

8 Foundation Class

8.1 General

Foundation provides a semantic backing to the SACM Model from UML/KerML like models to make mapping (e.g. through Profiles) into UML/SysMLv1 or SysMLv2 more regular. Many of the textual entries in this section come directly from the UML 2.5.1 metamodel.

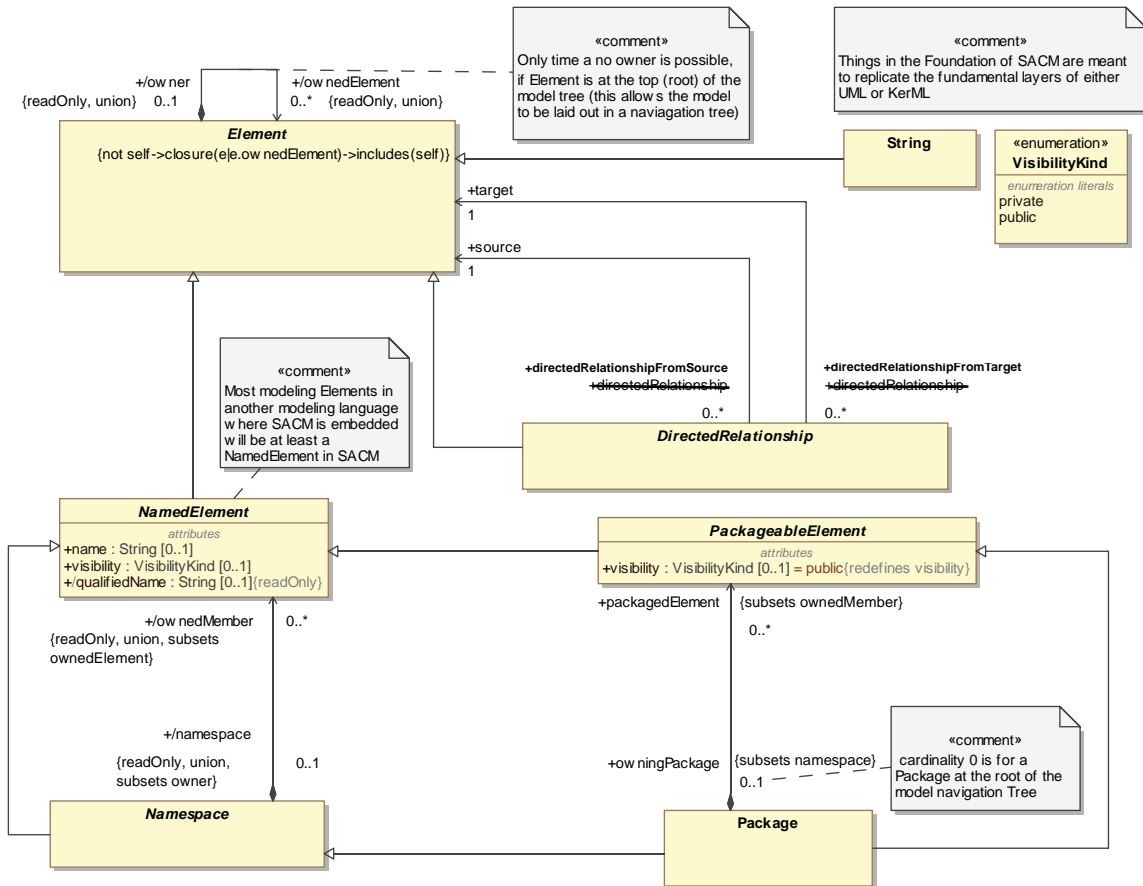


Figure 8.1 - Foundation Class

8.1 DirectedRelationship [Abstract Class]

A *DirectedRelationship* represents a relationship between a source model *Element* and target model *Element*. (This represents a more restrictive, than UML, *Binary Relationship*.)

Association Ends

source
~~/source~~ : Element [1]

Specifies the source *Element* of the *DirectedRelationship*.

target
~~/target~~ : Element [1]

Specifies the target *Element* of the *DirectedRelationship*.

8.2 Element [Abstract Class]

An Element is a constituent of a model. As such, it has the capability of owning other Elements.

Association Ends

/ownedElement : Element [0..*]{union,readOnly}

The Elements owned by this Element.

/owner : Element [0..1]{union,readOnly}

The Element that owns this Element.

Constraints

not_own_self

An element may not directly or indirectly own itself.

inv: not self->closure(e | e->ownedElement)->includes(self)

8.3 NamedElement [Abstract Class]

A NamedElement is an Element in a model that may have a name.

Attributes

name : String [0..1]

The name of the NamedElement.

/qualifiedName : String [0..1] {readOnly}

A name that allows the NamedElement to be identified within a hierarchy of nested Namespaces. It is constructed from the names of the containing Namespaces starting at the root of the hierarchy and ending with the name of the NamedElement itself. (Separator of names is "::".)

visibility : VisibilityKind [0..1]

Determines whether and how the NamedElement is visible outside its owning Namespace.

~~Association Ends~~

~~**/namespace : Namespace [0..1]{union}**~~

~~*Specifies the Namespace that owns the NamedElement.*~~

Constraints

visibility_needs_ownership

If a NamedElement is owned by something other than a Namespace, it does not have a visibility.

inv: (namespace = null and owner <> null) implies visibility = null

8.4 Namespace [Abstract Class]

A Namespace is an Element in a model that owns and/or imports a set of NamedElements that can be identified by name.

Association Ends

{union,readOnly,subsets ownedElement}

/ownedMember : NamedElement [0..*]~~{union, subsets Element::ownedElement}~~

A collection of NamedElements owned by the Namespace.

8.5 Package [Class]

A package can have one or more profile applications to indicate which profiles have been applied. Because a profile is a package, it is possible to apply a profile not only to packages, but also to profiles.

attributes

{subsets ownedElement}

packageableElement : PackageableElement [0..*]~~{subsets Namespace::ownedMember}~~

Specifies the packageable elements that are owned by this Package.

8.6 ~~8.5~~ PackageableElement [Abstract Class]

A PackageableElement is a NamedElement that may be owned directly by a Package.

Attributes

visibility : VisibilityKind [0..1] = public

A PackageableElement must have a visibility specified if it is owned by a Namespace. The default visibility is public.

VisibilityKind [Enumeration]

VisibilityKind is an enumeration type that defines literals to determine the visibility of Elements in a model.

Literals

public

A Named Element with public visibility is visible to all elements that can access the contents of the Namespace that owns it.

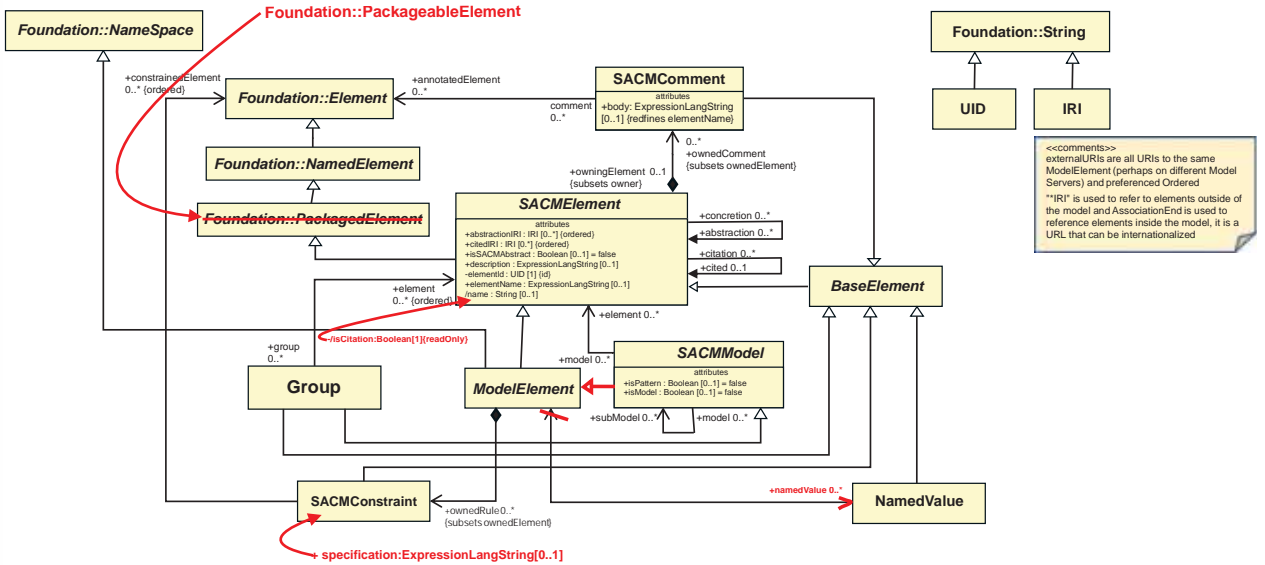
private

A NamedElement with private visibility is only visible inside the Namespace that owns it.

9 ~~8~~ Structured Assurance Case Base Classes

9 ~~8.1~~ General

This chapter presents the normative specification for the SACM Base Metamodel. It begins with an overview of the metamodel structure followed by a description of each element.

