Uniform Model Interface for Assurance Case Integration with System Models

Andrzej Wardziński[1,2] and Paul Jones[3]

[1] Gdańsk University of Technology, Gdańsk, Poland
Andrzej.Wardzinski@pg.edu.pl
[2] Argevide, Gdańsk, Poland
[3] US Food and Drug Administration, Silver Spring, MD, USA
Paul.Jones@fda.hhs.gov

Abstract. Assurance cases are developed and maintained in parallel with corresponding system models and therefore need to reference each other. Managing the correctness and consistency of interrelated safety argument and system models is essential for system dependability and is a nontrivial task. The model interface presented in this paper enables a uniform process of establishing and managing assurance case references to various types of system models. References to system metamodels are specified in an argument pattern and then used for assurance case instantiation. The proposed approach permits incremental development of assurance cases that maintain consistency with corresponding system models throughout the system development life cycle.

Keywords: Assurance case · Safety case · System models · Argument pattern

1 Introduction

When developing systems, engineers necessarily rely on models to facilitate comprehension, analysis, and communication of complex development details. Such models may represent design and development processes, system component architecture, behavior, and other types of development abstractions. We refer to each of these types of models in this paper collectively as *system models*.

Assurance cases may mirror these system models to varying levels of detail and refer to their elements. It is important that these references are unambiguous, complete, and correct so that someone creating, modifying or reviewing an assurance case can be confident of being directed to the right element or property. When a few assurance cases are developed for components of the system (e.g. system of systems) it is critical to ensure that the assurance cases refer to the same concepts, system models, model interfaces, and properties.

Our goal is to develop a generic model interface between an assurance case and system models which will allow establishing and maintaining assurance case references to elements of various system models. The interface should provide system model referencing services desired by the assurance case user (developer, assessor etc.) while hiding unnecessary details that may not add to comprehension. The idea is to not have

DOI: 10.1007/978-3-319-66284-8_4

the assurance case user track model element references manually but rather to assist the user by providing required information describing the desired system model(s). The proposed model interface provides:

- A uniform way of specifying assurance case references to system model elements,
- The ability to specify restrictions in a form of relations between referenced model elements to strengthen assurance case consistency,
- A mechanism for maintenance of the argument references when the models are modified,
- The possibility to develop an assurance case incrementally when system models are evolving throughout the system life cycle (the argument instantiation does not have to be carried out all at once, some parts of the pattern may stay un-instantiated until the corresponding system models are available).

Use of a uniform model interface for establishing and maintaining assurance case relations to system models will make it simpler to manage consistency between the two. The initial cost of this approach is in the development of the model interface. It must be implemented for each system model type to which the assurance case references. The implementation will depend on the specific data format used by each type of system model. System models can be represented in XML, a database, "flat" text file or a structured document. The system model can be also managed by any application with an API offering access to the model data (for example OSLC interface – Open Services for Lifecycle Collaboration).

In the next section, we will analyze the general problem of managing relations between assurance cases and system models. The concept of a model interface is presented in the third section. Section 4 describes the process how the model interface is used in assurance case integration with system models. We summarize the approach in Sect. 5.

2 Assurance Case Integration with System Models

Assurance cases may refer to many aspects of systems like systems goals and requirements, risks and mitigations, system structure, elements properties, life cycle activities and their products. The most common approach is to use textual references and manually manage their consistency with system models and real world artefacts. For example, textual references were proposed in developing assurance cases for software model-driven development [1].

One of the initial studies on managing explicit assurance case references to external models or ontologies was described in a safety argument for hospital treatment [2]. Górski et al. used UML to represent a claim model and a related context model.

Evidence argument elements can also be used to represent elements of system models. Sljivo et al. presented an extension of assurance case metamodels enabling use of evidence element references to a system component and safety contract metamodel [3].

