements.* 2007.
[7] Rail Safety and Standards Board. *Engineering Safety Management (The Yellow Book).* 2007.
[8] Tim Kelly and Rob Weaver. The goal structuring notation–a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, page 6. Citeseer, 2004.
[9] GSN Working Group Online - The Goal Structuring Notation. http://www.goalstructuringnotation.info/. Accessed: 26-04-2018.
[10] Paul Chinneck, David Pumfrey, and Tim Kelly. Turning up the heat on safety case construction. In *Practical Elements of Safety*, pages 223–240. Springer, 2004.
[11] Robin Bloomfield and Peter Bishop. Safety and assurance cases: Past, present and possible future–an adelard perspective. In *Making Systems Safer*, pages 51–67. Springer, 2010.
[12] Richard Hawkins, Ibrahim Habli, Dimitris Kolovos, Richard Paige, and Tim Kelly. Weaving an assurance case from design: a model-based approach. In *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, pages 110–117. IEEE, 2015.
[13] Structured Assurance Case Metamodel, Object Management Group. https://www.omg.org/spec/SACM/About-SACM/. Accessed: 06-04-2018.
[14] John Rushby. Runtime certification. In *International Workshop on Runtime Verification*, pages 21–35. Springer, 2008.
[15] Daniel Schneider and Mario Trapp. A safety engineering framework for open adaptive systems. In *Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on*, pages 89–98. IEEE, 2011.