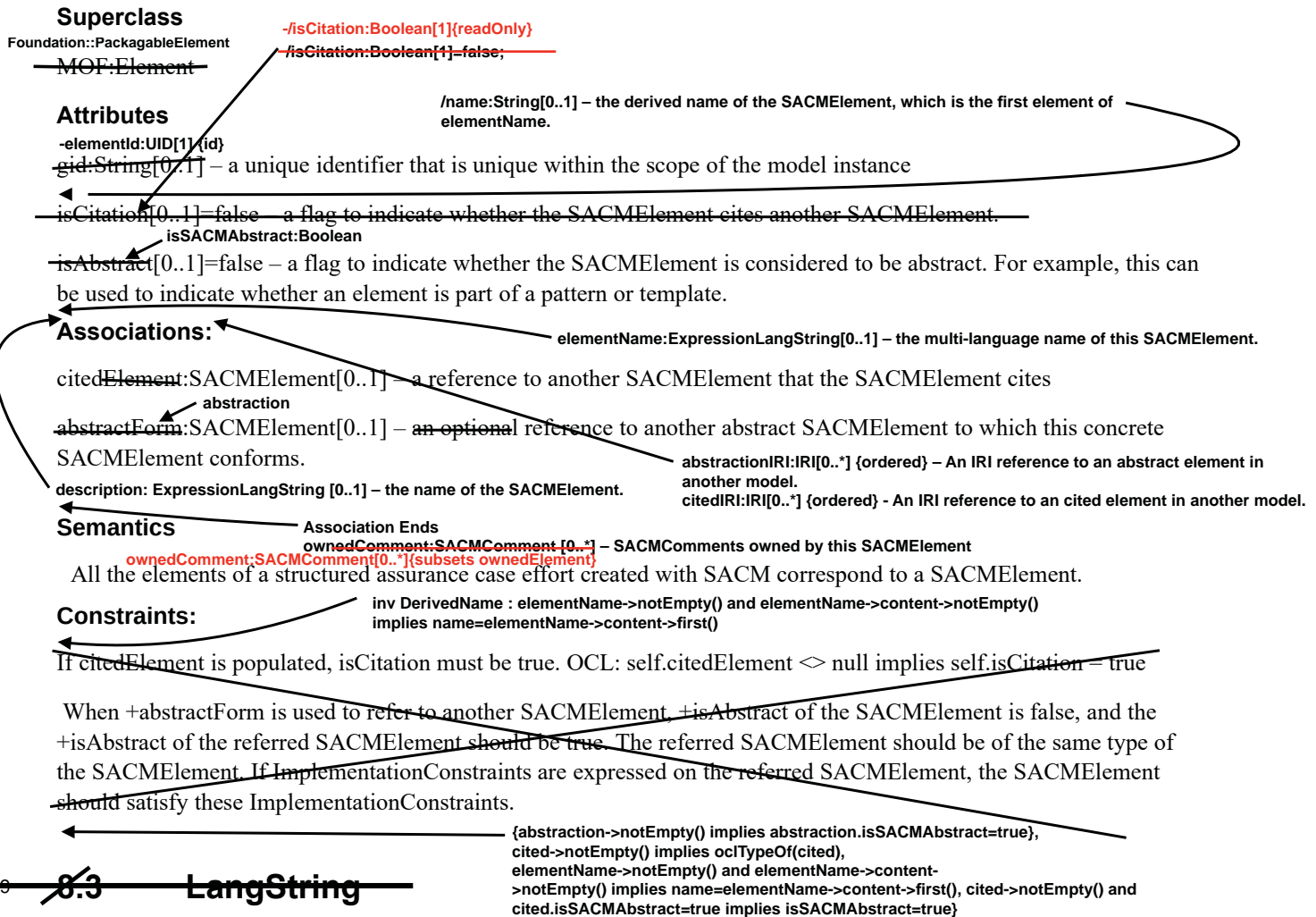


9 Figure ~~8.1~~ - Structured Assurance Case Base Classes Diagram

The Structured Assurance Case Base Classes express the foundational concepts and relationships of the base elements of the SACM metamodel and are utilized, through inheritance, by the bulk of the rest of the Structured Assurance Case Metamodel.

9 ~~8.2~~ SACMElement (abstract)

SACMElement is the base class for SACM.



9 ~~8.3~~ LangString

LangString is the format SACM uses for description. It serves the same purpose as String but with the additional specification of the language used for the content.

Superclass

MOF:Element

Attributes

lang:String[0..1] – a field to indicate the language used in the string.
content:String[0..1] – the content of the string

Semantics

LangString serves the same purpose as String, SACM uses LangString for description, which containing the information of the language it uses in the content.

Move to Terminology Class

9 ~~8.4~~ ExpressionLangString

ExpressionLangString is used to denote a structured expression, it ~~contains a description (LangString) and it also (optionally) points to an ExpressionElement in the Terminology Package.~~

18

Structured Assurance Case Metamodel, v2.3

inherits description (LangString) and it also (optionally) points to an ExpressionElement.

Superclass

MultiLangString

Attributes

expression:ExpressionEleme

~~expression:Terminology::ExpressionElement[1] (composition) – a reference to an ExpressionElement in the TerminologyPackage~~

Semantics

~~ExpressionLangString provides a means for description, it can also be used to link to an ExpressionElement in the Terminology package.~~

~~**Constraints**~~

~~If expression is not empty, then +content should be empty.~~

9 ~~8.5~~ **MultiLangString**

MultiLangString, as its name suggests, provides a means to describe things using different languages.

Superclass

~~Element~~ Foundation::Element

Associatiions

~~value:LangString[1..*] (composition) – contains the descriptions which bear the same meaning but in different languages~~

content:String[0..*] {ordered,nonunique} – the content of the string.

Semantics

MultiLangString provides a means to describing things using different languages. It contains a ~~list of LangString, which the user can specify their languages and the descriptions in the languages.~~

Constraints set of contents each of which is in a different language represented by

~~For each of the LangString in the value feature, their +lang must be unique.~~

9 ~~8.6~~ 3 **ModelElement (abstract)**

ModelElement is the base element for the majority of modeling elements.

Superclass

, Foundation::NameSpace

SACMElement

Associations

Attributes

~~ownedRule:SACMConstraint[0..*]{subsets ownedElement}~~

~~name:LangString[1] (composition) – the name of the ModelElement.~~

~~ownedRole: SACMConstraint [0..*] – SACMConstraints owned by this ModelElement.~~

~~implementationConstraint: ImplementationConstraint [0..*] (composition) – a collection of implementation constraints.~~

~~description: Description[0..1] (composition) – the description of the ModelElement.~~

~~note:Note[0..*] (composition) – a collection of notes for the ModelElement.~~

namedValue:NamedValue

NamedVal NamedVal

~~taggedValue: TaggedValue [0..*] (composition) – a collection of TaggedValues, TaggedValues can be used to describe additional features of a ModelElement~~

Semantics

All the individual and identifiable elements of a SACM model correspond to a ModelElement.

Moved to, and revised in, 8.2

LangIfContentMoreThan1
If content has more than one value, then lang must have string representing the languages of the values.
inv: content->size() > 1 implies lang->size() = content->size()

~~Constraints~~

~~ImplementationConstraints should only be specified if +isAbstract is true~~

~~OCL:~~

~~self.implementationConstraint->size() > 0 implies self.isAbstract = true~~

9 ~~8.7~~ 4 ~~BaseElement~~ **UtilityElement (abstract)**

~~BaseElement~~ UtilityElement is the base element for a number of auxiliary elements which can be added to ModelElements.

used in any diagram type.

~~Superclass~~

~~SACMElement~~

~~Associations~~

~~content:MultiLangString[0..1] (composition) – a MultiLangString to describe the content of the UtilityElement in (possibly) multiple languages~~

~~Semantics~~

~~UtilityElement supports the specification of additional information for a ModelElement.~~

9 ~~8.8~~ 5 ~~SACMConstraint~~ **ImplementationConstraint**

~~ImplementationConstraint~~ specifies details of any implementation constraints that must be satisfied whenever a referencing ModelElement is to be converted from *isAbstract = true* to *isAbstract = false*. For example in the context of a SACM pattern fragment, an element will need to satisfy the implementation rules of the pattern.

~~Superclass~~

~~BaseElement~~

~~UtilityElement~~

~~Semantics~~

~~ImplementationConstraints indicate the conditions to fulfill in order to allow an abstract ModelElement (isAbstract = true) to become non abstract (isAbstract = false).~~

Attributes

specification:ExpressionLangString [0..1] – specification details a Boolean results that must be true for a valid model.

AssociationEnds

constrainedElement:Element[0..*]{ordered} – the SACMElement which this constraint constrains. (for ordering see UML 2.5.1 section 7.6.4 with respect to dashed lines)

9 ~~8.9~~ 6 ~~Description~~

~~Description is used to specify a description that may be associated with a ModelElement. In many cases Description is used to provide the 'content' of a SACM element. For example, it would be used to provide the text of a Claim.~~

~~Superclass~~

~~UtilityElement~~

~~Semantics~~

~~A Description provides details about ModelElements in relation to aspects such as their content or purpose. Therefore, Descriptions can be used to both characterize ModelElements and facilitate their understanding.~~

Move to Artifact

9 ~~8.10~~ 7 **ArtifactElement (abstract)**

ArtifactElement acts as the base class for elements in other SACM packages. Essentially, all elements which extend ArtifactElement is considered to be an artifact, and therefore can be referenced using Argument:ArtifactReference.