Currently the most advanced solution for use of models to describe the context of an assurance case is a weaving metamodel proposed by Hawkings et al. [4] and applied in the D–MILS project [5]. The weaving metamodel captures dependencies between role bindings specified in an assurance case pattern and system models. Abstract dependency information captured in the weaving metamodel is used in the argument instantiation process.

The weaving metamodel describes:

- system model classes specific for a given system perspective (e.g. AADL or FMEA model),
- relations between model classes specified as UML associations,
- role bindings in the argument patterns, that is terms used in pattern element names to denote specific system model elements (e.g. "System" in a claim "{System} safety policy is enforced"),
- relations between role bindings resulting from the pattern argument structure (they are directed relations which describe the scope of a binding role context),
- relations between role bindings and system model classes.

The approach presented in this paper is similar to the use of the weaving model, however, we use two separate elements in place of the weaving model. The first one is a model interface describing system models in a unified way and the second one is a reference table describing argument relations to system models. This approach offers new possibilities described in the next sections.

3 The Concept of the Model Interface

The concept of a model interface arose from the observation that assurance cases refer to different types of system models but assurance case developers would prefer a standard way to establish and maintain the references. The model interface has been designed to satisfy these needs and facilitate a uniform reference management process in assurance case development and maintenance. The concept has been developed as an extension of a reference mechanism described in [6].

References to system models are first specified on an abstract level when an argument pattern is developed. Argument patterns may refer to abstract concepts like subsystems, components, events or hazards. To ensure abstract references are unambiguous, they should be specified in a context of a formally defined system metamodel. UML class models can be used for such specification. References to metamodel elements will be sufficiently precise to ensure unambiguity.

The argument pattern serves as a basis for development of a "well formed" argument appropriate for a specific system model. In the assurance case instantiation process each abstract reference should be replaced with a reference to an existing system model element which satisfies the conditions imposed by the abstract reference. Use of a formally defined system metamodel in abstract references helps ensure the consistency of the instantiated argument with the referred system. The model interface should operate on both levels: abstract system metamodels and concrete system models that describe a real system.

The model interface serves as an intermediary between the assurance case and system models. As presented in Fig. 1 it provides an interface for assurance cases (on the left of the diagram) to access system models. The model interface does not keep any information on the assurance case argument references to system models. All the reference data is stored in the assurance cases in an abstract and a concrete reference table.

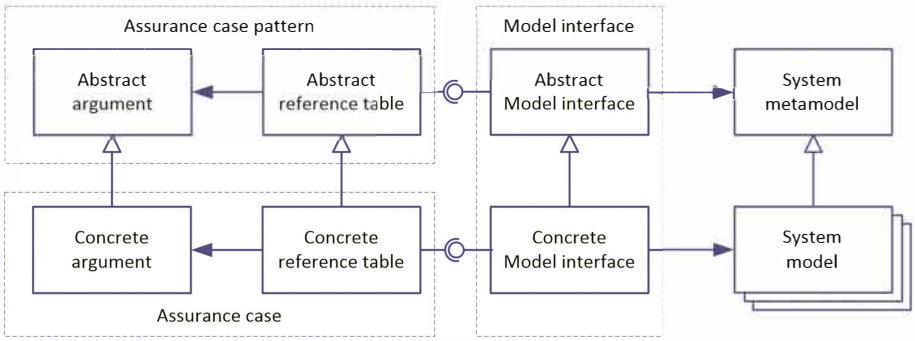

Fig. 1. Assurance case instantiation metamodel

You can notice two levels of the model interface presented on the diagram. The upper part works on an abstract level, i.e. the system metamodel used for argument pattern references. The lower part provides an interface to system models used when the concrete argument is instantiated. On the abstract level the model interface services return:

(a) a list of available model types,
(b) a list of element types for a given model type,
(c) a list of relations for a specific model element type.

