Using Process Models in System Assurance

Richard Hawkins, Thomas Richardson, and Tim Kelly

Department of Computer Science, The University of York, York, YO10 5GH, UK

Abstract. When creating an assurance justification for a critical system, the focus is often on demonstrating technical properties of that system. Complete, compelling justifications also require consideration of the processes used to develop the system. Creating such justifications can be an onerous task for systems using complex processes and highly integrated tool chains. In this paper we describe how process models can be used to automatically generate the process justifications required in assurance cases for critical systems. We use an example case study to illustrate an implementation of the approach. We describe the advantages that this approach brings for system assurance and the development of critical systems.

1 Introduction and Motivation

Systems used to perform critical functions require justification that they exhibit the necessary properties (such as for safety or security). The assurance of a system requires the generation of evidence (from the development and analysis of the system) and also a reasoned and compelling justification that explains how the evidence demonstrates the required properties are met. The evidence and justifications are often presented in an assurance case. A compelling justification will always require both a technical risk argument (reasoning about assurance mitigations of the system) and confidence arguments (documenting the reasons for having confidence in the technical argument). Although both technical arguments and arguments of confidence are included in most assurance cases, we find that often the focus is on the technical aspects of assurance and that confidence is often dealt with in very general terms. In [8] we discuss the need for confidence arguments to be specific and explicit within an assurance case. The confidence argument should consider all the assertions made as part of the technical argument. In this paper we focus on one important aspect of this - demonstrating the trustworthiness of the artefacts used as evidence in the technical argument.

As an example, figure 1 shows a small extract from an assurance argument that uses evidence from formal verification to demonstrate than an assurance property of the system is satisfied. Figure 1 is represented using Goal Structuring Notation (GSN). In this paper we assume familiarity with GSN, for details on GSN syntax and semantics we refer readers to [5] and [11].

Figure 1 can be seen to present a technical argument (the left-hand leg), and also a claim that there is sufficient confidence in the verification results that are presented in that technical argument (Goal: formalConf). The level of confidence

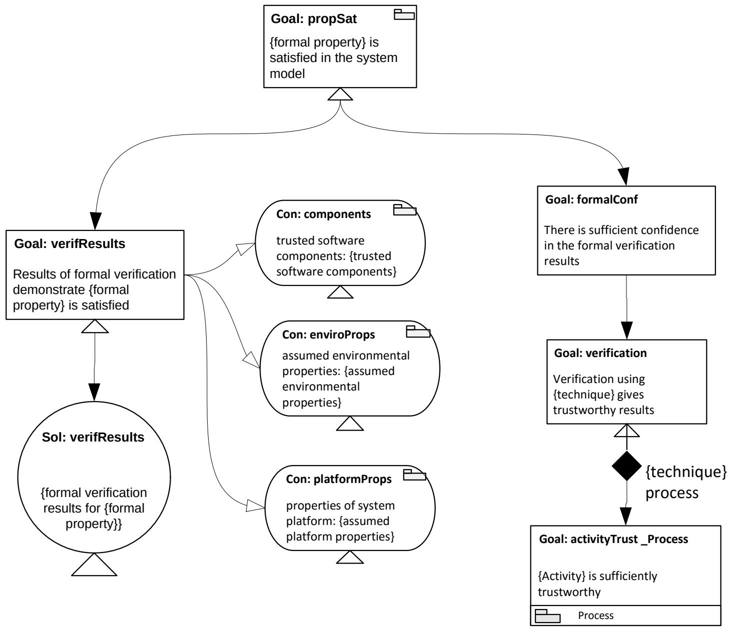
Goal: propSat
{formal property} is satisfied in the system model
Goal: verifResults
Results of formal verification demonstrate {formal property} is satisfied
Con: components
trusted software components: {trusted software components}
Con: enviroProps
assumed environmental properties: {assumed environmental properties}
Con: platformProps
properties of system platform: {assumed platform properties}
Sol: verifResults
{formal verification results for {formal property}}
Goal: formalConf
There is sufficient confidence in the formal verification results
Goal: verification
Verification using {technique} gives trustworthy results
{technique} process
Goal: activityTrust _Process
{Activity} is sufficiently trustworthy
Process