~~Superclass~~

~~ModelElement~~

Semantics

ArtifactElement corresponds to the base class for specifying all the identifiable units of data modelled and managed in a structured assurance case effort.

9 ~~8.11~~ 8 **Note**

This class specifies a generic note that may be associated with a ~~ModelElement~~ ^{an} ModelElement. For example a note may include a number of explanatory comments.

Superclass
~~SACMCoreElement~~
~~UtilityElement~~
 BaseElement

Attributes
~~body:ExpressionLangString[0..1]~~
~~AssociationEnds~~
 annotatedElement:Element[0..*] – Elements to which the SACMComment applies

Semantics

~~SACMComments~~
 Notes are used to specify additional (typically optional) generic, unstructured, untyped information about a ModelElement. An example of this kind of information could be a comment about a ModelElement.

9 ~~8.12~~ 9 **TaggedValue**

This class represents a ~~simple key/value pair~~ ^{attributes} that can be attached to any element in SACM. This is a simple extension mechanism to allow users to add attributes to each element beyond those already specified in SACM.

Superclass
~~BaseElement~~
~~UtilityElement~~
 BaseElement

Associations
 value :String [0..*] {ordered, nonunique}

Semantics

~~NamedValue~~
~~TaggedValues~~ can be used to specify attributes, and their corresponding values, for ModelElements.

9.10 Group

Group can be used to associate a number of SACMElements to a common group (e.g., representing a common type or purpose, or being of interest to a particular stakeholder).

Supertype
~~SACMElement~~
 BaseElement

Association End

element: SACMElement [0..*] {ordered} – a collection of SACMElements that comprise a group

9.11 ~~9.10~~ IRI

IRI is a international resource identifier is a String and is formatted according to RFC 3987.

Supertype
~~String~~
 Foundation::String

9.11 SACMModel
 An abstract subtype of ModelElement which allows one to define a model and/or a pattern.

Supertype
 ModelElement

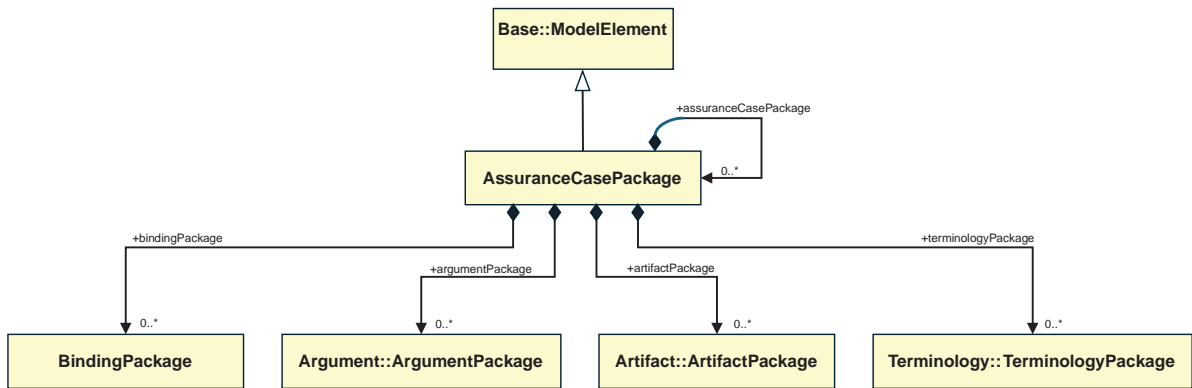
Attributes
 isPattern : Boolean [0..1] = false - if true elements in SACMModel are part of a pattern.
 isModel : Boolean [0..1] = false - if true elements in SACMModel are part of a model.

Association
 +subModel:SACMModel [0..*] - subModel is a recursive containment of elements in the model.
 +elements:SACMElement [0..*] - elements are items in the model.

10 ~~9~~ Structured Assurance Case Packages

10 ~~9.1~~ General

This chapter presents the normative specification for the SACM Packages Metamodel. It begins with an overview of the metamodel structure followed by a description of each element.



10
Figure 9.1 - Structured Assurance Case Packages Class Diagram

In SACM, the parent container element is AssuranceCasePackage. AssuranceCasePackages can be thought of assurance case 'modules'. Packages can contain other packages, including citations to other packages not contained within the same package hierarchy. Packages optionally can have a separately declared interface (AssuranceCasePackageInterface) (analogous to a public header file) that declares selected packages contained by a package.

Assurance cases (AssuranceCasePackages) consist of arguments (contained in ArgumentPackages), evidence descriptions (contained in ArtifactPackages) and Terminology definitions (contained in TerminologyPackages).

10 ~~9.2~~ AssuranceCasePackage

AssuranceCasePackage is an exchangeable element that may contain a mixture of artifacts, argumentation and terminology. When users exchange content, it is expected they use this as the top-level container. It is a recursive container, and may contain one or more sub-packages.

This follows the existing practice of considering an assurance case when fully completed to comprise both argumentation and evidence, although each may be exchanged individually.

~~AssuranceCasePackage is a sub-class of Base::ArtifactElement. Semantically an AssuranceCasePackage can be considered as an artifact of evidence (e.g., from the perspective of another AssuranceCasePackage).~~

Superclass

Base::ModelElement
~~Base::ArtifactElement~~

Associations

assuranceCasePackage: AssuranceCasePackage [0..*] (composition) – a collection of optional sub-packages

interface: AssuranceCasePackageInterface [0..*] – a number of optional assurance case package interfaces that the current package may implement

bindingPackage: BindingPackage [0..*] Sub-packages within any BindingPackage can be bound together by means of a BindingPackage. A Binding can include any combination of reference to Package or InterfacePackage.

artifactPackage: ArtifactPackage [0..*] (composition) – a number of optional artifact sub-packages

terminologyPackage: TerminologyPackage [0..*] (composition) – a number of optional terminology sub-packages

argumentPackage: Argument::ArgumentPackage[0..*] (composition) – a number of optional argument packages.

Semantics

AssuranceCasePackage is the root class for creating structured assurance cases.

~~9.3 AssuranceCasePackageInterface~~

~~AssuranceCasePackageInterface is a kind of AssuranceCasePackage that defines an interface that may be exchanged between users. An AssuranceCasePackageInterface will consist of at least one of an ArgumentPackageInterface, ArtifactPackageInterface, and/or a TerminologyPackageInterface. Conversely, any combination of an ArgumentPackageInterface, ArtifactPackageInterface, and/or a TerminologyPackageInterface will be contained within an AssuranceCasePackageInterface. The combination depends on what parts of the assurance case are to be made “public.” An AssuranceCasePackage may declare one or more ArtifactPackageInterfaces.~~

~~Superclass~~

~~AssuranceCasePackage~~

~~Associations~~

~~implements:AssuranceCasePackage[N] – the AssuranceCasePackage that the AssuranceCasePackageInterface declares.~~

~~Semantics~~

~~AssuranceCasePackageInterface enables the declaration of the elements of an AssuranceCasePackage that might be referred to (cited) in another AssuranceCasePackage. These declarations are provided by containing AssuranceCasePackageInterface(s)/ArgumentPackageInterface(s)/ArtifactPackageInterface(s)/TerminologyPackageInterface(s) to the packages contained by the AssuranceCasePackage for which the interface is provided.~~

~~Constraints~~

~~AssuranceCasePackageInterface are only allowed to contain the following: AssuranceCasePackageInterface, ArgumentPackageInterfaces, ArtifactPackageInterfaces, and TerminologyPackages.~~

~~OCL:~~

~~self.assuranceCasePackage->forall(acp|acp.ocIsTypeOf(AssuranceCasePackageInterface)) and
self.argumentPackage->forall(ap|ap.ocIsTypeOf(Argumentation::ArgumentPackageInterface))
and self.artifactPackage->forall(ap|ap.ocIsTypeOf(Artifact::ArtifactPackageInterface)) and
self.terminologyPackage->forall(tp|
tp.ocIsTypeOf(Terminology::TerminologyPackageInterface))~~

