Generation and Verification of Executable Assurance Case by Model-based Engineering

Fang Yan
Department of Computer Science
University of York
York, UK
email: fang.yan@york.ac.uk

***Abstract*—Assurance Cases (ACs) are used for justifying system confidence in important properties including safety, reliability, etc. Their manual generation is time-consuming and prone to errors. Also, AC update calls for more labour. However, there is not an automatic solution to guide the whole engineering process of AC generation and verification process. An executable AC is machine readable and checkable, and brings the benefit of efficiency and confidence of AC evolution. Thus, in this PhD, the Model-based Engineering (MBE) techniques are exploited for an automatic process for executable ACs. The first aim is to generate AC models automatically from system artefacts. Currently available approaches are usually constrained to specific modelling environments, or address only system model artefacts, or do not cover informal and unstructured artefacts. The second aim is to automate the evidence generation using formal verification. FM provides a rigorously mathematical proof. But current solutions to create formal assertions are manual and expertise-requiring. The paper discusses on the technical problem, and the proposed approach.**

***Index Terms*—Assurance Case, generation, SACM, model query, formal verification, assertion**

I. Introduction

The properties of reliability, safety, maintainability, etc. are critical for complex systems. The Assurance Case (AC) is a systematic way to argue that the system exhibits some of these properties supported by relevant evidences [1], and is required by ISO26262 [2] for safety-critical systems. AC generation involves large amounts of system lifecycle data and various types of verification methods; and is labour-intensive and error-prone if processed manually. As system design changes and evolves frequently during development and operational phase, ACs needs to be updated correspondingly to keep the validity of the argued property. For instance, when a system function is added, a new AC claim must be created and substantiated by argument; when the detailed design has been modified, the evidence to support the claim needs to be re-verified. It is important to provide an automatic approach and tool for AC creation. The problem we face is a missing of an automatic solution to guide the whole engineering process of AC generation and formal verification process [3].

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 812.788 (MSCA-ETN SAS). This publication reflects only the author's view, exempting the European Union from any liability. Project website: http://etn-sas.eu/.

An executable AC is machine readable and checkable, and brings benefits of efficiency and confidence of AC generation and evolution. This PhD is to provide an automatic solution for assurance engineers to generate executable AC models by Model-based Engineering (MBE) with formal verification support. We implement the approach using RoboChart [4], a UML-tailored language tailored for development of robotic controllers, which provides block-based architectural and state machine modelling notations. Since its development environment RoboTool is Eclipse-based, our method is generalizable to other Eclipse-based modelling environment. We collaborate with D-RisQ[1] to gather engineering requirements for developing AC process and verify our approach using their robot.

The research questions are: (1) For systems developed with RoboChart, what MBE techniques can be used for automatic AC generation supporting various system modelling languages? (2) For the AC claims that can be verified by Formal Methods (FM), what phases can be automated within RoboTool? And how do we realize the automation?

For question one, much work has addressed AC fragment generation from system data with different levels of automation. Thanks to the maturity of MBE application in system development process, it's possible to generate AC models with system models as inputs. For the purpose of automation of traceability establishment between AC elements and system data, model query techniques have been explored. However, the approaches in the literature on the traceability establishment either do not cover system design models [5]; or address design models (e.g., a component design model) with a limited applicability to different system development environments [6]. Moreover, unstructured system data, such as spreadsheet, is not covered. Thus, one challenge for the first research question is the automatic establishment of traceability between AC models and different formats of system data. This PhD first proposes to unify the formats of system data (including system models and unstructured data) using predefined metamodels, automate the provenance by model query, then generate AC structure automatically. Some AC evidence can also be created from model query results. The approach is independent of system data formats and modelling languages.

The AC evidence is generated from AC claim verification.

[1]D-RisQ Software Systems. http://www.drisq.com/.