Abstract functions of the model interface do not need existing system models to function because they return data on a metamodel level. To work with the concrete model interface one first needs to initiate it with a specific model (for example provide a model file name) and then the interface functions can be called to return:

(d) a list of models of a given type,
(e) a list of model elements of a given type,
(f) a list of elements which satisfy a given relation,
(g) detailed data of a given element (when its identifier is provided).

The presented set of functions is sufficient to specify abstract references in argument patterns and then instantiate them to produce concrete assurance cases. This process will be presented in the next section.

4 Process of Assurance Case Integration with System Models

The integration process consists of steps performed in two phases. The first phase is performed on the abstract level when the assurance case pattern is developed but no real system exists yet. This is what might be called a *pre-development* phase. Here, an abstract argument pattern with references to a system metamodel is established. The second phase is what might be called a *development* phase when an assurance case is instantiated with references to models of the developed system and then maintained throughout the system life cycle. In the following subsections, we will describe the steps of this process.

- Pre-development phase steps
  1. System metamodel specification
  2. Model interface development
  3. Argument pattern development
- Development phase steps
  4. System modeling
  5. Assurance case development
  6. System models and assurance case maintenance (iteration of steps 4 and 5)

The process covers the whole assurance case life cycle from the moment an argument pattern is created in the context of an abstract metamodel, to assurance case maintenance after a product has been placed on the market.

Details of the integration process are presented below with the use of a sample argument fragment that references a system risk model. The referenced system is a Patient Control Analgesia (PCA) infusion pump [7].

4.1 Step 1: System Metamodel Specification

The first step of the process is to specify a system risk metamodel and its data format to allow implementation of model interface functions.

In our example, we will use a *risk metamodel* presented in Fig. 2. as a UML class diagram (for simplicity class' attributes are not shown on the diagram). The risk model describes system hazards, their causes and control measures. The structure of the

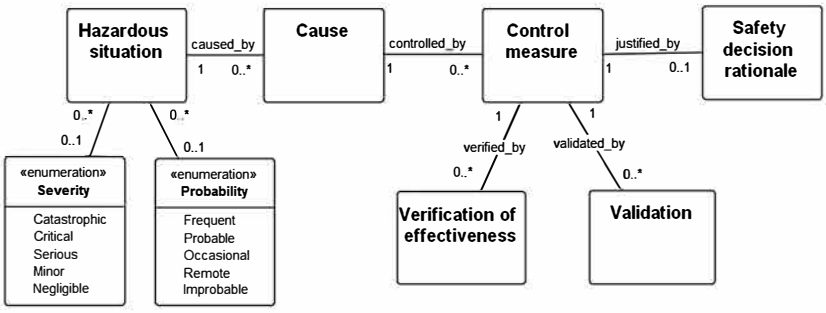

Fig. 2. The system risk metamodel

metamodel is based on the hazard table format specified in [8]. The model data format is an XML file and the XML schema is based on the presented risk metamodel.

The result of this step is a set of system metamodels along with detailed technical data on the model format necessary for implementing the model interface as described in the next step.

4.2 Step 2: Model Interface Development

A model interface is, in general, a software module that provides a uniform interface for access to any type of system models. It is assumed that the model interface is only allowed to read system models and cannot modify them. Access to system models is realized by instantiations of abstract classes *ModelType*, *ElementType* and *Expression*. An implementation of these classes is required for each system metamodel intended to be referenced from the assurance case.

