Hierarchical Assurance Patterns for Cyber-Resilient Systems Engineering

Isaac Amundson*, Darren Cofer*, David Hardin* and John Hatcliff†
*Applied Research and Technology, Collins Aerospace, USA
{isaac.amundson, darren.cofer, david.hardin}@collins.com
†Kansas State University, USA
hatcliff@ksu.edu

***Abstract*—On the DARPA Cyber Assured Systems Engineering (CASE) program, our team has developed BriefCASE, an open-source model-based engineering environment for cyber-resilient system design. BriefCASE is comprised of tools that emit evidence of correctness, which is maintained by the framework and can be used to substantiate assurance claims. In this paper, we describe hierarchical cyber-resiliency assurance patterns, which BriefCASE instantiates with the system under development. Evidence collected by the framework is automatically evaluated in the resulting assurance case to determine whether cyber-resiliency goals have been acceptably satisfied.**

I. Introduction

Developers of safety-critical embedded systems have become increasingly aware that their products are vulnerable and subject to cyber attacks similar to those common in large enterprise networks. In response, efforts are underway to incorporate *cyber-resiliency* into system design. Cyber-resiliency means that the system is tolerant to cyber attacks just as safety-critical systems are tolerant to random faults: they recover and continue to execute their mission function, or safely shut down, as requirements dictate. Unfortunately, systems engineers are currently given few development tools to help answer even basic questions about potential vulnerabilities and mitigations, and instead rely on process-oriented checklists and guidelines.

The DARPA Cyber Assured Systems Engineering (CASE) program was launched to research new methods and tools for design, analysis, and verification that enable systems engineers to *design-in* cyber-resiliency for complex cyber-physical systems. Our team developed BriefCASE, a framework for designing and assuring cyber-resilient embedded systems according to the CASE workflow depicted in Fig. 1. BriefCASE provides a development environment for modeling system architectures in AADL [1], analyzing the models for cyber-vulnerabilities, mitigating those vulnerabilities by applying automated model transformations, formally verifying security properties in the model, generating high-assurance component code from model specifications, building the system to a secure kernel target, and finally, generating a system cyber-resiliency assurance case.

For the development of high-assurance systems, multiple stakeholders must be convinced of the system's dependability prior to deployment. First and foremost, the developer must

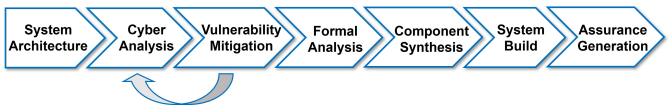

Fig. 1. Cyber Assured Systems Engineering workflow.

be confident that the system operates correctly with respect to the intent of the specification. Next, the certification or accreditation authority (where applicable) must be convinced. Finally the customer, end user (and often other stakeholders in between) will need *assurance* that the system will behave as intended. Assurance is defined as the "planned and systematic actions necessary to provide adequate confidence and evidence that a product or process satisfies given requirements" [2], and different stakeholders may require different means of assurance.

A structured *assurance argument* is one method for conveying assurance. In order to be effective, the argument should be well-formed, complete, and substantiated with evidence. Arguments can be constructed manually or based on assurance *patterns*, in which generic arguments are defined (ideally arrived at through consensus by a body of experts), then instantiated with a concrete system instance [3]. This is the approach taken by BriefCASE, which includes a collection of *hierarchical* cybersecurity assurance patterns that are instantiated with the system under development and incorporate evidence from artifacts generated by the framework. The assurance patterns are hierarchical in that the argument embodied in one pattern may be used to help substantiate the claim made by another pattern. The patterns are represented as *modules* that can be composed into larger, comprehensive patterns for addressing specific dependability concerns.

In this paper, we present a comprehensive collection of hierarchical CASE assurance patterns covering the generation and ingestion of cyber requirements, requirement satisfaction in the model, and requirement satisfaction in the realization of the model. In addition, we describe the mechanisms by which evidence generated as part of the BriefCASE workflow is incorporated into an instantiated assurance case.

The remainder of this paper is organized as follows. In Section II, we describe related research on cybersecurity assurance patterns and frameworks. In Section III, we present

an overview of the BriefCASE framework and workflow. Section IV describes our assurance patterns with respect to the workflow. Specifically, we focus on arguments for cybersecurity requirement correctness, model correctness, and implementation correctness. We provide concluding remarks and discuss future directions in Section V.

II. Related Work

Patterns for assurance case argumentation have been considered in [4], [5], [3], and [6]. An approach to apply and evolve assurance cases as part of system design is found in [7], which is similar to the process we use in BriefCASE. The high-level structure of our CASE assurance patterns is inspired in part by the D-MILS argument pattern [8], in which system dependability properties are assured via modules arguing component, compositional, and implementation correctness. Similar to BriefCASE, the authors demonstrate how to instantiate the D-MILS pattern from an AADL system model; however, they accomplish this via specification of an additional *weaving model*, which is not necessary in BriefCASE due to the tight coupling between the modeling environment and assurance tool.