Besides model query mentioned above, other techniques for AC verification exist including simulation, testing, review, FM, etc. FM provides a rigorously mathematical proof and has been applied on AC verification at various levels. But one challenge for the second question is that the formal assertions are usually created manually from AC claims and require FM expertise, which hinders the executable AC realization due to this automation gap in FM verification. This PhD proposes to realize an automatically checkable AC by first deriving formal assertions from claims using Controlled Natural Languages (CNL) and model transformation techniques, then deriving AC evidence from assertion checking results using MBE techniques. Other AC claim verification methods may also be processed automatically, but is out of my research scope.

We implement the approach using Epsilon [7] for model query, transformation, etc. The generated AC models can further be managed with MBE capabilities, such as review by query, comparison for update. These will be addressed in future work. System data is input of AC generation, but data collection from system development process is not addressed in my approach, and will be considered in future work.

We organize the rest of paper as follows. §II discusses related work. §III presents the approach, progress and plan. §IV discusses possible threats to validity. §V concludes.

II. Related Works

Hawkins et al. [8] used the model weaving technique [9] to build the relationship between system models and AC elements at the metamodel level. The AC is generated by the pattern instantiation. Its advantage is that the system models can be extracted automatically. Our approach is inspired by and expands this work to cover not only system design models but also the unstructured data such as hazard analysis result. Gacek et al. [6] generate AC models by querying AADL system models. The query environment is integrated with AADL modelling platform. We refer the model query concept in [6] at design model level. The difference is that our approach has no constraint on system modelling languages and is capable of assembling with AC structure generated from other sources. Šljivo et al. [10] derive ACs from system design pattern by MBE which is different from generating AC directly from system models in our work. Gallina and Nyberg [5] also utilize model query technique to generate AC. The objective models to be queried are the system data (e.g., test plan) compliant with OSLC (Open Services for Lifecycle Collaboration) [11] standard. However, OSLC does not support system design models, e.g., AADL models. The unstructured textual data is not addressed in the work either. Our approach complements this work. Denney and Pai [12] construct a complete AC by automatic pattern instantiation. But the system design data are not required to be models thus the method does not support the generation from system design models using model query.

For FM verification of AC claims, Diskin et al. [13] and Gleirscher et al. [14] both propose to formalize claims as assertions on system formal models, but the process automation is not addressed. Cârlan et al. [15] focuse on the consistency checking between system data and AC elements, and exploits model checking as one of the claim verification methods.

III. Research Approach and Progress

Our approach exploits MBE techniques to facilitate the assembly and verification of AC models in an automatic manner, as shown in Fig. 1. AC models will be compliant with the Structured Assurance Case Metamodel (SACM) [16]. SACM is a unifying standard for assurance cases to express argumentation, artifact traceability, and terminology [17].

The system data produced from system development process can be in any format, including models (e.g., EMF models), and unstructured data (such as spreadsheet, text, etc.). For the data used as AC inputs, we first process and convert the unstructured data to models with predesigned metamodels; then create AC models by querying different system models (hazard analysis, system models, etc.) respectively. The query rules are to be predefined for different scenarios, and kept in a library for reuse. The AC models then shall be integrated as a whole module. Further, the approach tackles the AC formal verification. For the claims to be verified by FM, the safety requirement used to generate these claims are further used to create formal assertions. The textual requirements will be rewritten with a CNL in its tool environment, then be exported in a structured format, such as XML. The assertion templates shall be defined to the target FM verification tools. Then, based on the templates, the formal assertions will be generated automatically by model-to-text transformation. The FM verification results from running these assertions are used to create the AC evidence models. The last step is to integrate the evidence models in to the AC module, or for re-verification, to replace the old evidence models with the newly created ones.

