Model Based System Assurance Using the Structured Assurance Case Metamodel

Ran Wei, Tim P. Kelly, and Richard Hawkins

University of York, United Kingdom
{ran.wei,tim.kelly,richard.hawkins}@york.ac.uk

Abstract. Assurance cases are used to demonstrate confidence in system properties of interest (e.g. safety and/or security). A number of system assurance approaches are adopted by industries in the safety-critical domain. However, the task of constructing assurance cases remains a manual, lengthy and informal process. The Structured Assurance Case Metamodel (SACM) is a standard specified by the Object Management Group (OMG). SACM provides a richer set of features than existing system assurance approaches. SACM lays a foundation for model-based system assurance, which bears great application potentials in growing technology domains such as Open Adaptive Systems. However, the intended usage of SACM has not been sufficiently explained. In addition, there has not been support to interoperate between existing assurance case models and SACM models.

In this paper, we explain the intended usage of SACM based on our involvement in the specification process of SACM. In addition, to promote a model based approach, we provide SACM compliant metamodels for existing system assurance approaches (the Goal Structuring Notation and Claims-Arguments-Evidence), and the transformations from them to SACM. We also briefly discuss the tool support for model-based system assurance which help practitioners make the transition from existing system assurance approaches to model-based system assurance using SACM.

Keywords: Model Driven Engineering; Structured Assurance Case Metamodel; Model Based System Assurance; Goal Structuring Notation; Claims, Arguments and Evidence

1 Introduction

Systems/services used to perform critical functions require justifications that they exhibit necessary properties (i.e. safety and/or security). *Assurance case*s provide an explicit means for justifying and assessing confidence in these critical properties. In certain industries, typically in the safety-critical domain, it is a regulatory requirement that an assurance case is developed and reviewed as part of the certification process [1]. An assurance case is a document that facilitates information exchange between various system stakeholders (e.g. between operator and regulator), where the knowledge related to the safety and security of the system is communicated in a clear and defend-able way [2].

Assurance cases are typically represented either textually - using natural languages; or graphically - using structured graphical notations such as the Goal Structuring Notation (GSN) [3] or Claims-Arguments-Evidence (CAE) [4]. Graphical notations have gain popularity due to their abilities to express clear and well structured argumentations. A number of tools exist which implement GSN and CAE to produce safety cases [5]. Some tools adopt Model-Driven Engineering (MDE) to produce models that conform to their own versions of GSN/-CAE metamodels [6–10].

To improve standardisation and interoperability, the Object Management Group (OMG) specified and issued the Structured Assurance Case Metamodel (SACM). SACM is developed by the specifiers of existing system assurance approaches (e.g. GSN and CAE), based on the collective knowledge and experiences of safety/security practitioners. Comparing to existing assurance case approaches, SACM provides additional features such as fine-grained modularity, controlled vocabulary, and argument-evidence traceability. Therefore, SACM is more powerful in terms of expressiveness. However, neither detailed explanation has been provided in the OMG specification to demonstrate how to use SACM, nor the relationships between existing assurance case approaches (i.e. GSN and CAE) and SACM has been discussed. This brings challenges to the adoption of SACM due to the complexity of SACM and the sophistication of its intended usage.

Model-based system assurance has attracted a significant amount of interests in recent years due to the benefits provided by MDE such as automation and consistency. Model-based system assurance is particularly important for concepts such as Open Adaptive Systems (OAS), where open (safety/security critical) systems connect to each other, and adapt to changing contexts at runtime.

As the principal contributors of SACM and the originators of GSN, in this paper, we provide a detailed explanation of SACM and discuss its relationship with existing system assurance approaches (i.e. GSN and CAE). The contributions of this paper are as follows:

- A definitive exposition of SACM version 2.0;
- A detailed discussion on how to use SACM to create an assurance case model.
- GSN and CAE metamodels that are compliant with SACM;
- Comprehensive mappings from GSN/CAE to SACM;

This paper is organised as follows. In Section 2 and Section 3, we provide the background and the motivation of our work. In Section 4 we provide detailed discussions about the facilities provided by SACM. In Section 5 we provide examples to illustrate the semantics of the elements provided in SACM, and how to use SACM to construct argumentation patterns, and how to integrate assurance cases. In Section 6, we discuss the relationship between existing notations and SACM. We provide SACM compliant metamodels for GSN and CAE and their mappings to SACM. In Section 7, we briefly discuss tool support for model-based system assurance. We finally conclude the paper in Section 8.

2 Background and Motivation

2.1 Safety Cases

The concept of assurance cases has been long established in the safety-related domains, where the term *safety case* is normally used. For many industries, the development, review and acceptance of a safety case form a key element of regulatory processes. This includes the nuclear [11], defence [12], civil aviation [13] and railway [14] industries. Safety cases are defined in [3] as follows: *A safety case should communicate a clear, comprehensible and defensible argument that a system is acceptably safe to operate in a particular context.*

Historically, safety arguments were typically communicated in safety cases through free text. However, there are problems experienced when text is the only medium available for expressing complex arguments. One problem of using free text is that the language used in the text can be unclear and poorly structured, there is no guarantee that system engineers would produce safety cases with a clear and well-structured language. Also, the capability of expressing cross-references for free text is very limited, multiple cross-references can also disrupt the flow of the main argument. Most importantly, the problem with using free text is in ensuring that all stakeholders involved share the same understanding of the argument to development, agree and maintain the safety arguments within the safety case [3].

To overcome the problems of expressing safety arguments in free text, graphical argumentation notations were developed. Graphical argumentation notations are capable of explicitly representing the elements that form a safety argument (i.e. requirements, claims, evidence and context), and the relationships between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). Amongst the graphical notations, the *Goal Structuring Notation* (GSN) [3] has been widely accepted and adopted [15]. The key benefit experienced by companies/organisations adopting GSN is that it improves the comprehension of the safety argument amongst all of the key project stakeholders (e.g. system developers, safety engineers, independent assessors and certification authorities), therefore improving the quality of the debate and discussion amongst the stakeholders and reducing the time taken to reach agreements on the argument approaches being adopted.

Another popular graphical argumentation notation is *Claims-Arguments-Evidence* (CAE) [4]. CAE views assurance cases as a set of *Claim*s supported by *Argument*s, which in turn rely on *Evidence*. Compared to CAE, GSN provides more granular decomposition of satety arguments, and supports additional features such as modularity and argument patterns [16]. In this paper, we focus on GSN (and its relationship to SACM) since we are the principal contributors to the standardisation of both GSN and SACM.

A number of graphical assurance case tools have been developed due to the popularity of graphical argumentation notations. A recent study [5] has looked into and compared assurance case tools that have been developed in the past twenty years (where 32 of them support GSN). The majority of the tools do not support model-based approach (i.e. creating model-based graphical assurance cases). Whilst graphical assurance cases are valuable in communicating the safety and/or security properties of the system, model-based assurance cases enable higher level model management operations to be performed.

2.2 Safety Cases and Model Driven Engineering

Model-Driven Engineering (MDE) is a contemporary software engineering approach. In MDE, *model*s are first class artefacts, therefore *driving* the development. There are two important aspects in MDE: domain specific modelling and model management. Domain specific modelling enables domain experts to capture the concepts in their systems in the form of *metamodel*s, which are then used to create models of their systems (that conform to the defined *metamodel*s). Model management enables a variety of operations to be performed on models in an automated manner, which include, but not limited to: Model Validation, Model-to-Model Transformation, Model-to-Text Transformation and Model Merging. MDE has been proven to improve consistency and productivity significantly due to the automation provided by model management operations [17, 18].

MDE is beneficial to system assurance case approaches. For example, model validation can be used to check well-formedness of assurance cases (e.g. In GSN, a *Strategy* cannot be directly supported by a *Solution*); model-to-text transformation can be used to generate assurance case report; and model merging can be used to bind assurance cases. A number of assurance case tools adopt MDE, such as AdvoCATE [6], D-Case Editor [7], ASCE [8], Astah GSN [9], and CertWare [10]. However, a common problem with these tools is that they define their own GSN metamodels. This is due to the fact that there has not been a standard GSN metamodel (apart from the mappings from GSN to SACM provided on the GSN website[1]). Thus, there may be interoperability problems when one wishes to import GSN models created by other tools to his/her own tool. Although the tools mentioned above all claim to support SACM, the support was for SACM version 1.0 (realsed in June, 2015), which was replaced by SACM version 2.0 (released in March, 2018). Since SACM version 1.0 was also not sufficiently explained (no work has been done in this aspect), there may be cognitive gaps between different tool developers, so that exported SACM models from the tools may differ. In addition, a considerable amount of changes have been introduced to SACM 2.0, the claimed support for SACM of these tools are out-dated.

2.3 Assurance Cases and the Structured Assurance Case Metamodel

There has been an increasing interest in the use of structured argumentation in other domains, particularly for demonstrating system security [19]. Such argu-

[1] http://www.goalstructuringnotation.info/

mentations are typically referred to as *security cases*. The similarities between safety and security cases have been highlighted in [20]. Therefore, the term *assurance case* is a broader definition: *An assurance case should communicate a clear, comprehensible and defensible argument that a system/service is acceptably safe and/or secure to operate in a particular context.*

To promote standardisation and interoperability, the Object Management Group (OMG) specified and issued the *Structured Assurance Case Metamodel* [21]. SACM is developed by the developers of existing system assurance approaches (e.g. GSN and CAE), based on the collective knowledge and experiences of safety and/or security practitioners over the period of two decades. Therefore, features that are not supported by GSN and CAE have been evaluated and included in SACM. A selection of such features are summarised in Table 1 (with an indicative but non-exhaustive list of works that motivate the features).

Feature	**Motivated By**
F1. Modularity	[22]
F2. Multiple Language Support	[23]
F3. Controlled Vocabulary	[24, 25]
F4. Describing the Level of Trust in Arguments	[26, 27]
F5. Counter-Arguments in Assurance Cases	[28]
F6. Traceability from Evidence to Artifact	[29]
F7. Automated Assurance Case Instantiation	[30, 31]

Table 1: Features added to SACM

Modularity (F1). It is important to promote modularity for assurance cases, so that system safety/security can be argued on a component basis (instead of having an enormous assurance case diagram) [22]. Modularity is supported by GSN, with the notion of *Module* and *ContractModule*. In SACM, a finer grained modularity is provided. The users are able to selectively declare the argument/artifact/terminology elements externally through *package interfaces*. And then integrate these packages using *package bindings*. In this way, engineers are able to understand more clearly how assurance cases are integrated.

Multiple Language Support (F2). Multiple language support seems trivial in assurance cases created using GSN and CAE. However, when creating SACM version 2.0 we realised that the importance of multiple language support not only lies in the ability to describe arguments in multiple natural languages, but also lies in the ability to describe arguments in formal languages [23]. Arguments described using formal language enable the possibility of (semi-) automated reasoning of system safety, which is particularly important in the context of runtime model-based system assurance.

Controlled Vocabulary (F3). Various studies have identified the importance of controlled vocabulary used in system assurance arguments [24, 25]. In SACM, the users are able to create controlled vocabulary (which can relay to actual model/model elements that define the vocabulary) and refer to them in the assurance argument.

Describing the Level of Trust in Arguments (F4). There is a need to argue the trustworthiness of arguments made in an assurance case, motivated in [26]. In GSN, an extension (Assurance Claim Points) was suggested to allow the association of confidence arguments to a primary argument [26]. However, in SACM, there is a facility specifically designed to enable the users to argue the level of trust for argument elements.

Counter-Arguments in Assurance Cases (F5). Sometimes, it is also important to present counter arguments in an assurance case [28]. However, in GSN and CAE there is no specific means to express counter arguments. In SACM, an assertion can be declared as *counter*, to declare a reversal argument.

Traceability from Evidence to Artifact (F6). In GSN and CAE, evidence in assurance cases are described using natural language, there is no built-in facility that enables the traceability from evidence to the actual artefact. Existing work achieves traceability through the use of an external metamodel [29]. In SACM, traceability is naturally supported without the need of an external model.

Automated Assurance Case Instantiation (F7). Assurance case templates are useful in capturing good practice in system assurance for re-use. GSN provides the notion of *GSN patterns*, which enables the users to create abstract safety cases (templates), and then *instantiate* the patterns based on actual system information to create concrete safety cases. In [30] and [31], a model based automated pattern instantiation approach is discussed and presented. This approach uses a intermediate *weaving model* to link GSN patterns with system models, and requires extensions to the GSN metamodel. In SACM, automated instantiation can be achieved without the introduction of these extensions.

As principal contributors of SACM and the originators of GSN, we acknowledge that SACM is more powerful than GSN and CAE in terms of expressiveness. Therefore, we feel the need to promote SACM, especially in the context of model-based system assurance. However, in the SACM specification there is limited information on the intended usage of SACM. In order to exploit SACM's full potential, and to promote the adoption of SACM, it is necessary to explain SACM in detail so that safety and security engineers can fully use SACM to achieve higher level goals (e.g. automated model validation to check either well-formedness or runtime safety certification).

In the current state of practice, graphical notations such as GSN remain the most popular approach for system assurance. SACM is designed to support GSN, but the OMG specification does not provide a mapping between GSN elements and SACM elements. This is due to the fact there has not been a SACM aligned GSN metamodel. Thus, in this paper we provide a GSN metamodel which aligns to SACM. In addition, there is also a need to translate from GSN to SACM. First of all, the OMG has not defined a concrete syntax (i.e. graphical notation) for SACM elements, which makes creating SACM models a tedious and error-prone process. Thus, to make the transition from GSN to SACM, it is good practice to use GSN notations to construct arguments and then transform to SACM using model-to-model transformation. Secondly, since GSN has been widely adopted in industry, practitioners can convert their legacy diagrams into GSN models, and then transform to SACM to enable model-based system assurance. Due to the reasons above, in this paper, we will also provide a mapping (model-to-model transformation) from GSN to SACM.

2.4 Model-Based System Assurance and Open Adaptive Systems

System assurance is typically difficult for Open Adaptive Systems (OAS). Increasing challenges for OAS assurance are observed due to the popularity of emerging concepts such as Cyber-Physical Systems (CPS). In particular, OAS dynamically connect to each other (openness) and adapt to a changing context at runtime (adaptivity) [32]. Industry sees huge economic potential in such systems - particularly because their openness and adaptivity enable new types of promising applications in various domains, such as automotive, health care, and home automation [33]. The majority of application domains of OAS are safety-critical, such as car2car scenarios and collaborative autonomous mobile systems. If these systems fail, they may cause harm and lead to temporary collapse of important infrastructures, with potential catastrophic consequences for industry and society. Therefore, it is imperative to ensure the dependability of OAS in order to realise their full potential. However, OAS poses significant challenges to their assurance due to their open and adaptive nature, as it is nearly impossible to sufficiently anticipate the concrete system structure, its capabilities and the environmental context at design time.