The VERDICT [9] framework was also developed on the CASE program and has some similarities with BriefCASE. Although VERDICT does generate and evaluate assurance arguments based on analyses performed as part of the tool workflow, the assurance arguments are only fragments of a comprehensive cybersecurity case, and focus primarily on whether applicable Common Attack Pattern Enumeration and Classification (CAPEC) [10] entries have been addressed in the design. In contrast, our CASE assurance patterns consider vulnerability mitigations in the system design *and* the implementation, and include arguments for several other aspects of cyber-resiliency assurance as well.

The Architecture-driven Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems (AMASS) [11] framework incorporates a workflow similar to BriefCASE, but has a scope that encompasses assurance for all types of dependability properties, and a more ambitious goal of driving down certification costs for high-assurance systems development. AMASS supports tools and processes for system design, verification and validation activities, and assurance generation, among others. It is extensible and has a growing ecosystem supported by a European consortium of researchers and practitioners. In contrast, BriefCASE is targeted at the development of *provably cyber-resilient* embedded systems, and although it does support analysis and assurance of other classes of dependability properties, the assurance patterns presented in this paper strictly focus on providing confidence in the cyber-resiliency of the system under development.

III. Cyber Assured Systems Engineering with BriefCASE

In this section, we provide an overview of the tools that comprise BriefCASE, as well as describe the built-in mechanism for automated assurance generation. BriefCASE is predicated on a model-based systems engineering (MBSE) process in which models are the primary vehicle for communication and understanding among the parties tasked with designing the system. The BriefCASE architecture is shown in Fig. 2. It is implemented as a collection of plugins in the Eclipse-based Open Source AADL Tool Environment (OSATE) [1], the reference AADL modeling tool maintained by the Software Engineering Institute (SEI) at Carnegie Mellon University.

A. BriefCASE Architecture

BriefCASE provides access to two architecture analysis tools, GearCASE [12] and DCRYPPS [13], that analyze AADL models for potential cyber vulnerabilities and generate cyber requirements for mitigation. Systems engineers are presented with a Requirements Management interface for viewing the generated requirements and importing them into the model so they can be addressed. Some requirements can be formalized as assume-guarantee contracts, enabling formal verification. Such a requirement will be imported into the model with an associated formal contract. BriefCASE maintains a log detailing which requirements have been imported and which were omitted, and prompts the engineer to provide rationale for the omitted requirements. The log is used in part to provide assurance evidence that the CASE workflow was properly followed (see Section IV).

To address a new cyber requirement, the architecture will need to be transformed in such a way as to harden the design against the vulnerability. BriefCASE provides an extensible library of model transformations for addressing common cyber vulnerabilities. The transformations are automated by the BriefCASE tool, resulting in a hardened model that is correct-by-construction. For example, the requirement that a component shall only receive well-formed messages can be satisfied by the insertion of a high-assurance filter. A BriefCASE transform wizard helps to configure the filter component properties, including the filter behavioral specification, which is represented as an assume-guarantee contract. BriefCASE then inserts a new filter component into the model, sets the component properties, and establishes the appropriate connections to source and destination components. The filter behavioral contract is also added to the model, enabling formal analysis of the model as well as providing the behavioral specification for a provably correct synthesis of the filter component implementation. The transformation also updates the assurance case with new evidential statements indicating how the associated goal has been satisfied, including the strategy used and context needed for assurance case evaluation.

The Assume Guarantee Reasoning Environment (AGREE) [14], is a compositional, assume-guarantee-style model checker for AADL models. AGREE attempts to prove properties about one layer of the architecture using properties allocated to subcomponents. The composition is performed in terms of assumptions and guarantees that are provided for each component. Once the system architecture

[1] https://www.osate.org

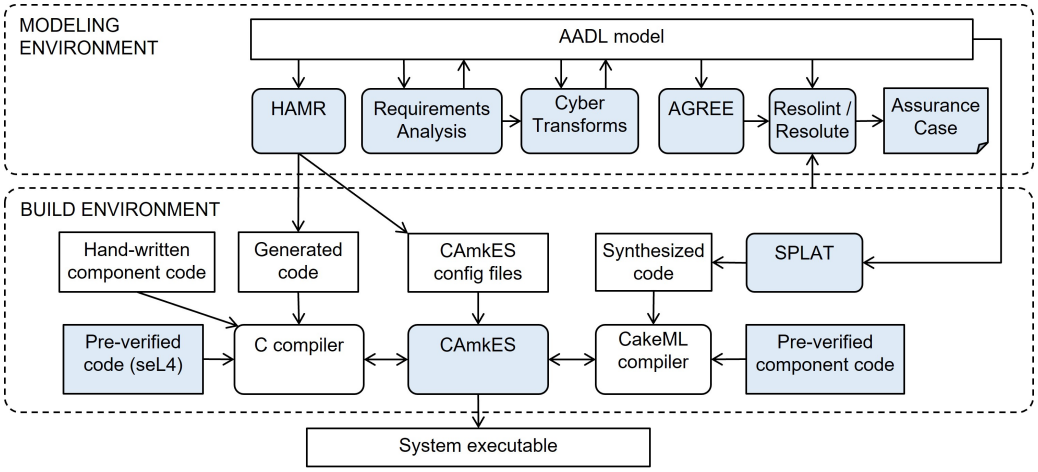

Fig. 2. BriefCASE architecture. Tools and artifacts in the BriefCASE workflow are shown in blue.

has been modeled in AADL and component assume-guarantee contracts have been specified, the AGREE model checker is used to verify the consistency of these contracts. AGREE results are automatically incorporated as evidence into the BriefCASE assurance case.

