Presented at SASSUR Sept. 2024; appears in SafeComp Workshops, Springer LNCS vol 14989, pp 173-187

Enabling Theory-based Continuous Assurance: A Coherent Approach with Semantics and Automated Synthesis

Srivatsan Varadarajan[1], Robin Bloomfield[2], John Rushby[3], Gopal Gupta[4], Anitha Murugesan[1], Robert Stroud[2], Kateryna Netkachova[2], Isaac Hong Wong[1], Joaquín Arias[5]

[1] Honeywell Aerospace; srivatsan.varadarajan@honeywell.com ✉
[2] Adelard (NCC); robin.bloomfield@nccgroup.com
[3] SRI International; rushby@csl.sri.com
[4] University of Texas at Dallas; gupta@utdallas.edu
[5] CETINIA, Universidad Rey Juan Carlos; joaquin.arias@urjc.es

Abstract. Assurance cases are gaining traction as a viable means of certification in various safety/security critical industries. In this paper, we introduce a theory-based, property-driven continuous assurance framework that eliminates ad-hoc case constructions through emphasis on validity and soundness of the arguments, confidence of the claims/arguments/evidences and the systematized specification of defeaters. We then describe tools and automation support for semantic analysis of assurance cases and their synthesis. Finally, we showcase a continuous assurance tools infrastructure through an example.

1 Introduction

Assurance cases are gaining popularity in most safety/security critical domains as an alternate means of compliance, as opposed to prescriptive, process-driven approaches such as DO-178C[16] in aerospace. Although these traditional approaches have a good track record and offers clarity to stakeholders on the exact criteria to meet, the quality of compliance and, as a result, the degree of confidence in the system safety/security is hard to judge. Furthermore, in systems whose integration and delivery are *continuous*, i.e., the system is updated frequently for potential quality improvements, incrementally preparing and presenting assurance documentation and evaluating them becomes unwieldy. On the other hand, assurance cases provide explicit certification arguments, that can be fine-tuned to the specific system in consideration, and hence it is more agile than process compliance in adapting to new techniques and applications and thus, well suited to accommodate system design refinements and development innovations.

One challenge, however, is that the strategy of structuring the assurance case is heavily influenced by the assurance authors' expertise and familiarity with the system at hand. Consequently, the evaluation of such assurance cases tends to be highly tailored, and context-specific, and hence, potentially time-consuming and biased based on the evaluators expertise. To this end, we present our approach – *Consistent Logical Automated Reasoning for Integrated System Software Assurance (CLARISSA)* [20] – developed for the DARPA ARCOS program [10]. CLARISSA's approach to assurance cases is based on a more rigorous and demanding *"Claims, Argument, Evidence"*

(CAE) [1] framework called *Assurance 2.0* [5]. This approach simplifies the development and assessment of cases because issues that were previously treated in an ad-hoc manner and were subject to contention and misinterpretation, are now made explicit and systematized.

CLARISSA's *continuous assurance* methodology and tools, the focus of this paper, embraces a *theory-based repeatable* approach, characterized by its *property-driven objectivity* and use of *automated reasoning*. *"Theories"* [19] serve as reusable assurance templates, each *defined* with self-contained *justifications* and exhaustively enumerated *defeaters* [4] that invalidates claims/arguments/evidence within it, offers the potential for pre-certification based on criteria for their *correct application*. Theories alleviate cognitive burden and enhance comprehension and efficiency, facilitating the *synthesis of assurance cases*. Within CLARISSA, constructing assurance cases becomes a seamless, error-free process by instantiating theories, integrating arguments, refuting defeaters and/or accepting inconsequential defeaters and providing supporting evidence.

Furthermore, CLARISSA upholds a stringent definition for a *valid* and *sound* assurance case. Leveraging *confirmation measures* for evidence scrutiny and systematic *confidence* propagation for soundness [19], the assurance cases with potential defeaters are also assessed for validity using novel *non-monotonic logic* formulations. CLARISSA tools provide sophisticated automation for semantic validation and assembly of assurance case elements. Semantic analysis proceeds by translating an assurance case, with appropriate semantic specifications, into corresponding logic predicates with diverse checks and defeater analyses [14,15]. Additionally, the CLARISSA tool suite features a *synthesis assistant*, which generates assurance cases by utilizing evidence and theories as ontologies, employing logic-based transformations. We integrate the CLARISSA tools into a *continuous assurance infrastructure* for handling CI/CD pipelines and illustrate their capabilities through an exemplar scenario involving design changes.

2 Methodological Foundations for Continuous Assurance

Assurance cases in CLARISSA follow the Assurance 2.0[5,6] approach, which builds upon the *CAE* [1] methodology, where the main component of an assurance case is a structured argument represented as a tree of claims linked by argument steps, and grounded on evidence. Details of the Assurance 2.0 methodology and the CLARISSA related enhancements are captured in [20]. In this section, we discuss the relevant concepts for a *semantic-driven, theory-based continuous assurance approach*.

2.1 Composing Assurance Cases with Theories and Defeater Patterns

Theories are essentially an *assurance sub-case*[19,6,20], defined as a *reusable template* but with *semantics* and *justifications*, that can be applied to various assurance cases. The motivation is that any assurance case for a complex system can mostly be built by instantiating various theories plus integrating arguments.