Fig. 1. Example assurance argument pattern

required in the verification results is determined by both the assurance required for the system as a whole, and the role of those verification results in the overall system argument. This issue of establishing confidence in an evidence artefact is a complex one. As discussed in [20], both the appropriateness of the artefact in supporting the argument claim and the trustworthiness of that artefact must be considered. In this paper we focus on the trustworthiness of the artefact. The notion of evidence trustworthiness has been widely discussed, such as in the Structured Assurance Case Metamodel standard (SACM) [16]. Trustworthiness (sometimes also referred to as evidence integrity) relates to the likelihood that the artefact contains errors. It has long been understood that the processes used to generate an artefact are one of the most important factors in determining how trustworthy an artefact is. This is discussed further in work such as [18], and is also seen in standards such as [9] and tool qualification levels in [10]. The basis for such an approach is that a trustworthy artefact is more likely to result form the application of a rigorous, systematic process undertaken by suitable participants using appropriate techniques and incorporating thorough evaluation. This includes consideration of the assessment and qualification of tools used as part of a tool chain. In figure 1 it is seen how the claim 'Goal: formalConf' can be supported by reasoning over the trustworthiness of the verification results (Goal:

verification), and then in turn by arguing over the formal verification process that generated that result (Goal: activityTrust_Process)[1].

Modern critical systems often require the use of complex processes involving the integration of multiple development tools and techniques. Creating a compelling justification for each process adopted can be a huge challenge, and indeed this may be a reason why this is often overlooked in favour of more general demonstrations of process compliance. We believe that it should be possible to make the generation of confidence arguments from processes easier and more systematic. This paper therefore provides the following solution:

"Using process models generated as part of system development, and a set of confidence argument patterns, the required confidence arguments for assurance artefacts can be automatically generated."

Firstly, in Section 2 we discuss the process models, in Section 3 we describe the confidence argument patterns, finally in Section 4 we describe how the process models and argument patterns can be linked together to create the required confidence arguments for the target system. We use an example throughout to illustrate our approach.

2 Process Models

Our approach permits the use of any process model in order to generate the process argument. This provides important flexibility for system developers to use any existing process models and tooling. A defined meta-model must however be provided for all models used (in order to create a weaving model for instantiation - see Section 4) and the process models must be valid instances of the defined meta-model. It should be noted that for most commonly used process modelling approaches such as SPEM [13] meta-models already exist. For the purposes of our example we have chosen to use the process meta-model that is summarised in Figure 2, which is based upon that created as part of the OPENCOSS project[2]. We used the OPENCOSS process meta-model [2] as the basis for this since it has been developed based upon a cross-domain consideration of safety standards and processes and with input from industrial partners from many industries.

Here we provide a summary of the main elements of the meta-model in Figure 2. Processes entail Activities, which may themselves entail other Activities (sub-activities). Any activity may have related Participants (which could be a Person, Tool or Organisation (see Figure 3)). Activities may require and produce Artefacts. Any artefact may be defined as a ManageableAssuraceAsset (defined as part of the evidence meta-model (see Figure 3)) for which evaluations (AssuranceAssetEvaluation) may be created (such as review or testing of the generated artefact). Activities may also be associated with a particular Technique that is used to carry out that activity.

In Figure 4 we show an example process model created from the meta-model described above. The example process used is the process of formally checking

[1] As described later, we use the term 'Activity' to refer to the relevant process.