Each high-assurance component inserted by a BriefCASE transformation must conform to its AGREE contract. This obligation is addressed by formal synthesis, using the Semantic Properties of Language and Automata Theory (SPLAT) tool [15]. SPLAT generates code to implement the AGREE contract and then proves that its implementation exactly preserves the meaning of the contract all the way down to the binary for the target platform. SPLAT uses the HOL4 theorem proving system to implement the synthesis and prove its correctness relative to the contract. The synthesis targets a dialect of Standard ML called CakeML [16] and uses CakeML's fully verified compiler to render the final binary.

BriefCASE employs the High Assurance Modeling and Rapid engineering for embedded systems (HAMR) tool [17], a multi-platform, multi-language AADL code generation framework. Using seL4 [18] as a foundation, HAMR enables AADL to be used as a model-based development and systems engineering framework for seL4-based applications [19]. The seL4 microkernel is a lightweight, fast, and secure operating system kernel. Its implementation is fully formally verified, from high-level security properties down to the binary level.

One of the primary objectives of HAMR is to support system builds that leverage seL4 separation and information flow guarantees to achieve the AADL-specified component isolation and inter-component communication needed for cyber-resiliency. For each AADL thread component, HAMR generates a thread code skeleton and APIs for communicating over the ports declared on the component. For components that are implemented manually, the developer fills out the thread skeleton with application code. HAMR generates component infrastructure and integration code implementing the semantics of AADL-compliant thread scheduling, thread dispatching, and port-based communication.

The seL4 deployment uses the Component Architecture for microkernel-based Embedded Systems (CAmkES) development framework to configure the microkernel. The HAMR-generated CAmkES directly encodes the AADL model's component and communication topology and includes the AADL run-time infrastructure with its thread scheduling. HAMR leverages the existing seL4 domain scheduler to enforce time partitioning and provide static cyclic scheduling. As part of its code generation process, HAMR produces flow graphs reflecting the inter-component information flow at both the AADL architecture level and the CAmkES level for the seL4 deployment. Visual representations are provided for manual inspection, and SMT-based representations are generated for formal reasoning. The SMT-based representations are used to prove that 1) all AADL modeled flows are in the CAmkES configuration, and 2) no extraneous flows have been added.

B. Assurance Case Generation in BriefCASE

Each of the BriefCASE tools contribute to some aspect of high-assurance system development, and each emit evidence of correctness that can be used to substantiate cyber-resiliency assurance goals. Resolute [20] is used to evaluate this evidence and incorporate it into a system cyber-resiliency assurance case. Resolute is a language and tool for embedding an assurance argument in an AADL system architecture model and evaluating the validity of the associated evidence. Because high-assurance products generally undergo certification at the system level, there is a natural mapping between a system design and the corresponding assurance argument. Resolute takes advantage of this alignment by enabling the specification of the assurance argument directly in an AADL *annex*. The assurance case is then automatically instantiated and evaluated with elements specified in the model.

BriefCASE projects contain a repository for cyber requirements. Imported requirements (e.g., those generated by GearCASE or DCRYPPS) are represented as assurance case goals to be satisfied. For example, a requirement that specifies that a target component shall only receive well-formed

messages is imported as the Resolute `goal` depicted in Fig. 3a. The goal is initially marked `undeveloped` since the requirement has yet to be addressed.

```
1 goal Req_Filter() <=
2   ** "Messages shall be well-formed" **
3   context Generated_By : "GearCASE";
4   context Generated_On : "2022-09-09-180532";
5   context Req_Component : "SW::FlightPlanner";
6   undeveloped
```

(a)

```
1 goal Req_Filter() <=
2   ** "Messages shall be well-formed" **
3   context Generated_By : "GearCASE";
4   context Generated_On : "2022-09-09-180532";
5   context Req_Component : "SW.FlightPlanner";
6   agree_property_checked("SW.FlightPlanner", "Req_Filter") and
7   add_filter("SW.FlightPlanner", "SW.Filter", "SW.c7", RF_Msg)
```

(b)

Fig. 3. (a) Cyber requirement imported as an `undeveloped` Resolute assurance goal. (b) Updated goal with logical rules for determining whether goal is satisfied.

The well-formed message requirement can be mitigated by performing an automated model transformation for inserting a filter. Each transformation has an associated assurance pattern that describes a strategy for determining from the model whether the requirement has been satisfied (see Section IV-B). These evidential statements are automatically added to the goal as the design is updated to address the requirement, as shown in Fig. 3b. For the insertion of a filter, Resolute must now check that AGREE formal analysis passes (line 6) and the filter was added correctly to the model (line 7). The `agree_property_checked()` and `add_filter()` function definitions are included with the built-in BriefCASE assurance pattern library and contain additional statements that instruct Resolute on how to determine the validity of the claims. Subsequent changes to the model that invalidate any of the assurance claims can then be detected and corrected.