Theories are similar in spirit to the *Safety Case Patterns* developed for *Goal Structured Notation (GSN)* [17] in *AdvoCATE* tool [11] whereby *multiple argument repeated structures* are defined and instantiated or the more generic variant *Assurance Case Templates (ACT)* [7]. In contrast, CAE-based theories are constructed within CLARISSA ASCE tool [2] where they are primed for a *logic-driven analysis* based upon elaborate *semantics* with *rigorous assessment criteria* (see Sections 2.2. 3.1 respectively) to ensure their *correct instantiations*.

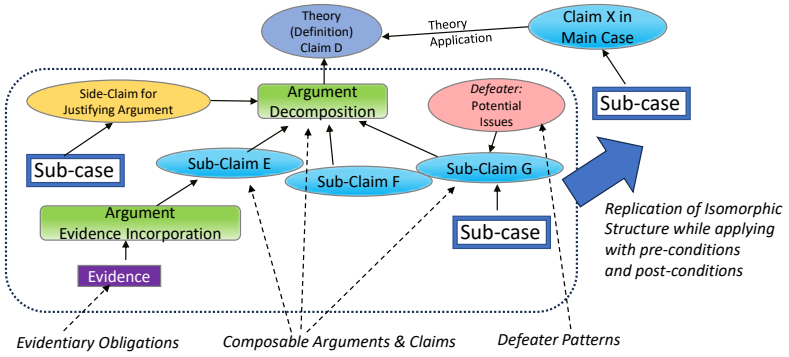

Fig. 1. Generic Theory Definition

Theories then provide a method to control the size of the argument through *compositional reasoning* and aids development of the overall assurance case in a *modular* fashion. This approach is akin to function definitions and function calls in computer programming. Self-contained models and theories, with objective analysis that are better developed, analyzed, justified, and evaluated (i.e., *pre-certified*) separately without cluttering the main assurance case, can be separately managed by experts and presented and assessed by the scientific and engineering methods traditional to their fields.

The generalized structure of a theory is shown in Figure 1. The theory *definition* is described under *Claim D* while the theory *application* is instantiated under *Claim X* in the main assurance case. The idea is that the definition structure under Claim D on the left of the figure is *isomorphically replicated* when theory is applied to Claim X on the right of the figure. In essence, the theory name, once defined, can be utilized in the main assurance case like a short hand representation of the sub-case in it's definition. The theory definition can utilize the full breadth of *CAE* blocks available in the *ASCE* tool. This includes *defeaters* [19] and whose pattern can be introduced at various CAE nodes in the definition of the theory to throw well-known concerns typically accompanying the theory definitions.

Defeaters could be utilized to capture uncertain *doubts*, or to employ *refutational reasoning* in terms of counter-claims/counter-arguments/counter-evidence or to resort to *eliminative argumentation* for a proof-by-contradiction type approach [4,20]. CLARISSA ensures systematized logic-based analysis support for the various defeaters unlike its counterparts in literature. The intention then is that the defeaters enumerated in the theory definition must subsequently during application either be (i) *refuted* during application with evidentiary support for *defeating the defeaters* or (ii) *sustained* with evidence for proving the defeaters and justification for the *acceptability of the residual risk* due to the presence of the defeaters.

Additionally, *pre-conditions*, as either side-claims (justifications) or some subsets of sub-claims, can be specified in the theory definition that then needs to be satisfied *before* application of the theory. The evidence in its definition become *evidentiary obligation* that has to be supplied at the time of application of the theory. Also, *consistent semantic object-property-environment* specification [19,20], of the theory application has to match with theory definition. The latter two items along with sustainment/refuta-

tion of defeaters constitute the *post conditions* that needs to be satisfied after applying the theory. Any successful application of the defined theories within Assurance 2.0 enforced through the CLARISSA tools, requires satisfaction of all pre-conditions and post-conditions for each individual theory.

2.2 Assessing Assurance Case for Soundness and Validity

Assurance 2.0 requires assurance cases to satisfy two logical criteria: *validity* and *soundness*. Validity says the argument *makes logical sense*, assuming its premises (i.e., reasoning and evidential steps) are true, no matter what the claims mean (strictly, it is true in all interpretations). Soundness says that in addition the premises are true for the actual claims. Detailed treatments of these topics can be found in [6,4,20]. Reversing order, we address soundness first and then validity.