[2] See http://www.opencoss-project.eu/

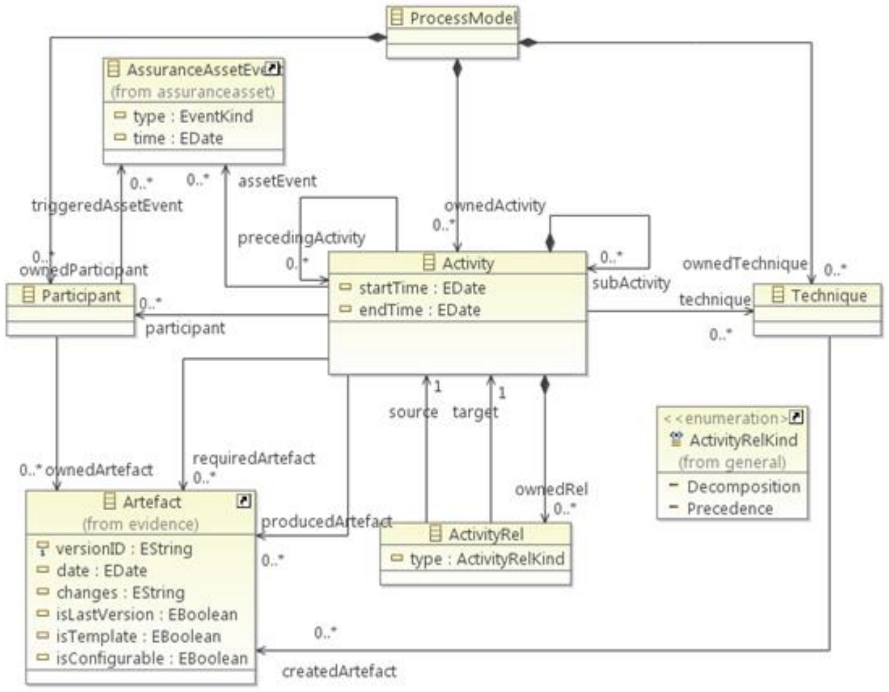

Fig. 2. EMF [19] core meta-model of processes

contracts specified using OCRA [14]. The results of the contract checking can be used to provide evidence as part of an assurance justification for the system by demonstrating that important security properties hold. As seen in Figure 4, the contract checking activity can be broken down to two sub-activities. Firstly the system model specified in AADL [15] must be translated to an OCRA specification. The second sub-activity is to perform the refinement check on the OCRA specification. The translation activity uses a tool called Compass [1], that has been evaluated for its correctness through testing. This activity requires the AADL specification, and produces a specification in the form of OCRA contracts. The contract specification is evaluated using consistency checking. The refinement activity requires the OCRA contract specification and uses another tool, the OCRA tool, in order to do the refinement. This tool has also been tested.

3 Confidence Argument Patterns

Patterns are widely used in software engineering as a way of abstracting the fundamental design strategies from the details of particular designs [4]. The use of patterns as a way of documenting and reusing successful assurance argument

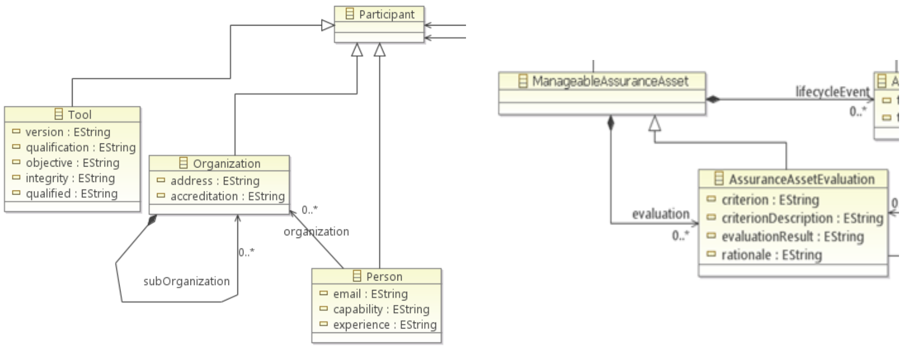

Fig. 3. Sub-types of Participant available in the process meta-model (left) and Extract from the evidence meta-model used in creating process models (right)

structures was pioneered by Kelly [11]. Assurance argument patterns provide a way of capturing the required form of an assurance argument in a manner that is abstract from the details of a particular argument. It is then possible to use the patterns to create specific arguments by instantiating the patterns in a manner appropriate to the application. Assurance argument patterns are a very useful technique as they can help to ensure a consistent approach is applied when similar assurance claims are required in different systems. It also provides a way of sharing experience across projects.

Figure 5 shows an assurance argument pattern we have developed that can be used to argue the trustworthiness of a process activity. This could be used to support the argument we presented in Figure 1. This argument pattern could be instantiated for an activity using information in a process model such as that shown in Figure 4.

This argument structure can be seen to make claims over the trustworthiness of the participants of the activity, the required and produced artefacts, the techniques used and the sub-activities. For each of these elements of the process, the argument shows they are sufficiently trustworthy through consideration of their demonstrable attributes. The notion of what is sufficiently trustworthy for a process element is driven firstly by the confidence required in the artefact being generated. As discussed in section 1, this is determined by the assurance required for the system as a whole, and the role of the artefact in the overall system argument. Errors in some evidence artefacts will have less impact on the assurance of the system, and the level of confidence required in such cases is correspondingly reduced.

The trustworthiness of the process must reflect the confidence required in the artefact itself. For each element of the process, it is necessary to take account of the role that the process element itself plays as part of the process to generate the artefact. For example, errors in a tool that generates an input file for an activity may be mitigated by other elements of the process, such as manual review of