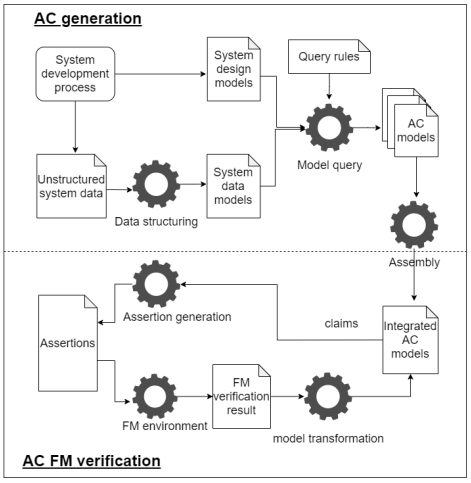

Fig. 1: Automating Generation and FM Verification of ACs

A. AC generation by model query

To create an AC module, many kinds of system data are needed, such as hazard analysis, system specification, system architecture, verification plan and results, most of which are unstructured. We first convert the unstructured data into

structured models with metamodels and query rules. Each type of document shall have its own metamodel and query rules. The metamodels can be designed following industrial standards such as OSLC or company internal standards.

With all relevant data converted into EMF models, we generate AC models based on SACM metamodel, as shown in Fig. 2. For each type of system data (e.g., hazard log, test plan), a set of query rules is designed to create relevant AC elements. Then, these AC models will be integrated into one module according to their built-in relationships.

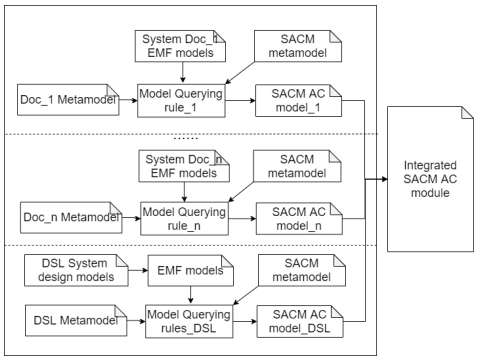

Fig. 2: SACM AC generation by model query

For system design model query, we use RoboChart for system modelling, then obtain EMF models from RoboChart models for model query. Consider a scenario as follows,

Claim: All states that satisfy Condition 1 shall have a transition that satisfies Condition 2.

Evidence: The transition exists for each state.

The query rules of this scenario search all the states in RoboChart models meeting Condition 1, and create claims for each of these states. Then, transitions of each state are checked to identify the ones meeting Condition 2, an evidence and a link to the claim are built for the right transitions. In between, a strategy and a claim for verification are inserted. An AC structure is automatically generated as follows,

Claim 1: {State S.1} that satisfies Condition 1 shall have a transition that satisfies Condition 2.

Inference Strategy 1: Argue over verification and validation methods of {Claim 1}.

> *Claim 1.1: {Claim 1} is verified by model query result.*
>
> *Evidence 1.1: {Transition T.X} exists for {State S.1}.*

...

Claim N: {State S.N} that satisfies Condition 1 shall have a transition that satisfies Condition 2.

Strategy N: Argue over verification and validation methods of {Claim N}.

> *Claim N.1: {Claim N} is verified by model query result.*
>
> *Evidence N.1: {Transition T.Z} exists for {State S.N}.*

The name of states and transitions "State S.i" and "Transition T.X...Z" will be instantiated with concrete RoboChart models. The numbers "N" is the exact number of states that satisfy Condition 1. In this example, the evidence to the claims is also generated by model query which substitutes the manual model review. This again improves the efficiency and avoids errors. The content to be instantiated is presented with {}. Whenever the RoboChart models are modified, e.g., new states added, transitions deleted, AC models can be updated directly from system models automatically. It is noted that it is unrealistic to have a complete set of query rules for design models. We've preliminarily implemented the approach of this section to a D-RisQ robot performing underwater maintenance tasks. An example metamodel of a hazard log spreadsheet for this robot with its query rules are provided online[2].

B. Claim verification by FM

After generating AC structure from system data, this section discusses the automation of claim FM verification within RoboTool. The automation covers not only formal assertions generation for different FM methods, but evidence model generation from FM verification results, as shown in Fig. 3.