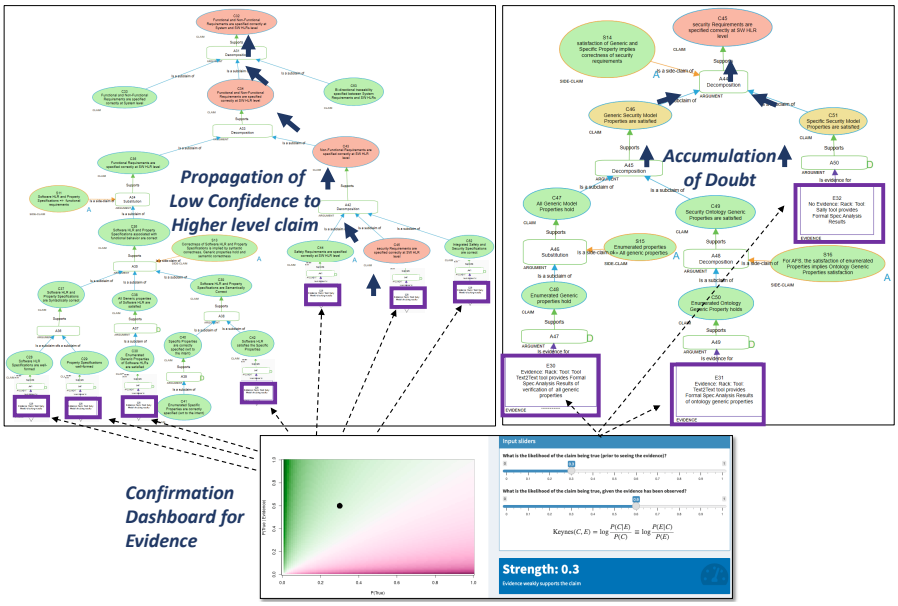

Fig. 2. Illustration of Confirmation Measures and Confidence Propagation

Confidence in Evidential Support for Claims and Confidence Propagation: Given a valid argument, soundness adds the requirement that its actual premises must be true. In an Assurance 2.0 argument, the premises are the evidence incorporation steps and the reasoning steps. As fallible humans, we cannot know that these are true, but we can attempt to establish indefeasible belief that they are so with confidence. Soundness is achieved through the concepts of confirmation measures, confidence values associated with evidence and confidence values propagation through the assurance case – bottom-up from sub-claims and side-claims to parent-claims.

Confirmation Measures are attached to Evidence Incorporation argument blocks to indicate weight of confidence in evidence justifying leaf level claims in a CAE graph. A natural measure of confidence in a claim C given the evidence E is the subjective

posterior probability $P(C \mid E)$, which may be assessed numerically or qualitatively (e.g., *"low"*, *"medium"*, or *"high"*). However, confidence in the claim is not the same as confidence that it is justified by the evidence. For the latter, while we explore many possibilities in [6], we limit to primarily two confirmation measures:

1. $Keynes(C,E)$: how much does the evidence increase my confidence in the claim indicated by $log\frac{P(C|E)}{P(C)}$
2. $Good(C,E)$: how well does this evidence distinguish between the claim and counter-claim indicated by $log\frac{P(E|C)}{P(E|\neg C)}$

Confidence Propagation: the assessment criterion or *"stopping rule"* for an assurance case in Assurance 2.0 is indefeasible soundness (which includes full validity). In CLARISSA, we use subjective probabilistic assessments of *confidence* to record and propagate these judgements. As mentioned above, when assessing soundness of evidential claims we use a confirmation measure $Keynes(C,E)$ or $Good(C,E)$ rather than the posterior probability $P(C \mid E)$ because we wish to evaluate the discriminating power, or "weight" of the evidence, and confirmation measures do this. But once we have assessed soundness, it is reasonable to use the posterior as our measure of probabilistic confidence in the claim C and it is this that will be propagated through the probabilistic valuation of the case. The propagation rule we favor is *sum of doubts* which estimates the *probabilistic doubt* (i.e., 1 - probabilistic confidence) of a parent claim as the sum of probabilistic doubts over its sub-claims and side-claim.

Bottom of Figure 2 shows the use of confirmation measures to the evidence nodes which are the leaf nodes with the assurance cases. Confidence values are propagated automatically upwards from the leaf nodes to sub-claims and sub-claims to parent-claims and all the way up to the top-claim. The nodes are colorized, for ease of readability to green, amber and red signifying *high*, *medium* and *low* confidence nodes respectively. The colorization further ensures problematic parts of the case are not lost/forgotten and direct focus to address them. The top-left of Figure 2 illustrates how lower confidence claims (red nodes) impacts higher confidence claims nodes (green) as these confidence values get propagated upwards. The top-right of Figure 2 shows how the sum of doubts model of confidence calculation impacts claims. Although all nodes beneath the left hand orange node in Figure are green, the small doubts in them accumulate to turn the node orange. The addition of medium confidence in the right hand node causes the top claim shown to be red and this low confidence will be propagated up the tree.

Logical Validity Assessment for Positive Case with Negative Defeaters: The focus of this section is the *validity* criteria while the *soundness* is addressed in the previous section. Because the structure of an Assurance 2.0 argument is so restricted (its logical interpretation is simply propositional calculus over definite clauses), validity simply reduces to a structural check: viewed as a graph, the argument must be connected, with a single top claim, and its leaves must be evidence incorporation nodes or assumptions. Sub-cases may either be expanded in place and assessed as part of the main argument (i.e., as *"macros"*) or be separately assessed and represented by their own top claim (i.e., they function as lemmas). CLARISSA/ASCE tool [19] eliminates many common errors and ensures that the argument is logically valid *"by construction"* [4]. We saw in Section 2.1, that investigation of a refuted defeater could introduce a second-level

defeater and, investigation and resolutions of defeaters may in-turn lead to defeaters at multiple levels. Validity checking becomes more complicated in the presence of defeaters, at multiple levels, and we address them in this section.