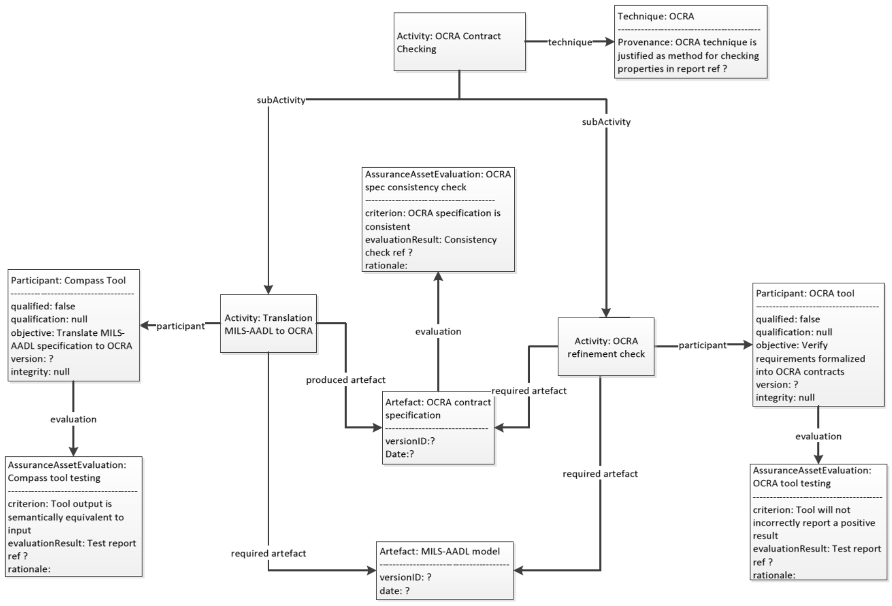
Activity: OCRA Contract Checking
technique
Technique: OCRA
Provenance: OCRA technique is justified as method for checking properties in report ref ?
subActivity
subActivity
AssuranceAssetEvaluation: OCRA spec consistency check
criterion: OCRA specification is consistent
evaluationResult: Consistency check ref ?
rationale:
Participant: Compass Tool
qualified: false
qualification: null
objective: Translate MILS-AADL specification to OCRA
version: ?
integrity: null
participant
Activity: Translation MILS-AADL to OCRA
evaluation
Activity: OCRA refinement check
participant
Participant: OCRA tool
qualified: false
qualification: null
objective: Verify requirements formalized into OCRA contracts
version: ?
integrity: null
produced artefact
required artefact
Artefact: OCRA contract specification
versionID:?
Date:?
evaluation
evaluation
AssuranceAssetEvaluation: Compass tool testing
criterion: Tool output is semantically equivalent to input
evaluationResult: Test report ref ?
rationale:
required artefact
AssuranceAssetEvaluation: OCRA tool testing
criterion: Tool will not incorrectly report a positive result
evaluationResult: Test report ref ?
rationale:
required artefact
Artefact: MILS-AADL model
versionID: ?
date: ?

Fig. 4. Process model for OCRA contract checking

that input file or the provision of multiple inputs. In such cases the level of trust required for that element may be reduced. What this means is that the claim that the element is sufficiently trustworthy must be interpreted for each element based upon a consideration of its role in the process. In some domains, and some standards, the notion of sufficiently trustworthy evidence is codified, such as requirements for testing and review to be performed by independent persons in DO-178C [17] and accepted and established notions of competency and tool qualification. Where such guidance exists this can also be used to help ensure proportionality in the process argument.

We have created argument patterns for all the process elements considered in the argument pattern in Figure 5, full details of these patterns are provided in [3]. Figure 6 shows one of these patterns, the argument pattern for creating arguments regarding the artefacts required by a process. This argument uses the evaluations performed on the artefact, plus attributes of the artefact, such as its version number and defined evaluation criterion, to form the confidence argument. The argument patterns for all of the elements of a process can similarly be instantiated from a process model such as the one in Figure 4.

4 Instantiating Argument Patterns

Instantiating an assurance argument pattern involves identifying the necessary information relating to the target system, required to choose and instantiate

