The Need for a Weaving Model in Assurance Case Automation

R. Hawkins, I. Habli, T. Kelly

The Department of Computer Science, The University of York, York, YO10 5GH. Tel: +44 1904 325463; email: {richard.hawkins|ibrahim.habli|tim.kelly}@york.ac.uk

Abstract

*In this paper we describe how the automated instantiation of assurance case arguments will require information to be extracted from multiple models of a system and its environment and engineering processes, e.g. safety and verification processes. For this to be done successfully the dependencies between the models must be explicitly, completely and correctly captured. We describe how a model-based approach, **model weaving**, provides an excellent mechanism for modelling the correspondences that exist between models and discuss how model weaving can be applied in the context of assurance cases.*

1 Introduction

Assurance cases provide an explicit means for justifying and assessing confidence in critical properties of interest such as safety or security properties. An assurance case should contain a reasoned and compelling argument, supported by a body of evidence [1]. We are concerned with the challenge of how to make it easier for system developers to create valid and compelling arguments for their systems. To help to guide assurance argument development, the concept of providing reusable patterns of argument and evidence was developed [2]. Assurance argument patterns allow the desired structure of the argument to be captured, whilst abstracting from the details of a particular target system. An assurance argument can be created for a system by instantiating the argument pattern with information about the target system. Assurance argument patterns have been shown to be useful in helping developers create arguments [3]. However current practice is mainly to instantiate argument patterns manually.

There are a number of advantages to be gained from automating the generation of assurance arguments:

- Human error in instantiating patterns is eliminated.
- The argument can be generated directly from, and is therefore consistent with and traceable to, the design and development models of the system themselves.
- Instantiations can be produced quickly and easily to reflect the current state of development.
- Change management of the argument becomes automatic.
- Consistent, reusable instantiation rules can be established, ensuring consistent and repeatable pattern instantiation.

Any approach for automating assurance argument generation requires as a minimum:

- model(s) of the required assurance argument structure - for this we use the assurance argument patterns;
- model(s) of the system - containing the information necessary to instantiate the patterns, often including models of the environment and development processes
- transformation rules to generate the output model (the assurance argument).

If we assume that we have available the required assurance argument patterns, the challenge becomes one of identifying the necessary system models, and defining a set of transformation rules. These are the focus of this paper. Section 2 discusses the system models that are required to generate an assurance argument. Section 3 discusses an approach to defining transformation rules – model weaving. Section 4 describes how model weaving can be applied to assurance cases. In section 5 we discuss related work and describe our conclusions.

2 System Models for Assurance

Assurance argument patterns can be captured using the graphical notation GSN [1]. Instantiation of assurance argument patterns involves both instantiating ‘roles’ in the argument patterns, and making instantiation choices. Roles are instantiable entities within elements of the argument pattern. They represent an abstract entity that needs to be replaced with a concrete instance appropriate for the target system. For example in Figure 1, the role within this assurance claim, represented in curled braces is ‘Function’. This entity must be replaced with the name of the relevant function of the system. In addition, argument patterns will often include multiplicity relations, where the number of required argument elements must also be determined (e.g. an entity created for each of the functions present in the system design).