Therefore, existing design time system assurance activities are inappropriate to enable dynamic system assurance for OAS at runtime. In [32], the authors identify the importance of system assurance at runtime for OAS and propose the idea of Models@Runtime, in the sense that system assurance information is exchanged when OASs interconnect with each other to reason about the dependability of the to-be-formed system of systems.

SACM provides a model basis that could underpin AssuranceCase@Runtime in this context, this is backed by the fact that SACM is used in the DEIS project [33] as a backbone for its Open Dependability Exchange metamodel (ODE), to ensure the dependability of Cyber-Physical Systems. SACM provides an open, universal assurance case modelling language which establishes a framework for reasoning about the assurance of a wide range of system properties, linked to heterogeneous models and evidence.

2.5 Summarised Motivations

The motivations of our work can be summarised as follows.

- **The need for model-based approach**. As previously discussed, MDE is beneficial to system assurance. High level operations such as model validation, mode-to-model transformation and model merging can be performed on model based assurance cases. In addition, model based assurance cases are the key to assure safety/security related Open Adaptive Systems.

- **Heterogeneous GSN metamodels and misalignment to SACM**. Although a number of model-based assurance case tools exist, they implement their own versions of GSN, their mappings from GSN to SACM version 1.0 are not unanimous due to the fact that SACM version 1.0 was not sufficiently explained. In addition, due to the release of SACM version 2.0, the claimed supports for SACM of existing model based GSN/CAE tools become out-dated.
- **SACM and its application in Open Adaptive Systems**. SACM provides more features than GSN/CAE, which makes it more powerful in terms of expressiveness. SACM is fundamental to runtime system assurance, which is the key for safety/security related Open Adaptive Systems.
- **Insufficient explanation of SACM**. SACM has been developed over the period of 10 years, and is the result of significant deliberation and consultation. As a consequence, there are many use cases behind the features (known to the authors of this paper as principal authors of the SACM standard) that are present that may not be apparent to users on first inspection. E.g. to support existing concepts such as modularity, patterns, and meta argumentation. SACM and the relationship between GSN/CAE and SACM are not sufficiently explained, which hinders its adoption.

3 The Goal Structuring Notation

It is necessary to discuss the Goal Structuring Notation (GSN) before discussing SACM as SACM was developed based on the concepts in GSN. GSN is a well established graphical argumentation notation to represent safety arguments in a structured way. GSN is widely adopted within safety-critical industries for the presentation of safety arguments within safety cases. The core elements of GSN are shown in Figure 1.

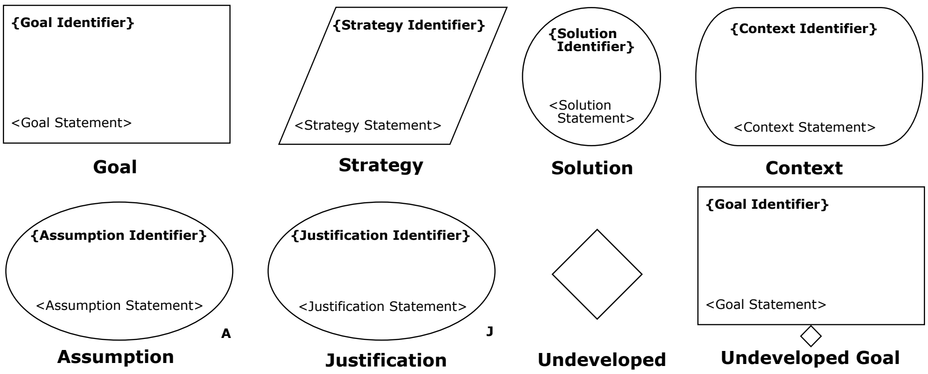


Fig. 1: Core GSN elements.

A *Goal* represents a safety claim within the argumentation. A *Strategy* is used to describe the nature of the inference that exists between a goal and its supporting goal(s). A *Solution* represents a reference to an evidence item or multiple evidence items. A *Context* represents a contextual artefact, which can be a statement, or a reference to contextual information. An *Assumption* represents an assumed statement made within the argumentation. A *Justification* represents a statement of rationale. An element can be *Undeveloped*, which means that a line of argument has not been developed yet, it can apply to *Goals* and *Strategies*, the *Undeveloped Goal* in Figure 1 provides an example.

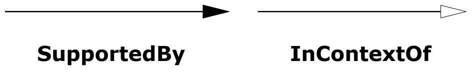


Fig. 2: GSN connectors.

Core elements of GSN are connected with two types of connectors, as shown in Figure 2. The *SupportedBy* connector allows inferential or evidential relationships to be documented. The *InContextOf* relates contextual elements (i.e. *Context*, *Assumption* and *Justification*) to *Goals* and *Strategies*.

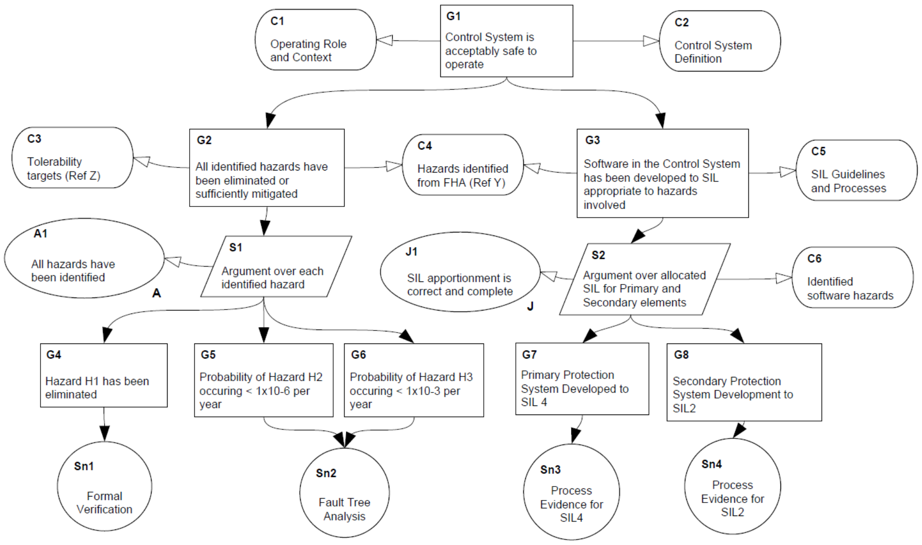

C1
Operating Role and Context
G1
Control System is acceptably safe to operate
C2
Control System Definition
C3
Tolerability targets (Ref Z)
G2
All identified hazards have been eliminated or sufficiently mitigated
C4
Hazards identified from FHA (Ref Y)
G3
Software in the Control System has been developed to SIL appropriate to hazards involved
C5
SIL Guidelines and Processes
A1
All hazards have been identified
A
S1
Argument over each identified hazard
J1
SIL apportionment is correct and complete
J
S2
Argument over allocated SIL for Primary and Secondary elements
C6
Identified software hazards
G4
Hazard H1 has been eliminated
G5
Probability of Hazard H2 occuring < 1x10-6 per year
G6
Probability of Hazard H3 occuring < 1x10-3 per year
G7
Primary Protection System Developed to SIL 4
G8
Secondary Protection System Development to SIL2
Sn1
Formal Verification
Sn2
Fault Tree Analysis
Sn3
Process Evidence for SIL4
Sn4
Process Evidence for SIL2

Fig. 3: An example goal structure [34].

When the elements of the GSN are linked together in a network, they are often referred to as a *goal structure*. The purpose of any goal structure is to show how *Goals* are successively broken down into sub-*Goals* until a point is reached where *Goal*s can be supported by direct reference to available evidence (*Solution*s). An example goal structure is shown in Figure 3.

Figure 4 shows how GSN supports modularity. Goal structures can be organised in *Module*s. For example, for a system that consists of two components A and B, it is possible to organise the safety case of component A in Module A and safety case of component B in Module B. Modularity promotes re-use, so that safety cases for system components can be re-used when different components integrate to form a system. When integrating system safety cases, a *Contract Module* can be used to *bind* different *Module*s together. Binding is done via the use of *Away Goal*s, *Away Context*s and *Away Solution*, where *Goal*s, *Context*s and *Solution*s from an external *Module* can be referenced. Like other GSN elements, away elements can be connected using *SupportedBy* and *InContextOf* connectors.

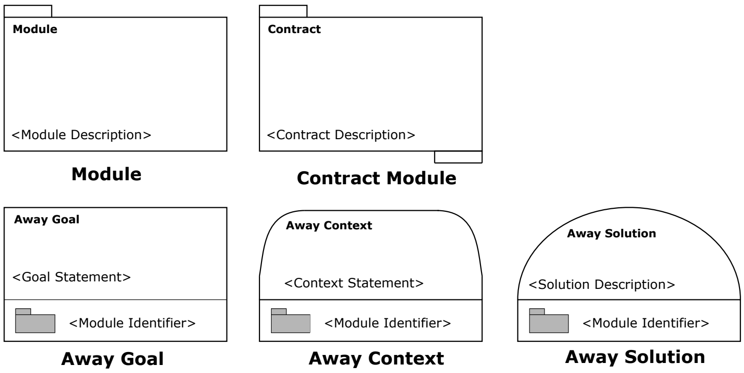


Fig. 4: Modular GSN elements.

For a successful GSN safety case, people tend to define a template of the safety case to re-use its structure in the future. GSN provides the extension for users to define templates that are called *GSN patterns*. In [35], the use of patterns is discussed, patterns are a means of documenting and reusing successful assurance argument structures. Safety case argument patterns provide a way of capturing the required form of a safety 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. Pattern instantiation refers to the process of constructing concrete system safety cases by populating the templates provided in the GSN pattern with actual system information. Figure 5 shows the elements in GSN which enable the users to create patterns.

The *Uninstantiated* indicator marks that an element is abstract and to be instantiated, at some later stage, the abstract element needs to be replaced with a more concrete instance. *Uninstantiated indicator* can be associated to any GSN element, Figure 5 demonstrates how it can be associated to a *Goal*.

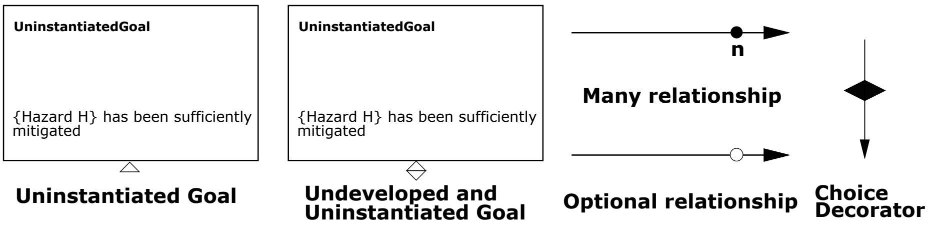

UninstantiatedGoal
{Hazard H} has been sufficiently mitigated
Uninstantiated Goal
UninstantiatedGoal
{Hazard H} has been sufficiently mitigated
Undeveloped and Uninstantiated Goal
n
Many relationship
Optional relationship
Choice Decorator

Fig. 5: GSN pattern elements.

The *Undeveloped and Uninstantiated* indicator marks an element (in particular, *Goal*s and *Strategies*) to be both abstract (to be instantiated) and needs more development (needs supporting argument), Figure 5 demonstrates its usage on a Goal. In GSN patterns, the *SupportedBy* and *InContextOf* connector can bear more information, the *Many* decorator on a connector indicates that when the pattern is instantiated, the connector can be multiplied n times (expressed in the label) based on the actual system information provided. The *Optional* decorator indicates that when the pattern is instantiated, the connector can connect to one or zero element. The *Choice* decorator on a connector can be used to denote possible alternatives in satisfying a relationship. It can represent 1-of-n and m-of-n selection.

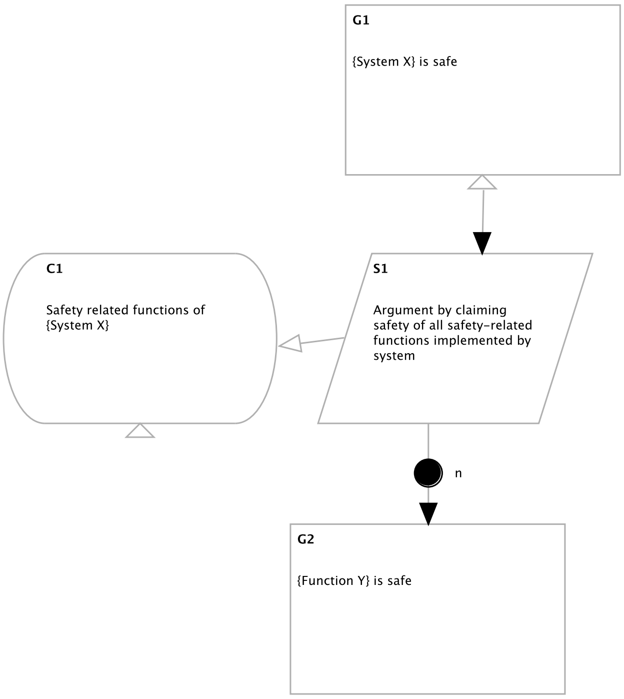

G1
{System X} is safe
C1
Safety related functions of {System X}
S1
Argument by claiming safety of all safety-related functions implemented by system
n
G2
{Function Y} is safe

Fig. 6: An example of a GSN Pattern [35]

Figure 6 shows an example of a GSN pattern (adopted from [35]). The contents in the curly brackets are *role*s in GSN terms, they are place holders which when the pattern is instantiated, will be replaced by actual system information. {System X} in G1 will be replaced with the actual name of the system when the pattern is instantiated. Similarly, the *SupportedBy* connector between S1 and G2 specifies that when the pattern is instantiated, there should be *n* (the number of safety-related functions implemented by the system) *Goal*s and *n SupportedBy* connected to S1. Pattern instantiation is often a manual process that involves comprehension of the GSN patterns and replacing the *role*s with actual system information. However, there has been work on automating the pattern instantiation process using MDE with the use of a *weaving model* [31].

In this section, we discussed briefly the elements provided by GSN. GSN is powerful in representing arguments in a structured way, which enables better comprehension of system safety arguments. GSN promotes modularity and abstraction in the sense that good practice in safety case construction can be captured and re-used. In the following sections, we will discuss SACM and its relationship with GSN.