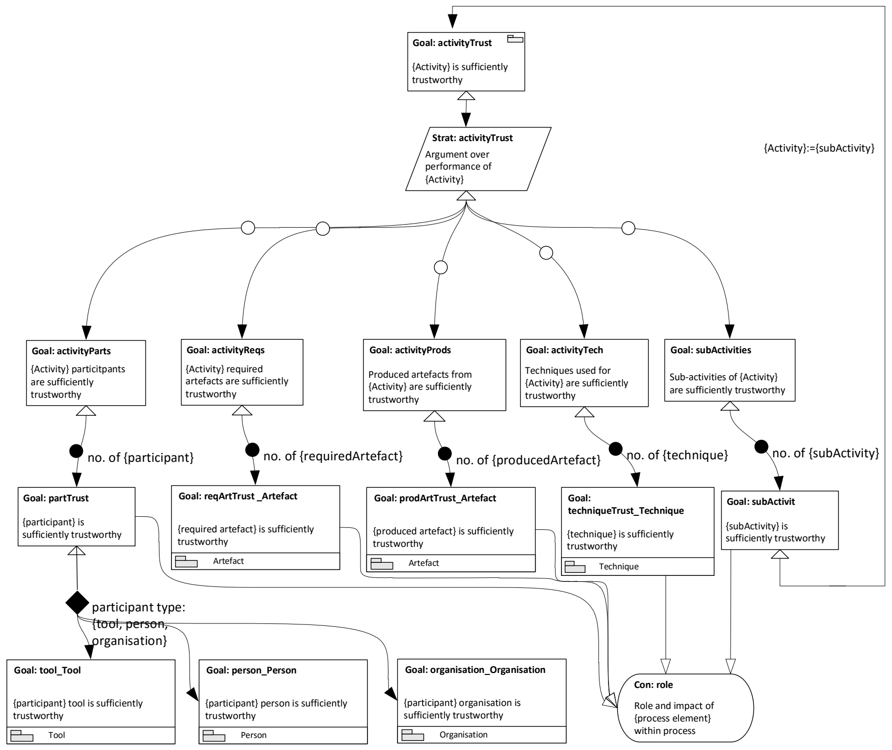
Goal: activityTrust
{Activity} is sufficiently trustworthy
Strat: activityTrust
Argument over performance of {Activity}
{Activity}:={subActivity}
Goal: activityParts
{Activity} particitpants are sufficiently trustworthy
Goal: activityReqs
{Activity} required artefacts are sufficiently trustworthy
Goal: activityProds
Produced artefacts from {Activity} are sufficiently trustworthy
Goal: activityTech
Techniques used for {Activity} are sufficiently trustworthy
Goal: subActivities
Sub-activities of {Activity} are sufficiently trustworthy
no. of {participant}
no. of {requiredArtefact}
no. of {producedArtefact}
no. of {technique}
no. of {subActivity}
Goal: partTrust
{participant} is sufficiently trustworthy
Goal: reqArtTrust _Artefact
{required artefact} is sufficiently trustworthy
Artefact
Goal: prodArtTrust_Artefact
{produced artefact} is sufficiently trustworthy
Artefact
Goal: techniqueTrust_Technique
{technique} is sufficiently trustworthy
Technique
Goal: subActivit
{subActivity} is sufficiently trustworthy
participant type: {tool, person, organisation}
Goal: tool_Tool
{participant} tool is sufficiently trustworthy
Tool
Goal: person_Person
{participant} person is sufficiently trustworthy
Person
Goal: organisation_Organisation
{participant} organisation is sufficiently trustworthy
Organisation
Con: role
Role and impact of {process element} within process

Fig. 5. Assurance argument pattern for confidence arguments

the assurance claims and to provide the required evidence. In this sense the instantiable elements of the patterns define requirements for information. It is possible to manually obtain this information and instantiate the argument patterns; this is current practice. A manual approach however is often not ideal. The instantiation is often repetitive and mechanistic in nature and prone to human error. Manual instantiation can also be time consuming and inconsistent. In [7] we described a model-based approach to automated instantiation of assurance argument patterns, based upon the specification of a weaving model that describes the dependencies between abstract elements of the argument pattern and elements of various system models for the target system. At instantiation, information is extracted from the relevant model elements of the target system to create the assurance argument. Our previous work on applying our approach ([7], [6]) has focussed predominantly on the automated instantiation of the technical argument. To move to a more complete automation of the assurance case, automation of the confidence argument is also required. Below we describe how the process model and confidence argument patterns presented can be used to create an assurance argument for a claim regarding a formal property of our

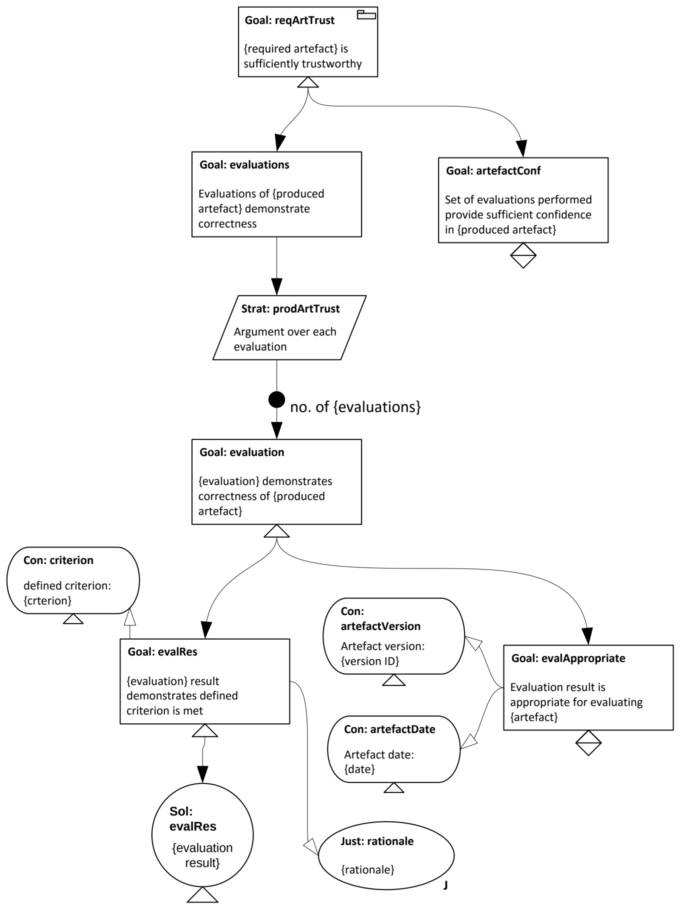
Goal: reqArtTrust
{required artefact} is sufficiently trustworthy
Goal: evaluations
Evaluations of {produced artefact} demonstrate correctness
Goal: artefactConf
Set of evaluations performed provide sufficient confidence in {produced artefact}
Strat: prodArtTrust
Argument over each evaluation
no. of {evaluations}
Goal: evaluation
{evaluation} demonstrates correctness of {produced artefact}
Con: criterion
defined criterion: {crterion}
Goal: evalRes
{evaluation} result demonstrates defined criterion is met
Con: artefactVersion
Artefact version: {version ID}
Goal: evalAppropriate
Evaluation result is appropriate for evaluating {artefact}
Con: artefactDate
Artefact date: {date}
Sol: evalRes
{evaluation result}
Just: rationale
{rationale}
J