Resolute has recently been updated to enable evaluation of artifacts external to the modeling workspace, which facilitates manual specification of additional assurance goals and required evidence. BriefCASE includes an Artifact Management tool for specifying how Resolute should parse documents with specific formats (such as test results, review forms, etc.) to determine whether they support specific assurance claims. For each type of document, users can specify a regular expression that will be matched against the document contents, such that a correct match indicates the validity of the evidence in supporting a specific claim.

Resolute also includes a linter tool for AADL models called Resolint [21]. Resolint provides a language for specifying rules that correspond to modeling guidelines, as well as a checker for evaluating whether a model complies with the rules. Results of the Resolint analysis are displayed to the user, and can be directly incorporated as evidence in a Resolute assurance argument.

IV. BriefCASE Cyber-Resiliency Assurance Patterns

Although verifying functional correctness, safety, and other dependability properties is necessary for a comprehensive system assurance case, the CASE patterns presented in this section only addresses cyber-resiliency. The intention is for the resulting instantiated assurance argument to be integrated into a full system dependability assurance case, when applicable. We present our assurance patterns as GSN Patterns [3], [22] (with slight abuse of notation for brevity).

The high-level CASE argument structure is depicted in Fig. 4, with the top-level goal stating "The system is acceptably cyber-resilient". Two contextual elements are required here: the system under development and the domain-specific guidance that defines "acceptable cyber-resilience" for the current development effort. This goal is then substantiated by arguments that cyber-resiliency requirements have been appropriately identified and then satisfied, both in the system model and the *realization* of the system model as a built, deployable system.

A. Cyber Requirement Correctness and Completeness

The assurance argument for cybersecurity requirement correctness and completeness is shown in Fig. 5. In the figure, it can be seen that in order to support the claim, we must provide evidence that the full set of cyber requirements passed through a review process, were imported into the BriefCASE environment as Resolute goals or omitted with rationale, and that successive analyses on updated versions of the model found no new vulnerabilities. The latter reflects the iterative step in the workflow (depicted by the left-pointing arrow in Fig. 1), in which a modified model must be re-analyzed after applying a mitigation for a previously generated requirement. This is necessary in order to demonstrate that the mitigation of one vulnerability does not inadvertently introduce new vulnerabilities. To argue that the current model was analyzed appropriately, we must be able to demonstrate first that the model is well-formed (i.e., it complies with modeling guidelines), that the analysis was indeed performed on the current version of the model, and that the analysis does not produce any new applicable requirements.

B. Cyber Requirements are Satisfied in the System Model

BriefCASE includes a library of automated model transformations corresponding to common cyber requirement classes. Each transformation modifies the model to harden it against a specific vulnerability, thereby mitigating the associated threat and addressing the driving requirement. In addition, the transformations automatically update the corresponding Resolute goals with logical statements that enable Resolute to evaluate whether the goal is supported by the necessary evidence. Because the transformations modify the architecture model in different ways, assuring that a specific requirement is satisfied

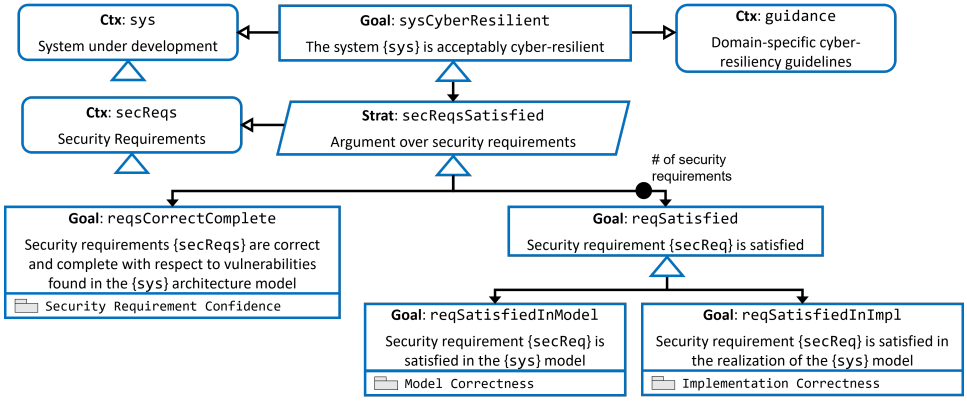

Fig. 4. Top-level assurance pattern structure.