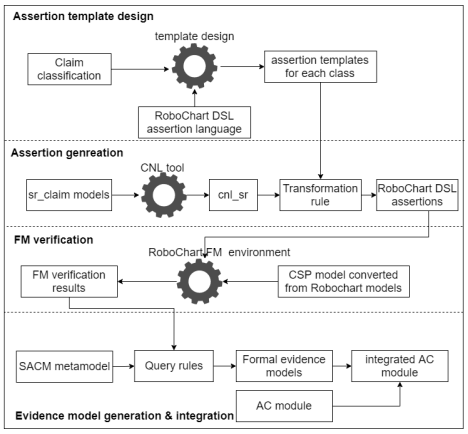

Fig. 3: FM Verification of AC claims within RoboTool

To implement FM verification, taking refinement checking as an example, we need a formal specification, a formal implementation model, and the assertion itself. RoboChart has a formal semantics in the process algebra CSP [18] , and RoboTool can generate their CSP formal models automatically for verification. RoboChart is also supported by an assertion Domain Specific Language (DSL) developed atop Machine-Readable CSP (CSPM). The sophisticated assertions can be translated into CSPM assertions to be run in FDR model checker. With these advantages, we design the approach as four phases. We first classify AC claims according to their CSPM specification patterns, then design three types of templates for each class including CSPM specification model template, CSPM implementation model template, and RoboChart DSL assertion template. We identify the claims to be formally verified in AC models, and rewrite the requirements (e.g., safety requirement, reliability requirement) which those claims are derived from within a CNL tool, and further export them in a structured format. This step will be a manual process.

[2]https://github.com/laila-fangyan/SACM-AC-generation-use-case.git

A software requirement management tool Kapture[3] which is based on CNL is exploited in the approach implementation, and the requirements are exported in XML. The assertion generation starts from extracting information from exported XML requirements, then with the transformation rules designed for the corresponding type of assertion template, assertions are generated. The FM assertion checking within RoboTool will be triggered automatically by an Eclipse plug-in. The AC evidence models then can be derived by querying verification results. This evidence model needs to be integrated to the AC module generated in Section III-A. Further, to ensure the quality of AC modules, the syntactic checking will be performed on AC models using Epsilon Validation Language (EVL). In this PhD, the approach will be implemented on the RoboTool supported FDR model checker and a probabilistic model checker PRISM, and the theorem prover Isabelle/HOL.

C. Progress and plan

At this stage, the AC generation approach has been implemented. The AC formal verification approach has been applied on FDR model checker within RoboTool. Claim classification is carried out, the templates are partially developed. For the next stage, formal verification approach needs to be applied to PRISM for probabilistic properties and to Isabelle/HOL theorem prover. More templates need to be developed, so as the system model query rules. The whole approach will be evaluated with D-RisQ robot use case and be reviewed by engineering practitioners to assess the effectiveness of the proposed approach in practice and against the current available methods discussed in §II.

IV. Threat to Validity

AC models can be generated directly by querying design models, but not all claims are suitable to this query method, i.e. the query rules cannot cover all claims. For the claims not covered, the manual process might be involved. The query rules for the system design models are the basis for AC generation. However, it's not realistic to provide a complete set of rules. New claims may require new rules, but the approach is still valid. Our approach of AC generation can be implemented to the modelling languages backed by EMF metamodels; it can also be applicable to languages backed by other metamodels. For those other languages, we first transform the models to EMF models, then can follow the same process hereafter. Formal verification automation is proposed and implemented for RoboTool, and is not fully generalizable because CSP assertions are of a specific form. In future work, we'd like to address its generalization to other platforms.