- *Assumptions*: we simply state the *claim* as a fact, provided it is not defeated.
  claim :– *–defeater*
  It is worth examining the cases here. If the *defeater* is *false* or absent, the *claim* is *true*.
  If the defeater is *true*, then *–defeater* is *false*, but this does not make *claim false*
  (the body affects the head only when it is *true*).
  Finally, if the *defeater* is unproved, then so is the *claim*.
- *Unsupported claims* (i.e., claims with no sub-case): we say nothing about the claim; s(CASP) will treat it as *unproved*.
- *External sub-cases*: the *parent-claim* inherits whatever the subcase delivers for its *top-claim*.
- *Evidence Incorporation*:
  measured-parent-claim :– *evidence-present*, *–defeater*
  Here, the *evidence-present* flag is set *true* (i.e., stated as a fact) when the evidence is present and is not mentioned otherwise.
- *General Argument Blocks* (concretion, substitution, conjunctive-decomposition, and calculation):
  parent-claim :– *sub-claims*, *side-claims*, *–defeater*
  Here, *sub-claims* are a list of one or more claims and *side-claims* is also a list of zero or more.
- *Special case for substitution blocks* delivering *evidentially useful parent-claims*:
  Justification should reference confirmation measures, then then *confirmation* can be set *true*, *unproved*, *false* (i.e., respectively stated as a fact, not mentioned, stated as a negated fact) according to whether the confirmation measure is strongly positive, neutral, or strongly negative.
  if confirmation measure is *strongly positive*:
  useful-parent-claim :– *side-claims*, *measured-claim*, *–defeater*
  if confirmation measure is *strongly negative*:
  –useful-parent-claim :– *side-claims*, *measured-claim*, *–defeater*
  Note that this causes the evidentially useful *parent-claim* to be set *false* when the evidence is strongly negative.
- *Ordinary (non-exact) defeaters (includes doubts)*:
  if the defeater is *unsupported* (i.e., is a doubt), then it is not mentioned and will default to unproved.
  Otherwise, it must be the parent node of some argument node and will be set according to its type using the rules above.
- *Exact defeaters*:
  parent-claim :– *–defeater-claim*
  –parent-claim :– *defeater-claim*
  Note that we need two rules here: one to propagate *true* and the other to propagate *false*

Table 1. Logic Program Interpretation of Validity Propagation Rules

Logic Program Interpretation of Validity Propagation Rules: We extend the Assurance 2.0 methodology to support propagation rules for validity assessments of assurance cases with multiple levels of defeaters using *three-valued non-monotonic logic*: *true*, *false*, and *unsupported*. We then enforce the propagation rules in [4] to ensure validity. Argument blocks are interpreted as a material implication on *logic*, which can be rewritten as: $\textit{parent-claim} \vee \neg \textit{side-claim} \vee \neg \textit{sub-claim}_1 \vee \cdots \vee \neg \textit{sub-claim}_n$. In *logic program* such s(CASP) [3] this can be written as: $\textit{parent-claim} \ :- \ \textit{side-claim}, \textit{sub-claim}_1, \ldots, \textit{sub-claim}_n$. Table 1 shows propagation rules interpretations.

3 Tools Support for Continuous Assurance

CLARISSA Tools Architecture is shown in Figure 3, which consists of: (i) *Assurance and Safety Case Environment (ASCE)* [2] which is the most widely adopted commercial software for the creation and management of safety and security assurance cases, and (ii) a goal-directed top-down *solver for Constraints Answer Set Programs s(CASP)* [3,15] for reasoning about assurance cases using an enhanced *Prolog* engine.

ASCE has full support of Assurance 2.0 framework its enforcement while it also facilitates systematic creation of Assurance 2.0 cases leveraging theories, ensuring the validity and soundness of the logical arguments with justifications while enabling active search for defeater and either sustaining or refuting them. Libraries of theories and defeaters are maintained as active repository of knowledge and known vulnerabilities. ASCE performs *structural analysis* to ensure their *correct and complete* construction while automatically analyzing specific *syntactic* elements of assurance cases including adherence to notations, grammar/spell-checks within natural language descriptions.

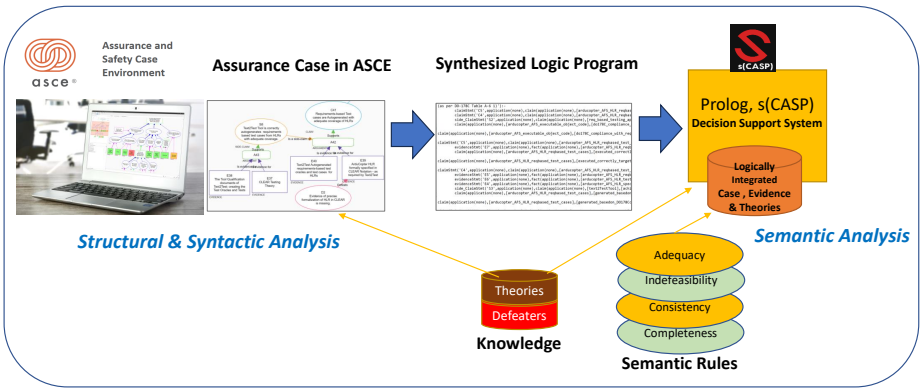

Fig. 3. CLARISSA Tools Architecture

ASCE also *automatically converts the assurance case to an equivalent logic program* to support systematically reasoning with the s(CASP) engine. The s(CASP) engine reasons over the *semantics* or underlying meaning of the claims, arguments, and evidence presented in assurance cases which includes various properties of the assurance case such as *consistency* (i.e. absence of logical contradictions), *indefeasibility* (i.e. absence of defeaters) and *completeness* (i.e. state of encompassing all the requisite elements), etc.