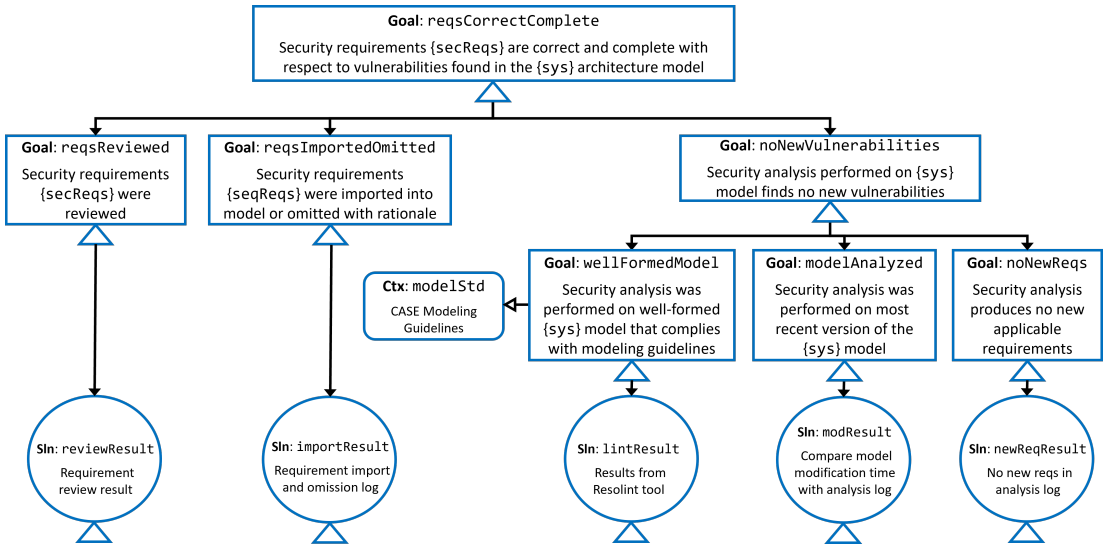

Fig. 5. Assurance pattern for security requirement correctness and completeness.

in the model will be argued according to a transformation-specific pattern.

For example, the well-formed message requirement (introduced in Section III) can be addressed by inserting a filter on the communication channel upstream of the target component. The corresponding assurance pattern for this mitigation is shown in Fig. 6. Here, we argue that the well-formed message requirement has been satisfied in the model by showing that a filter component was inserted and formally verified on the current version of the model. For evidence that the filter was properly added, we rely on Resolute's model traversal functions to verify that the filter component was indeed added to the model upstream of the target component and that there are no communication channels that can bypass the filter. For formal verification, we first desire evidence that the requirement is stated in terms of the AADL component interfaces and publicly disclosed state. Although this is not strictly necessary in the general case, it is included in this pattern for confidence that the CASE workflow was followed correctly. For a formally specified functional requirement, this may be substantiated with evidence that the formal specification has been validated to align with the requirement's intent (e.g., failure and success cases for the formal specification are provided) and the requirement is correctly specified in the formal language (e.g., by manual review).

Assurance patterns corresponding to all the BriefCASE model transformations have been defined and are packaged with the framework.

C. Cyber Requirements are Satisfied in the Realization of the System Model

In the CASE workflow, a software component implementation could have various origins. It could be legacy, third-party, or manually implemented code. It could also be generated from a behavioral model (i.e., Simulink) or be synthesized directly from the component's contract. In BriefCASE, the latter is performed by the SPLAT tool.

Application infrastructure code and the operating system itself must also be implemented and integrated into a deployable system. The HAMR tool generates the infrastructure code,

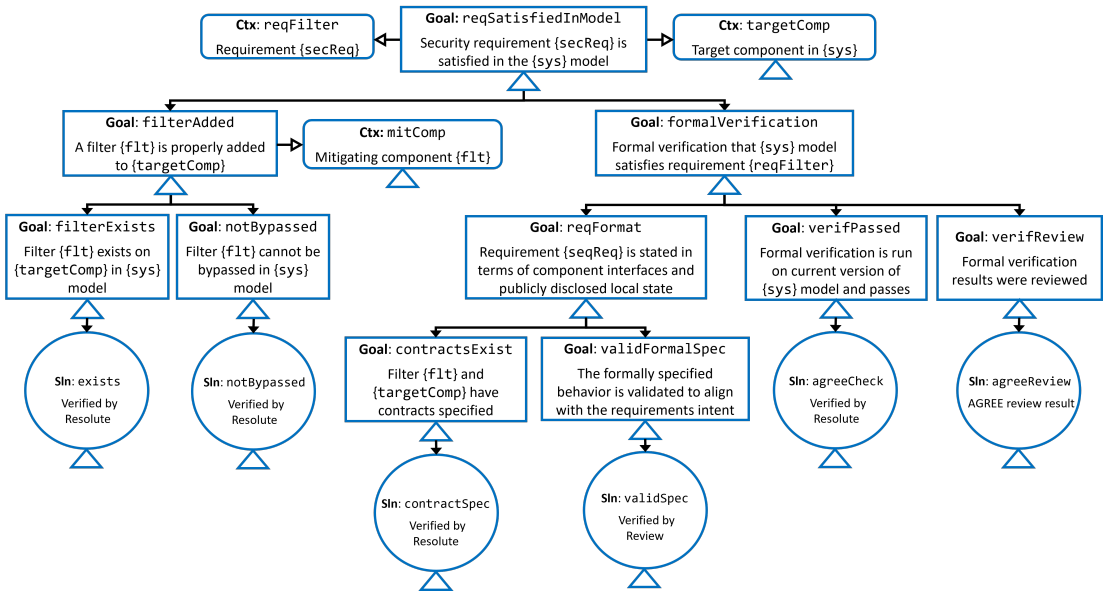

Fig. 6. Assurance pattern for proper filter insertion in the architecture model.

along with correspondence proofs that the inter-component connections specified in the model are maintained in the implementation and that no new connections have been created. For high-assurance systems, this is made possible in part by building to a target platform running the formally verified seL4 microkernel, which provides time and space partitioning guarantees.