10.3

~~9.4~~

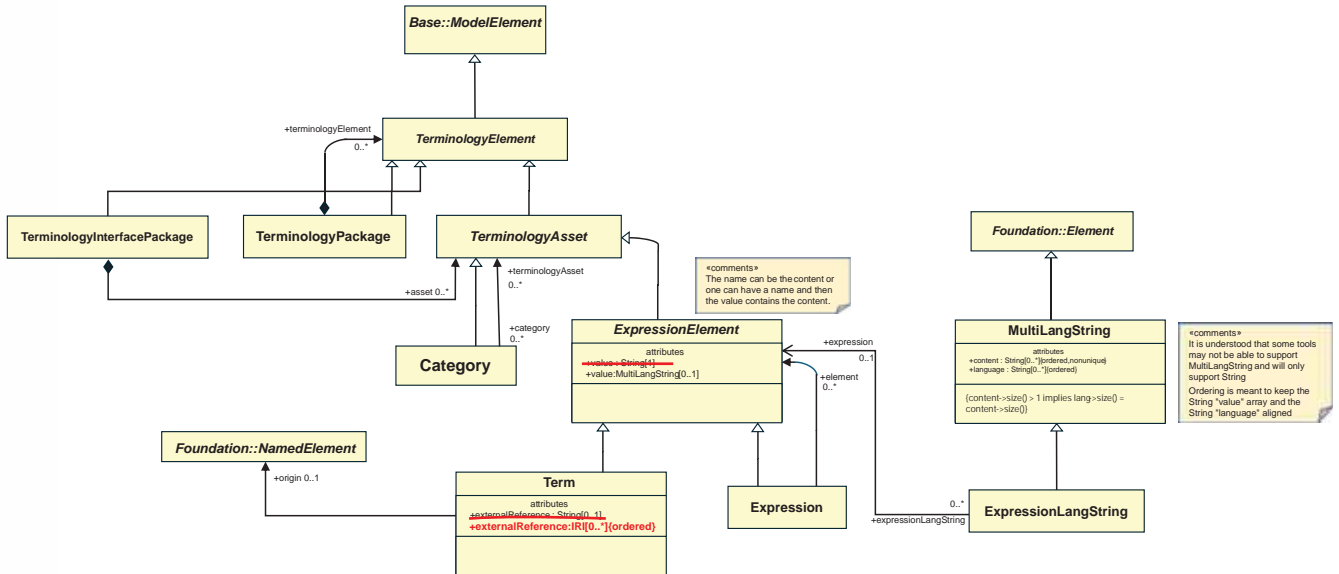
~~AssuranceCasePackageBinding~~

~~Sub-packages within any AssuranceCasePackage can be bound together by means of AssuranceCasePackageBindings. A Binding can include any combination of reference to Package or PackageInterface of the same type. An AssuranceCasePackageBinding can also include Package(s) or PackageInterface(s) for Argument, Artifact and/or Terminology. Each Package or PackageInterface referenced by a PackageInterface can be a participant Package in a Binding. AssuranceCasePackageBindings bind the participant packages by means of ArgumentPackageBindings/TerminologyPackageBindings/ArtifactPackageBindings elements that bind the contained packages of the participant packages.~~

11 ~~10~~ Structured Assurance Case Terminology Classes

11 ~~10.1~~ General

This chapter presents the normative specification for the SACM Terminology Metamodel. It begins with an overview of the metamodel structure followed by a description of each element



11 ~~10.1~~ Figure 10.1 - Terminology Class Diagram

This portion of the SACM metamodel describes and defines the concepts of term, expression and an external interface to terminology information from others. This area of the Structured Assurance Case Metamodel also provides the starting foundation for formalism in the assembly of terms into expressions without mandating the formalism for those that do not need it.

11 ~~10.2~~ TerminologyElement (abstract)

TerminologyElement is an abstract class that serves as a parent class for all SACM terminology assets (TerminologyAsset) and the grouping of TerminologyElements (TerminologyGroup). TerminologyElement extends Base::ArtifactElement, this implies that all elements in the Terminology package are artifacts.

Superclass

Base::ModelElement

~~Base::ArtifactElement~~

Semantics

TerminologyElement is the base class for specifying the terminology aspects of an assurance case (AssuranceCasePackage).

10.3 TerminologyGroup

TerminologyGroup can be used to associate a number of TerminologyElements to a common group (e.g., representing a common type or purpose, or being of interest to a particular stakeholder).

Superclass

TerminologyElement

Associations

terminologyElement[0..*] – an optional collection of TerminologyElements that are organised within the TerminologyGroup.

Semantics

TerminologyGroup can be used to associate a number of TerminologyElements to a common group (e.g. representing a common type or purpose, or being of interest to a particular stakeholder). The name and the description of the TerminologyGroup should provide the semantic for understanding the TerminologyGroup. TerminologyGroups serve no structural purpose in the formation of the argument network, nor are they meant as a structural packaging mechanism (this should be done using TerminologyPackages).

11 10.4³ TerminologyPackage

The TerminologyPackage is the container element for SACM terminology assets.

Superclass

TerminologyElement

Associations

TerminologyElement:TerminologyElement[0..*] (composition) – TerminologyElements contained in the TerminologyPackage, it can be either TerminologyPackage (and its sub-types) or TerminologyAssets (or its sub-types).

Semantics

TerminologyPackage contains the TerminologyElements that can be used within the naming and description of SACM arguments and artifacts. TerminologyPackages can be nested.

11 10.5⁴ TerminologyPackageInterface

TerminologyInterfacePackage

TerminologyPackageInterface is a kind of TerminologyPackage that defines an interface that may be exchanged between users. An TerminologyPackage may declare one or more TerminologyPackageInterfaces.

Superclass

TerminologyElement
~~TerminologyElement~~

Associations

+asset : terminologyAsset[0..*]
~~implements:TerminologyPackage[1] – the TerminologyPackage that the TerminologyPackageInterface declares.~~

Semantics

TerminologyPackageInterface enables the declaration of the elements of an TerminologyPackage that might be referred to (cited) in another TerminologyPackage, thus the elements can be used for assurance in the scope of the latter AssuranceCasePackage. A TerminologyPackageInterface resides inside the TerminologyPackage to which it refers. It refers to TerminologyElements using isCitation=True that reside within the same TerminologyPackage as itself.

28

Constraints
All Elements that are cited in a BindingPackage must be contained in either the owner of that BindingPackage or a (recursively) sibling Package of that BindingPackage
inv CitedElementsAreScoped: element->forAll(e|e.cited->notEmpty()) implies e.cited->closure(g.owningPackage).asSet()->one(p|p=self.owningPackage)
Elements that are not either a Diagram or a BaseElement must be a citation.
inv MustBeCited: element->forAll(e|not e->oclKindOf(SACMDiagram) and not e->oclKindOf(BaseElement) implies e.isCitation=true)

Structured Assurance Case Metamodel, v2.3

10.6 TerminologyPackageBinding

Elements within the TerminologyPackage can be bound together by means of TerminologyPackageBindings. TerminologyPackageBindings bind the participant packages by means of terminology elements that connect the cited elements of the participant packages.

Superclass

TerminologyPackage

Semantics

TerminologyPackageBinding binds TerminologyPackages together to indicate the relationship between two TerminologyPackages. A TerminologyPackageBinding resides within an AssuranceCasePackageBinding. It contains references, using isCitation=True to each TerminologyElement and connects TerminologyElements from different TerminologyPakages.

Constraints

The participantPackages should be either TerminologyPackage or TerminologyPackageInterface

OCL:

```
self.participantPackage->forall(pp|pp.ocIsKindOf(Terminology::TerminologyPackage))
```

11 10.7⁶ TerminologyAsset (abstract)

The TerminologyAsset Class is the abstract class for the different types of terminology elements represented in SACM.

Superclass

TerminologyElement

Semantics

TerminologyAssets represent all of the elements required to model and categorize expressions in SACM (expressions and terminology categories).

11 10.8⁷ Category

The Category class describes categories of ExpressionElements (Terms and Expressions) and can be used to group these elements within TerminologyPackages.

Superclass

TerminologyAsset

Association Ends
expressionElement:ExpressionElement[0..*]{ordered} – elements contained in the Category

Semantics

Terms, Categories, and ExpressionElements can be said to belong to Categories. Categories can group Terms, Expressions, or a mixture of both and Categories can contain Categories. For example, a Category could be used to describe the terminology associated with a specific assurance standard, project, or system.

11 ~~10.9~~⁸ ExpressionElement (abstract)

The ExpressionElement class is the abstract class for the elements in SACM that are necessary for modeling expressions.

Superclass

TerminologyAsset

Attributes

~~value:String[1] — the value of the expression.
value:MultiLangString[0..1]~~

~~Associations~~

~~category:Category [0..*] — optionally associates the ExpressionElement with one or more terminology categories.~~

Semantics

ExpressionElements are used to model (potentially structured) expressions in SACM.

← The name can be the content or one can have a name and then the value contains the content.

11 ~~10.10~~⁹ Expression

The Expression class is used to model both abstract and concrete phrases in SACM. Abstract Expressions are denoted by the inherited isAbstract:Boolean attribute being set true. A concrete expression (denoted by isAbstract:Boolean being false) is one that has a literal string value and references only concrete ExpressionElements.

Superclass

ExpressionElement

Associations

element:ExpressionElement[0..*]{ordered}

~~element:ExpressionElement [0..*] — an optional reference to other ExpressionElements forming part of the structured Expression.~~

Semantics

Expressions are used to model phrases and sentences. These are defined using the value feature. Alternatively, the expression can also be defined (using the value feature) as a production rule involving other ExpressionElements. In this case, the value must use a suitable (string) form for denoting the position of involved ExpressionElements (e.g. “\$<ExpressionElement.name>\$”) within the production rule, and expressing production rule operators (e.g. Extended Backus-Naur Form operators).

Constraints

Where an Expression has associated ExpressionElements (the +element feature), these should be referenced by name within the +value feature.