Related Work: CLARISSA, like it's counterpart AACE tool [9], provides better automation support for synthesizing assurance case using library of patterns/templates/theories with a more precise characterization of semantics (e.g. distinction between inductive vs deductive refinements of claims) than other tools in this area (e.g. AdvoCATE [12]). But in contrast to AACE or AdvoCATE, CLARISSA provides better support for defeaters (e.g. doubts, refutational reasoning, eliminative argumentation) and a more advanced logic support in s(CASP) with complex probabilistic reasoning for defeaters as opposed to a much simpler bayesian reasoning available in these tools.

3.1 Property-driven Semantics with LLM Support and Synthesized Prolog Logic based Analysis

Assurance cases, traditionally rely, primarily on free-form natural language to document claims, arguments and evidence. Hence, despite being well-structured and syntactically correct in the graphical representation of the assurance case, ensuring semantic correctness (the top-level claim logically follows from its sub-claims, arguments, and

evidence), and that there are no logical inconsistencies or fallacies is intellectually demanding for both authors and evaluators, due to the inherent ambiguity, and inconsistency of natural language. Consequently, automating semantic reasoning would greatly reduce human effort and enhance the quality and confidence in these cases.

We take a two-step approach, where we first categorically ground the terms used in the descriptions in a intuitive and *"minimally"* formal way that is easily *extensible*. Then the assurance case is transformed into a logical notation that can be subject to various formal analyses at the back end. The claim/evidence language formalisms, the assurance case transformation to logic programs and subsequent semantic analysis capabilities are detailed in [14,15].

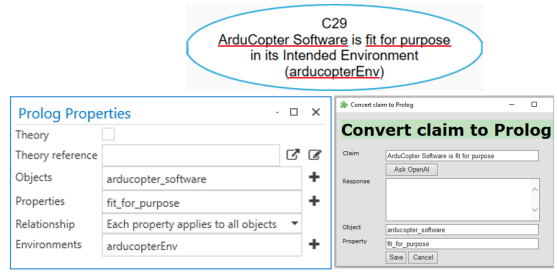

Fig. 4. Objects-Properties-Environments Formalism and LLM Support

Step 1: Objects-Properties-Environment (OPE) as Semantic Specifications

Fundamentally, assurance cases consist of blocks or "nodes" that describe the properties or relationships applicable to objects in a certain environment. Leveraging this general structure of description, we first categorically specify the terms used in the descriptions in terms of *objects* of the system, *properties* they possess, and optional *environments* in which the properties of the given objects are valid. e.g., the claim: *"software is correct w.r.t. requirements"* can be ex-pressed in the object-property-environment formalism as: $object(software)$, $property(correct\_wrt\_requirements)$, $environment(env)$, $claimstmt(software, correct\_wrt\_requirements, env)$.

When multiple objects and properties are associated with a node, it becomes necessary to specify their relationships, such as each property is applicable to all objects or each object or a specific object. This relationship is preserved and meticulously converted into appropriate logic predicates. Currently, our ASCE assurance case authoring tool enables users to explicitly define objects, properties, and environments, along with their relationships to the assurance nodes, in addition to providing natural language descriptions. While the tool currently supports this dual specification, we have provided methods leveraging LLMs, to automatically extract *Objects-Properties-Environment (OPE)* information from the descriptions, thereby streamlining the process. The Prolog Properties dialog box was added to ASCE tool to capture the semantic formalism as shown in Figure 4.

Step 2: Transformation to Prolog Logical Notation

In order to convert the assurance case into a logical notation suitable for the desired semantic analysis, we opt for the domain of Logical Programming (LP). Then, in the second step, the assurance case is transformed into a logic program notation (see bottom of Figure 3), that can be subject to various formal analyses at the back-end using the

s(CASP) system that is based on LP in Prolog, a widely used logical programming language. By utilizing the Objects-Properties-Environment specification provided in each assurance case node, the ASCE tool automatically translates the case into predicates in Prolog. The transformation process follows the guidelines presented in Table 2.

In Table 2, the term 'ClaimPredicate' refers to a Prolog predicate represented as the claim([O], [P], [E]), where [O], [P], and [E] represent comma-separated lists of Objects, Properties, and Environments associated with each assurance node. As we formalize the properties, the object-property relationships specified during the definition of objects and properties are preserved. This preservation ensures precision in the analysis. Consequently, the 'PropertyList' is derived from the ClaimPredicate based n the specified object-property relationship, as enumerated in Figure 4. Furthermore, to maintain the structure of the case during export and ensure traceability, certain metadata (such as node identifiers, descriptions, etc.) is also included within the exported predicates. Description of the automation with various analyses and examples can be found in [14,15].