To help ensure that system executables conform to AADL model semantics and that semantics are consistent across different AADL-aligned code generation frameworks, AADL defines principles for structuring application code and specifies key semantic steps in the form of Run-Time Services (RTS). AADL RTS are library functions, some of which are called by AADL infrastructure code while others may be called by application code (e.g., to access values on component ports). Working off of an AADL instance model generated by OSATE, HAMR generates a CAmkES specification of the deployment topology and other kernel configuration information. For each AADL thread, HAMR generates infrastructure code that implements the AADL thread dispatch semantics. This includes (a) infrastructure code for linking entry point application code to the underlying seL4 scheduling framework, for implementing the storage associated with ports, and for realizing the buffering and notification semantics associated with event and event data ports; and (b) developer-facing code including thread code skeletons for which the developer will write application code, and port APIs that the application code uses to send and receive messages over ports.

The structure of the `Implementation Correctness` module of the assurance pattern (shown in Fig. 7) therefore necessarily focuses on evidence of correctness in terms of behaviors observed at component interface deployment observation points associated with the BriefCASE workflow. To support the claim that the deployed software component satisfies the cyber requirement, we must demonstrate that the component application code conforms to both its declared interface and requirements (goal `interfaceConformance`), that the component's AADL runtime infrastructure code satisfies AADL port and threading semantics (goal `aadlSemantics`), and that the component's platform deployment context achieves its required assurance properties (goal `platformDeployment`).

Goal `interfaceConformance` is substantiated by the argument in Fig. 8. Here it is critical that we show the application code interfaces correctly with the infrastructure code. This includes evidence that application code only communicates through correct port APIs, runs to completion and produces an output (or drops the input where appropriate) upon being dispatched, and satisfies information flow and worst-case execution time specifications. Evidence supporting most of these goals will typically be in the form of manual inspection and review; however, some evidence such as analysis and verification results can be automatically evaluated by the framework.

For goal `aadlSemantics` (expanded argument not pictured), to demonstrate that the component's AADL runtime infrastructure code satisfies AADL port and threading semantics, we must show that for each component port declared in the AADL model, HAMR correctly generates (a) an API for application code (aligned with the AADL standard) to use when interacting with that port, and (b) an implementation of the port API (aligned with the AADL standard's description of port semantics) that communicates values between the application code APIs and the boundary of the platform deployment of the component. Evidence to substantiate these claims comes from manual inspection of model-to-code traceability information as well as a justification of alignment with the AADL standard.

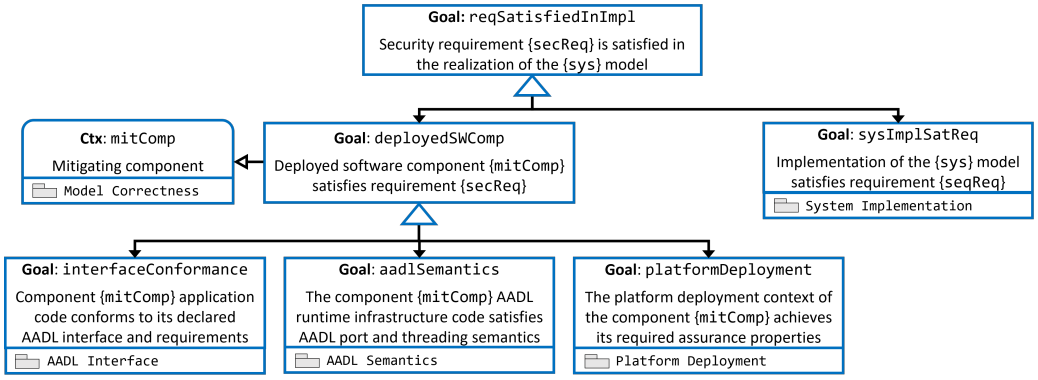

Fig. 7. Assurance pattern for arguing a requirement is satisfied in the realization of the model.