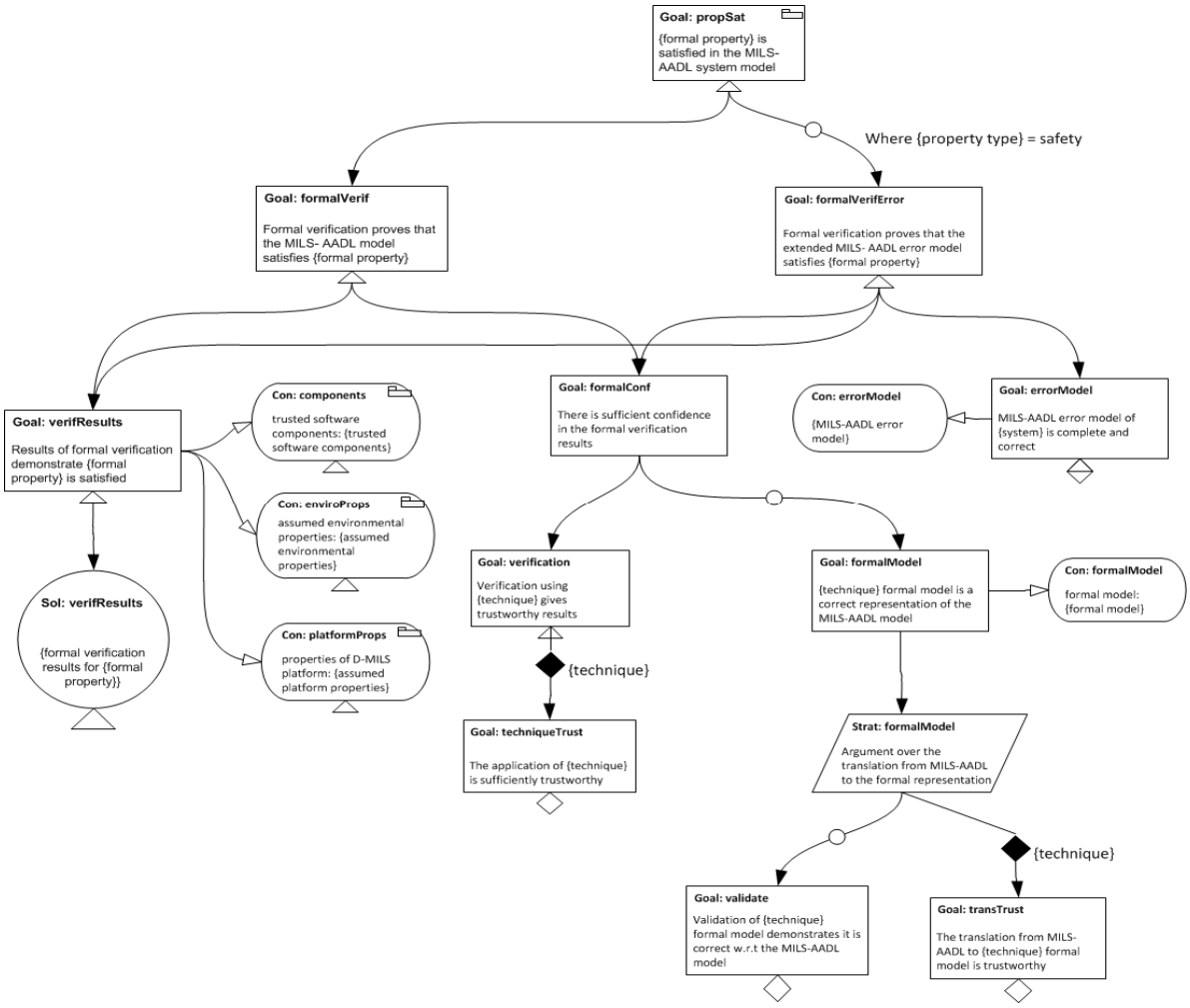

Figure 2. An example pattern for part of a D-MILS system assurance case

Assurance argument patterns will also often represent choices for different argument approaches that may be adopted. At instantiation, the assurance claims most appropriate for the target system must be chosen from the options provided in the pattern. A more detailed example of an assurance argument pattern is provided in Figure 2.

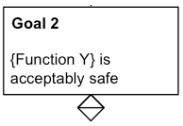

Figure 1. A GSN Argument Element Requiring Instantiation

All of these instantiation decisions are made using information about the system. The nature of the claims made in an assurance case can vary enormously between systems and domains, but in all cases there will be a requirement to include two types of argument, technical risk arguments and confidence arguments [4]. The technical arguments reason about risk reduction and the mitigation of system hazards. These will include consideration of specific design features and properties of the system. The technical argument requires consideration of design, analysis and verification artefacts. Arguments of confidence document the reasons for having confidence in the technical argument. The confidence argument will in general require consideration of the processes used to generate the development artefacts.

In most cases it is not possible to acquire all the information that is required for a complete and compelling assurance case including both technical and confidence arguments from a single model of the system. In work such as [5] it is described how it is possible to extract a lot of information required to create an assurance argument from system specifications such as AADL models. However such specifications would not contain all the information required for the assurance case. For example, although development artefacts themselves, such as safety analyses, are often integrated into such system specifications (e.g. as AADL error models), information to support a confidence argument (about the way in which those artefacts were generated) is not included (and it wouldn't be appropriate to do so). Information regarding verification is also not commonly included in such specifications. Clearly multiple models will be required to generate a complete assurance argument.

As an example we present in Figure 2 an example argument pattern that we created to form part of the assurance argument for a Distributed MILS (D-MILS) system [6]. There can be seen in this argument pattern to be a number of roles that it was possible for us to instantiate using

information extracted from an extended MILS-AADL model of the system, such as:

- formal properties (the properties to be demonstrated);
- trusted software components
- assumed platform properties.