4 Structured Assurance Case Metamodel

The *Structured Assurance Case Metamodel* (SACM) is standardised by the Object Management Group (OMG). The intention of the metamodel is to promote a model-based approach in the process of *System Assurance*, which is currently a manual approach that produces artefacts (i.e. Assurance Cases) that are (mostly) not model-based, where higher level operations (such as model query) on these artefacts are not enabled. SACM is created to support existing well established graphical argumentation notations, the *Goal Structuring Notation* (GSN) and *Claims-Arguments-Evidence* (CAE).

In this section, we discuss the packages in SACM and explain their intended usage. We provide two types of examples to demonstrate how SACM can be used: for simple concepts to explain without further context we will use in-place examples; and for complex concepts which requires the context of understanding the entirety of SACM we will use concrete examples, provided at the end of this section. Since OMG has not standardised the concrete syntax of SACM[2], we will use object diagrams in the examples.

4.1 SACM Overview

SACM is organised in five components, as illustrated in Figure 7. The *AssuranceCase* component captures the concepts of *Assurance Case* in system assurance. In SACM, an *AssuranceCase* package contains a number of *Argumentation* packages, *Terminology* packages and *Artifact* packages. The *AssuranceCase* component is discussed in Section 4.2. The *Base* component provides the foundation

[2] We are aware that concrete syntax for SACM is in development.

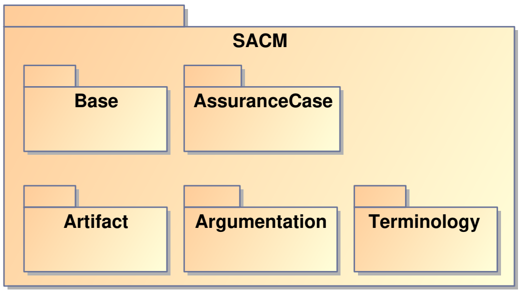


Fig. 7: Components of SACM

of SACM, which will be discussed in Section 4.3. The *Artifact* component captures the concepts used in providing evidence for the arguments made for system properties. The *Artifact* component is discussed in Section 4.4. The *Terminology* component captures the concepts used in expressing the arguments regarding system properties, which is discussed in Section 4.5. The *Argumentation* component captures the concepts used in arguing system properties (such as safety and/or security)[3], which is discussed in Section 4.6.

4.2 SACM AssuranceCase Component

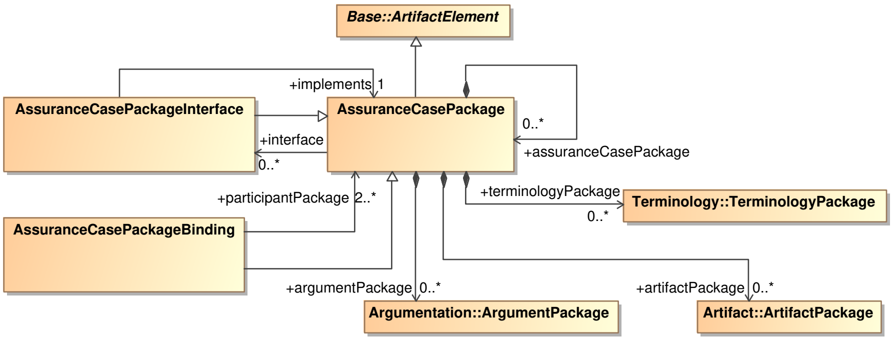


Fig. 8: SACM AssuranceCase Component

It is necessary to discuss the *AssuranceCase* component first as it provides an insight on how an *Assurance Case* in SACM is organised. The structure of the *AssuranceCase* component is shown in Figure 8.

[3] System properties refer to the safety and/or security in the context of this paper, hereafter.

The core element in the *AssuranceCase* component is the *AssuranceCasePackage* element, which extends the *ArtifactElement* in the *Base* component. The implication is that an *AssuranceCasePackage* can be considered to be an artefact. An *AssuranceCasePackage* can hold a number of *ArgumentPackage*s, *TerminologyPackage*s and *ArtifactPackage*s, which hold the argumentation with regards to system safety/security, the controlled vocabularies used in the argumentation, and the artifact that back the argumentation as evidence. In this way, SACM provides more detailed support for modularity than GSN[4].

Sometimes, the developer of an *AssuranceCasePackage* may want to make part of the *AssuranceCasePackage* available externally so that they can be reused. Consider the scenario where a system is composed of components A and B and *AsssuranceCasePackage*s ACPA and ACPB are created respectively for A and B (which contain structured argumentations with regard to safety and/or security properties for A and B). The developer may want to make parts of the argumentations public so that during system integration, where A and B are integrated to form a system, their assurance cases ACPA and ACPB can also be integrated to form a new *AssuranceCasePackage*. To disclose only necessary contents externally, one need to make use of the *AssuranceCasePackageInterface* to do so. The premise of system integration in safety related domains is to integrate assurance cases of systems to form an overall assurance case. SACM handles this scenario with the *AssuranceCasePackageBinding*, which binds two or more *AssuranceCasePackageInterface*s together to form an overall *AssuranceCasePackage*. This particular scenario is discussed in Section 5.4.

4.3 SACM Base Component

The *Base* component captures the fundamental concepts of SACM, the structure of the *Base* package is shown in Figure 9. The base element of all SACM elements is *Element*. Its direct children are *LangString*, *MultiLangString* and *SACMElement*.

LangString is used as an equivalence to *String* (the value of which is held in the *+content* feature), except it captures an additional feature *+lang* which allows the users to define what language is used in the *LangString*. *ExpressionLangString* is used to not only record a *String* in SACM, but also refer to its corresponding *Expression* organised in a *TerminologyPackage*. The usage of *ExpressionLangString* is discussed in Section 4.5 *MultiLangString*, as its name suggests, is used to express the same semantics using different languages. For example, to express ‘hazard’ in both English and German, the user can create a *MultiLangString* with two *LangString*, as shown in Figure 10.

The *MultiLangString* can then be associated to other SACM elements to denote the same meaning. What is more important than multiple natural language support is the support for computer languages[5]. As previously discussed, for open adaptive systems, system assurance needs to be performed at runtime.

[4] Feature **F1** in Section 2.3.

[5] Feature **F2** in Section 2.3.

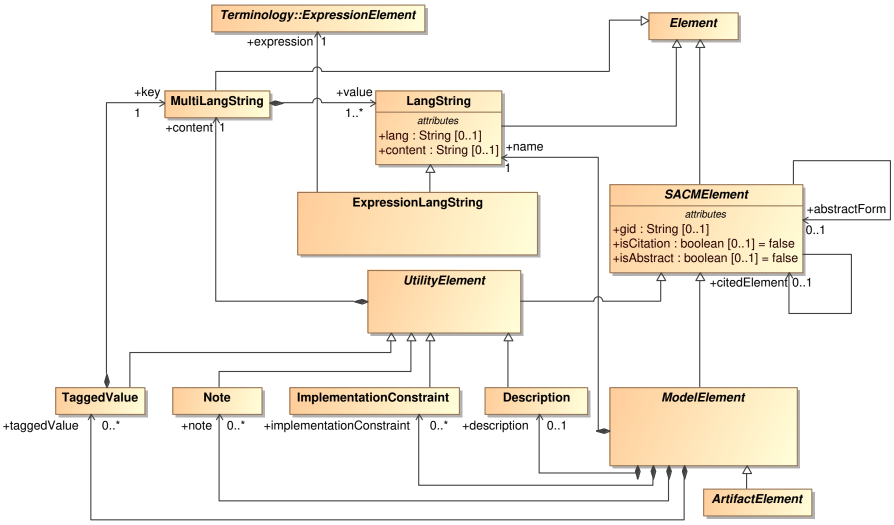


Fig. 9: SACM Base Component

Hence, automation is needed at runtime to reason about the safety of open adaptive systems. A first step towards this direction is the use of formal languages in assurance cases, so that system assurance can be (semi-) automated, in the sense that automated reasoning can be performed on the argumentation. In this case, *MultiLangString* can be used to hold both natural languages and computer languages (e.g. formal languages) to support automated reasoning of argumentations.

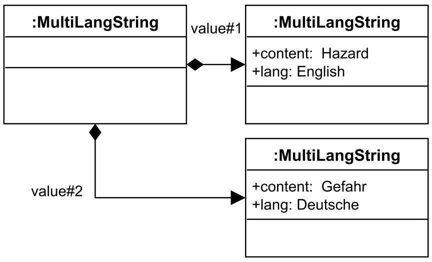


Fig. 10: Example *MultiLangString*.

SACMElement contains the foundational features of all SACM elements. *SACMElement* can record a *+gid* (*global identification*). *SACMElement* is also able to refer to (or 'cite') other *SACMElement*s, which is useful for implicit references, discussed in Section 5.1. The *+citedElement* and *+isCitation* properties are used for this purpose. A *SACMElement* can also be abstract, denoted by the *+isAbstract* property. A *SACMElement* can also be an *+abstractForm* of another, which is discussed in Section 5.3.

ModelElement further refines *SACMElement* which contains a *name* (of type *LangString*) and a set of *UtilityElements*. A *ModelElement* can contain a *Description* which describes its contents. Like previously mentioned, a *Description* can be expressed in any language via its usage of *MultiLangString*. A *ModelElement* can also contain an *ImplementationConstraint*, which is used to specify the instantiation rules for assurance case templates (that of similar to safety case patterns). A *ModelElement* can also contain a number of *Notes*, to hold additional information rather than descriptions. Finally, a *ModelElement* can also contain a number of *TaggedValues*, which are essentially {key, value} pairs. *TaggedValue* can be considered as an extension mechanism to allow the users to associate additional features to a *ModelElement* (other than the features modelled in the current version of SACM).

The *Base* component also defines the *ArtifactElement*, in the sense that all elements that extend *ArtifactElement* are considered to be *Artifacts*. The reason for this is discussed in Section 5.2.

In summary, the *Base* component lays the foundation for SACM, it provides facilities not only to express assurance cases (to be precise, assurance case models) in natural language, but also in computer languages. The *Base* component also provides a number of *UtilityElements* so that the user can use them to describe *ModelElements* as precisely as possible.

4.4 SACM Artifact Component

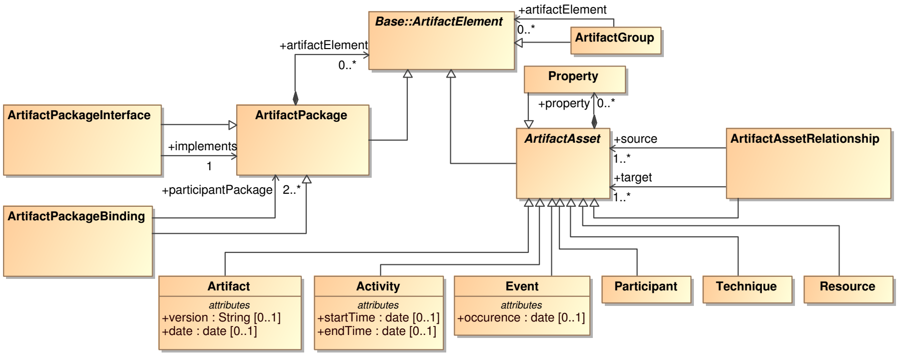


Fig. 11: SACM Artifact Component

Before delving into the *Argumentation* component, it is necessary to discuss the *Artifact* component. The structure of the *Artifact* component is shown in Figure 11. All elements in the *Artifact* component extend *ArtifactElement* in the *Base* component. *ArtifactElement*s are organised in *ArtifactPackage*s to promote modularity. Assurance case integration needs also be performed at the *ArtifactPackage* level. *ArtifactPackageInterface* and *ArtifactPackageBinding* are used for this purpose. *ArtifactGroup* is a new concept introduced in SACM 2.0. As *ArtifactPackage* can contain rather a large number of *ArtifactElement*s, the *ArtifactGroup* provides the user with a means to selectively group *ArtifactElement*s, so that the user can group/view *ArtifactElement*s with their defined criteria.

ArtifactAsset allows the users to create corresponding artefact elements in SACM, it can contain a number of *Property*-ies to hold user-defined properties. *Artifact* is used to record a piece of information (e.g. hazard log, failure logic models, etc). *Activity* is used to record an activity (e.g. specification of requirements). *Event* is used to record an event (e.g. creation/modification of *Artifacts*). *Participant* is used to record participants involved in *ArtifactAsset*s. *Technique* is used to record the techniques used in *Activities. Resource* is used to record a piece of resource, usually in the form of some electronic file. And finally, *ArtifactAssetRelationship* is used to link *ArtifactAsset*s (e.g. connecting *Activity* to *Participants*). Note that the *ArtifactAssetRelationship* is a generic relationship, however, the user can choose to use *Property* to specify the purpose of an *ArtifactAssetRelationship.*

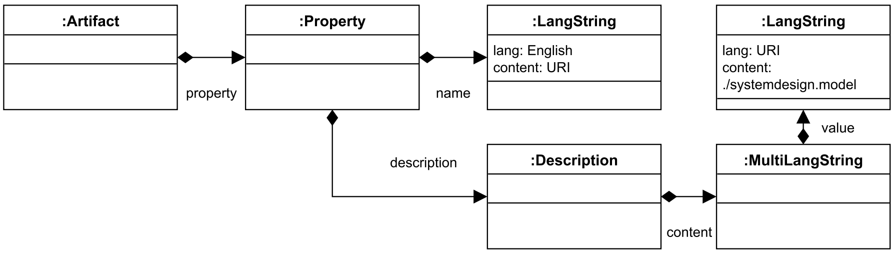


Fig. 12: External reference using *Property.*

One open question regarding the *Artifact* component is how to refer to external materials (such as system requirement, system design model, system failure logic model, etc.). To achieve automation, it is necessary to have a means to refer to external materials (especially models) so that all information can be gathered together to form an assurance case. The user may make use of the *Property* element, as shown in Figure 12, where a *Property* is associated to an *Artifact*[6], which has a *name* (of type *LangString*): 'URI' and a *Description*,

[6] The *Artifact* should have its own features such as name and description, which is omitted here.

which in turn specifies a file on local disk (systemdesign.model). In this way, the *Property* element is used as a reference to a local file.

SACM does not restrict how external materials should be referenced, the description provided above is one way of achieving it. It also depends on tool implementations on how external references should be handled. It is possible to have finer grained reference to (a collection) of model elements for a model (e.g. a Fault Tree Analysis model). However, OMG has not standardised how this can be done.