V. Conclusion

It's at a medium stage of my PhD, and the paper covers the technical problem and the proposed approach. This PhD is to design a technical solution for automating AC process by MBE to improve efficiency and reduce errors. The AC models are directly derived from system data based on the traceability built in the model query rules. This traceability triggers the automatic update of AC models when system data changes. The formal assertion generation is addressed to automate the formal verification process and avoid the need of FM expertise. The approach extends the existing work, and will contribute a whole AC automation process, a series of query rules and model transformation rules for different process phases. The tentative date of the thesis defense is in December 2022.

[3]https://www.drisq.com/product-kapture

Acknowledgment

I would like to thank the supervisory team for this PhD: Dr. Simon Foster and Dr. Ibrahim Habli from Department of Computer Science at University of York.

References

[1] Assurance Case Working Group, "GSN Community Standard. Version 2," 2018.

[2] ISO, "ISO 26262 Road vehicles–Functional Safety, Version 1," 2011.

[3] F. Yan, S. Foster, and I. Habli, "Safety case generation by model-based engineering: State of the art and a proposal," in *The Eleventh International Conference on Performance, Safety and Robustness in Complex Systems and Applications, proceedings*. International Academy, Research, and Industry Association, 2021.

[4] A. Miyazawa, P. Ribeiro, W. Li, A. Cavalcanti, J. Timmis, and J. Woodcock, "Robochart: modelling and verification of the functional behaviour of robotic applications," *Software & Systems Modeling*, vol. 18, no. 5, pp. 3097–3149, 2019.

[5] B. Gallina and M. Nyberg, "Pioneering the creation of iso 26262-compliant oslc-based safety cases," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2017, pp. 325–330.

[6] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen, "Resolute: an assurance case language for architecture models," in *ACM SIGAda Ada Letters*, vol. 34, no. 3. ACM, 2014, pp. 19–28.

[7] D. S. Kolovos, R. F. Paige, and F. A. Polack, "The epsilon transformation language," in *International Conference on Theory and Practice of Model Transformations*. Springer, 2008, pp. 46–60.

[8] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly, "Weaving an Assurance Case from Design: A Model-Based Approach," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. IEEE, 2015, pp. 110–117.

[9] M. D. Del Fabro, J. Bézivin, and P. Valduriez, "Weaving models with the eclipse amw plugin," in *Eclipse Modeling Symposium, Eclipse Summit Europe*, vol. 2006, 2006, pp. 37–44.

[10] I. Šljivo, G. J. Uriagereka, S. Puri, and B. Gallina, "Guiding assurance of architectural design patterns for critical applications," *Journal of Systems Architecture*, vol. 110, p. 101765, 2020.

[11] "Open Services for Lifecycle Collaboration." http://open-services.net/, online; accessed 15th August, 2021.

[12] E. Denney and G. Pai, "Tool support for assurance case development," *Automated Software Engineering*, vol. 25, no. 3, pp. 435–499, 2018.

[13] Z. Diskin, T. Maibaum, A. Wassyng, S. Wynn-Williams, and M. Lawford, "Assurance via model transformations and their hierarchical refinement," in *Proc. the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018, pp. 426–436.

[14] M. Gleirscher, S. Foster, and Y. Nemouchi, "Evolution of Formal Model-Based Assurance Cases for Autonomous Robots," *Lecture Notes in Computer Science*, vol. 11724 LNCS, pp. 87–104, 2019.

[15] C. Cârlan, D. PetriŞor, B. Gallina, and H. Schoenhaar, "Checkable safety cases: Enabling automated consistency checks between safety work products," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2020, pp. 295–302.

[16] Object Management Group (OMG), "Structured Assurance Case Metamodel (SACM), Version 2.1 beta," 2020.

[17] R. Wei, T. P. Kelly, X. Dai, S. Zhao, and R. Hawkins, "Model based system assurance using the structured assurance case metamodel," *Journal of Systems and Software*, vol. 154, pp. 211–233, aug 2019.

[18] A. W. Roscoe, *Understanding concurrent systems*. Springer Science & Business Media, 2010.