However there can also be seen to be other claims within the argument where information will be required that is not available from the MILS-AADL model. For example the claim that the application of a particular technique to verify a formal property is sufficiently trustworthy will require information about the process for applying the technique, and about the tools used (similarly for claims regarding the translation from informal to formal representations). We obtained this information from models produced of the verification process and tool chain. Another example is the formal verification results, which are not part of the MILS-AADL model, but contained within a separate verification model.

3 Model Transformation

In the previous section we described how the instantiation of assurance argument patterns will normally require information from multiple source models. There will be (often complex) relationships between these models. Relationships will exist both between the source information models and the instantiable elements of the argument pattern models, and also between elements of the different source models. Successful pattern instantiation requires that the relationships between model elements are correctly specified.

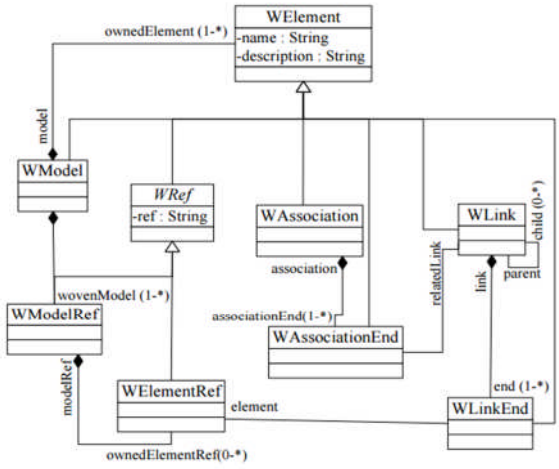

Figure 3. The weaving metamodel

Model weaving, is described in [7] as "*a method of establishing correspondences with semantic meaning between model elements*". The central concept is a weaving model which is "*a special kind of model used to save these correspondences*". Like all other models the weaving model must conform to a weaving metamodel. The basic form of the weaving metamodel, taken from [8] is shown in Figure 3. Weaving models can be created to define links between model elements. The semantics of the link can be defined for specific links in the weaving model. The weaving metamodel also includes associations that can define relationships between the links in the weaving model. In Section 4 we describe how associations and links may be used in a weaving model for an assurance case.

The weaving model that is created can then be used as the specification for model transformations to generate the output model or models from the set of source models. Model weaving can bring a number of advantages when compared with other approaches to model transformation. The weaving model specification is independent of implementation, which means that the same weaving model can be used to create multiple transformations. The semantics of the transformations in the weaving model are defined by the user. This allows much greater flexibility when applying the weaving model. In addition, as the weaving model is itself a model, it allows a seamless model-driven approach to be adopted for all aspects of the assurance case process.

4 Applying Model Weaving to Assurance Cases

In [9] we have described an approach that uses a weaving model to create an assurance argument from assurance argument pattern(s) and a set of system models. Figure 4 provides an overview of our current prototype tool that implements this approach. Below we briefly describe each of the elements.

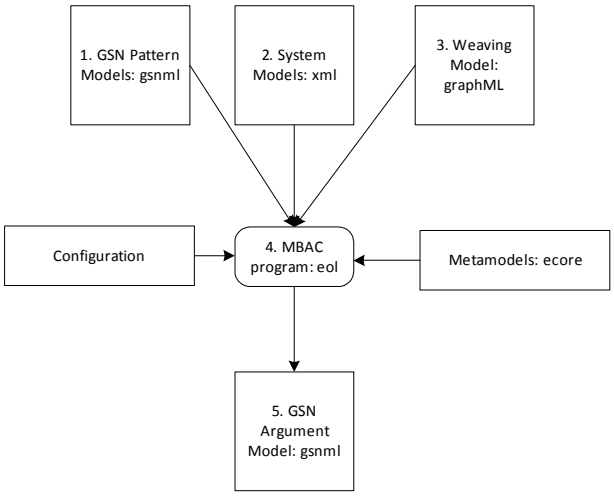

Figure 4. An implementation of a model weaving approach for assurance cases

1. The argument patterns must be provided in machine-readable format. For this we have developed 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 the GSN metamodel) as GSNML files.

2. Any system models that conform to a defined metamodel may be taken as input.

3. 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. Future development of the tool will include the creation of weaving models directly from the metamodels, rather than graphically. The weaving model is represented using typed nodes and edges with properties declared to specify additional attributes such as the metamodel element type or the name of the target model. Figure 5 shows an example weaving model created in this manner. The nodes on the left hand side represent roles within the argument patterns whereas the nodes on the right hand side are elements of the source metamodels. The edges represent weaving links and associations.