4.5 SACM Terminology Component

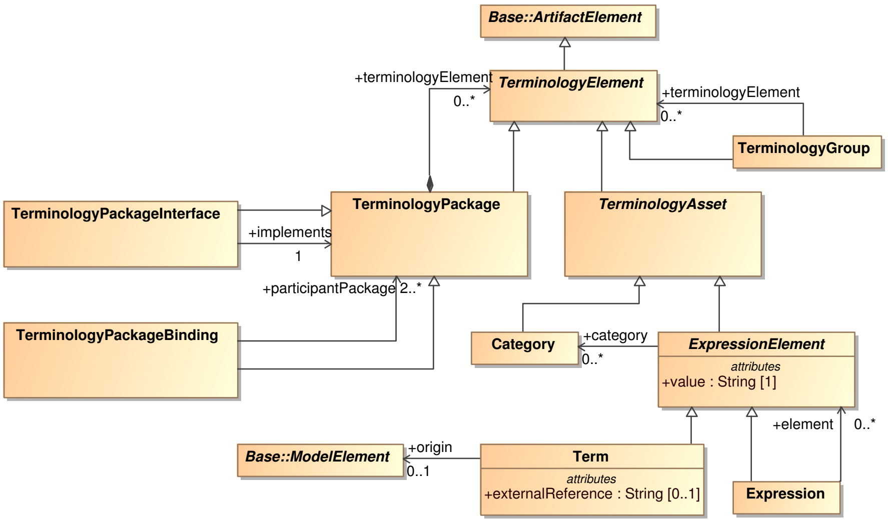


Fig. 13: SACM Terminology Component

The *Terminology* component captures concepts that enable the users of SACM to define controlled vocabularies[7] to describe their argumentations in greater precision. The structure of the *Terminology* package is shown in Figure 13.

The root element of the *Terminology* component is *TerminologyElement* which also extends *ArtifactElement* in the *Base* package. *TerminologyAsset*s are organised in *TerminologyPackage*s. *TerminologyInterface* and *TerminologyPackageBinding* are used for assurance case integration. *TerminologyGroup* enables to users to group *TerminologyAsset*s selectively.

[7] Feature **F3** in Section 2.3.

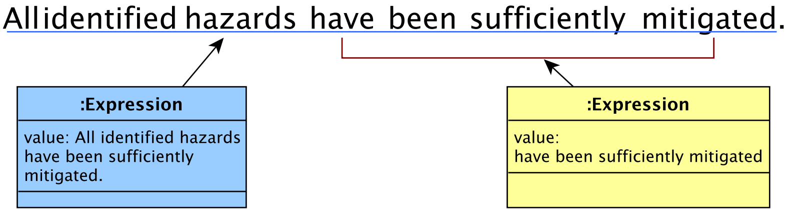

All identified hazards have been sufficiently mitigated.
:Expression
value: All identified hazards have been sufficiently mitigated.
:Expression
value:
have been sufficiently mitigated

Fig. 14: Example use of *Expression*.

ExpressionElement captures the concept of two kinds of expressions used in argumentations. *ExpressionElement* holds a *value* (of type String). *ExpressionElement* can either be an *Expression* or a *Term*. An *Expression* is used to model phrases and sentences. For example, when stating a sentence: **All identified hazards have been sufficiently mitigated.**, the *Expression* element can be used to to capture this sentence as shown in Figure 14 (element rendered in cyan). Alternatively, the users of SACM may choose to capture arbitrary phrases in the sentence as an *Expression*, as shown in Figure 14 (element rendered in yellow). The idea behind *Expression* is to store phrases/sentences in the *TerminologyPackage* like a set of controlled vocabularies, so that the *Expressions* can be re-used. Of course the example in Figure 14 does not add too much value to the assurance case, in realistic scenarios, only phrases that can be re-used should be captured in *Expressions*. This completely on the use cases and tool implementers of SACM.

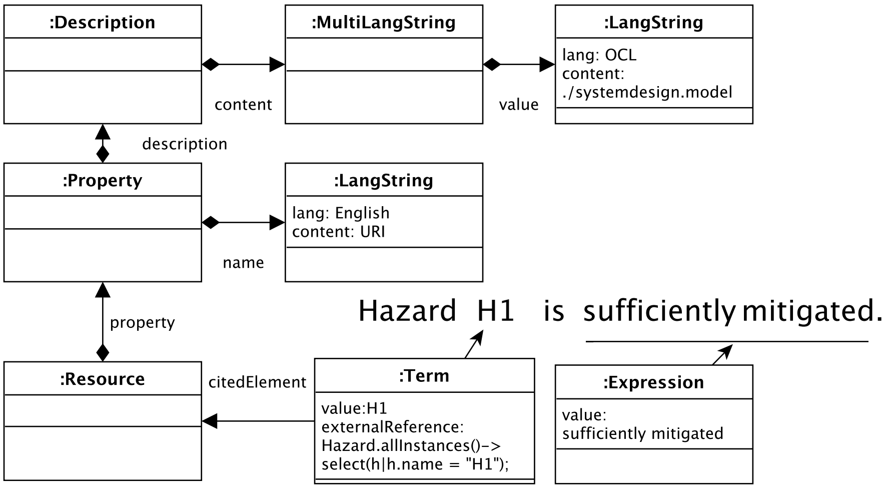


Fig. 15: Example use of *Expression* and *Term*.

The *Term* element is used to capture a terminology used in the process of system assurance. For example, in Figure 15 (first consider elements rendered in white), a statement is made: **Hazard <u>H1</u> is sufficiently mitigated.** Within the sentence, <u>H1</u> may cross-references to an identified hazard in a hazard log. Thus, <u>H1</u> can be captured using a *Term*. The location of the hazard log can be captured using a *Resource*[8] in the *Artifact* component, where the location of the *Resource* is captured using *Property*. In order to point to a specific model element (in this case, model element <u>H1</u> in the hazardLog.model), the user may use OCL in the *+externalReference* feature of the *Term*. This is illustrated in Figure 15. Again, SACM does not restrict how external references should be handled, the description provided above is one way of achieving it. It also depends on tool implementations on how external references are handled. Sometimes, a term may refer to a *ModelElement* within the assurance case (via the *+origin* feature). This will be discussed in Section 4.6.

The users of SACM may also organise *ExpressionElement* into *Categories*. For example, the users may create a *Category* of Hazard, and associate the *Term* <u>H1</u> to it.

4.6 SACM Argumentation Component

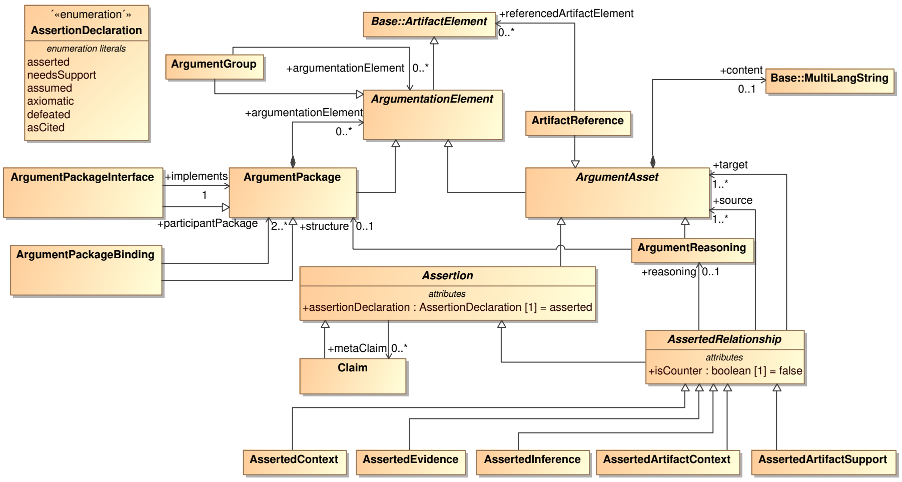


Fig. 16: SACM Argumentation Component

The *Argumentation* component captures the concepts required to model structured arguments regarding system properties. The structure of the *Argumentation* component is shown in Figure 16. The root element of the *Argumentation* component is the *ArgumentationElement*, which is a direct child of

[8] The details of the *Resource* is omitted.

ArtifactElement in the *Base* component. This implies that all elements in the *Argumentation* package are also considered to be artifacts.

ArgumentationPackage can contain a number of *ArgumentationElement*, which can either be *ArgumentPackage*s (and their children) or *ArgumentAsset*s. *ArgumentAsset* can store a *content*, as discussed in Section 4.3, the content can be in any language[9].

ArgumentAsset and its children are the elements that form the structured argumentation in the *Argumentation* package. An *Assertion* has an *AssertionDeclaration* to distinguish different kinds of *Assertion*s. The *AssertionDeclaration*s are as follows:

- **asserted** - the default declaration, means that the *Assertion* is made and is supported by evidence;
- **needsSupport** - a flag indicating that the *Assertion* is not supported yet (needs further development);
- **assumed** - a flag indicating that the truth of the *Assertion* is assumed although no supporting evidence is provided;
- **axiomatic** - a flag indicating that the truth of the *Assertion* is axiomatically true without further supporting evidence;
- **defeated** - a flag indicating that the truth of the *Assertion* is invalidated by a counter-evidence and/or argumentation;
- **asCited** - when an *Assertion* 'cites' another *Assertion* (via the +citedElement property in *SACMElement*), the truth of the *Assertion* is transitively derived from the value of the cited *Assertion.*

The use of *AssertionDeclaration* will be illustrated in Section 5.1.

Sometimes it is necessary to argue the confidence (level of trust) in the arguments provided in the assurance case[10]. This can be achieved using the *+metaClaim* feature of the *Assertion* element. *+metaClaim* as its name suggests, is a *Claim* about an *Assertion* to argue the soundness/trustworthiness of the *Assertion.* Within the meta *Claim*, one may write 'full confidence in Claim C1 is achieved via...' in its description, within which the *Term* G1 refers to another *Claim* in the same SACM model. This is where the *+origin* feature of the *Term* is used, in the sense that the *Term* G1 refers to a *Claim* G1 within the same SACM model.

Claim and *AssertedRelationship* are the core elements of the *Argumentation* package. *AssertedRelationship* are used to connect *ArgumentAsset*s to form structured argumentation (*AssertedRelationship*s can also be *counter* relationships, indicated by the *+isCounter* feature to present counter-arguments[11]):

- **AssertedContext** - this relationship is used to connect contextual *Assertion*s to an *Assertion*;

[9] Via the usage of *MultiLangString*

[10] Feature **F4** in Section 2.3.

[11] Feature **F5** in Section 2.3.

- **AssertedEvidence** - this relationship is used to connect evidence (referenced via *ArtifactReference*) to an *Assertion*;
- **AssertedInference** - this relationship records the inference between on or more *Assertion* and another *Assertion*;
- **AssertedArtefactContext** - this relationship is used to connect contextual artefacts (via *ArtifactReference*) to an artefact (via *ArtifactReference*);
- **AssertedArtefactSupport** - this relationship is used to connect supporting artefacts (via *ArtifactReference*) to an artefact (via *ArtifactReference*).

ArtifactReference is a type of *ArgumentAsset*, which is able to refer to an *ArtifactElement*. *ArtifactReference* is typically used to refer to evidence stored in an *Artifact* package. In addition, it can refer to any element that extends *ArtifactElement* (all elements in the *AssuranceCase*, *Argumentation*, *Artifact* and *Terminology* packages are *ArtifactElement*s)[12].

ArgumentReasoning is also a type of *ArgumentAsset*, it is used to provide a explanatory description for an *AssertedRelationship*. A detailed discussion of *ArgumentReasoning* is in Section 5.2.

4.7 Summary

In this section, we discussed the components provided by SACM. We explained the intended semantics of the elements in the packages and we used some in-place examples to illustrate how SACM can be used to construct a system assurance case with argumentations regarding system properties (i.e. safety and/or security) and its supporting evidence/artefact (using the *Artifact* and *Terminology* packages of SACM). In the next section, we will illustrate how SACM can be used with more concrete examples.

5 SACM: Examples

In this section, we provide concrete examples on how SACM is used to construct structured argumentation, and how SACM can be used to form argumentation patterns (similar to GSN argument patterns). Since GSN notations are widely accepted and understood, we compare GSN depictions with their equivalent model elements in SACM to illustrate how SACM elements can be used to denote the same meaning carried by their GSN counterparts.

5.1 Example: Making Claims and Citations

Figure 17 provides an example on how to construct a *Claim*. A *Claim* can have a name and a description, captured by *LangString* and *Description* respectively. A *Claim* is **asserted** if it is supported by other *Claim*s. In this example, it is connected by an *AssertedInference* with another *Claim* (details ommitted). The

[12] Feature **F6** in Section 2.3.

equivalent of *Claim* C1 in Figure 17 in GSN is provided in Figure 18, where the *Goal* G1 bears the same semantics with C1 (and how *Goal* G1 is supported by another *Goal*[13]).

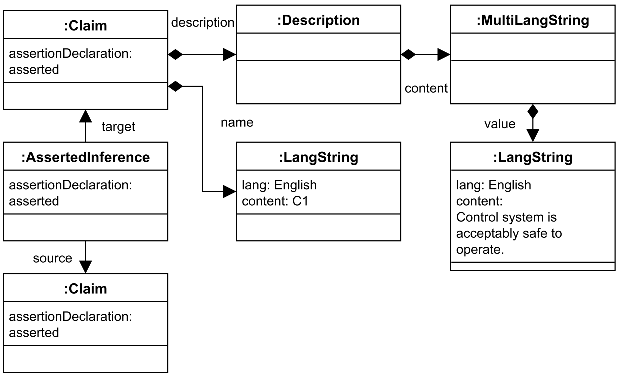


Fig. 17: An **asserted** *Claim* in SACM.

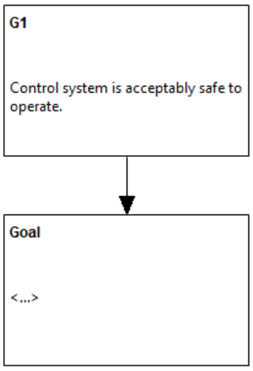


Fig. 18: A *Goal* structure in GSN.

Like previously mentioned, a *Claim* can be marked as **needsSupport**. Figure 19 illustrates the use of this declaration. A *Claim* that is not supported by any argument/evidence should be marked as **needsSupport**. The equivalence of this *Claim* in GSN is shown in Figure 20, where *Goal* G1 bears the same semantics of *Claim* C1. When a *Goal* is not supported by argument/evidence, it is marked as *undeveloped*.