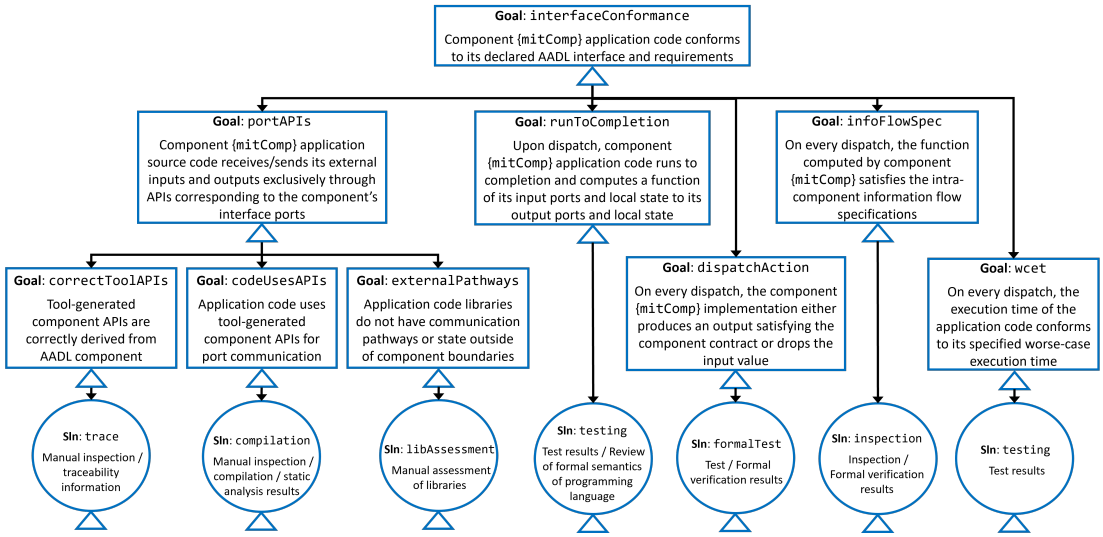

Fig. 8. Assurance pattern for arguing component code conforms to the specified interface and requirements.

Finally, it must be shown that the component's platform deployment context achieves its required assurance properties (goal `platformDeployment`). This is supported by the argument in Fig. 9, in which we must assure the realization of component interfaces and data encodings on the seL4 platform. This includes appropriate separation of the deployment interface into application and infrastructure interfaces and the correct composition of the application and the AADL runtime onto the platform. Some supporting evidence such as HAMR and seL4 proof artifacts can be evaluated automatically by the framework, but otherwise manual artifact inspection is required.

In addition to demonstrating that deployed software components satisfy their cyber requirements, we must also argue that the deployed system implementation preserves them. We do not further expand the `System Implementation` argument module here since the evaluation criteria are typical of established methods in practice today.

V. Conclusion

We have presented a collection of hierarchical cyber-resiliency assurance patterns, which are bundled with our BriefCASE framework and instantiated with a specific system under development. Automated instantiation and evaluation of these patterns provides us with confidence that we have adequately analyzed the system for cyber vulnerabilities and addressed the corresponding cyber requirements in the system design *and* implementation. Although these patterns mainly correspond to the automated BriefCASE features that support the CASE workflow outlined in Fig. 1, BriefCASE is not required to use them; with minor adaptation they can be applied to any tool chain that supports a similar workflow.

We have demonstrated the utility of our tools and methods on several real-world use cases that were subjected to red-team adversarial evaluation[2]. Nonetheless, it will not often be the case that an entire high-assurance system can be developed in this fashion. Not all cyber vulnerabilities can

[2]On the DARPA CASE program, BriefCASE was applied to a section of CH-47 mission control software, as well as an AFRL UxAS application.

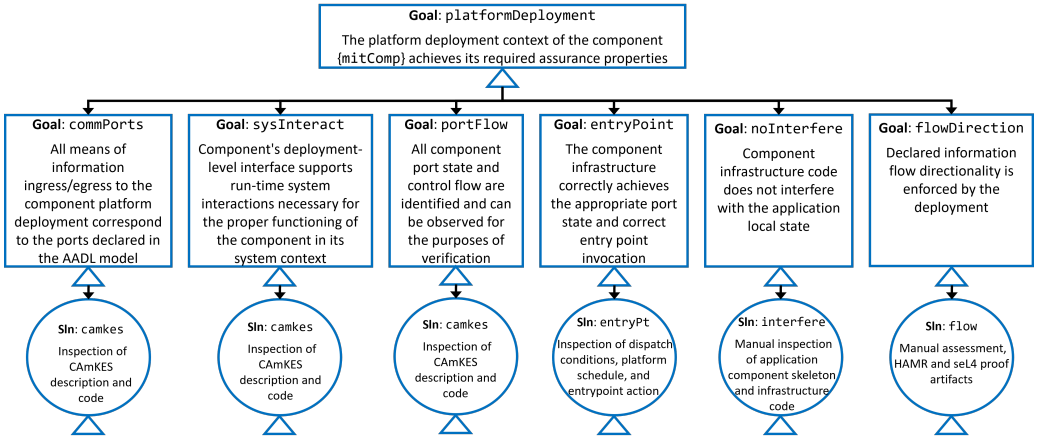

Fig. 9. Assurance pattern for arguing platform deployment context achieves the required assurance properties.

be mitigated by an automated model transformation. Not all component implementations can be synthesized in a provably correct manner. And not all evidential development artifacts can be automatically evaluated. But, for many systems, we are confident that the technologies described in this paper can be usefully employed to improve cyber resilience, even when some elements of the system (e.g., legacy components) resist rigorous analysis and/or automated synthesis.