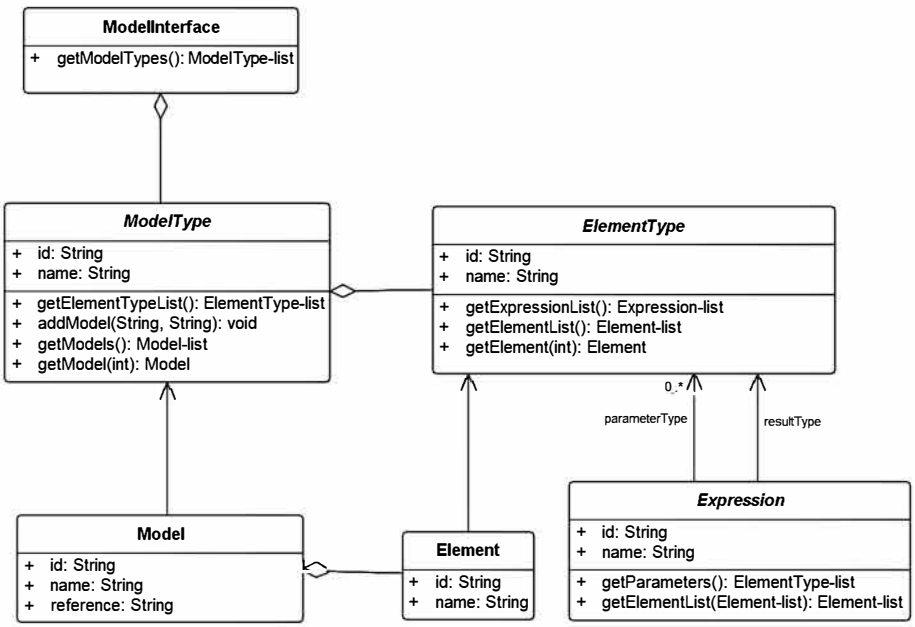

Fig. 3. Model interface metamodel

The model interface implementation for the risk metamodel presented in Fig. 2 encompasses an instantiation of *ElementType* class for each risk model element like *Hazard*, *Cause*, *Severity*. The model interface should also include implementation of the *Expression* class for each relation specified in the metamodel. For example it may contain an expression *causesOfHazard*(*Hazard*) to denote the relation *caused_by* between classes *HazardousSituation* and *Cause*. This interface function takes one

parameter of *Hazard* type and returns a set of elements of *Cause* type. Given that the risk models are represented in XML files, we chose to use XQuery scripts to implement access to the model data.

```
for $causeId in doc($ModelRef)//relationship[@source
    =$HazardId]/@target
for $cause in doc($ModelRef)//hazardElement[@xmi:id
    =$causeId and @xsi:type="HA:Cause"]
return <result><id>{data($cause/@xmi:id)}</id>
    <name>{data($cause/@content)}</name></result>
```

The script returns the result in XML format, for example:

```
<result><id>C1</id><name>Sensor failure to detect air
bubble</name></result>
<result><id>C2</id><name>Safety subsystem failure to stop
the pump</name></result>
<result><id>C4</id><name>Pump does not stop on request
</name></result>
```

The result consists of model elements identifiers and names. This is transformed into a collection of *Element* objects and returned by the model interface. The element name will be presented for the assurance case user and the identifier will be used for traceability of the referenced model element.

The complete model interface implements all the functions specified in Sect. 3 and its scope covers all the system model classes and relations between them. The presented example refers to a risk model, but model interface implementations for other types of models (e.g. AADL, EAST-ADL) are also possible.

4.3 Step 3: Argument Pattern Development

In this step an argument pattern with references to the system metamodel is developed. The model interface should provide operations which return available system model types, their element types and relation, permitting the user to specify correct references

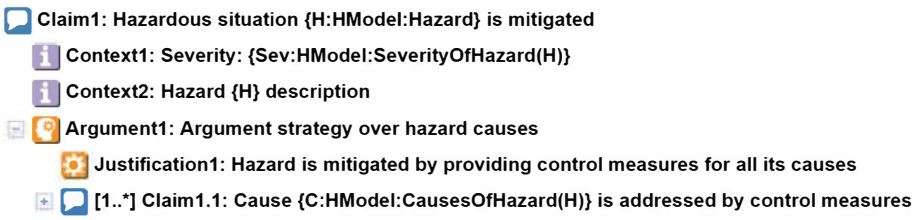

Fig. 4. Argument template

to a system metamodel. A definition of an abstract reference consists of three attributes: a reference name, a model type and an element selector. The reference name is used internally in the assurance case pattern while the model type and the element selector are used to identify the referred element of the metamodel. For example, reference "H" in *Claim1* in Fig. 4 refers to elements of the *Hazard* class in *HModel*. The presented argument fragment uses textual hierarchical notation. Labels of the argument elements indicate the type for each element.

