

## 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.

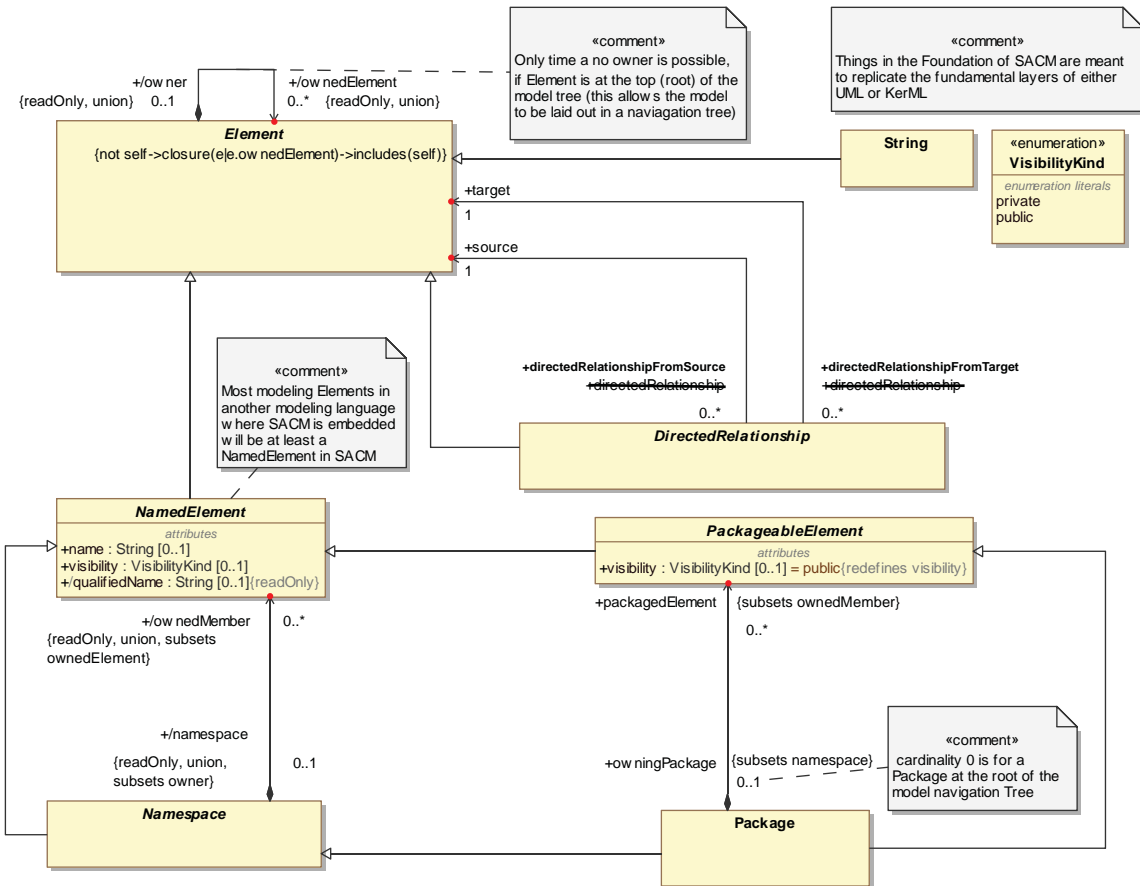


Figure 8.1 - Foundation Class

#### 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*.

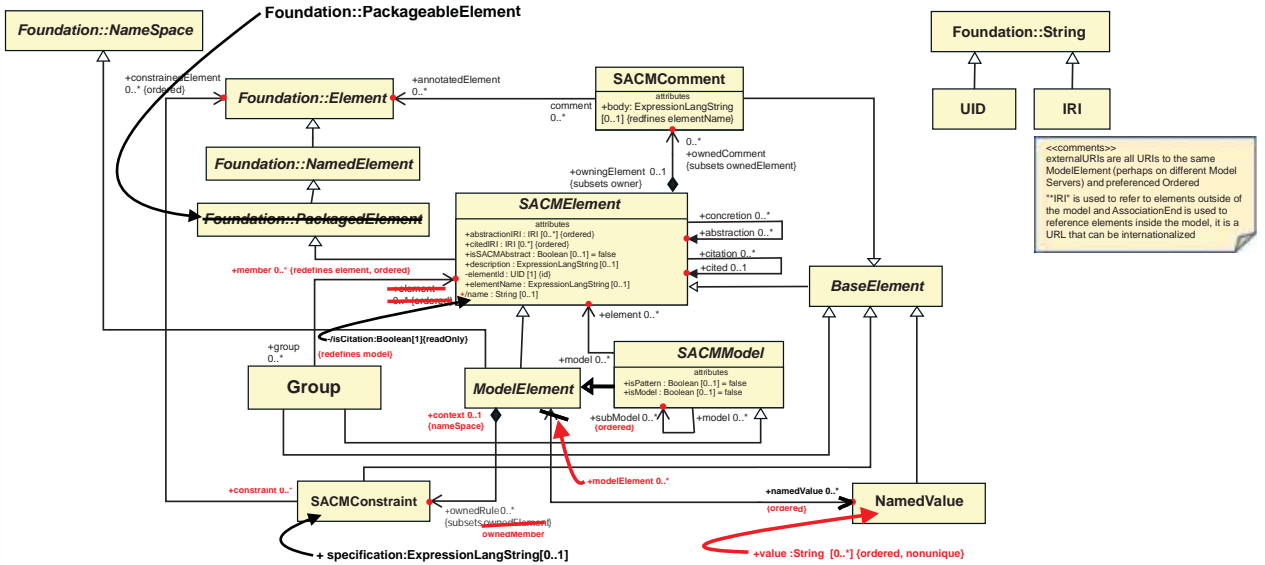
**target**  
~~target~~ : Element [1]

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

# 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.

**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**

lang:String[0..\*] {ordered} – a field to indicate the language used in the string.

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 the lang attribute.

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

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

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()

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

**Superclass**

, Foundation::NameSpace

SACMElement

~~Associations~~

~~Attributes~~

~~ownedRule:SACMConstraint[0..\*] (subsets ownedElement) – (subsets ownedMember)~~

~~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 {ordered} NamedValues NamedValues~~

~~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

## 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~~ <sup>an Element</sup>. For example a note may include a number of explanatory comments.

**Superclass** BaseElement

Attributes body:ExpressionLangString[0..1]{redefines elementName}

~~SACMCoreElement~~

~~body:ExpressionLangString[0..1]~~

~~UtilityElement~~

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**

NamedValue

attributes

This class represents a ~~simple key/value pair~~ 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~~

Attributes

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

#### Associations

~~key:MultiLangString[1] (composition) – the key of the TaggedValue.~~

#### 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 ← , SACMModel

#### Association End

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

~~member:SACMElement [0..\*] {redefines element, ordered} – a collection of SACMElements that compromise a group~~

