

8 Structured Assurance Case Base Classes

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.

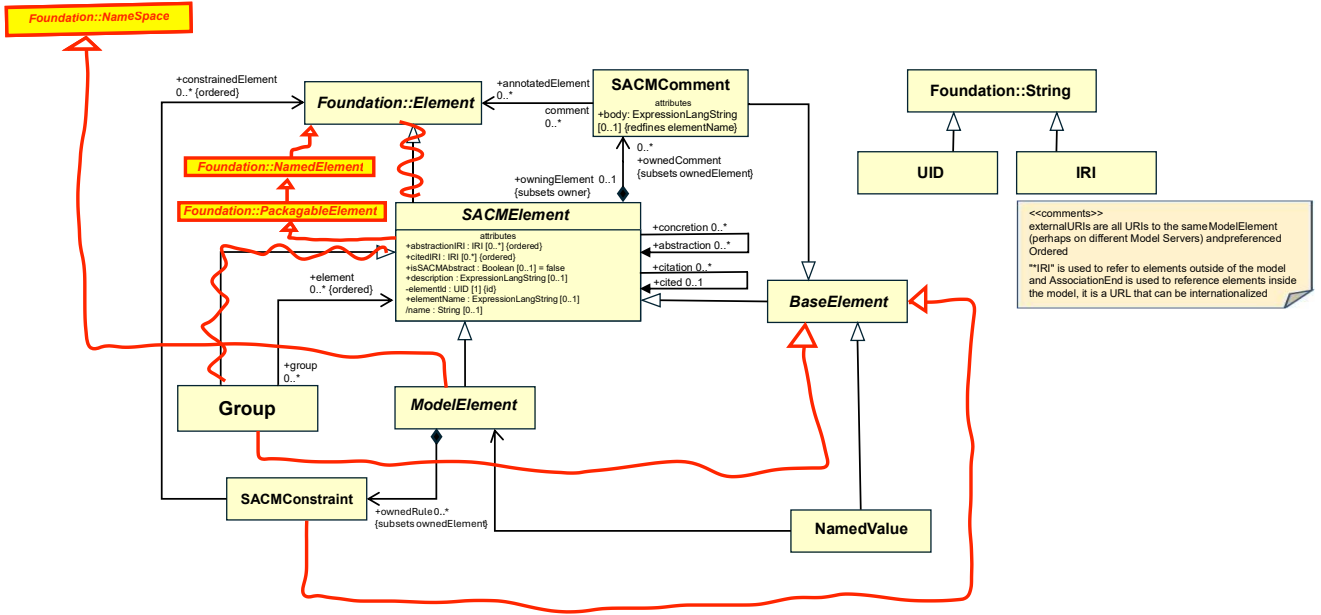


Figure 8.1 - Structured Assurance Case Base Classes Diagram

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.

Superclass

Foundation::PackagableElement

~~MOF:Element~~

/isCitation: Boolean[1]=false;

Attributes

-elementId: UID[1]{id}

-gid: String[0..1] – a unique identifier that is unique within the scope of the model instance

/name: String[0..1] – the derived name of the SACMElement, which is the first element of elementName.

~~isCitation[0..1]=false – a flag to indicate whether the SACMElement cites another SACMElement.~~

isSACMAbstract: Boolean

~~isAbstract[0..1]=false – a flag to indicate whether the SACMElement is considered to be abstract. For example, this can be used to indicate whether an element is part of a pattern or template.~~

Associations:

elementName: ExpressionLangString[0..1] – the multi-language name of this SACMElement.

~~citedElement: SACMElement[0..1] – a reference to another SACMElement that the SACMElement cites~~

abstraction

~~abstractForm: SACMElement[0..1] – an optional reference to another abstract SACMElement to which this concrete SACMElement conforms.~~

abstractionIRI: IRI[0..*] {ordered} – An IRI reference to an abstract element in another model.

description: ExpressionLangString [0..1] – the name of the SACMElement.

citedIRI: IRI[0..*] {ordered} – An IRI reference to an cited element in another model.

Semantics

Association Ends

ownedComment: SACMComment [0..*] – SACMComments owned by this SACMElement

All the elements of a structured assurance case effort created with SACM correspond to a SACMElement.