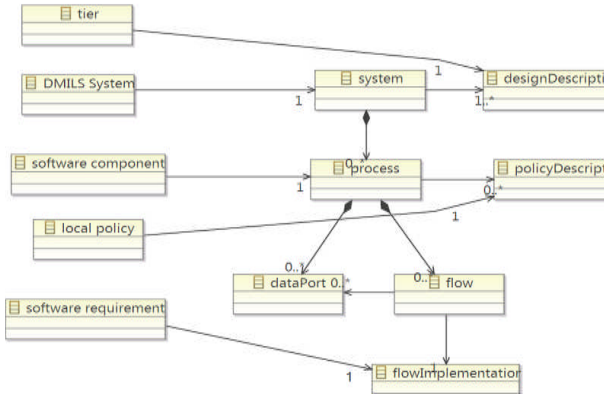

Figure 5. An example weaving model for an assurance argument pattern instantiation

Note that changes to the system models should not normally require changes to the weaving model, so long as no changes are made to the existing argument patterns and only system models conforming to the same metamodels are used. This means that changes to the system design can be quickly reflected in the assurance case.

4. The MBAC (Model Based Assurance Case) program is an Epsilon Object Language (eol) program [10] that runs on the Eclipse platform. It takes the GSNML argument pattern files, the system models and corresponding metamodels, 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.

5. 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.

The GSN argument model can also be used as the basis for performing verification of the assurance argument structure, as well as validation of the argument with respect to the system models. These verification and validation activities are the subject of on-going research.

5 Conclusions

It is a shared goal of many researchers [11, 12, 13] to increase automation in the generation and maintenance of assurance arguments. Our approach complements these approaches, but crucially, it does not depend on having to extract and pre-process assembly and instantiation data. By automatically extracting information directly from the design and safety analysis models themselves, a model weaving approach ensures traceability between the sources of information, e.g. in design, process and analysis models, and the assurance case. Automation in this way also has the potential to support the coevolution of system design and assurance cases.

The correct definition of the weaving model is of course crucial to the success of this approach. Although our initial work has demonstrated the feasibility of the approach, further work is required to more fully understand and model the relationships and constraints that exist between system design models (such as AADL) and other models required for the assurance case (such as process models).

Acknowledgements

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

References

[1] GSN Community Standard Working Group (2011), *GSN Community Standard*, Available at www.goalstructuringnotation.info/.

[2] T. Kelly and J. McDermid (1997), *Safety Case Construction and Reuse Using Patterns*, in proc. Safecomp 97, pp 55-69, Springer.

[3] R. Hawkins et. al. (2011), *Using a Software Safety Argument Pattern Catalogue: Two Case Studies.* In proc. of Safecmp 11, Springer.

[4] R. Hawkins et. al. (2011), *A New Approach to Creating Clear Safety Arguments*, In proc. of the Nineteenth Safety-Critical Systems Symposium, pp 3-23, Springer.

[5] A. Gacek et. al. (2014), *Resolute: An Assurance Case Language for Architecture Models*, In proc. of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology. pp 19-28

[6] J. Rushby (2008). *Separation and Integration in MILS (The MILS Constitution,*. Technical Report, SRI International

[7] M. Didonet et. al. (2005), *Applying generic model management to data mapping*, in proc. Bases de Données Avancées (BDA05).

[8] M. Didonet et. al. (2005), *AMW: A generic model weaver*, in proc. 1ères Journées sur l'Ingénierie Dirigée par les Modèles.

[9] R. Hawkins et. al. (2015), *Weaving an Assurance Case from Design: A Model-Based Approach*, In proc. of 16th IEEE International Symposium on High Assurance Systems Engineering.

[10] D. Kolovos et. al. (2013), *The Epsilon Book*, available at http://www.eclipse.org/epsilon/doc/book/.

[11] E. Denney et. al. (2012), *Advocate: An Assurance Case Automation Toolset*. in proc. Workshop on Next Generation of System Assurance Approaches for Safety Critical Systems, pp 8-21.

[12] Y. Matsuno and S. Yamamoto (2013), *An implementation of GSN community standard*, In proc. of Assurance Cases for Software-Intensive Systems (ASSURE).

[13] J. Rushby (2013), *Mechanized support for assurance case argumentation*, in proc. 1st International Workshop on Argument for Agreement and Assurance (AAA 2013), Springer LNCS.