Once a reference is specified, it can also be used for other argument elements. It can be used directly as a reference, for example *Context2* refers to the same hazard *H* as *Claim1*. A reference can also be used as a parameter for a selector. Hazard *H* is used as a parameter for references in *Context1* and *Claim1.1* (Fig. 4). This method of reference specification ensures that instantiated *Claim1.1* will refer only to causes of a hazard specified by the instantiation of its parent *Claim1*.

Model interface operation *getModelTypes()* (compare Fig. 3) helps to ensure that the argument pattern references relate to existing model types. Operations *getModelElementTypeList()*, *getExpressionList()* and *getParameterList()* assist in managing correct references to the system metamodel.

All the abstract references defined in the argument pattern are recorded in the *abstract reference table* (Table 1) which is an integral part of the assurance case pattern.

Table 1. Abstract reference table

Pattern element id	Reference name	Model type	Element selector
Claim1 Context2	H	HModel (the *risk model*)	Hazard
Context1	Sev	HModel (the *risk model*)	SeverityOfHazard(H)
Claim1.1	C	HModel (the *risk model*)	CausesOfHazard(H)

The result of the pattern development step is a complete argument pattern with references to the system metamodel represented in the abstract reference table. The pattern is not specific to any system and it can be used for developing assurance cases for a class of systems.

4.4 Step 4: System Modeling

The development phase begins with the system modeling step. The goal is to develop models of a real system that comply with the corresponding system metamodels to which the assurance case will refer. Each system model, when ready, can be used for building safety arguments (described in the next step).

One of system models often used in safety critical systems is the risk model. In Table 2 we present a fragment of the risk model in the form of a hazard table.

Table 2. Excerpt of the PCA infusion pump hazard table

Hazardous situation	Severity	Cause	Control measure
Air in line	Critical	Sensor failure to detect air bubble	Sensor failure rate 10E-6 for air bubbles with the size greater than 1 ml
		Safety subsystem failure to stop the pump	Safety subsystem failure rate 10E-6/h
		Pump does not stop on request	Pump design ensures stopping the flow in the absence of control signal

The risk model is recorded in an XML file and its file format is based on the metamodel presented in Sect. 4.1. The model interface will read these XML files to get information on referenced model elements. An XML file excerpt is presented below:

```
<hazardElement content="Air in line"
  xsi:type="HA:HazardousSituation" xmi:id="H1"/>
<hazardElement
  content="Sensor failure to detect air bubble"
  xsi:type="HA:Cause" xmi:id="C1"/>
<hazardElement
  content="Safety subsystem failure to stop the pump"
  xsi:type="HA:Cause" xmi:id="C2"/>
<relationship xsi:type="HA:CausedBy" xmi:id="S1C1"
  source="H1" target="C1"/>
<relationship xsi:type="HA:CausedBy" xmi:id="S1C2"
  source="H1" target="C2"/>
```

The result of this step of the process is a set of system models in a format readable by the model interface. One does not need to have all the system models developed before starting the argument instantiation. An assurance case can be developed incrementally and can refer to models or parts of a model that are ready at a given time.

4.5 Step 5: Assurance Case Development

The objective of this main step is to develop an assurance case based on the argument pattern (see step 3) and establish references to models of a particular system. To do this the model interface must be initialized with concrete models of a real system. The user selects an argument pattern and then specifies the file locations or links to system models to which the assurance case will refer.