Where the +value feature references ExpressionElement by name, these ExpressionElements should be associated (using the +element feature) with Expression. A concrete expression should have references to only concrete ExpressionElements

OCL:

self.isAbstract = false implies self.element->forall(expr|expr.isAbstract = false).

11 ~~10.11~~₀ Term

Term is used to model both abstract and concrete terms in SACM. Abstract Terms can be considered placeholders for concrete terms and are denoted by the inherited `isAbstract:Boolean` attribute being set true. A concrete term is denoted by `isAbstract:Boolean` being false.

Superclass

`ExpressionElement`

Attributes

`externalReference:IRI[0..*]{ordered}`

~~`externalReference:String[0..1]`~~ – an attribute recording an external reference (e.g., URI) to the object referred to by the Term

Associations

`Foundation::NamedElement`

~~`origin:Base::ModelElement[0..1]`~~ – a reference which points to the origin of the Term.

Semantics

Term class is used to model both abstract and concrete terms in SACM. Abstract Terms can be considered placeholders for concrete terms and are denoted by the inherited `isAbstract:Boolean` attribute being set true. A concrete term is denoted by `isAbstract:Boolean` being false.

The `externalReference` attribute enables the referencing of the object signified by the term (i.e., the signifier). It also provides a mechanism whereby terms can reference concepts and terms defined in other ontology and terminology models.

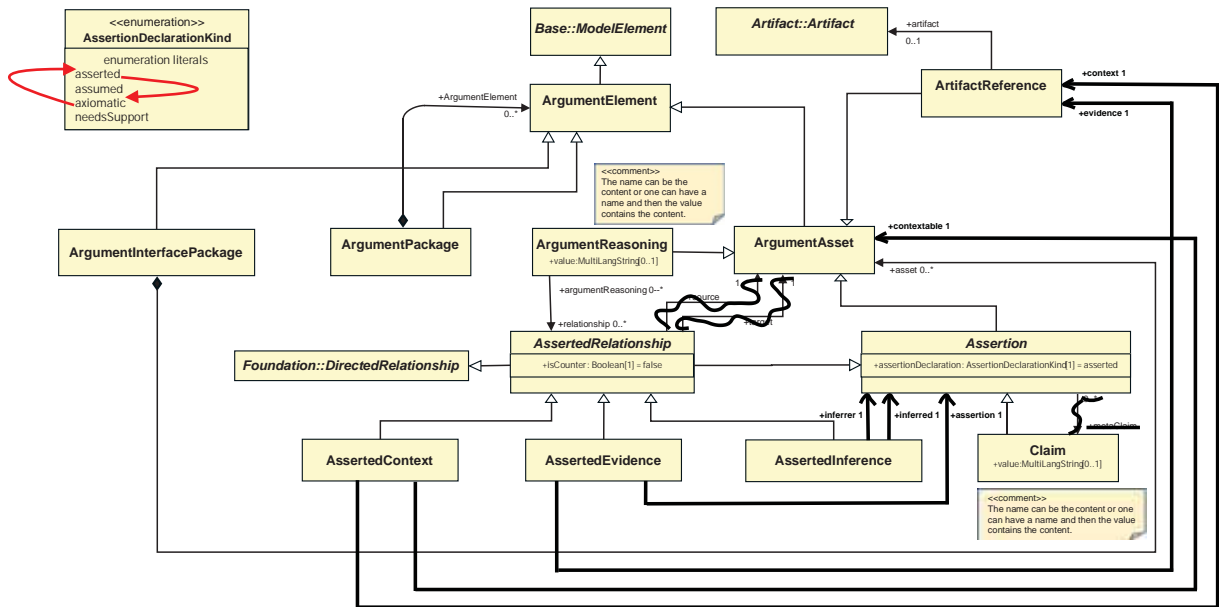
Argument

12.11 SACM Argumentation Metamodel

12.11.1 General

Argument

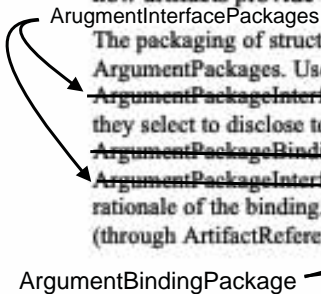
This chapter presents the normative specification for the SACM Argumentation Package. It begins with an overview of the metamodel structure followed by a description of each element.



12 Argument
Figure 12.1 - Argumentation Package Diagram

This portion of the SACM model describes and defines the concepts required to model structured arguments. Arguments are represented in SACM through explicitly representing the Claims and citation of artifacts (e.g., as evidence) (ArtifactReference), and the 'links' between these elements – e.g., how one or more Claims are asserted to infer another Claim, or how one or more artifacts (referenced by ArtifactReference) are asserted as providing evidence for a Claim (AssertedEvidence). In addition to these core elements, in SACM it is possible to provide additional description of the ArgumentReasoning associated with inferential and evidential relationships, represent counter-arguments and counter-evidence (through isCounter: Boolean), and represent how artifacts provide the context in which arguments should be interpreted (through AssertedContext).

The packaging of structured arguments into 'modular' argument packages is enabled through ArgumentPackages. Users are able to declare interfaces for their packages through the use of ArgumentPackageInterface. Within an ArgumentPackageInterface, users create citations of the argumentation elements they select to disclose to external parties. Users are able to integrate ArgumentPackages through the use of ArgumentPackageBinding. An ArgumentPackageBinding binds ArgumentPackages together by including the declared ArgumentPackageInterfaces for the ArgumentPackages, it may contain additional argument structures to provide the rationale of the binding. It is also possible within a package to cite elements contained within other argument packages (through ArtifactReference).



argument

11.2 ArgumentGroup

ArgumentGroup can be used to associate a number of ArgumentElements to a common group (e.g., representing a common type or purpose, or being of interest to a particular stakeholder).

Superclass

~~Argument~~
~~ArgumentationElement~~

Associations

~~argumentAsset:ArgumentAsset[0..*]{ordered}~~
~~argumentationElement:ArgumentationElement[0..*]~~ an optional collection of ~~ArgumentationElements~~ organised within the ArgumentGroup.

Semantics

~~ArgumentGroup can be used to associate a number of ArgumentElements to a common group (e.g., representing a common type or purpose, or being of interest to a particular stakeholder). The name and the description of the ArgumentGroup should provide the semantic for understanding the ArgumentGroup. ArgumentGroups serve no structural purpose in the formation of the argument network, nor are they meant as a structural packaging mechanism (this should be done using ArgumentPackages).~~

12 11.3 ArgumentationElement (abstract)

An ArgumentationElement is the top level element of the hierarchy for argumentation elements. ArgumentationElement extends Base::ArtifactElement. Subsequently, all argument elements are considered artifacts.

Superclass

~~Base::ModelElement~~
~~Base::ArtifactElement~~

Semantics

The ArgumentationElement is a common class for all elements within a structured argument.

12 11.4 ArgumentPackage Class

ArgumentPackage is the containing element for a structured argument represented using the SACM Argumentation Metamodel.

Superclass

ArgumentationElement

Associations

argumentationElement:ArgumentationElement[0..*] (composition) – a collection of ArgumentationElements forming a structured argument

Semantics

ArgumentPackages contain structured arguments. These arguments are composed of ArgumentAssets. ArgumentPackages elements can also be nested.

Constraints

If an ArgumentPackage has nested ArgumentPackages, then it is only allowed to contain ArgumentPackages.

12 11.5 ArgumentPackageBinding

ArgumentElement within the ArgumentPackage can be bound together by means of ArgumentPackageBinding. An ArgumentPackage can provide ArgumentPackageInterfaces, which export ArgumentationElements to be used by other ArgumentPackages. ArgumentPackageInterfaces contain citations to ArgumentationElements (e.g. an ArgumentPackageInterface may contain an 'asCited' Claim, with its 'citedElement' pointing to the Claim inside the

ArgumentPackage). An ~~ArgumentPackageBinding~~ binds the participant packages (i.e., ~~ArgumentPackageInterfaces~~) by means of structured arguments, with ~~ArgumentationElements~~ citing the contents inside the participant packages.

Superclass

ArgumentPackage

Associations

participantPackage:ArgumentPackage[2..*] - the ArgumentPackages being mapped together by the ~~ArgumentPackageBinding~~.

Semantics

ArgumentPackageBindings can be used to map resolved dependencies between the Claims of two or more ArgumentPackages.

For example, one ArgumentPackage may contain a claim that needsSupport (i.e. currently has no supporting argument). An ArgumentPackageBinding can be used to record the mapping by means of containing a structured argument linkingArgumentElements that cite the claims in question.

~~ArgumentPackageBinding~~. ~~ArgumentPackageBinding~~ is a sub type of ArgumentPackage, it is used to record the argument that connects the arguments of two or more ArgumentPackages.

An ~~ArgumentPackageBinding~~ resides within an AssuranceCasePackageBinding. It contains references, using isCitation=True to each ArgumentationElement used in its argument structure that relates ~~ArgumentationElements~~ from different ArgumentPackages.

Constraints

The participantPackages should be only ArgumentPackages

OCL: self.argumentElement

self.participantPackage->forall(pp|pp.ocllsTypeOf(Argument::ArgumentPackageInterface)) and self.argumentationElement->forall(e|e.isCitation = true and e.citedElement <> null

The ArgumentElements contained by an ~~ArgumentPackageBinding~~ must be ArgumentElement citations to ArgumentElements contained within the ArgumentPackages associated by the participantPackage association.