Constraints:

inv DerivedName : elementName->notEmpty() and elementName->content->notEmpty()
implies name=elementName->content->first()

~~If citedElement is populated, isCitation must be true. OCL: self.citedElement <> null implies self.isCitation = true~~

~~When +abstractForm is used to refer to another SACMElement, +isAbstract of the SACMElement is false, and the +isAbstract of the referred SACMElement should be true. The referred SACMElement should be of the same type of the SACMElement. If ImplementationConstraints are expressed on the referred SACMElement, the SACMElement should satisfy these ImplementationConstraints.~~

~~{abstraction->notEmpty() implies abstraction.isSACMAbstract=true,
cited->notEmpty() implies oclTypeOf(cited),
elementName->notEmpty() and elementName->content->notEmpty() implies name=elementName->content->first(), cited->notEmpty() and
cited.isSACMAbstract=true implies isSACMAbstract=true}~~

~~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.~~

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~~

Semantics

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

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

LangIfContentMoreThan1

If content has more than one value, then lang must have string representing the languages of the values.

~~the content->size() > 1 implies lang->size() = content-~~

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

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

Superclass

, Foundation::NameSpace

SACMElement

Associations

Attributes

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

~~ownedRole: SACMConstraint [0..*] – SACMConstraints~~

~~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

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**

SACMComment

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

Superclass BaseElement

~~SACMCoreElement~~

~~UtilityElement~~

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**

NamedValue

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~~

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 BaseElement

~~SACMElement~~

Association End

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

10 Structured Assurance Case Terminology Classes

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

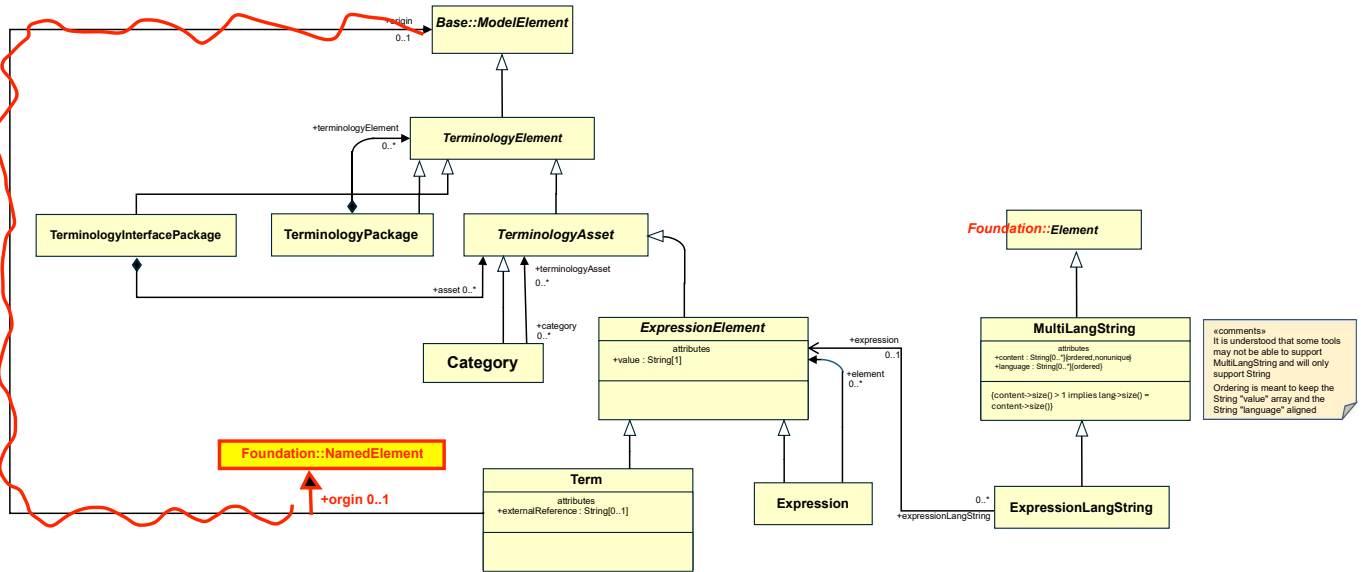


Figure 10.1 - Terminology Class Diagram

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.

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::ArtifactElement

Semantics

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

10.1~~1~~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: 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

11 SACM Argumentation Metamodel

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.