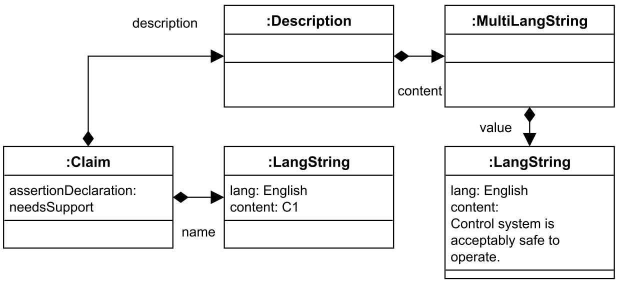


Fig. 19: A **needsSupport** *Claim* in SACM.

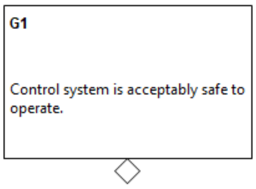


Fig. 20: A undeveloped *Goal* in GSN.

A *Claim* marked as **assumed** is used to state an assumption in the argumentation. Figure 21 illustrates an **assumed** *Claim* in SACM (where the user

[13] Details of subsequent *Goals* are omitted.

assumes that *all hazards have been identified*). The users are responsible to declare that a *Claim* is **assumed** when they make assumptions about the system. The equivalence for **assumed** *Claim* is an *Assumption* in GSN, as shown in Figure 22.

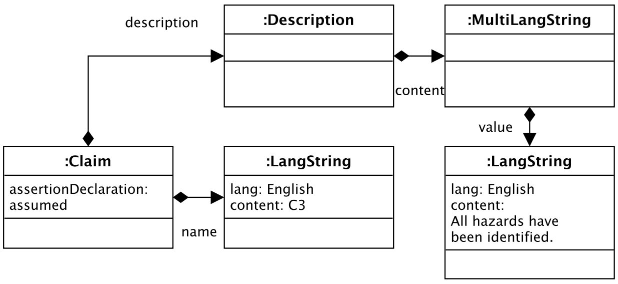


Fig. 21: An **assumed** *Claim* in SACM.

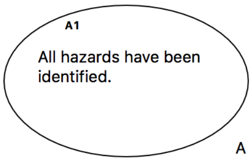


Fig. 22: An *Assumption* in GSN.

A *Claim* marked as **axiomatic** is used to state a well agreed fact (presumably among stakeholders), which does not need any further support by arguments/evidence. The equivalent of an **axiomatic** *Claim* in GSN is a *Justification*, which does not need further supporting argument/evidence to support its content. Figure 23 and Figure 24 illustrates the use of **axiomatic** *Claim* and *Justification* respectively.

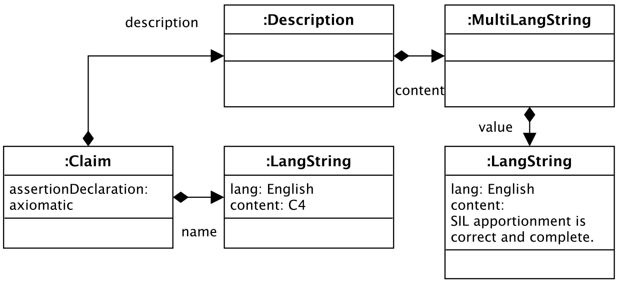


Fig. 23: An **axiomatic** *Claim* in SACM.

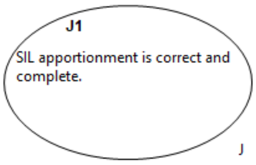


Fig. 24: A *Justification* in GSN.

A *Claim* is marked as *defeated* if the truth of the claim is proven to be false by supporting argument/evidence. Figure 25 provides an example of a **defeated** *Claim*, the *Claim* C1 is connected with another *Claim* (we do not consider the detail of this *Claim* for this example) by an *AssertedInference*, but with its *+isCounter* feature to be *true*. This means that the *AssertedInference* is

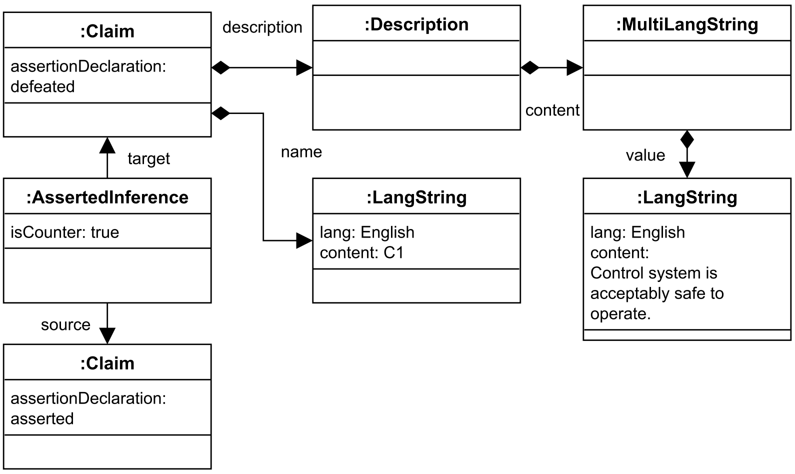


Fig. 25: A **defeated** *Claim.*

a counter-argument, which negates the truth of *Claim*, if the *+source* of the *AssertedInference* is *true*. There is no equivalence for *defeated Claims* in GSN.

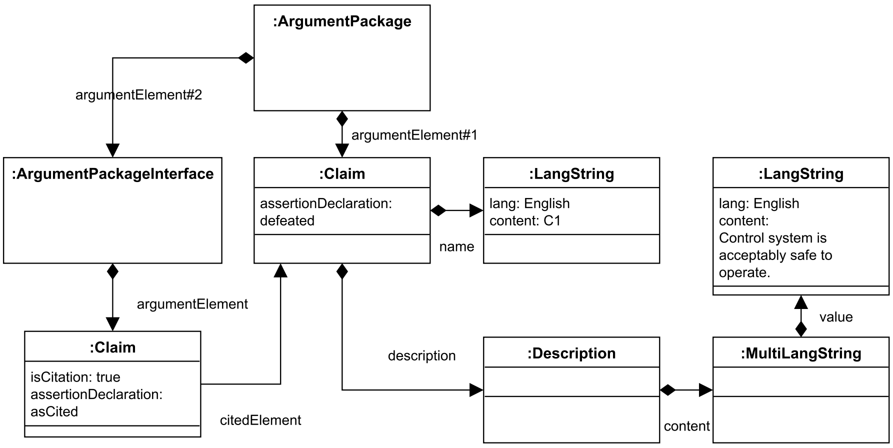


Fig. 26: An **asCited** *Claim.*

As previously discussed, the users of SACM are able to selectively disclose the content of an *ArgumentPackage* via the use of *ArgumentInterface*. Figure 26 illustrates how an *ArgumentPackageInterface* can be used. In this example, a *Claim* C1 is held within an *ArgumentPackage*, which contains an *Argument-*

PackageInterface that in turn contains another *Claim*, which is a citation (its *+isCitation* is true). The *Claim* ‘cites’ C1 in the *ArgumentPackage* via the *+citedElement* feature (which is defined in the *SACMElement* element in the *Base* package). Hence, the *asCited* declaration on the *Claim* should be used, which means that the truth of the *Claim* as a citation, depends on the *Claim* that it cites (in this case, *Claim* C1). There is no equivalence of *cited Claim* in GSN.

The *+isCitation* and *+citedElement* features can be used in the same way in *ArtifactPackageInterface* and *TerminologyInterface*, should the developers of assurance cases wish to disclose information in their corresponding packages.

5.2 Example: AssertedRelationships and ArgumentReasoning

The intent of *Claims* are distinguished using *AssertionDeclarations*. Different types of *Claims* are connected using different *AssertedRelationships* to form a structured argumentation.

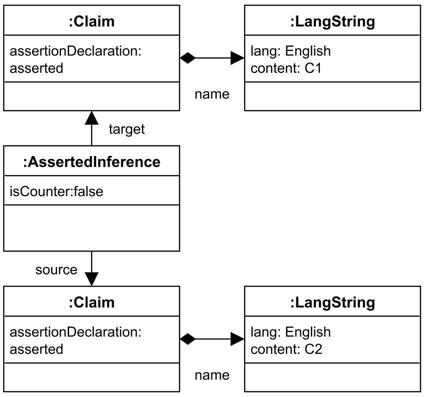


Fig. 27: An example of **AssertedInference**.

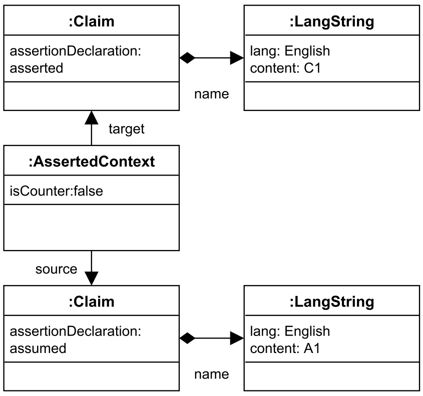


Fig. 28: An example of **AssertedContext**.

The *AssertedInference* denotes the inference between one or more *Assertions* and another *Assertion*, Figure 27 provides an example, where the truth of *Claim* C1[14] is inferred from the truth of Claim C2.

The *AssertedContext* connects contextual *Assertions* to an *Assertion*, Figure 28 provides an example, where the *Claim* A1 provides contextual information for *Claim* C1.

The *AssertedEvidence* connects evidence to an *Assertion*, Figure 29 provides an example, where the *ArtifactReference* S1 (which refers to an *Artifact* organised in an *ArtifactPackage*, details omitted) provides evidence for the *Claim* C1.

[14] Description details are omitted to make the model clearer.

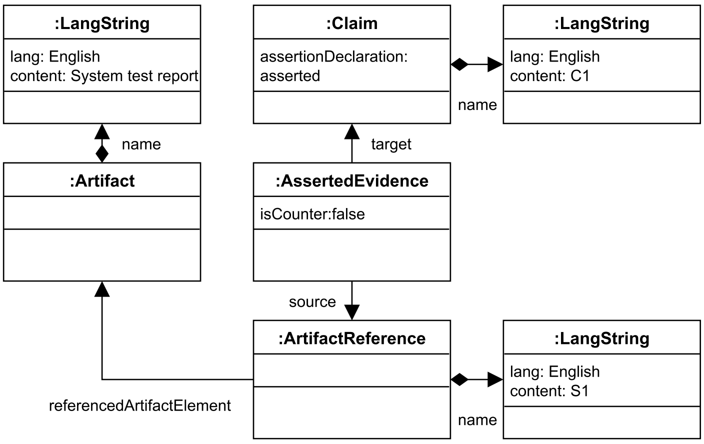


Fig. 29: An example of **AssertedEvidence**.

The *AssertedArtifactSupport* connects supporting artefacts to an artefact, Figure 30 provides an example, where *ArtifactReference* S2 (which cites an *Artifact System test plan*) supports the *ArtifactReference* S1 (which cites the *Artifact System test report*), in the sense that the system test plan supports the system text report. The *AssertedArtifactContext* is also used on *ArtifactReferences*, except that it connects contextual artefacts to another artefact.

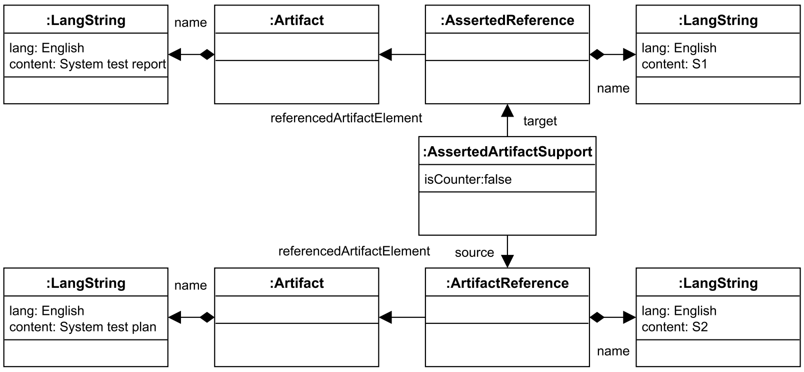


Fig. 30: An example of **AssertedArtifactSupport**.

As previously mentioned, all elements that extend *ArtifactElement* can be considered an artifact, this is due to the fact that there are *supporting* and *contextual* relationships between elements rather than those in the *Artefact* package.

For example, an assurance case developer may want to express that one *ArgumentPackage* bears supporting arguments to another *ArgumentPackage*, in this case, the user may want to use *AssertedArtifactSupport* to connect two *ArtifactReference* which reference these two *ArgumentPackages*.

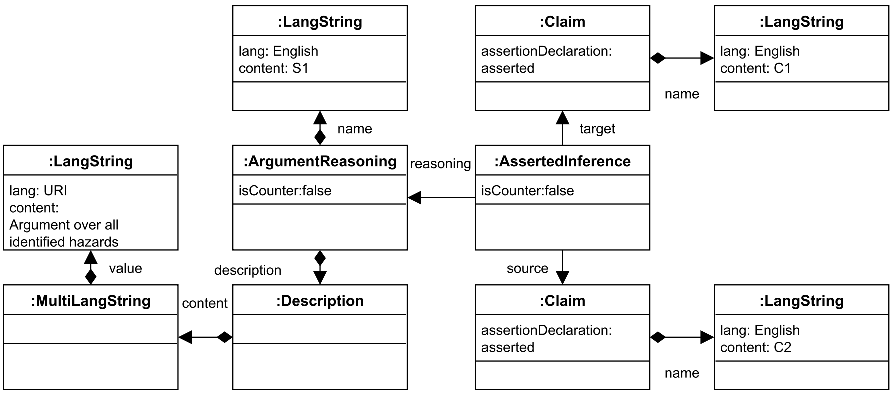


Fig. 31: An example of **ArgumentReasoning**.

Sometimes, a reason of *AssertedRelationships* can be attached to further clarify the reasons of the *AssertedRelationship*, then *ArgumentReasoning* is used for this purpose. Figure 31 illustrates the use of *ArgumentReasoning*, where top level *Claim* C1 is backed by its sub-*Claim* C2, which are connected by an *AssertedInference*, the user may choose to give the *AssertedInference* a reason with the use of *ArgumentReasoning*. In this case, an *ArgumentReasoning* S1 is attached to the *AssertedInference*, which states that the argument is made by *Argument over all identified hazards*. Supportive and contextual information can also be associated to *ArgumentReasoning* via the use of *AssertedSupport* and *AssertedContext*. Since an *ArgumentReasoning* is not an *Assertion*, there is little value in arguing the soundness of the *ArgumentReasoning*.