12.11.6 ⁵ **ArgumentPackageInterface**

ArgumentInterfacePackage
~~ArgumentPackageInterface~~ is a kind of ArgumentPackage that defines an interface that may be exchanged between users. An ArgumentPackage may declare one or more ~~ArgumentPackageInterface~~.
 ArgumentInterfacePackage

Superclass

~~ArgumentElement~~
~~ArgumentPackage~~

Associations

asset : ArgumentAsset[0..*]
 implements:ArgumentPackage[1] – a reference to the ArgumentPackage which the ~~ArgumentPackageInterface~~ declares.
 ArgumentInterfacePackage

Semantics

ArgumentInterfacePackages
~~ArgumentPackageInterface~~ can be used to declare (by means of containing ArgumentElement based citations) the ArgumentAssets contained in an ArgumentPackage that form part of the explicit, declared, interface of the ArgumentPackage.

For example, whilst an ArgumentPackage may contain many Claims, it may be desirable to create an

All Elements that are cited in a BindingPackage must be contained in either the owner of that BindingPackage or a (recursively) sibling Package of that BindingPackage

inv CitedElementsAreScoped: element->forAll(e|e.cited->notEmpty() implies e.cited->closure(g|g.owningPackage).asSet()->one(p|p=self.owningPackage))

Elements that are not either a Diagram or a BaseElement must be a citation.

inv MustBeCited: element->forAll(e|not e->oclKindOf(SACMDiagram) and not e->oclKindOf(BaseElement) implies e.isCitation=true)

ArgumentPackageInterface that cites only a subset of those claims that are intended to be mapped / used (e.g. by means of an ArgumentPackageBinding) by other ArgumentPackages. There may be more than one ArgumentPackageInterface for a given ArgumentPackage that reveal different aspects of the ArgumentPackage for different audiences.

An ArgumentPackageInterface resides inside the ArgumentPackage to which it refers. It refers to ArgumentationElements using isCitation=True that reside within the same ArgumentPackage as itself. Similar relationships exist for an ArtifactPackageInterface and for a TerminologyPackageInterface.

Constraints

~~ArgumentPackageInterfaces are only allowed with isCitation=true and CitedElement refer to ArgumentAssets within the ArgumentPackage implementation referred to by implements.~~

12.11.7 ArgumentAsset (abstract)

ArgumentAsset is the abstract base element for the elements of any structured argument represented in SACM.

Superclass

ArgumentationElement

Semantics

ArgumentAssets represent the constituent building blocks of any structured argument contained in an ArgumentPackage.

For example, ArgumentAssets can represent the Claims made within a structured argument contained in an ArgumentPackage.

12.11.8 AssertionDeclaration (Enumeration)

AssertionDeclaration provides a list of declarations which can be used to declare the state of an Assertion.

Superclass

N/A

Enumeration Literals

asserted – ~~the default enumeration literal~~ ^(default), indicating that an Assertion is asserted.

needsSupport – a flag indicating that further argumentation has yet to be provided to support the Assertion.

assumed – a flag indicating that the Assertion being made is declared by the author as being assumed to be true rather than being supported by further argumentation.

axiomatic – a flag indicating that the Assertion being made by the author is axiomatically true, so that no further argumentation is needed.

~~defeated – a flag indicating that the Assertion is defeated by counter evidence and/or argumentation.~~

~~asCited – a flag indicating that because the Assertion is cited, the AssertionDeclaration should be transitively derived from the value of the AssertionDeclaration of the cited Assertion.~~

Semantics

~~AssertionDeclaration provides a list of declarations which indicate the state of an Assertion.~~

12.11.9 ArtifactReference

ArtifactReference enables the citation of an artifact as information that relates to the structured argument.

Superclass

ArgumentAsset

~~Associations~~

~~referencedArtifactElement:Base::ArtifactElement[0..*] - reference to a collection of ArtifactElements.~~

Semantics

It is necessary to be able to cite artifacts that provide supporting evidence, context, or additional description within an argument structure. ArtifactReferences allow there to be an objectified citation of this information within the structured argument, thereby allowing the relationship between this artifact and the argument to also be explicitly declared.

12 ~~11.10~~ Assertion (abstract)

Assertions are used to record the propositions of Argumentation (including both the Claims about the subject of the argument and the structure of the Argumentation being asserted). Propositions can be true or false, but cannot be true and false simultaneously.

Superclass

ArgumentAsset

Attributes

assertionDeclaration:AssertionDeclaration[1] = asserted – the declaration indicating the state of the Assertion.

~~Associations~~

~~metaClaim:Claim[0..*] - references Claims concerning (i.e., about) the Assertion (e.g., regarding the confidence in the Assertion)~~

Semantics

Structured arguments are declared by stating claims, citing evidence and contextual information, and asserting how these elements relate to each other.

12 ~~11.11~~ Claim

Claims are used to record the propositions of any structured argument contained in an ArgumentPackage. Propositions are instances of statements that could be true or false, but cannot be true and false simultaneously.

Superclass

Assertion

Attributes

value:MultiLangString[0..1]

Semantics

The core of any argument is a series of claims (premises) that are asserted to provide sufficient reasoning to support a (higher-level) claim (a conclusion). The name can be the content or one can have a name and then the value contains the content.

~~The core of any argument is a series of claims (premises) that are asserted to provide sufficient reasoning to support a (higher-level) claim (a conclusion).~~

~~A Claim that is intentionally declared without any supporting evidence or argumentation can be declared as being assumed (i.e., assertionDeclared = assumed). It is an assumption. However, it should be noted that a Claim that is not 'assumed' (i.e., assertionDeclaration = asserted) is not being declared as false. However, there is the expectation of the provision of a supporting argument structure (e.g., it may represent part of an incomplete structure).~~

~~A Claim that is intentionally declared as requiring further evidence or argumentation can be denoted by setting +assertionDeclaration to "needsSupport".~~

~~A Claim that is being declared as axiomatically true can be denoted by setting +assertionDeclaration to "axiomatic".~~

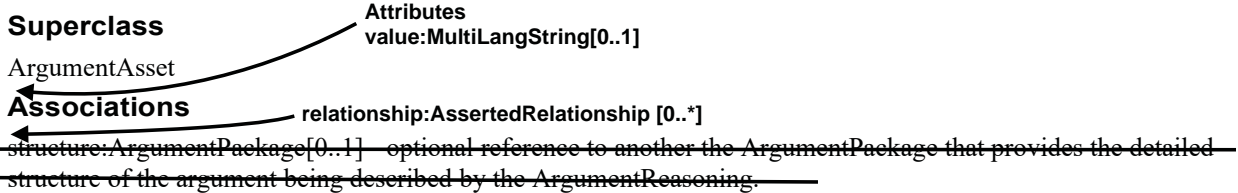
~~A Claim that is defeated by counter evidence or counter argument can be denoted by setting +assertionDeclaration to "defeated".~~

~~A Claim which cites another claim and supported by the cited claim can be denoted by setting +assertionDeclaration to "asCited".~~

The name can be the content or one can have a name and then the value contains the content.

12.11.12 ArgumentReasoning

ArgumentReasoning can be used to provide additional description or explanation of the asserted relationship. For example, it can be used to provide description of an AssertedInference that connects one or more Claims (premises) to another Claim (conclusion). ArgumentReasoning elements are therefore related to AssertedInferences, AssertedContexts, and AssertedEvidence. It is also possible that ArgumentReasoning elements can refer to other structured Arguments as a means of documenting the detail of the argument that establishes the asserted inferences, contexts, and evidence.



Semantics

The AssertedRelationship that relates one or more Claims (premises) to another Claim (conclusion), or evidence cited by an ArtifactReasoning to a Claim, may not always be obvious. In such cases ArgumentReasoning can be used to provide further description of the reasoning involved.

← The name can be the content or one can have a name and then the value contains the content.

12.11.13 AssertedRelationship (abstract)

AssertedRelationship is the abstract association class that enables the ArgumentAssets of any structured argument to be linked together. The linking together of ArgumentAssets allows a user to declare the relationship that they assert to hold between these elements.



Attributes

isCounter:Boolean[1] = false – a flag indicating whether the AssertedRelationship counters its declared purposes (e.g. setting isCounter = true for an AssertedEvidence indicates that the relationship is a counter-evidence).

Associations

~~source:ArgumentAsset[1..*] - reference to the ArgumentAsset(s) that are the source (starting point) of the relationship.~~

~~target:ArgumentAsset[1] - reference to the ArgumentAsset(s) that are the target (ending point) of the relationship.~~

~~reasoning:ArgumentReasoning[0..1] - an optional reference to the a description of the reasoning underlying the AssertedRelationship.~~

Semantics

In SACM, the structure of an argument is declared through the linking together of primitive ArgumentAssets. For example, a sufficient inference can be asserted to exist between two claims (“Claim A implies Claim B”) or sufficient evidence can be asserted to exist to support a claim (“Claim A is evidenced by Evidence B”). An inference asserted between two claims (A – the source – and B – the target) denotes that the truth of Claim A is said to infer the truth of Claim B.