Node	Prolog Translation

Claim	`claimStmt(Claim_ID, Application, ClaimPredicate, Description):-`
	`  [claimStmt | evidenceStmt | side_ClaimStmt], ...,`
	`  [-Defeater], ClaimPredicate,`
	`  theory(Theory_ID, Application, Claim).`
	`ClaimPredicate :- PropertyList.`
	`theory('Theory_ID', Application, ClaimPredicate):-.`
	`  [claimStmt|evidenceStmt|side_ClaimStmt], ...`
	`  ClaimPredicate.`

Evidence	`evidenceStmt(Evidence_ID, Application, ClaimPredicate, Artefact, URI):-`
	`  [-Defeater], ClaimPredicate.`
	`ClaimPredicate :- PropertyList.`

Side Claim	`side_ClaimStmt(Side_Claim_ID, Application, ClaimPredicate, Justification):-`
	`  [claimStmt | evidenceStmt | side_ClaimStmt], ... ,`
	`  [-Defeater], ClaimPredicate.`
	`ClaimPredicate :- PropertyList.`

Defeater	`defeater(Defeater_ID, Application, defeats(Node_ID), ClaimPredicate, Description):-`
	`  [claimStmt|evidenceStmt|side_ClaimStmt], ... ,`
	`  [-Defeater],`
	`  [theory('Theory_ID', Application, Claim)].`
	`ClaimPredicate :- PropertyList.`
	`PropertyList.`

Table 2. Assurance 2.0 Node Mapping to Prolog

Structural and Syntactic Analysis in ASCE: The ASCE tool is built-in with the capability to perform various structural and syntactic analyses on assurance cases, including:

- *Indefeasibility*: report all unretired/unresolved defeaters in the case.
- *Confidence*: capability to assess quantitative and qualitative confidence (high, low, medium) at both node level and for the overall case.

Semantic Analysis with S(CASP): The following are some of the semantic properties of the assurance case that are analyzed in s(CASP) for a synthesized Prolog from assurance case:

- *Theory Application Correctness*: This property guarantees that the theories are precisely instantiated in the assurance case; in particular, the properties and the instantiations (objects and environment) align with their respective theory definitions.
- *Harmonious Composition of Theories*: when multiple theories are linked and applied within an assurance case, there is a risk of conflicting properties among their definitions or in their concrete applications. We define novel rules that will allow

s(CASP) to check for the presence of conflicts or determine that the theories can harmoniously coexist within that assurance case and can be composed correctly.

3.2 Synthesis Assistant for Generating Assurance Cases

We use a logic based language and associated tools to develop a capability to automatically synthesis CAE based cases, add identified evidence and relevant defeaters. It is based on theories as ontologies and logic-based transformations. The underlying motivations for synthesis automation include: (i) reduce errors in case construction, (ii) increase tempo of case production (and reduce cost), respond to DevSecOps and compile to combat challenges, and (iii) remove unnecessary variability in cases, so easier to review. The synthesis workflow is shown at the top of Figure 5.

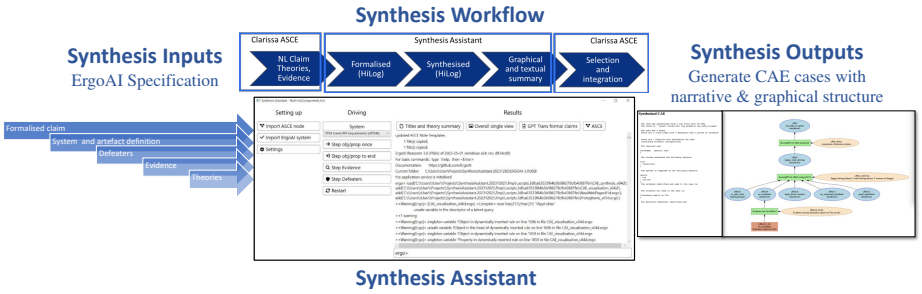

Fig. 5. Illustration of Synthesis Assistant

To generate the CAE structure and assurance case, we need 5 key inputs as shown in left of Figure 5. First, the basis for the synthesis is a top-level claim in a formalized specification. Second, we need an understanding of what the case is about so a description of the system and its artefacts. System definitions represent the structure of the system (subsystems, components, hardware, software etc.) and the associated life-cycle artefacts (e.g. code, requirements, tests etc.) which, might form part of the case. Third, we need the the rules for developing the argument steps. In assurance 2.0 these are based on theories, a form of parameterized template that has defined semantics discussed in Section 2.1. In addition, we want to identify possible defeaters to the case and, if there is evidence available, show how it can be incorporated in the case. Finally, With the five synthesis inputs, shown in Figure 5, we can synthesis candidate CAE structures, generate full cases, providing a graphical CAE and textual narrative summary as synthesis outputs. Design/Implementation details of the synthesis assistant can be found in [20].