### 9.11 ~~9.10~~ IRI

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

#### 9.11 SACMModel

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

Supertype ~~String~~

Supertype ModelElement

Foundation::String

#### 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.

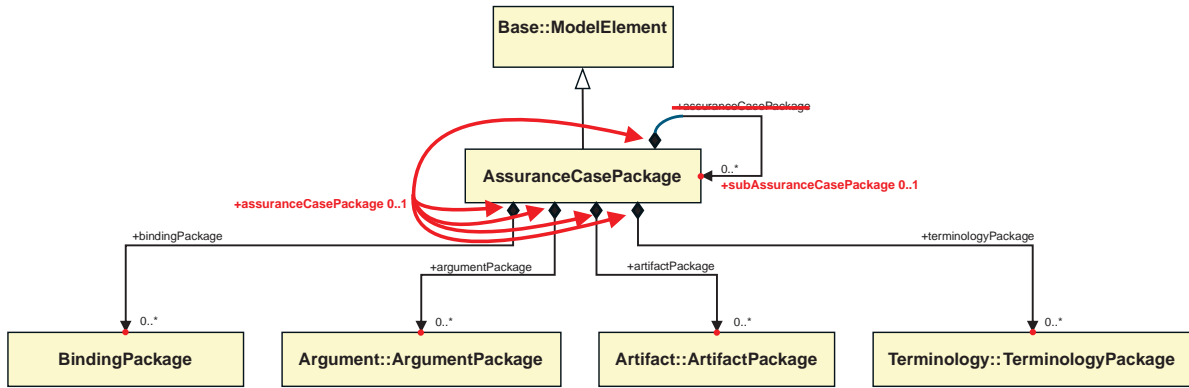
10 ~~9~~

# 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

~~subAssuranceCasePackage~~

~~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[1] – 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.oclIsTypeOf(AssuranceCasePackageInterface)) and
self.argumentPackage->forall(ap|ap.oclIsTypeOf(Argumentation::ArgumentPackageInterface))
and self.artifactPackage->forall(ap|ap.oclIsTypeOf(Artifact::ArtifactPackageInterface)) and
self.terminologyPackage->forall(tp|
tp.oclIsTypeOf(Terminology::TerminologyPackageInterface))
```

## 10.3

### ~~9.4~~

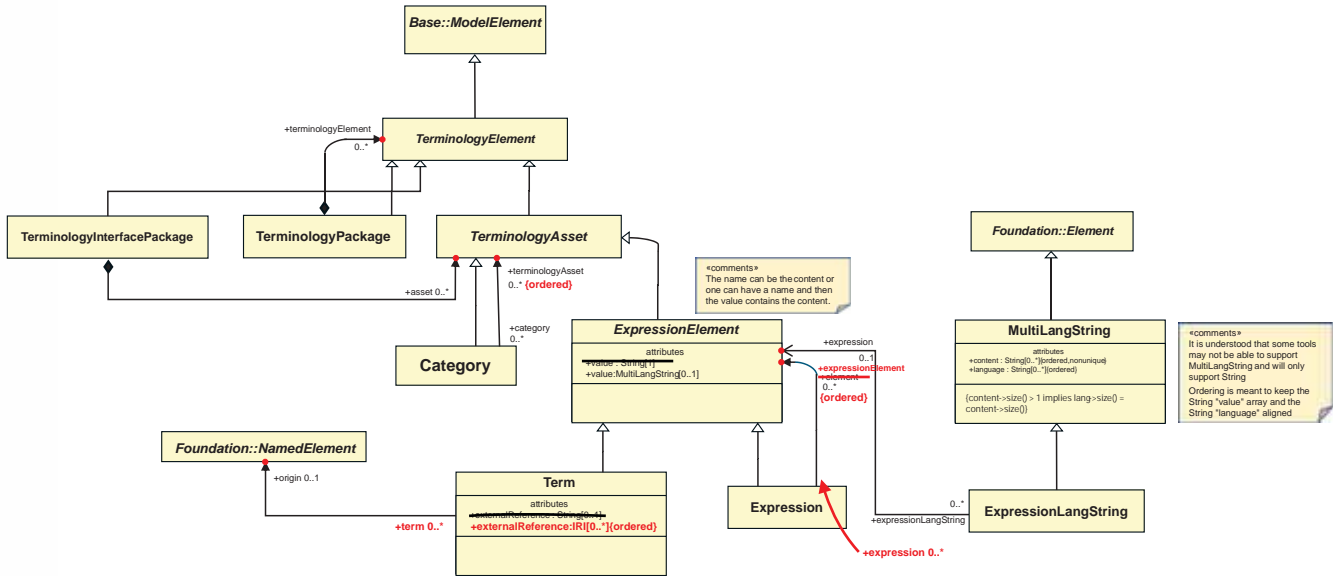
## ~~AssuranceCasePackageBinding~~

Sub-packages within any <sup>BindingPackage</sup> AssuranceCasePackage can be bound together by means of <sup>a BindingPackage.</sup> 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).