5.3 Example: Argumentation Patterns

In GSN, there is a concept of *GSN Patterns* [35], which are essentially successful GSN safety case templates that can be re-used. SACM provides similar concepts so that the user can construct patterns in SACM, and at a later stage, to *instantiate* the patterns by populating actual system information in the patterns. Figure 32 shows an argumentation pattern in SACM. For comparison, the content of the pattern is identical to the GSN pattern shown in Figure 6. Note that some details are omitted due to the complexity of the SACM model.

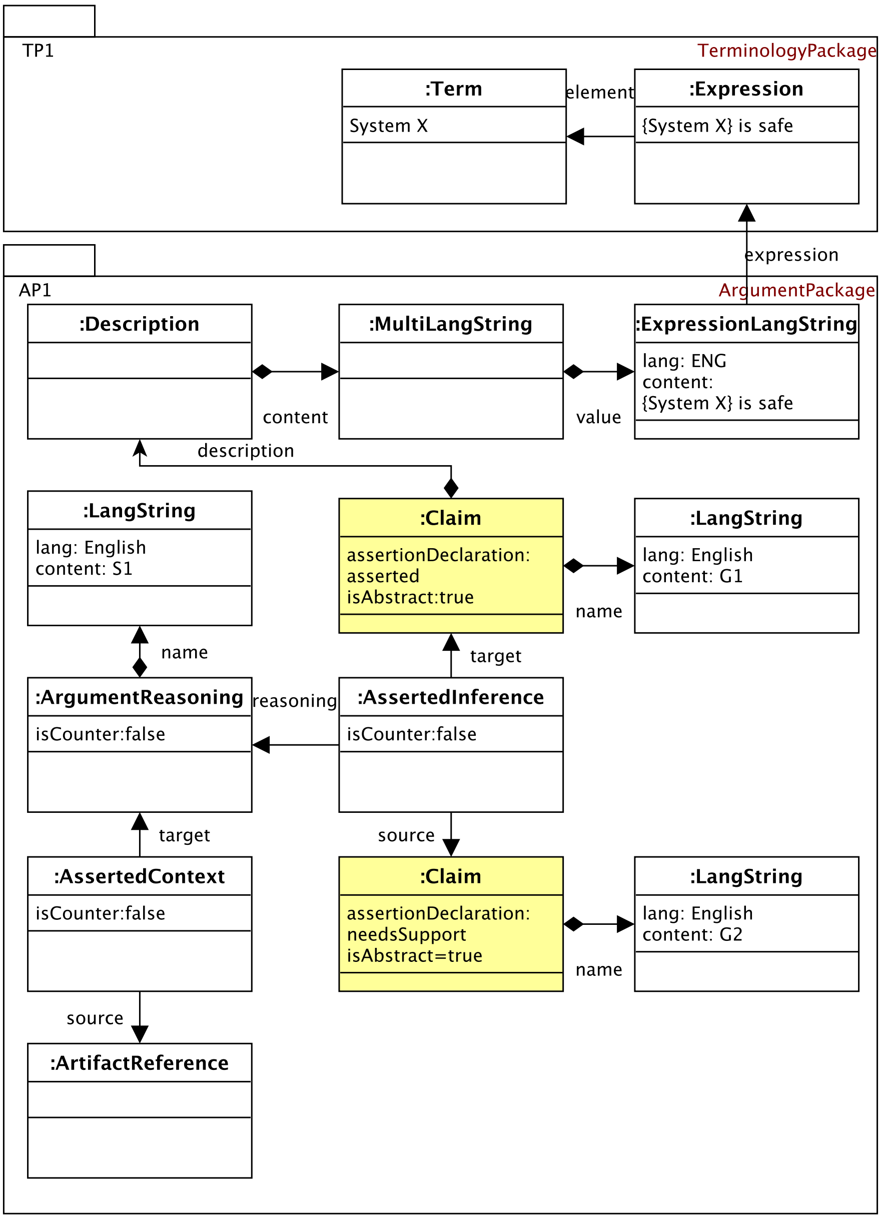


Fig. 32: An example SACM argumentation pattern.

There is a *TerminologyPackage* in the upper part of Figure 32 named TP1, within which contains an *Expression* and a *Term*[15]. In the lower part of the figure, there is an *ArgumentPackage* AP1, which contains the structured argumentation pattern (N.B. the packages are placed in this way to make the figure more readable and does not denote the priority of the packages).

The top level *Claim* of this pattern is G1, which maps to the *Goal* G1 in Figure 6. Note that G1's *+isAbstract* is set to *true* since this is an abstract *Claim* (a template). Now we focus on the *description* of G1, within which an *ExpressionLangString* is used. The *ExpressionLangString* refers to the *Expression* in *TerminologyPackage* TP1 (which contains value: {*System X*} *is safe*). The curly brackets are a convention in GSN, which are called *role*s in GSN. *Role*s are essentially place holders in the pattern, the contents enclosed in the curly brackets will be replaced by actual system information if the pattern gets instantiated. In this case, when the pattern is instantiated, {System X} will be replaced with the name(s) of actual system(s).

In the *TerminologyPackage* TP1, the *Expression* refers to the *Term* *System X*. This is a typical use of *Term* in structured argumentation. When the pattern is *instantiated*, the *Term* should have its *+externalReference* feature populated to import actual system information.

The structure of the argumentation in Figure 32 follows that in Figure 6, where the *Claim* G2 maps to *Goal* G2, and the *ArgumentReasoning* S1 maps to *Strategy* S1. Note that the *Claim*s rendered in yellow are the templates, which have their *+isAbstract* feature set to true.

GSN pattern instantiation is often a manual procedure as safety case developers need to comprehend the GSN pattern and replace the *role*s in the pattern with actual system information. In [31], a model-based approach is proposed, which makes use of a *weaving model* to link the elements in the GSN with elements in system models. This is typically due to the fact that in GSN there are no means to specify instantiation rules for GSN patterns. In SACM, as previously mentioned in Section 4.3, the element *ImplementationConstraint* can be used to specify the instantiation rules for patterns[16]. The users of SACM can associate *ImplementationConstraint* to any element and the use of language is not restricted. Evidently, pattern instantiation needs tool support and an model management engine to execute the *ImplementationConstraint*s. The language used in *ImplementationConstraint* are not limited to computer languages. *ImplementationConstraint*s described in natural languages can also be used as instantiation rules, except that the instantiation procedure is limited to manual only.

When a SACM model instantiates a SACM pattern, it can relate to the SACM pattern via the *+abstractForm* feature. For example, if a *Claim* C1 created by instantiating the template G1, it can refer to G1 via the *+abstractForm* for future reference.

[15] The containment is described in this way to improve comprehensibility.

[16] Feature **F7** in Section 2.3.

5.4 Example: A case study on the European Train Control Systems (ETCS)

As previously discussed, in SACM, assurance cases can be integrated to form an overall assurance case. Integrating assurance cases is a typical task performed when system components (or sub-systems) are integrated to form an overall system. Sometimes, system components/sub-systems are developed independently, together with their assurance cases. Hence, in safety-related domains, the premise of system integration is integration of assurance cases of components/sub-systems, to argue the dependability of the to-be-integrated system.

To illustrate how assurance case integration is performed in SACM. We provide an example[17] taken from an engineering story in European Train Control System (ETCS) we encountered in the research carried out in the DEIS project [33]. In this example, we consider the scenario where the assurance cases of on-board and side-track components of ETCS are integrated to form an overall assurance case.

The example SACM model for the ETCS case study is shown in Figure 33. For simplicity, we only show the top level *Claim*s of the assurance cases. In Figure 33, there are three *AssuranceCasePackage*s. On-Board ACP (at the top of the figure) is the assurance case for the on-board component of ETCS, Track-Side ACP (at the bottom of the figure) is the assurance case for the side-track component of ETCS, and Integration ACP (in the middle of the figure) is the integration assurance cases which integrates the two component assurance cases together.

For On-Board ACP, *ArgumentPackage* AP1 contains the argument regarding the safety of the on-board component. As discussed in Section 4.2, system engineers may wish only to disclose the top level *Claim* externally, hence *ArgumentPackageInterface* AP1 is used which contains a citation of G2 which will be referenced externally. Same principle is applied to Track-Side ACP, where the top level *Claim* G3 is cited in the *ArgumentPackageInterface* API2. It is to be noted that both On-Board ACP and Track-Side ACP contains their *ArtifactPackage*s and *TerminologyPackage*s, which are not shown due to the complexity of the model structure.

To integrate On-Board ACP and Track-Side ACP, *AssuranceCasePackage* named Integration ACP is created. Integration ACP contains an *AssuranceCasePackageBinding* (Integration ACPB) specifically to bind On-Board ACP with Track-Side ACP. Within Integration ACPB, an *ArgumentPackageBinding* (APB1) is used to bind API1 and API2 via the *+participantPackage* feature. In APB1, top level *Claim* G1 argues the safety of ETCS and two supporting *Claim*s G2 and G3 are in place. Note that G2 and G3 are citation *Claim*s which cites G2 in API1 (which in turn cites G2 in AP1) and G3 in API2 (which in turn cites G3 in AP2). It is also to be noted that within the *AssuranceCasePackageBinding* (Integration ACPB), there are also *ArtifactPackageBinding*s and

[17] We reduced the complexity of the model structure by incorporating the name and description of the elements directly in the elements themselves.

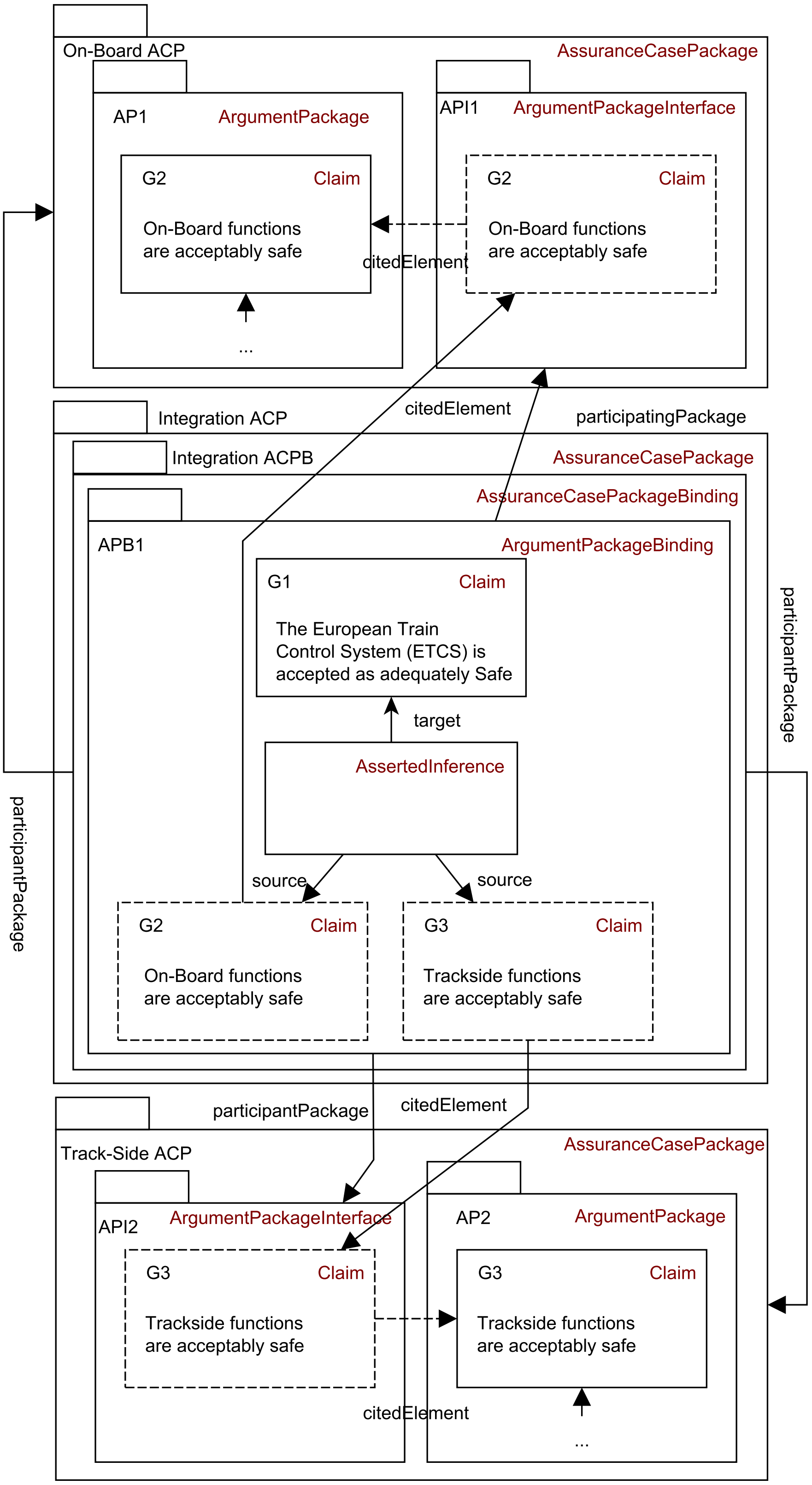


Fig. 33: A case study on the assurance case integration of the European Train Control Systems (ETCS).

*TerminologyPackageBinding*s which binds the artifacts and expressions used in the on board component and track side component.

The integration of assurance cases in SACM is achieved using various package bindings. It is also possible to include additional arguments in the binding *AssuranceCasePackage* when deemed necessary. Users of SACM may also argue the trustworthiness of the cited *Claim*s in other packages to ensure the confidence in citing argument elements.

This example is kept simple to illustrate that the users of SACM are able to integrate assurance cases using the facilities provided. It is to be noted that assurance case composition is a complex task, for there may be subtle dependencies among the systems. Modular assurance case construction makes no assumption that the safety/security of the whole system is guaranteed by a composition of arguments about the safety/security of the parts. It remains the responsibility of the assurance case developers to ensure that each assurance case module correctly identifies its dependencies on other assurance cases that must be satisfied in order to assure the composed system.

6 Metamodel for existing notations and the transformations from them to SACM

SACM is designed to support existing safety notations such as GSN and CAE. In previous sections, we briefly demonstrated the usage of SACM elements by comparing them with GSN notations. In this section, we provide a GSN metamodel and a CAE metamodel that are compliant to SACM. We also discuss the transformation from GSN and CAE to SACM.