3.3 Continuous Assurance for CI/CD Software Designs using ETB

Continuous Integration and Continuous Delivery (CI/CD) is the practice in the software industry of frequently merging a software developers local code repository to a main repository, and then ensuring the merged changes can be reliably released as a product at any time. This often involves automation pipelines that incrementally build, test, and deploy any changes. We envision that assurance case development can also make use of the CI/CD principles to incrementally change, test, and review the assurance case. We consider the added step of evaluating the merits of changes in evidence, tools,

and assurance case arguments as *"Continuous Assurance" (CA)*, leading to a more comprehensive *CI/CD/CA* software pipeline process.

The DesCert team [18] has developed a tool called the Evidential Tool Bus (ETB2), which is a framework that allows CI/CD/CA pipelines (i.e. *workflows*) to be defined [13,8]. ETB2 tracks all input artifacts and artifacts produced by each service. Changes in these artifacts are detected by ETB2, so that downstream services that rely on these artifacts can be identified as in need of updating. Incremental changes in input and tool output can thus be propagated down the workflow to ensure that all the intermediate artifacts and the final assurance case are up to date. The details of how ETB2 works as well as how CLARISSA tools are integrated into it can be found in [18,20]. Figure 6 illustrates this CI/CD/CA process with a workflow specification that includes appropriate tools from Evidence Generation (EG), Evidence Curation (EC), and Assurance Generation (AG), which are the CLARISSA tools from this paper, integrated with ETB2.

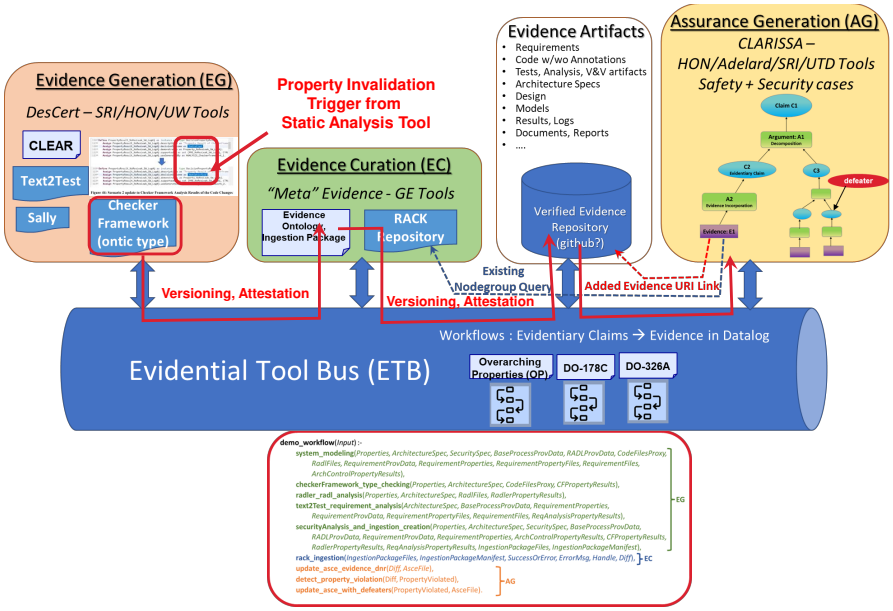

Fig. 6. CLARISSA Tool Integration with ETB2

Bottom of Figure 6 shows an example ETB2 workflow specification. ETB2 triggers based on the rules specified, upon various claims being invalidated due to changes in code/requirement/analysis results etc., the appropriate EG/EC/AG tools. The workflow we defined starts with the results of manual system modeling and requirements definition. These files are the input to the workflow and are used by several evidence generation tools to create evidence and artifacts. These artifacts are then used as input to evidence curation tools, which stores information about the artifacts and generates an output of whether the current set of generated evidence is different from the previously generated set of evidence, and what has changed. This output is then used as input to the CLARISSA tools integrated into ETB2. We have integrated features of the ASCE tool and the Synthesis Assistant (SA) tool into ETB2 as individual services.

We emulate a scenario whereby an update to the source code introduces a dependency on a log4j library that has a known security vulnerability. After the source code is committed, ETB2 will detect a change in the files and identify the requisite services that depend on the change. In this case, it is the Checker Framework service followed by the security analysis service, and so they will both rerun to update their output. Because the source code now introduces a known vulnerability, the result of the analysis will conclude that one of the security properties are violated. Top left of Figure 6 shows the change in Checker Framework analysis results due to the updated source code, which invalidates the log4j no resource leak property. That property invalidation from static analysis tool (Checker Framework tool from EG tools) triggers the workflow as shown in the top left of Figure 6. ETB Workflows subsequently triggers Evidence Curation (EC) tools to store the meta evidence in RACK repository and actual evidence artifacts in github repositories. Finally CLARISSA Synthesis Assistant tool service runs and generates an assurance case fragment from the claim node invalidated by the changed property. This assurance case fragment is a defeater case that is merged with the main assurance case by attaching to that claim (seen as red claim in top right of Figure 6).

4 Conclusion

Assurance 2.0 framework, with concepts such as reusable theories and defeater patterns, aims to improve the science underpinning assurance case construction and increased confidence in their assessment to enhance their adoption as a means of certification. CLARISSA tools built-in automation fulfills this objective by ensuring semantic coherence across assurance cases through rigorous formalism and analysis support. It also provides infrastructure for systematically synthesizing cases that can be integrated into CI/CD pipelines for continuous assurance. Our future work, among others, includes assurance case ontology development, synthesis for candidate design alternatives with design criterion and leveraging Generative AI and Logic for enhancing assurances.