## ~~10.6 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 ~~10.7~~<sup>6</sup> 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 ~~10.8~~<sup>7</sup> Category

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

### Superclass

TerminologyAsset

### Semantics

Association Ends  
~~expressionElement:ExpressionElement~~[0..\*]{(ordered)} – elements contained in the Category  
 terminologyAsset:TerminologyAsset

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.

## 10.9 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.

### 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.

## 10.10 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..\*] – 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:

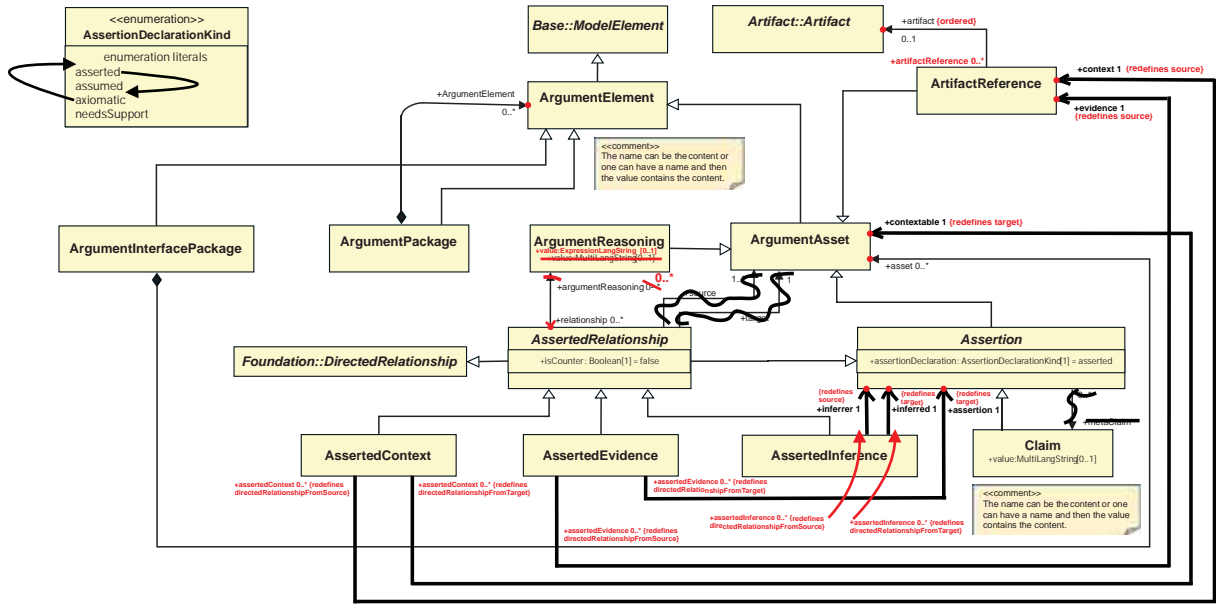
self.isAbstract = false implies self.element->forall(expr|expr.isAbstract = false).

# Argument

## 12.11 SACM Argumentation Metamodel

### 12.11.1 General

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.