6.1 The GSN Metamodel and the inteoperability from GSN to SACM

As previously discussed, SACM provides a richer set of features compared to GSN, which includes the ability to standardise evidential and informational artifacts in the models, the ability to standardise controlled vocabularies (expressions and terminologies), as well as modular organisation and integration of artifacts and terminologies. In general, creating a metamodel for GSN is a simple task, for there are only several concepts that GSN captures. However, it is ideal to create the GSN metamodel by extending SACM elements, so that not only can the GSN metamodel inherit features provided by SACM, but it also makes the interoperability from GSN to SACM easier.

Our version of the GSN metamodel is shown in Figure 34[18]. In GSN, argumentations are organised in *Module*s, which is made as a sub-type of *ArgumentPackage* in SACM; *ContractMoudle* are essentially contracts that bind *Module*s together, thus it is a sub-type of *ArgumentPackageBinding*.

Elements *Goal*, *Assumption*, *Justification* and *AwayGoal* are made sub-types of *Claim* in SACM. A *Goal* can be *uninstantiated*, which basically means it is

[18] All GSN elements are rendered in blue for readability.

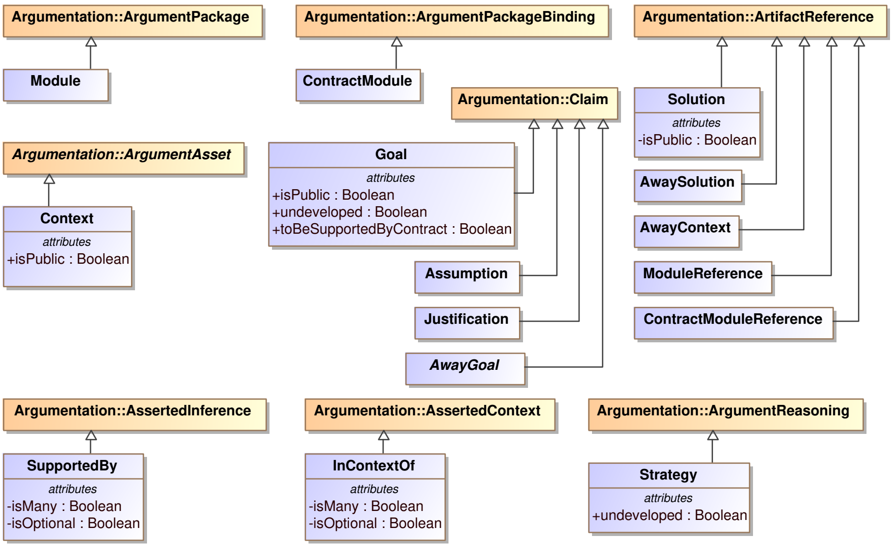


Fig. 34: SACM compliant GSN metamodel.

abstract, this is captured by the *+isAbstract* feature in SACM's *SACMElement* class. A *Goal* can be *public*, which is deprecated in SACM, a *Goal* can also be *undeveloped* and *toBeSupportedByContract*, which are captured individually.

Elements *Solution*, *AwaySolution*, *AwayContext*, *ModuleReference* and *ContractModuleReference* are sub-types of *ArtifactReference* in SACM as they refer to artefacts that contain information they represent. *Context* is a slightly complicated concept, as it can either be a statement stating the context of a *Claim*, or it can refer to contextual information stored in an artefact. Thus, *Context* is made a sub-type of *ArgumentAsset*.

SupportedBy is made a sub-type of *AssertedInference* and *InContextOf* is made a sub-type of *AssertedContext*. *Strategy* is made a sub-type of *ArgumentReasoning* for it explains the intention of an *AssertedRelationship*.

The way that the GSN metamodel is created makes it inherently compatible with SACM, so that it can be used as an *ArgumentPackage*, organised in an *AssuranceCasePackage*, with its supporting artefacts organised in *ArtifactPackage*s and controlled vocabularies organised in *TerminologyPckage*s.

To enable interoperability from GSN to SACM, we provide a model-to-model transformation[19] defined using the Epsilon Transformation Language (ETL) [36]. Most of the transformation is straight forward - instances of the GSN elements are transformed into instances of the SACM elements that the GSN elements extend. There is one exception: the transformation from *Strategy* to *ArgumentReasoning*. *Strategy* in GSN is a node in the structured argumentation, where

[19] Available at: https://github.com/wrwei/MDERE/blob/master/technical

ArgumentReasoning is a node associated to an *AssertedRelationship* (which is an edge). Hence, additional analysis is needed to perform the GSN2SACM transformation. This requires analysis to be performed during the transformation, which is shown in Algorithm 1.

Algorithm 1: Transforming Rule Strategy2ArgumentReasoning

```
 1 let strategy = Strategy to be transformed;
 2 let argumentReasoning = new ArgumentReasoning;
 3 argumentReasoning.name = strategy.name.equivalent();
 4 argumentReasoning.description = strategy.description.equivalent();
 5 if strategy.uninstantiated == true then
 6 |   argumentReasoning.isAbstract = true;
 7 end
 8 let incomingSupportedBy = the incoming SupportedBy to strategy;
 9 let outgoingSupportedBys = all outgoing SupportedBys from strategy;
10 let supportedByFromGoal = Goal from incomingSupportedBy
11 let supportedByToGoals = all Goals from outgoingSupportedBys that connects
     to strategy;
12 if supportedByToGoal is not empty then
13 |   let assertedInference = new AssertedInference;
14 |   assertedInference.target = supportedByFromGoal.equivalent();
15 |   for goal in supportedByToGoals do
16 |   |   assertedInference.source.add(goal.equivalent());
17 |   end
18 end
19
```

Algorithm 1 defines the transformation rule *Strategy2ArgumentReasong*. Line 1 retrieves the *Strategy* to be transformed; Line 2 creates a new *ArgumentReasoning*; Line 3 and 4 transform the *name* and *description* of the *Strategy* to the *ArgumentReasoning* (the equivalent() operation in ETL divert control to corresponding transformation rules that transform *Name* to *Name*, and returns the transformed model element). Line 5 checks if the *Strategy* is uninstantiated, and then in line 6, make the *ArgumentReasoning abstract*. Line 8 retrieves the incoming *SupportedBy* for the *Strategy*. Line 9 retrieves all outgoing *SupportedBy* for the *Strategy* (In GSN, the flow of an argument goes from top to bottom. Thus, for a *Strategy*, there is one incoming *SupportedBy* and many outgoing *SupportedBy*s). Line 10 retrieves the *Goal* from the incoming *SupportedBy*. Line 11 retrieves the *Goal*s from the outgoing *SupportedBy*s. Then, in line 13 an *AssertedInference* is created. It is worth noting that in SACM, the inference flows from bottom to top. Thus the source and target of the *AssertedInference* are the transformed counterparts of *supportedByToGoals* in line 11 and the transformed counterparts of *supportedByFromGoal* in line 10, respectively.

It is worth noting that some developers use *Solutions* to directly support *Strategy*. This is forbidden in the GSN standard (permitted SupportedBy connections are: *Goal*-to-*Goal*, *Goal*-to-*Strategy*, *Goal*-to-*Solution* and *Strategy*-to-

Goal). This is one of the reasons why model-based assurance case is desired - automated model validation can be performed to check the well-formedness of assurance cases.

6.2 The CAE metamodel and the interoprability from CAE to SACM

Claims-Arguments-Evidence (CAE) [4] is another widely used graphical notation for assurance case construction. Concepts in CAE are similar to those in GSN. There has not been an official metamodel defined for CAE. Thus, we provide our own version of CAE that extends SACM, shown in Figure 35[20].

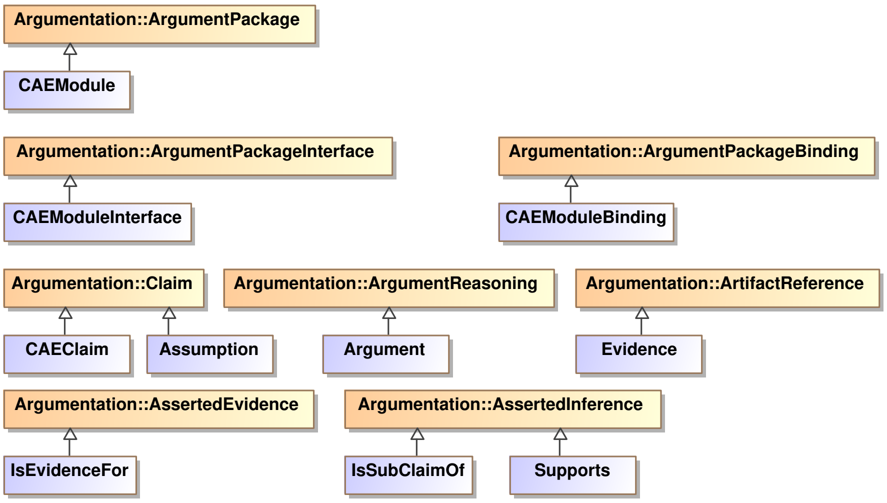

Argumentation::ArgumentPackage
CAEModule
Argumentation::ArgumentPackageInterface
CAEModuleInterface
Argumentation::ArgumentPackageBinding
CAEModuleBinding
Argumentation::Claim
CAEClaim
Assumption
Argumentation::ArgumentReasoning
Argument
Argumentation::ArtifactReference
Evidence
Argumentation::AssertedEvidence
IsEvidenceFor
Argumentation::AssertedInference
IsSubClaimOf
Supports

Fig. 35: SACM compliant CAE metamodel.

Unlike GSN and SACM, CAE does not support modularity. Therefore, we introduced three new concepts in CAE, *CAEModule*, *CAEModuleInterface* and *CAEModuleBinding*, which extend *ArgumentPackage*, *ArgumentPackageInterface* and *ArgumentPackageBiding* respectively. In CAE, there is a notion of *Claim*, which is semantically identical to *Claim* in SACM. We therefore create a *CAEClaim* element that extends *Claim* in SACM. The reason for this redundancy is that we want the CAE metamodel to be non-invasive to SACM. The *Argument* elements in CAE provide a description of the argument approach, which is functionally equivalent to *ArgumentReasoning*, thus it is made a sub-type of *ArgumentReasoning*. *Evidence* is made a sub-type of *ArtifactReference*

[20] All CAE elements are rendered in blue for readability.

because it is a reference to evidential materials. In CAE, there is a notion of *Assumption*, which is made a sub-type of *Claim*.

In CAE, there are three types of relationships, the *IsEvidenceFor* connects *Evidence* with *Claims*, thus is made a sub-type of *AssertedEvidence*; the *IsSubClaimOf* relationship connects sub-*Claims* to *Claim* and is made a sub-type of *AssertedInference*; the *Supports* relationship connects *Arguments* to *Claim* and is also made a sub-type of *AssertedInference*. Since there is no notion of modules in CAE, the argumentation is contained in *ArgumentPackages* inherited from SACM.

The transformation from CAE to SACM is similar to the transformation from GSN to SACM (which is also implemented in Epsilon Transformation Language), with the same issue that *Argument* (which is a node in the structured argumentation) needs to be mapped to *ArgumentReasoning* (a node associated to an edge). The detailed transformation is made publicly available[21].

7 Tool Support and Future Work

7.1 Assurance Case Modelling Environment - ACME

With all its power in model-based system assurance, there is one drawback for SACM at the moment, which is the lack of concrete syntax, i.e. graphical representations of SACM[22]. Without graphical representations, it is typically difficult for engineers to construct SACM. Hence, in order to exploit the benefits provided by SACM whilst provide support for existing assurance case approaches, we started developing a tool (Assurance Case Modelling Environment, ACME[23]) based on SACM and the GSN metamodel we discussed in Section 6.

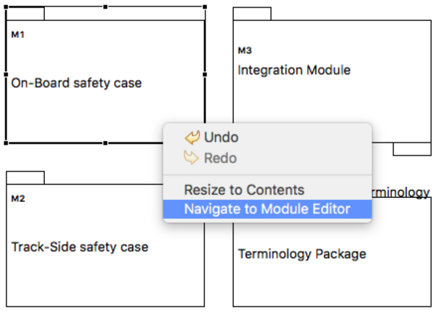


Fig. 36: The Assurance Case Package View of ACME.

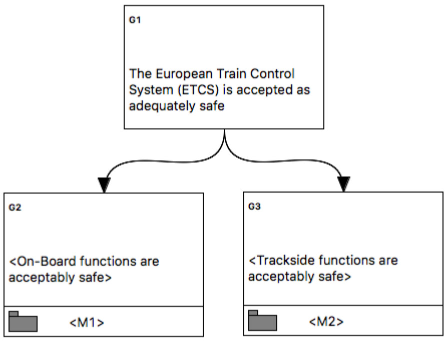
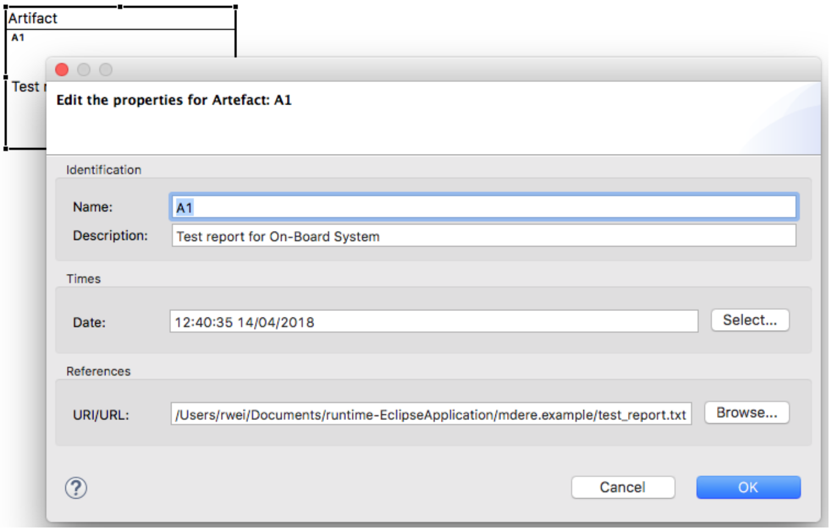

G1
The European Train Control System (ETCS) is accepted as adequately safe
G2
<On-Board functions are acceptably safe>
<M1>
G3
<Trackside functions are acceptably safe>
<M2>

Fig. 37: The Module View of ACME.

[21] Available at: https://github.com/wrwei/MDERE/blob/master/technical
[22] Graphical syntax for SACM is being developed by us
[23] Available upon request.