Acknowledgment

CLARISSA is supported by DARPA under contract number FA875020C0512. Distribution Statement “A”: Approved for Public Release, Distribution Unlimited. 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. Adelard: Claims Arguments Evidence, `https://claimsargumentsevidence.org/`
2. Adelard LLP: Assurance and Safety Case Environment, `http://www.adelard.com/asce`
3. Arias, J., Carro, M., Salazar, E., Marple, K., Gupta, G.: Constraint Answer Set Programming without Grounding. TPLP **18**(3-4), 337–354 (2018)
4. Bloomfield, R., Netkatchova, K., Rushby, J.: Defeaters and Eliminative Argumentation in CLARISSA. arXiv preprint (2024), `https://arxiv.org/abs/2405.15800`
5. Bloomfield, R., Rushby, J.: Assurance 2.0: A manifesto. arXiv preprint (2021), `https://arxiv.org/abs/2004.10474v3`
6. Bloomfield, R., Rushby, J.: Assessing Confidence with Assurance 2.0. arXiv preprint (2024), `https://arxiv.org/abs/2205.04522v4`

7. Chowdhury, T., Lin, C., Kim, B., Lawford, M., Shiraishi, S., Wassyng, A.: Principles for systematic development of an assurance case template from ISO 26262. In: Proceedings - 2017 IEEE 28th Intl. Symp. on Software Reliability Engineering Workshops, ISSREW 2017
8. Cruanes, S., Heymans, S., Mason, I., Owre, S., Shankar, N.: The Semantics of Datalog for the Evidential Tool Bus, pp. 256–275. Springer Berlin Heidelberg (2014)
9. Daw, Z., Oh, C., Low, M., Wang, T., Amundson, I., Pinto, A., Chiodo, M., Wang, G., Hasan, S., Melville, R., Nuzzo, P.: AACE: Automated Assurance Case Environment for Aerospace Certification. In: 2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)
10. Defense Advanced Research Projects Agency (DARPA): Automated Rapid Certification Of Software (ARCOS), `https://www.darpa.mil/program/automated-rapid-certification-of-software`
11. Denney, E., Pai, G.: A Formal Basis for Safety Case Patterns. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) Computer Safety, Reliability, and Security. pp. 21–32. Springer Berlin Heidelberg (2013)
12. Denney, E., Pai, G., Pohl, J.: AdvoCATE: An Assurance Case Automation Toolset. In: Ortmeier, F., Daniel, P. (eds.) Computer Safety, Reliability, and Security. pp. 8–21. Springer Berlin Heidelberg (2012)
13. fortiss: Evidential Tool Bus (ETB). `https://www.fortiss.org/en/research/projects/detail/evidential-tool-bus` (2024)
14. Murugesan, A., Wong, I.H., Stroud, R., Arias, J., Salazar, E., Gupta, G., Bloomfield, R., Varadarajan, S., Rushby, J.: Semantic Analysis of Assurance Cases using s(CASP). Goal Directed Execution of Answer Set Programs (GDE) Workshop in ICLP (2023)
15. Murugesan, A., Wong, I.H., Varadarajan, S., Arias, J., Salazar, E., Gupta, G., Stroud, R., Bloomfield, R., Rushby, J.: Automating Semantic Analysis of System Assurance Cases using Goal-directed ASP. Submitted to the Int'l Conf. on Logic Programming (ICLP) (2024)
16. Radio Technical Commission for Aeronautics (RTCA): DO-178C: Software Considerations in Airborne Systems and Equipment Certification.
17. Safety-Critical Systems Club's (SCSC) Assurance Case Working Group (ACWG) - `https://scsc.uk/gc`: Goal Structuring Notation(GSN), `https://scsc.uk/gsn`
18. Shankar, N., Bhatt, D., Ernst, M.D., Kim, M., Varadarajan, S., Millstein, S., Sánchez, H.A., Murugesan, A., Wong, I., Ren, H., Ruess, H., Siu, K., Beyene, T., Varanasi, S.C., Bouchekir, R.: Continuous Safety & Security Evidence Generation, Curation and Assurance Case Construction Using the Evidential Tool Bus. In: To appear in 43rd AIAA/IEEE Digital Avionics Systems Conference (DASC) (October 2024)
19. Varadarajan, S., Bloomfield, R., Rushby, J., Gupta, G., Murugesan, A., Stroud, R., Netkachova, K., Wong, I.: Clarissa: Foundations, tools & automation for assurance cases. In: 42nd AIAA/IEEE Digital Avionics Systems Conference (DASC) (10 2023)
20. Varadarajan, S., Bloomfield, R., Rushby, J., Gupta, G., Murugesan, A., Stroud, R., Netkachova, K., Wong, I.: Consistent Logical Automated Reasoning for Integrated System Software Assurance (CLARISSA), DARPA ARCOS Final Report. To appear shortly. Tech. rep. (June 2024)