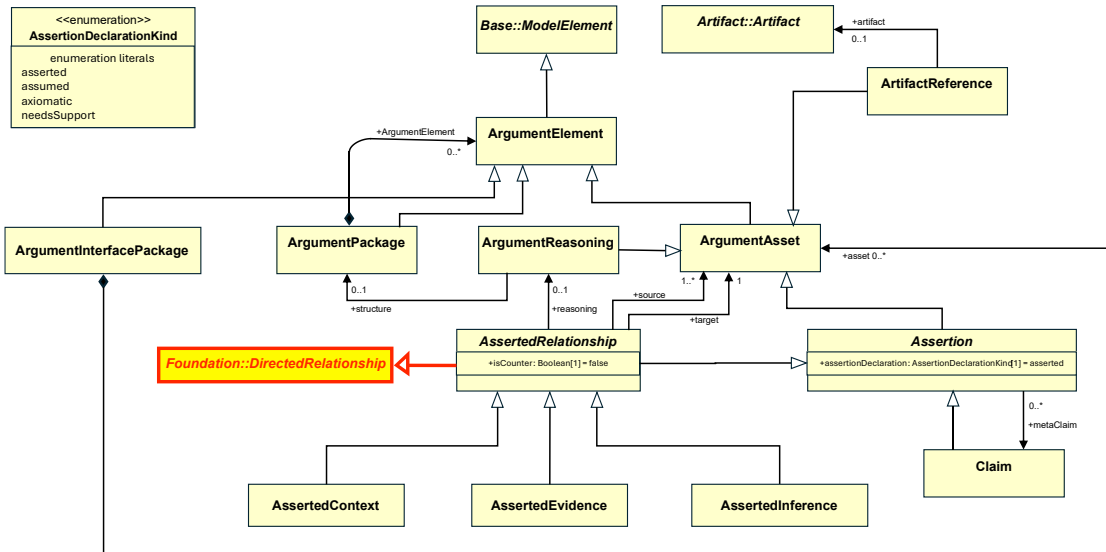


Figure 11.1 - Argumentation Package Diagram

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`).

ArgumentInterfacePackages

ArgumentBindingPackage

argument

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.

Superclass

ArgumentAsset

Associations

structure:ArgumentPackage[0..1] - optional reference to another the ArgumentPackage that provides the detailed structure of the argument being described by the ArgumentReasoning.

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.

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.

Superclass

Assertion , Foundation::AssertedRelationship

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.

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

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

12 Artifact Classes

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.

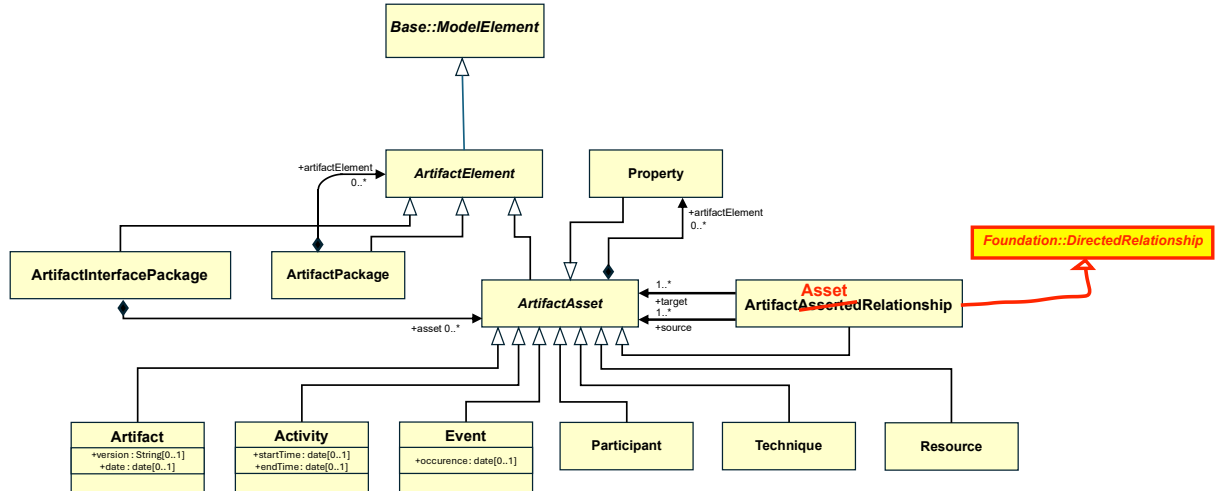


Figure 12.1 - Artifact Package Diagram

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

12.11⁰ Activity

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

Superclass

ArtifactAsset

Attributes

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

endTime: 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.

12.12¹ 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.

12.13² 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.

12.14³ 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