12.11.14 AssertedInference

AssertedInference association records the inference that a user declares to exist between one or more Assertion (premise) and another Assertion (conclusion). It is important to note that such a declaration is itself an assertion on behalf of the user.

Superclass

AssertedRelationship

Semantics

Associations

+inferred:Assertion [1] - the inferred must be undefeated in order for the inferred to be undefeated

+inferred:Assertion [1] - the inferred must be undefeated in order for the inferred to be undefeated

The core structure of an argument is declared through the inferences that are asserted to exist between Assertions (e.g., Claims). For example, an AssertedInference can be said to exist between two claims (“Claim A implies Claim B”). An

{redefines source}

{redefines target}

AssertedInference between two claims (A – the source – and B – the target) denotes that the truth of Claim A is said to infer the truth of Claim B.

12.11.15¹⁴ ~~11.15~~ AssertedEvidence

AssertedEvidence association records the declaration that one or more artifacts of Evidence (cited by ArtifactReference) provide information that helps establish the truth of a Claim. It is important to note that such a declaration is itself an assertion on behalf of the user. The artifact (cited by an ArtifactReference) may provide evidence for more than one Claim.

Superclass

AssertedRelationship

Associations

+assertion:Assertion [1] - the assertion is that the evidence for this Assertion is enough to make the AssertedEvidence undefeated.

+evidence:ArtifactReference [1] - the assertion is that the evidence for this Assertion is enough to make the AssertedEvidence undefeated.

Semantics

Where evidence (cited by ArtifactReference) exists that helps to establish the truth of a Claim in the argument, this relationship between the Claim and the evidence can be asserted by an AssertedEvidence association. An AssertedEvidence association between an artifact cited by an ArtifactReference and a Claim (A – the source evidence cited – and B – the target claim) denotes that the evidence cited by A is said to help establish the truth of Claim B.

Constraints

The source of AssertedEvidence relationships must be ArtifactReference.

OCL:

```
self.source->forall(s|s.ocllsTypeOf(ArtifactReference))
```

12.11.16¹⁵ ~~11.16~~ AssertedContext

AssertedContext can be used to declare that the artifact cited by an ArtifactReference(s) provides the context for the interpretation and scoping of a Claim or ArgumentReasoning element. In addition, the AssertedContext can be used to declare a Claim asserted as necessary context (i.e. a precondition) for another Assertion or ArgumentReasoning.

Superclass

AssertedRelationship

Associations

+context:ArtifactReference [1] - the context provides further clarification or constraint of the contexttable

+contexttable:ArgumentAsset [1] - the context provides further clarification or constraint of the contexttable

Semantics

Contextual information often needs to be cited in order to make clear the interpretation and scope of an Assertion and supporting argumentation. For example, a Claim can be said to be valid only in a defined context (“Claim A is asserted to be true only in a context as defined by the ArtifactReference B or conversely ArtifactReference B is the asserted context for Claim A”).

Contextual Claims often need to be cited as preconditions for an Assertion. For example, a Claim may be asserted only in the context of another claim (“Claim A is asserted to be true only in a context where Claim B is true”).

12.11.17¹⁶ ~~11.17~~ AssertedArtifactSupport

AssertedArtifactSupport records the assertion that one or more artifacts support another artifact.

Superclass

AssertedRelationship

Semantics

The truth of the assertions associated with an artifact are supported by the assertions that are associated with one or more other artifacts. Note: this can be an ambiguous relationship if the nature of these Assertions is unclear. In such cases, it would be clearer to declare explicit AssertedInferences between Claims drawn out from the ArtifactReference.

Constraints

The source and target of AssertedArtifactSupport must be of type ArtifactReference.

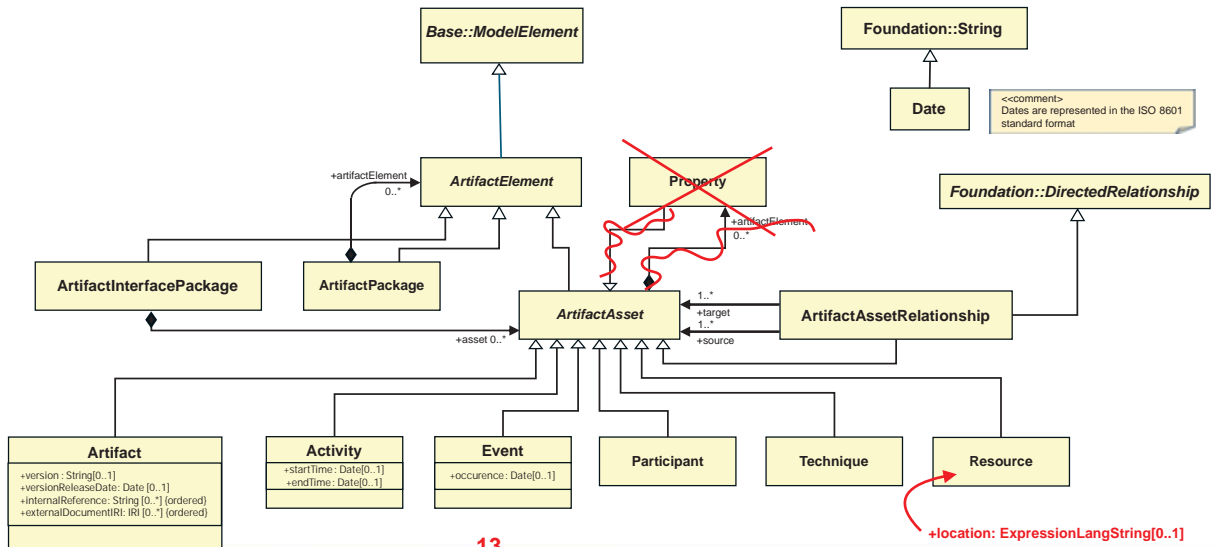
12.11.18¹⁷ ~~11.18~~ AssertedArtifactContext

AssertedArtifactContext records the assertion that one or more artifacts provide context for another artifact.

13.12 Artifact Classes

13.12.1 General

This chapter presents the normative specification for the SACM Artifact Package. It begins with an overview of the metamodel structure followed by a description of each element.



13
Figure 13.1 - Artifact Package Diagram

Artifacts correspond to the main evidentiary elements of an assurance case. By means of assertions (AssertedEvidence with isCounter = true/false), artifacts can be referenced (using ArtifactReferences) as supporting claims and arguments.

In general, artifacts are managed when the corresponding objects are available. For example, a test case is linked to the requirement that validates once the test case has already been created. However, artifact management might also require the specification of patterns (or templates) in order to allow a user, for instance, to indicate that a given artifact must be created but it has not yet. A common scenario of this situation corresponds to the process during which a supplier and a certifier have to agree upon the artifacts that the supplier will have to provide as assurance evidence for a system. As a result of this process, artifact patterns could be specified, and such patterns would need to be made concrete during the lifecycle of the system. Artifact patterns are specified by means of the attribute 'isAbstract' (SACMElement). For example, a supplier and a certifier might agree upon the need for maintaining a hazard log during a system's lifecycle. Such a hazard log would initially be modeled as an Artifact that is abstract. Once created, the value of this attribute of the hazard log would be 'false'. The specification of artifact patterns also facilitates their reuse, as the corresponding artifacts might have to be created in the scope of more than one assurance case effort. Using again hazard logs as an example, their structure might be the same for several systems, thus all the corresponding hazard logs might be based on a same abstract Artifact.

When made concrete, an Artifact can relate to many different types of information necessary for developing confidence in the Artifact and thus for assurance purposes. Such information can be regarded as meta-data or provenance information about an Artifact, provides information about its management, and is specified with the rest of specializations of ArtifactAsset. Using a design specification as an example, properties (Property) could be specified regarding its quality (completeness, consistency...), and it would have a lifecycle with events such as its creation and modifications. The specification could be created by using UML (Technique) in an Activity named 'Specify system design', stored in a Resource corresponding to a diagram created with some modeling tool, and later used as input for another Activity called 'Verify system design'. A given person (Participant) playing the role of system designer could

be the owner of the design specification, which would also relate to other artifacts: the requirements specification that satisfies, the architecture that implements, its verification report, etc. Associations between Artifacts and Activities /Events/Participants/ Resources/Techniques can be recorded by means ArtifactAssetRelationships.

13.12.2 ArtifactPackage

ArtifactPackage is the containing element for artifacts involved in a structured assurance case.

Superclass

Base::ArtifactElement

Associations

artifactElement:Base::ArtifactElement[0..*] (composition) – a collection of ArtifactElements forming an artifact package in a structured assurance case.

Semantics

ArtifactPackages contain ArtifactElements that represent the artifact forming part of a structured assurance case. ArtifactPackages can also be nested.

12.3 ArtifactGroup

ArtifactGroup can be used to associate a number of ArtifactElements to a common group (e.g., representing a common type or purpose, or being of interest to a particular stakeholder).

Superclass

Base::ArtifactElement

Associations

artifactElement:ArtifactElement[0..*] – an optional collection of ArtifactElements organised within the ArtifactGroup.