The instantiation process is performed top down starting with the top pattern element. For each abstract reference and multiplication operator the user has to decide how a given pattern element should be instantiated. For each abstract reference the model interface can search existing system models for elements which satisfy the reference conditions and the user may choose a model element for instantiated

(concrete) reference. When a multiplication operator is used, a separate argument section can be created for each reference value (e.g. for all causes of a hazard).

The risk model fragment presented in Table 2 consists of one hazard and three causes. The instantiation process starts with the top claim (Fig. 4). It refers to a model element *H* of class *Hazard*. The model interface function *getElementList()* returns a list of hazards defined in the hazard table and the user can select any hazard from the list. The reference can be instantiated to the hazard 'air in line' specified in the hazard table. The next argument element to be instantiated is *Context1*. It refers to a model element of Severity class in relation to hazard *H*. The model interface will return an element with value 'Critical'. For the next pattern element *Claim1.1* the multiplication operator [1..*] enables the user to choose a set of referenced elements. The model interface will return a list of causes for hazard H and all of them can be used in the claim instantiation. The result is presented in Fig. 5 (the identifiers of the instantiated argument have been reset).

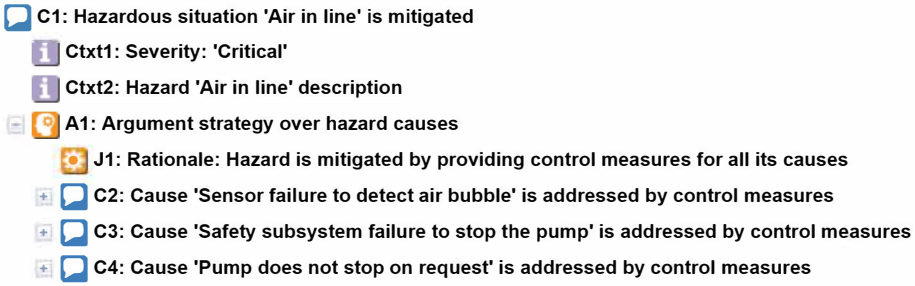

Fig. 5. Instantiated argument

The final result of this step is the instantiated argument along with the reference table describing all the relations to system models. The reference table specifies model element values and identifiers which can be used to track model changes (Table 3).

Table 3. Concrete reference table

Argument element id	Reference name	Model name	Model element id	Element name
C1 Ctxt2	H	PCAHazardTable.xml	H1	Air in line
Ctxt1	Sev	PCAHazardTable.xml	S1	Critical
C2	C	PCAHazardTable.xml	C1	Sensor failure to detect air bubble
C3	C	PCAHazardTable.xml	C2	Safety subsystem failure to stop the pump
C4	C	PCAHazardTable.xml	C4	Pump does not stop on request

4.6 Step 6: System Models and Assurance Case Maintenance

The objective of this step to accommodate the evolution of assurance cases throughout the system life cycle. An assurance case is usually not developed all at once; rather it is developed gradually during system development and is subject to many changes. System models are also developed gradually. In fact, a change in one often affects a change in the other.

Steps 4 and 5 can be repeated to gradually develop system models and the corresponding assurance case. The model interface provides features to facilitate this process in the following way:

- References to new models can be added at any time in the assurance case maintenance process. The user can change an existing reference to make it refer to a new model or add a new argument branch in the pattern where a multiplication operator is used. The new argument section can refer to new or already existing system model elements.
- Assurance case reference consistency with system models can be verified at any moment of time. For each reference the model interface functions *getElementList()* and *getElement()* (compare Fig. 3) can be used to check if the current reference value refers to a correct model element. The model interface can also return the current list of model elements which satisfy the condition specified by the abstract reference. When the system model is modified then new model elements can be reported. The user may want to add new argument elements with such new references. In some cases the model interface may report that an existing reference value is not a valid model element. Broken or inconsistent references can be reported to allow the user to correct them.
- In case the system model element name is changed, the assurance case can be automatically updated. The model element identifier stored in the reference table can be used as a parameter for the function *getElement()* to get its current data. When the system model element name is modified, it can be updated in the assurance case. In this way changes in system models can be propagated to the assurance case.