Fig. 6. Assurance argument pattern for artefacts

example system as part of an assurance case for that system. Firstly we identify a claim we wish to support as part of the assurance case for the example system. In this case a claim is required for each formal property specified as part of the AADL specification of the system. One such claim is shown in Figure 7, which follows the form presented in Figure 1. In this case the formal property to be satisfied is “always (outL > high_bound)”. This is one of a number of specified properties of the AADL model required in order to guarantee the security of the system. The result of an OCRA contract check is used to demonstrate this property. Following the structure of Figure 1, the trustworthiness of the OCRA contract checking must be demonstrated for this argument to be compelling (Goal: activityTrust_Process). The OCRA checking process model in Figure 4 can be used in conjunction with the confidence argument patterns to create an argument to support this claim. An extract showing just the top level of the resulting argument is shown in Figure 8. This argument structure instantiation is completed using the patterns for each aspect of the model such as Figure 6.

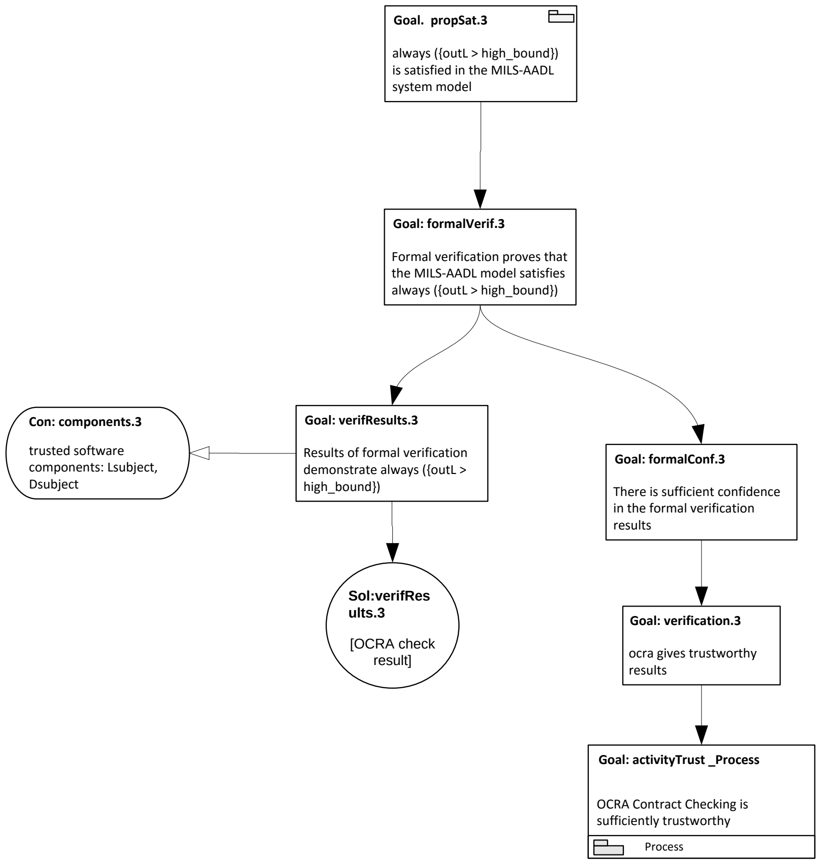
Goal. propSat.3
always ({outL > high_bound}) is satisfied in the MILS-AADL system model
Goal: formalVerif.3
Formal verification proves that the MILS-AADL model satisfies always ({outL > high_bound})
Con: components.3
trusted software components: Lsubject, Dsubject
Goal: verifResults.3
Results of formal verification demonstrate always ({outL > high_bound})
Goal: formalConf.3
There is sufficient confidence in the formal verification results
Sol:verifResults.3
[OCRA check result]
Goal: verification.3
ocra gives trustworthy results
Goal: activityTrust _Process
OCRA Contract Checking is sufficiently trustworthy
Process

Fig. 7. Part of the assurance argument for an example system

To make the process of instantiating the confidence argument patterns from the process models for a system easier and less error prone it is possible to make use of the model-based assurance case tool that we have developed to automatically generate the confidence argument from the process model. Below we briefly describe how the tool works.