Semantics

ArtifactGroup can be used to associate a number of ArtifactElements to a common group (e.g., representing a common type or purpose, or being of interest to a particular stakeholder). The name and the description of the ArtifactGroup should provide the semantic for understanding the ArtifactGroup. ArtifactGroups serve no structural purpose in the formation of the argument network, nor are they meant as a structural packaging mechanism (this should be done using ArtifactPackage).

13.12.4² ArtifactPackageBinding

The ~~ArtifactPackageBinding~~ is a sub type of ~~ArtifactPackage~~ used to record ~~ArtifactAssetRelationships~~ between the ~~ArtifactAssets~~ of two or more ~~ArtifactPackages~~.

Superclass

~~ArtifactElement~~
~~ArtifactPackage~~

Associations

participantPackage:ArtifactPackage[2..*] - the ~~ArtifactPackages~~ containing the ~~ArtifactAssets~~ being related together by the ~~ArtifactPackageBinding~~.

Semantics

~~ArtifactPackageBindings~~ can be used to map dependencies between the cited ~~ArtifactAssets~~ of two or more ~~ArtifactPackages~~. For example, a binding could be used to record a 'derivedFrom' ~~ArtifactAssetRelationship~~ between the ~~ArtifactAsset~~ of one package to the ~~ArtifactAsset~~ of another. ~~An ArtifactPackageBinding resides within an~~

~~AssuranceCasePackageBinding. It contains references, using isCitation=True to each ArtifactAsset needed and defines relationships among ArtifactAssets from different ArtifactPackages.~~

Constraints

~~ArtifactBindingPackages
ArtifactPackageBindings must only contain ArtifactAssetRelationships with source and target Artifacts, with isCitation = true citing ArtifactAssets contained within the ArtifactPackages associated by participantPackage.~~

13.12.5⁴ ArtifactPackageInterface

~~ArtifactInterfacePackage
ArtifactPackageInterface is a kind of ArtifactPackage that defines an interface that may be exchanged between users. An ArtifactPackage may define one or more ArtifactPackageInterfaces.~~

Superclass

~~ArtifactElement
ArtifactPackage~~

Associations

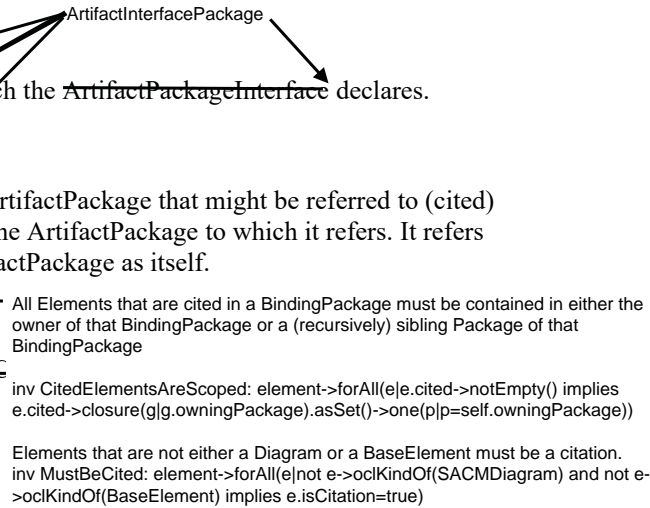
~~+asset : ArtifactAsset[0..*]
implements:ArtifactPackage[1] - a reference to the ArtifactPackage which the ArtifactPackageInterface declares.~~

Semantics

~~ArtifactPackageInterface enables the declaration of the elements of an ArtifactPackage that might be referred to (cited) in another ArtifactPackage. An ArtifactPackageInterface resides inside the ArtifactPackage to which it refers. It refers to ArtifactAssets using isCitation=True that reside within the same ArtifactPackage as itself.~~

Constraints

~~ArtifactPackageInterfaces are only allowed to contain Artifacts with +isCitation=True
ArtifactPackage with which this ArtifactPackageInterface is associated.~~



13.12.6⁵ ArtifactAsset (abstract)

ArtifactAsset represents the artifact-specific pieces of information of an assurance case, in contrast to the argument-specific pieces of information.

Superclass

~~ArtifactConcept
Base::ArtifactElement~~

Association

~~property:Property[0..*] (composition) - an optional collection of Property(ies) which enable the specification of the characteristics of an ArtifactAsset.~~

Semantics

Information about artifacts is essential for any assurance case. The artifacts correspond, for instance, to the evidence provided in support of the arguments and claims of an assurance case. It is also important to have access to related pieces of information such as the provenance of an artifact, its lifecycle, and its properties. All this information might have to be consulted for developing confidence in the validity of an assurance case.

13.12.7⁶ Artifact

Artifact represents the distinguishable units of data used in a structured assurance case.

Superclass

ArtifactAsset

internalReference : String [0..*] {ordered} - refence can be further restricted to some internal part for what the Artifact represents.
externalDocumentIRI : IRI [0..*] {ordered} - IRI to the external document that this Artifact element represents.
IRIs are ordered to allow preferred references appearing earlier in the ordering. All references should be to the same external document.

Attributes

version: String[0..1] - the version of the artifact
versionReleaseDate: Date[0..1]
~~date: date[0..1] - the date on which the artifact was created.~~

Semantics

Artifacts correspond to the main evidentiary support for the arguments and claims of an assurance case: an Artifact can play the role of evidence of a Claim (AssertedEvidence), or of counterevidence (AssertedCountedEvidence with isCounter = true). An Artifact can take several forms, such as a diagram, a plan, a report, or a specification, both in electronic (e.g., a pdf file) or physical (e.g., a paper document) formats. Typical examples of Artifacts include system lifecycle plans, dependability (e.g., safety) analysis results, system specifications, and V&V results.

~~13.12.8~~⁷ Property

~~Property enables the specification of the characteristics of an Artifact.~~

~~Superclass~~

~~ArtifactAsset~~

~~Semantics~~

~~An Artifact can have different, specific characteristics independent of the argumentation structure in which the Artifact is used. Some can be objective (e.g., the result of a test case execution, as passed or not passed) and others can be based on a person's judgement (e.g., regarding a quality aspect of a report).~~

~~13.12.9~~⁸ Event

Event enables the specification of the events in the lifecycle of an Artifact.

Superclass

ArtifactAsset

Attributes

date: ~~date~~^Ddate[0..1] - the date on which the Event occurred.

Semantics

Artifacts change during their lifecycle, and different types of happenings can occur at different moments: creation, modification, revocation... Events serve to maintain a history log of an Artifact, and can be consulted to know how an Artifact has evolved and to develop confidence in its adequate management.

~~13.12.10~~⁸ Resource

Resource corresponds to the tangible objects representing an Artifact.

Superclass

ArtifactAsset

Attributes

location: ExpressionLangString[0..1]

~~location: Base::MultiLangString (composition)~~ - the path or URL specifying the location of the Resource, can be in multiple languages.

Semantics

Artifacts are located and accessible somewhere, usually in the form of some electronic file for an assurance case. Such information is specified by means of Resources.

13 ~~12.11~~ Activity

Activity represents units of work related to the management of ArtifactAssets.

Superclass

ArtifactAsset

Attributes

startTime: ~~D~~date[0..1] - time when the activity started.

endTime: ~~D~~date[0..1] - time when the activity ended.

Semantics

The Artifacts used in an assurance case are the result of and managed via the execution of processes, which consist of Activities: specification of requirements, design of the system, integration of system components, etc. ArtifactActivityRelationships can be used to specify the relationship between Activities and Artifacts. Activities can, for instance, be described as using a given Artifact as input or producing an Artifact as output. Activities can be related to one another using ActivityRelationships (e.g., 'preceding'). The purpose of an activity can be specified in its description.

13 ~~12.12~~ Technique

Technique represents techniques associated with Artifacts (e.g., associated with the creation, inspection, review or analysis of an Artifact).

Superclass

ArtifactAsset

Semantics

Artifacts are created, or managed from a more general perspective, via some method whose use results in specific characteristics for the Artifacts. For example, the use of UML (as a Technique) for designing a system results in a design specification with a set of UML diagrams that could represent static and dynamic internal aspects of the system.

13 ~~12.13~~ Participant

Participant enables the specification of the parties involved in the management of ArtifactAssets.

Superclass

ArtifactAsset

Semantics

Different parties can participate in an assurance case effort, such as specific people, organizations, and tools.

13 ~~12.14~~ ArtifactAssetRelationship

ArtifactAssetRelationship enables the ArtifactAssets of a structured assurance case to be linked together. The linking together of ArtifactAssets allows a user to specify that a relationship exists between the assets.

Superclass

ArtifactAsset ← , Foundation::DirectedRelationship

Associations

source:ArtifactAsset[1..*] - the source of the ArtifactAssetRelationship

target:ArtifactAsset[1..*] - the target of the ArtifactAssetRelationship

Semantics

An ArtifactAsset can be related to other ArtifactAssets. This kind of information is specified by means of ArtifactAssetRelationships name and description of the ArtifactAssetRelationship can be used to describe the semantics of the ArtifactAssetRelationship.

~~12.14~~¹³ **Date**

Date is a type, whose primitive type is String and represented in the ISO 8601 standard format