Although the assurance patterns described herein provide confidence that (a) the CASE workflow was properly followed for a specific system development configuration and (b) the resulting deployable system is acceptably cyber-resilient, additional patterns are still necessary to support typical development efforts we see in practice today. Our cyber-resiliency patterns are structured hierarchically, enabling straightforward insertion of additional pattern fragments corresponding to new cyber vulnerability mitigations, processes, and workflows. We anticipate working towards supporting these patterns in future research projects, with contributions encouraged from the wider security assurance community.

VI. Acknowledgment

This work was funded by DARPA contract HR00111890001. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*, 1st ed. Addison-Wesley Professional, 2012.

[2] RTCA, *Software considerations in airborne systems and equipment certification*, 2011.

[3] T. Kelly and J. McDermid, "Safety case construction and reuse using patterns," in *Proceedings of the 1997 International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, 1997.

[4] E. Denney and G. Pai, "A formal basis for safety case patterns," in *Proceedings of the 2013 International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, September 2013.

[5] R. Hawkins, K. Clegg, R. Alexander, and T. Kelly, "Using a software safety argument pattern catalogue: Two case studies," in *Proceedings of the 2011 International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, September 2011.

[6] L. Sun, O. Lisagor, and T. Kelly, "Justifying the validity of safety assessment models with safety case patterns," in *Proceedings of the 6th IET System Safety Conference*, September 2011.

[7] P. Graydon, J. Knight, and E. Strunk, "Assurance based development of critical systems," in *2007 International Symposium on Dependable Systems and Networks (DSN)*, June 2007.

[8] R. Hawkins, T. Kelly, and I. Habli, "Developing assurance cases for D-MILS systems," in *International Workshop on MILS: Architecture and Assurance for Secure Systems*, 2015.

[9] B. Meng, D. Larraz, K. Siu, A. Moitra, J. Interrante, W. Smith, S. Paul, D. Prince, H. Herencia-Zapana, M. F. Arif, M. Yahyazadeh, V. Tekken Valapil, M. Durling, C. Tinelli, and O. Chowdhury, "Verdict: A language and framework for engineering cyber resilient and safe system," *Systems*, vol. 9, no. 1, 2021.

[10] MITRE. Common attack pattern enumerations and classifications. [Online]. Available: https://capec.mitre.org/

[11] J. L. de la Vara, A. Ruiz, and G. Blondelle, "Assurance and certification of cyber–physical systems: The AMASS open source ecosystem," *Journal of Systems and Software*, vol. 171, 2021.

[12] T. Patten, D. Mitchell, and C. Call, "Cyber attack grammars for risk-cost analysis," in *Proceedings of the 15th International Conference on Cyber Warfare and Security*, 2020.

[13] R. Laddaga, P. Robertson, H. E. Shrobe, D. Cerys, P. Manghwani, and P. Meijer, "Deriving cyber-security requirements for cyber physical systems," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1901.01867

[14] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *NASA Formal Methods*, A. E. Goodloe and S. Person, Eds. Springer, 2012, pp. 126–140.

[15] E. Mercer, K. Slind, I. Amundson, D. Cofer, J. Babar, and D. Hardin, "Synthesizing verified components for cyber assured systems engineering," in *24th International Conference on Model-Driven Engineering Languages and Systems (MODELS)*, October 2021.

[16] R. Kumar, M. O. Myreen, M. Norrish, and S. Owens, "CakeML: a verified implementation of ML," in *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (POPL)*, S. Jagannathan and P. Sewell, Eds. ACM, January 2014, pp. 179–192.

[17] J. Hatcliff, J. Belt, Robby, and T. Carpenter, "HAMR: An AADL multi-platform code generation toolset," in *10th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, ser. LNCS, vol. 13036, 2021, pp. 274–295.

[18] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: formal verification of an OS kernel," in *Proceedings of the 22nd ACM Symposium on Operating Systems*

Principles (SOSP), J. N. Matthews and T. E. Anderson, Eds. ACM, October 2009, pp. 207–220.

[19] J. Belt, J. Hatcliff, Robby, J. Shackleton, J. Carciofini, T. Carpenter, E. Mercer, I. Amundson, J. Babar, D. Cofer, D. Hardin, K. Hoech, K. Slind, I. Kuz, and K. Mcleod, "Model-driven development for the seL4 microkernel using the HAMR framework," *Journal of Systems Architecture*, vol. 134, no. C, February 2023.

[20] A. Gacek, J. Backes, D. D. Cofer, K. Slind, and M. Whalen, "Resolute: an assurance case language for architecture models," in *Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology (HILT)*, M. Feldman and S. T. Taft, Eds. ACM, October 2014, pp. 19–28.

[21] I. Amundson, "Checking compliance of AADL models with modeling guidelines using Resolint," in *SAE Technical Paper 2023-01-0995*, March 2023.

[22] SCSC-141C, *Goal Structuring Notation Community Standard (Version 3)*. The Assurance Case Working Group, 2021.