– Argument patterns are created in machine-readable format using a graphical editor that creates a model in an XML form from a graphical representation of the argument pattern in GSN. We refer to these files (that are compliant with a GSN meta-model) as GSNML files.

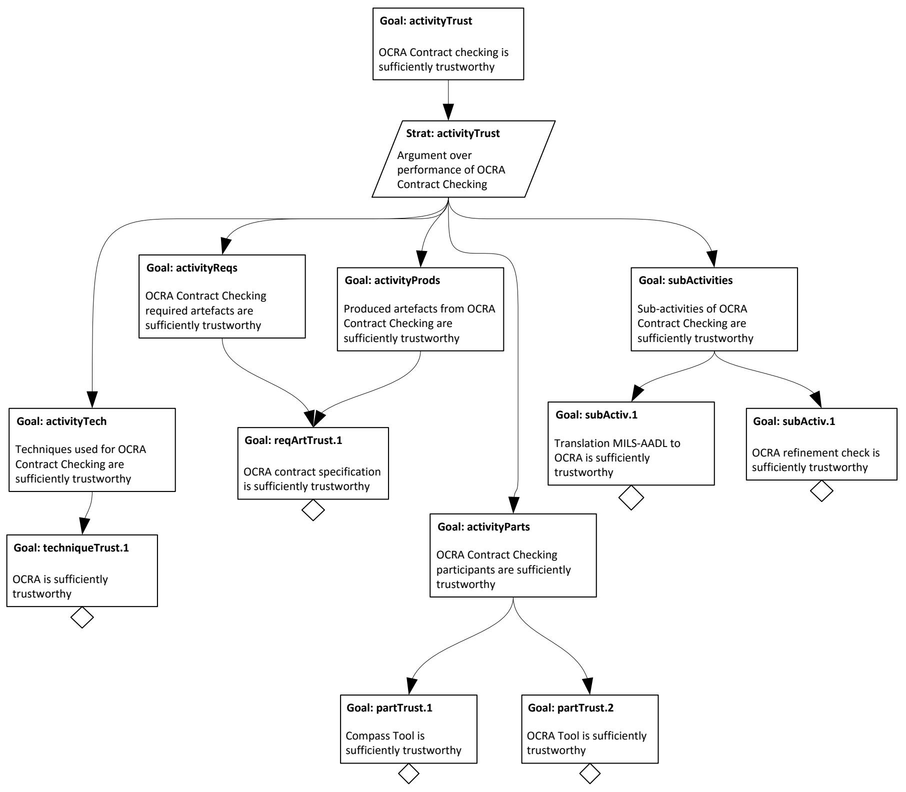
Goal: activityTrust
OCRA Contract checking is sufficiently trustworthy
Strat: activityTrust
Argument over performance of OCRA Contract Checking
Goal: activityReqs
OCRA Contract Checking required artefacts are sufficiently trustworthy
Goal: activityProds
Produced artefacts from OCRA Contract Checking are sufficiently trustworthy
Goal: subActivities
Sub-activities of OCRA Contract Checking are sufficiently trustworthy
Goal: activityTech
Techniques used for OCRA Contract Checking are sufficiently trustworthy
Goal: reqArtTrust.1
OCRA contract specification is sufficiently trustworthy
Goal: subActiv.1
Translation MILS-AADL to OCRA is sufficiently trustworthy
Goal: subActiv.1
OCRA refinement check is sufficiently trustworthy
Goal: techniqueTrust.1
OCRA is sufficiently trustworthy
Goal: activityParts
OCRA Contract Checking participants are sufficiently trustworthy
Goal: partTrust.1
Compass Tool is sufficiently trustworthy
Goal: partTrust.2
OCRA Tool is sufficiently trustworthy

Fig. 8. Part of the confidence argument for OCRA contract checking

– A weaving model is created to define links between elements in other models. In this case links are specified between GSN pattern models and the system or process models. The weaving model is then used as the specification for model transformations to generate the output model (instantiated assurance argument). The current version of the tool uses an interim solution for creating weaving models that involves creating the weaving models graphically and importing them to the tool as graphML files.
– The MBAC (Model Based Assurance Case) program is executed. This is an Epsilon Object Language (eol) program [12] that runs on the Eclipse platform. It takes the GSNML argument pattern files, the system and process models and corresponding meta-models, and the weaving model as inputs. The output is a GSN argument model for the target system that has been instantiated using information extracted from the system models.
– The argument model is generated as a GSNML file. This GSNML file can then be used to present information to the user in a number of ways. Firstly, the argument model can be represented graphically as a GSN structure.