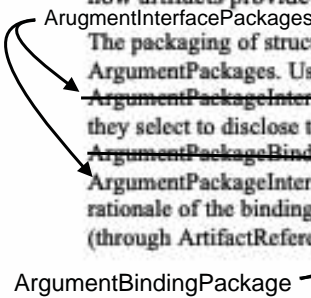


12 Argument

Figure 11.1 - 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).

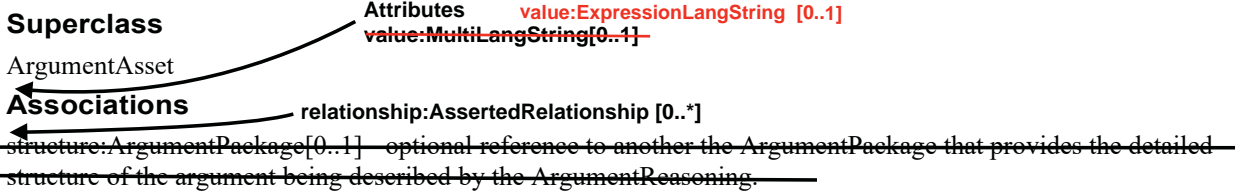


argument

# 11

## 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

## 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.~~

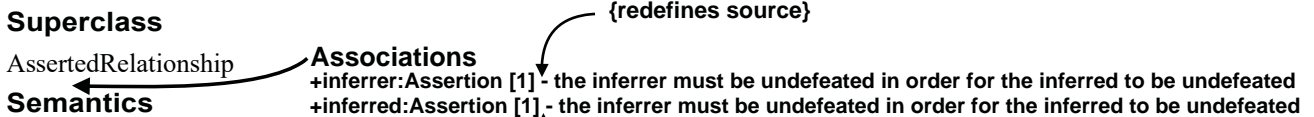
### Semantics

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.

# 13

## 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.



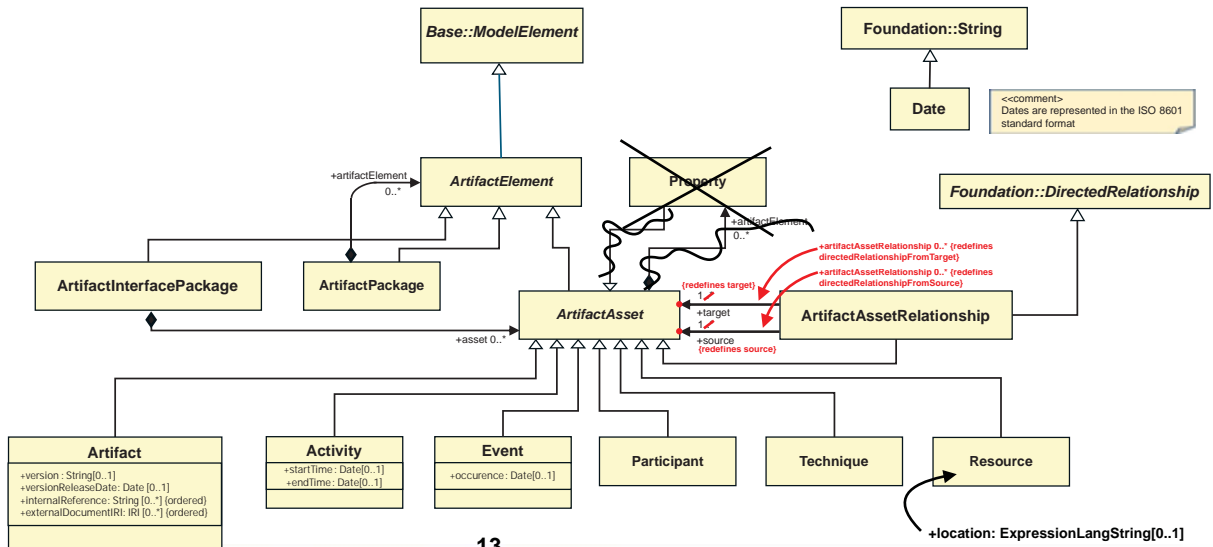
The core structure of an argument is declared through the inferences that are asserted to exist between Assertions (e.g., Claims). For example, an AssertedInference can be said to exist between two claims (“Claim A implies Claim B”). An

{redefines target}

# 13.1.2 Artifact Classes

## 13.1.1 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

## 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.

13<sup>3</sup>  
~~12.14~~ Date

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