ACME is implemented using the Graphical Modelling Framework (GMF) [37], which supports the creation of editors based on metamodels defined using the Ecore metamodel provided by the Eclipse Modelling Framework (EMF) [38]. As previously discussed, we created metamodels for GSN and CAE, the approach we take for ACME is to support GSN for the argumentation part of the assurance case (since there is no graphical syntax for the *Argumentation* component of SACM). In this sense, the users of ACME are able to create an assurance case using SACM, use GSN for the arguments, and then use SACM's *Artifact* and *Terminology* components for evidence-artefact traceability and controlled vocabulary. In this way, system assurance case practitioners are able to make the transition from GSN/CAE to SACM, from (mostly) non-model-based approach to uniformed model-based approach.

Figure 36 shows the Assurance Case Package View, within which the users are able to create Modules and Contract Modules of GSN, as well as elements defined in the *AssuranceCase* component of SACM. Figure 37 shows the Module view of ACME, where the users are able to create GSN elements.



Edit the properties for Artefact: A1

Identification

Name: A1

Description: Test report for On-Board System

Times

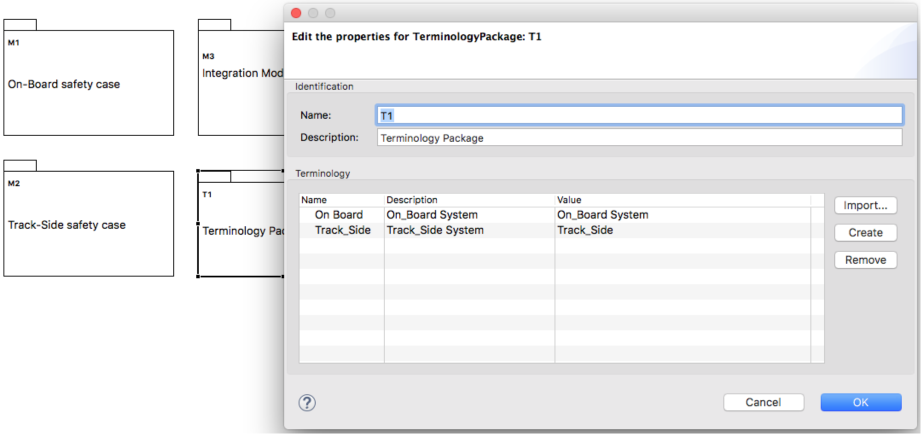
Date: 12:40:35 14/04/2018

References

URI/URL: /Users/rwei/Documents/runtime-EclipseApplication/mdere.example/test_report.txt

Fig. 38: Creation of Artifact.

The users are also able to create elements in the *Artifact* component of SACM (i.e. *Artifact*, *Activity*, *Event*, *Participant*, *Technique*, *Resource*, *Property*, and *ArtifactAssetRelationship*), Figure 38 demonstrates how an *Artifact* element can be created/edited. Creation tools are also provided for the elements in the *Terminology* component, where a *TerminologyPackage* is represented as a table in ACME, Figure 39 demonstrates how a *TerminologyPackage* and its contents can be created/edited.



Name: T1

Description: Terminology Package

Name	Description	Value
On Board	On_Board System	On_Board System
Track_Side	Track_Side System	Track_Side

Fig. 39: Editing a *TerminologyPackage*.

The idea behind ACME is to provide a transition for practitioners from GSN (and/or CAE[24]) to SACM, before the OMG standardises the graphical syntax of SACM. In this way, assurance case practitioners can continue use GSN, whilst exploiting the features provided by SACM (explained in Section 2.3).

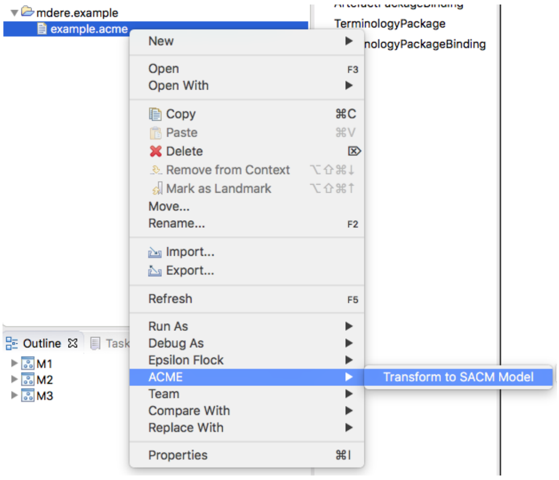


Fig. 40: M2M function to transform a GSN model to a SACM model.

The users of ACME are also able to transform their GSN models to SACM, based on the model-to-model transformation from GSN to ACME. ACME integrates with the Epsilon platform, so that model management operations are

[24] CAE editor for ACME is under development

supported on GSN/SACM models. A first step towards this direction is the support for GSN2SACM transformation, as shown in Figure 40.

ACME is the first step towards an integrated modelling environment for SACM. ACME provides a transitional solution to model-based system assurance, in the sense that it enables existing assurance case approaches to be used in conjunction with SACM, in order to exploit the features provided by SACM such as evidence-artefact traceability, controlled vocabularies and multiple language support. ACME as it is at the moment, illustrates what can be done with SACM and model-based assurance cases created with SACM. However, it does not guarantee that assurance cases provided by ACME are compelling ones. It is still the responsibility of assurance case developers to develop accurate and correct assurance cases.

7.2 Future Work

With regard to future work for ACME, we aim at achieving the following;

- Support for CAE. We aim to create the editor for CAE and integrate to ACME
- Support for legacy assurance cases. We aim to develop a set of model extraction mechanisms to extract information from assurance cases provided by existing tools and convert them to models that conform to our GSN/CAE metamodels.
- Model management programs. We aim to develop a set of model validation rules to check the well-formedness of GSN/CAE/SACM models. We also aim to develop a set of model-to-text transformation rules for assurance case report generation. Finally,
- Support for automated pattern instantiation. We integrate the Epsilon platform runtime[25] in ACME for various model management operations. We aim to explore how *ImplementationConstraint*s can be utilised to hold Epsilon programs for pattern instantiation.
- Concrete syntax for SACM. We are contributing to the development of graphical notations for SACM at the moment, and will add the support for the notations in ACME once they are developed and evaluated.

8 Conclusion

In this paper, we identified the importance of model-based system assurance cases, for that they enable high level model management operations to be performed, and its potential applications in Open Adaptive Systems. We also identified the importance of SACM for its role in model-based system assurance. SACM is more powerful than existing system assurance approaches (such as GSN and CAE), for its additional features;

[25] https://www.eclipse.org/epsilon/

- Fine grained modularity, for component based system assurance, as well as assurance case integration;
- Multiple language support, to support multiple natural languages, as well as computer languages;
- Controlled Vocabulary, to standardise terminologies used in assurance cases;
- Ability to argue the trustworthiness of arguments, so that assurance case reviewers are able to determine the level of trust to argument elements;
- Ability to express counter-arguments, so that the argumentation process becomes more comprehensible;
- Traceability from evidence to artefact, so that change of system information/argumentation can be propagated throughout the assurance case, to enable incremental certification;
- Automated assurance case instantiation, to link system information with failure modes to create concrete assurance cases.

We also provided a definitive exposition of SACM to explain its intended usage via examples. SACM has been sufficiently explained in this paper although extensive examples cannot be fully provided.

We also provided our version of GSN and CAE metamodels, which are compliant to SACM in the sense that users of these metamodels are able to exploit the facilities provided by SACM whilst still using GSN/CAE notations that they are familiar with. We also provide comprehensible model-to-model transformations from GSN/CAE to SACM to enable the interoperability from GSN/CAE to SACM.

We also briefly discussed the Assurance Case Modelling Environment (ACME) - a graphical modelling tool created based on our version of the GSN metamodel. With ACME, we explained what is supported when assurance cases become model-based, and what can be done in the future work of ACME ACME acts as a transitional solution from conventional GSN diagram creation to SACM model-based system assurance. ACME provides easy-to-use support for SACM facilities, as well as automated model-to-model transformation from GSN to SACM.

SACM provides a solid foundation for model-based system assurance, due to the variety of features that have been evaluated and added to it, based on experiences of using two well-established assurance case notations: GSN and CAE. Model-based assurance case is the key to assure Open Adaptive Systems (such as Cyber-Physical Systems), for it enables system assurance to be performed at runtime, which entails automated system assurance case integration, and automated reasoning of assurance case argumentations.

Acknowledgements This work is supported by the European Union's Horizon 2020 research and innovation programme through the DEIS project (grant agreement No 732242).

References

1. Health Foundation. *Using safety cases in industry and healthcare.* December 2012.
2. Richard Hawkins, Ibrahim Habli, Tim Kelly, and John McDermid. Assurance cases and prescriptive software safety certification: A comparative study. *Safety science*, 59:55–71, 2013.
3. Tim Kelly and Rob Weaver. The goal structuring notation–a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, page 6. Citeseer, 2004.
4. Peter Bishop and Robin Bloomfield. A methodology for safety case development. In *Safety and Reliability*, volume 20, pages 34–42. Taylor & Francis, 2000.
5. Mike Maksimov, Nick Fung, Sahar Kokaly, and Marsha Chechik. Two decades of assurance case tools: A survey. In *Proc. 6th International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE 2018)*, 2018.
6. Ewen Denney and Ganesh Pai. Tool support for assurance case development. *Automated Software Engineering*, pages 1–65, 2017.
7. Yutaka Matsuno, Hiroki Takamura, and Yutaka Ishikawa. A dependability case editor with pattern library. In *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*, pages 170–171. IEEE, 2010.
8. Kateryna Netkachova, Oleksandr Netkachov, and Robin Bloomfield. Tool support for assurance case building blocks. In *International Conference on Computer Safety, Reliability, and Security*, pages 62–71. Springer, 2014.
9. Xabier Larrucea, Alastair Walker, and Ricardo Colomo-Palacios. Supporting the management of reusable automotive software. *IEEE Software*, (3):40–47, 2017.
10. Matthew R Barry. Certware: A workbench for safety case production and analysis. In *Aerospace Conference, 2011 IEEE*, pages 1–10. IEEE, 2011.
11. Health and Safety Executive (HSE). *Safety Assessment Principles for Nuclear Facilities.* 2006.
12. UK Ministry of Defence (MoD). *Defence Standard 00-56 Issue 4: Safety Management Requirements for Defence Systems.* 2007.
13. Safety Regulation Group Civil Aviation Authority (CAA). *CAP 670 - Air Traffic Services Safety Requirements.* 2007.
14. Rail Safety and Standards Board. *Engineering Safety Management (The Yellow Book).* 2007.
15. Paul Chinneck, David Pumfrey, and Tim Kelly. Turning up the heat on safety case construction. In *Practical Elements of Safety*, pages 223–240. Springer, 2004.
16. Tim Kelly and Simon Bates Meng. The costs, benefits, and risks associated with patternbased and modular safety case development," to appear. In *in Proceedings of the UK MoD Equipment Safety Assurance Symposium.* Citeseer, 2005.
17. Ari Jaaksi. Developing Mobile Browsers in a Product Line. *IEEE software*, 19(4):73–80, 2002.
18. Juha Kärnä, Juha-Pekka Tolvanen, and Steven Kelly. Evaluating the Use of Domain-Specific Modeling in Practice. In *Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling*, 2009.
19. Robin Bloomfield and Peter Bishop. Safety and assurance cases: Past, present and possible future–an adelard perspective. In *Making Systems Safer*, pages 51–67. Springer, 2010.
20. Samantha Lautieri, David Cooper, and David Jackson. Safsec: Commonalities between safety and security assurance. In *Constituents of Modern System-safety Thinking*, pages 65–75. Springer, 2005.

21. Structured Assurance Case Metamodel, Object Management Group. https://www.omg.org/spec/SACM/About-SACM/. Accessed: 06-04-2018.
22. George Despotou and T Kelly. Investigating the use of argument modularity to optimise through-life system safety assurance. 2008.
23. Ewen Denney and Ganesh Pai. A formal basis for safety case patterns. In *International Conference on Computer Safety, Reliability, and Security*, pages 21–32. Springer, 2013.
24. Yaping Luo, Mark van den Brand, and Alexandre Kiburse. Safety case development with sbvr-based controlled language. In *International Conference on Model-Driven Engineering and Software Development*, pages 3–17. Springer, 2015.
25. Katrina Claire Attwood, Tim Kelly, and Philippa Conmy. The use of controlled vocabularies and structured expressions in the assurance of cps. *Ada User Journal*, pages 251–258, 2014.
26. Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. A new approach to creating clear safety arguments. In *Advances in systems safety*, pages 3–23. Springer, 2011.
27. Jane Fenn and Brian Jepson. Putting trust into safety arguments. In *Constituents of Modern System-safety Thinking*, pages 21–35. Springer, 2005.
28. James M Armstrong and Stephen E Paynter. The deconstruction of safety arguments through adversarial counter-argument. In *International Conference on Computer Safety, Reliability, and Security*, pages 3–16. Springer, 2004.
29. Kenji Taguchi, Souma Daisuke, Hideaki Nishihara, and Toshinori Takai. Linking traceability with gsn. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 192–197. IEEE, 2014.
30. Richard David Hawkins, Ibrahim Habli, and Tim Kelly. The need for a weaving model in assurance case automation. *Ada User Journal*, pages 187–191, 2015.
31. Richard Hawkins, Ibrahim Habli, Dimitris Kolovos, Richard Paige, and Tim Kelly. Weaving an assurance case from design: a model-based approach. In *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, pages 110–117. IEEE, 2015.
32. Mario Trapp, Daniel Schneider, and Peter Liggesmeyer. A safety roadmap to cyber-physical systems. In *Perspectives on the future of software engineering*, pages 81–94. Springer, 2013.
33. Ran Wei, Tim P Kelly, Richard Hawkins, and Eric Armengaud. Deis: Dependability engineering innovation for cyber-physical systems. In *Federation of International Conferences on Software Technologies: Applications and Foundations*, pages 409–416. Springer, 2017.
34. GSN Committee et al. Draft gsn standard version 1.0, 2010.
35. Tim P Kelly and John A McDermid. Safety case construction and reuse using patterns. In *Safe Comp 97*, pages 55–69. Springer, 1997.
36. Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. The epsilon transformation language. In *International Conference on Theory and Practice of Model Transformations*, pages 46–60. Springer, 2008.
37. Graphical Modeling Framework. https://www.eclipse.org/gmf-tooling/. Accessed: 06-04-2018.
38. Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.