Secondly, the model can be queried in order to provide a particular view on the assurance case. For example it is possible to just select those argument elements that remain undeveloped, requiring additional support from the system developer. Finally an instantiation table can also be generated that summarises how the pattern has been instantiated in tabular form, rather than having to consult the entire argument structure.

Using the model-based assurance case tool described above it becomes possible to:

- Automatically select the appropriate process model relevant to the evidence artefact cited in the assurance argument.
- Automatically populate the confidence argument pattern using information extracted from the process model.

It is important to note that when adopting this approach, thorough review of the assurance argument is still, as always, essential. However, rather than focussing review on the correctness of each argument created, the review effort can instead be focussed on the sufficiency of the pattern structure and the validity of the weaving model. Both of these, once reviewed can then be re-used for each instantiation. Another important focus for review becomes whether the role of each element of the process has been correctly interpreted, and whether what has been generated corresponds to this interpretation. We believe that in contrast to needing to review for correctness each time, this shift in focus helps to achieve more value from the review effort.

5 Conclusions

When creating an assurance justification for a system, the focus is often on the technical aspects of the assurance argument. The important confidence aspects are often addressed only in very general terms. Assurance cases are improved through provision of more focussed confidence arguments that address the integrity of specific artefacts through justification of the processes used. Creating such confidence arguments can be an onerous task for systems using complex processes and highly integrated tool chains. In this paper we have described how compelling confidence arguments can be developed directly from existing process models with the help of confidence argument patterns. We have described how existing tools can be used to automatically generate these arguments.

6 Acknowledgements

This work was part funded by the European Union FP7 D-MILS project (www.d-mils.org).

References

1. The COMPASS Project Web Site. http://compass.informatik.rwth-aachen.de/.
2. Opencoss Consortium. Common Certification Language: Conceptual Model D4.4 version 1.4. Available at: http://www.opencoss-project.eu/, 2015.
3. Integration of Formal Evidence and Expression in MILS Assurance Case. Technical Report D4.3, D-MILS Project, March 2015. http://www.d-mils.org/page/results.
4. E. Gamma, R. Johnson, R. Helm, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
5. Goal Structuring Notation Working Group. GSN Community Standard Volume 1, 2011.
6. R. Hawkins, I. Habli, and T. Kelly. The Need for a Weaving Model in Assurance Case Automation. *Ada User Journal*, 36(3):187–191.
7. R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly. Weaving an Assurance Case from Design: A Model-Based Approach. In *Proceedings of the 16th IEEE International Symposium on High Assurance Systems Engineering*, 2015.
8. R.D. Hawkins, T.P. Kelly, J. Knight, and P. Graydon. A New Approach to Creating Clear Safety Arguments. In *Advances in Systems Safety*, pages 3–23. Springer London, 2011.
9. IEC. IEC 61508 - Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Technical Report IEC 61508, The International Electrotechnical Commission, 1998.
10. ISO. ISO 26262 - Road Vehicles Functional Safety. Technical Report ISO 26262, ISO, Geneva, Switzerland, 2011.
11. T. Kelly. *Arguing Safety A Systematic Approach to Safety Case Management*. PhD thesis, The University of York.
12. D. Kolovos, L. Rose, A. Garcia-Dominguez, and R. Paige. The Epsilon book, 2013. http://www.eclipse.org/epsilon/doc/book/.
13. Object Management Group. Software and Systems Process Engineering Metamodel Specification (SPEM) version 2.0, 2008.
14. The Othello Contract Refinement Analysis (OCRA) Tool. https://es.fbk.eu/tools/ocra.
15. International Society of Automotive Engineers. Architecture Analysis and Design Language Annex (AADL), Volume 1. SAE Standard AS 5506/1, SAE, June 2006.
16. Object Management Group (OMG). Structured Assurance Case Metamodel (SACM), Version 1.0, 2013.
17. RTCA. DO-178C - Software Considerations in Airborne Systems and Equipment Certification. Technical Report DO-178C, RTCA, 2011.
18. S.Nair, N. Walkinshaw, T. Kelly, and J.L. de la Vara. An Evidential Reasoning Approach for Assessing Confidence in Safety Evidence. In *Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering (ISSRE 2015)*, 2015.
19. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley, 2008.
20. Linling Sun. *Establishing Confidence in Safety Assessment Evidence*. Phd thesis, University of York, 2012.