Use of a model interface allows keeping the assurance case up to date with systems models and to evolve in accordance with progress in system development throughout the system life cycle.

5 Summary

The presented concept of the model interface and the integration process facilitates assurance case consistency with system models. In particular it enables:

- A uniform process of definition and instantiation of assurance case relations to various system models independent of technical model representations (XML format, databases, files or external systems) provided that a model interface is implemented. This simplifies managing references to diverse system model types by the assurance case developer.

- Improved internal assurance case consistency by use of explicitly defined relations between system model elements. Those relations help in managing consistency between different references in the argument.
- Improved assurance case maintainability thanks to the possibility of incremental assurance case instantiation and establishing references to new system models.
- Better traceability because specified model element references can be verified for changes at any moment.
- Improved verifiability of the assurance case thanks to the possibility of analysis of consistency between the assurance case and system models.

The presented approach has been verified with a prototype tool that implements the model interface for the risk model and a subset of AADL models developed with Osate2. The prototype performs assurance case instantiation and exports the argument in XML format compliant with OMG SACM metamodel.

The model interface metamodel can be compared to the terminology classes in OMG SACM 2.0 metamodel [9]. The terminology classes in SACM consist of *Category* class which can correspond to a model type, *Term* class which can relate to model elements and *Expression* class which can be equivalent to *Expression* class in the model interface. Further research is required to determine if OMG SACM 2.0 should be extended to cover the model interface and references to models.

The presented process assumes that the argument pattern is static when the assurance case is developed for a given system. Usually system evolutionary life cycles span years requiring changes in the argument structure. Such changes would be introduced to the argument pattern as well and then propagated to the assurance case. Maintaining assurance case consistency with an evolving argument pattern may be challenging and requires further work.

The presented concept of a model interface is new to assurance case development. It offers the possibility of more robust assurance cases that map directly to system models, facilitating the development of unambiguous arguments.

References

1. Jee, E., Lee, I., Sokolsky, O.: Assurance cases in model-driven development of the pacemaker software. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010. LNCS, vol. 6416, pp. 343–356. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16561-0_33
2. Górski, J., Jarzębowicz, A., Leszczyna, R., Miler, J., Olszewski, M.: Trust case justifying trust in an IT solution. Reliab. Eng. Syst. Saf. **89**, 33–47 (2005)
3. Sljivo, I., Gallina, B., Carlson, B., Hansson, H., Puri, S.: A method to generate reusable safety case argument-fragments from compositional safety analysis. J. Syst. Softw. **131**, 570–590 (2017). doi:10.1016/j.jss.2016.07.034. Elsevier
4. Hawkins, R., Habli, I., Kolovos, D., Paige, R., Kelly, T.: Weaving an assurance case from design: a model-based approach. In: IEEE 16th International Symposium on High Assurance Systems Engineering (2015)
5. Compositional assurance cases and arguments for distributed MILS, D-MILS Project deliverable D4.2, University of York (2015)

6. Wardziński, A., Jarzębowicz, A.: Towards safety case integration with hazard analysis for medical devices. In: Skavhaug, A., Guiochet, J., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9923, pp. 87–98. Springer, Cham (2016). doi:10.1007/978-3-319-45480-1_8
7. Larson B.R., Hatcliff, J.: Open Patient-Controlled Analgesia Infusion Pump System Requirements, Kansas State University, SAnToS TR 2014-6-1 (2014)
8. Jones, P.L., Taylor, A.: Medical device risk management and safety cases. Bio-Med. Instrum. Technol. **49**, 45–53 (2015)
9. Structured Assurance Case Metamodel (SACM), version 2.0 – Beta, Object Management Group (2016)