8.6 ModelElement (abstract)
ModelElement is the base element for the majority of modeling elements.

**Superclass**
SACMElement

**Associations**
- 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.
- 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.

**Constraints**
ImplementationConstraints should only be specified if +isAbstract is true. OCL: self.implementationConstraint->size() > 0 implies self.isAbstract = true

8.7 UtilityElement (abstract)
UtilityElement is an abstract element for a number of utility elements.

**Superclass**
SACMElement

**Associations**
- expression: Expression [1] – the expression object containing the value of the UtilityElement (see Terminology section 10).

**Semantics**
UtilityElement supports the specification of additional information for a ModelElement.

8.8 ImplementationConstraint
This class 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**
UtilityElement

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

**Constraints**
ImplementationConstraints should only be specified if isAbstract is true.

8.9 Description
This class specifies 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.

8.10 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.
8.11 Description

This class specifies 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.

8.12 Note

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

8.12 TaggedValue

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.
9 Structured Assurance Case Packages

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.

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).

9.2 ArtifactElement (abstract)

ArtifactElement is an abstract class that serves as a parent class for Artifacts and AssuranceCasePackage elements.

- Superclass
  - ModelElement

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

9.3 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).
9.3 **AssuranceCasePackageInterface**

AssuranceCasePackageInterface is a kind of AssuranceCasePackage that defines an interface that may be exchanged between users. An AssuranceCasePackage may declare one or more ArtifactPackageInterfaces.

**Superclass**

AssuranceCasePackage

**Associations**


**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 provided).

**Constraints**

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

**OCL:**

\[
\text{self.assuranceCasePackage->forall(acp|acp.oclIsTypeOf(AssuranceCasePackageInterface)) and self.argumentPackage->forall(ap|ap.oclIsTypeOf(Argumentation::ArgumentPackageInterface)) and self.artifactPackage->forall(ap|ap.oclIsTypeOf(Artifact::ArtifactPackageInterface)) and self.terminologyPackage->forall(tp|tp.oclIsTypeOf(Terminology::TerminologyPackageInterface))}
\]

9.4 **AssuranceCasePackageBinding**

Sub-packages within the AssuranceCasePackage can be bound together by means of AssuranceCasePackageBindings. AssuranceCasePackageBindings bind the participant packages by means of ArgumentPackageBindings/TerminologyPackageBindings/ArtifactPackageBindings elements that bind the contained packages of the participant packages.

**Superclass**

AssuranceCasePackage

**Associations**

participantPackage:AssuranceCasePackage[2..*] – references to AssuranceCasePackages which the AssuranceCasePackageBinding binds together.

**Semantics**

AssuranceCasePackageBinding binds peer AssuranceCasePackages together to indicate the relationship between these AssuranceCasePackages. The bindings between AssuranceCasePackages consist of the bindings of the packages (i.e.
10.7 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).

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

**Superclass**
TerminologyAsset

**Semantics**
Terms and ExpressionElements can be said to belong to Categories. Categories can group Terms, Expressions, or a mixture of both. 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

**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. All ModelElements contain a Description whose value is provided by means of an Expression.

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 attribute being set true. A concrete expression (denoted by isAbstract being false) is one that has a literal string value and references only concrete ExpressionElements.

**Superclass**
ArtifactElement

**Attributes**
isAbstract: Boolean

**Associations**
element: ExpressionElement [0..*] – an optional reference to other ExpressionElements forming part of the StructuredExpression.

**Semantics**
Expressions are used to model phrases and sentences. These are defined using the value attribute. The value attribute can be a simple literal string. Alternatively, the expression can also be defined (using the value string) as a production rule involving other ExpressionElements. In this case, the value string 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).
(the +value property), these should be referenced by name within the +value property. Where the +value property references ExpressionElement by name, these ExpressionElements should be associated (using the +element property) with Expression. A concrete expression should have references to only concrete ExpressionElements.

OCL: self.isAbstract = false implies self.element->forAll(expr|expr.isAbstract = false).

**Constraints**

Where an Expression has associated ExpressionElements these should be referenced by name within the Expression.value. Where an Expression.value references ExpressionElements by name, these ExpressionElements should be associated (using the element association) with Expression.

**10.11 Term**

The **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` attribute being set true. A concrete term is denoted by `isAbstract` being false.

**Attributes**

- **value**: String – An attribute recording the value of the Term.
- **externalReference**: String – An attribute recording an external reference (e.g., URI) to the object referred to by the Term.

**Superclass**

ExpressionElement

**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` attribute being set true. A concrete term is denoted by `isAbstract` being false.

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

**Associations**

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

11.2 Argumentation Class Diagram

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, an optional declaration of an interface for the package (ArgumentPackageInterface) that organizes a specific selection of the ArgumentElements contained within the package, and the ability to link (by means of an argument) two or more argument packages (through an ArgumentPackageBinding). It is also possible within a package to cite elements contained within other argument packages (through AssertedContext).
packages (through an ArgumentPackageBinding). It is also possible within a package to cite elements contained within other argument packages (through ArtifactReference).

11.2 **11.3 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**
ArgumentationElement

**Associations**
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).

11.3 **11.4 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::ArtifactElement

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

11.4 **11.5 ArgumentPackage Class**

The ArgumentPackage Class is the container class for a structured argument represented using the SACM Argumentation Metamodel.

**Superclass**
ArgumentationElement

**Associations**
argumentAsset:ArgumentAsset[0..*] (composition) – a collection of ArgumentAssets forming a structured argument
argumentPackage:ArgumentationPackage[0..*] (composition) – a collection of ArgumentPackages
interface:ArgumentationPackage[0..*] (composition) – a reference to the declared interface for the ArgumentPackage.

**Semantics**
ArgumentPackages contain structured arguments. These arguments are composed of ArgumentAssets. ArgumentPackages can be nested, and can contain citations (references) to other ArgumentPackages. For example, arguments can be established through the composition of Claims (propositions) and the AssertedInferences between those Claims.

11.5 **11.6 ArgumentPackageBinding**

ArgumentElement within the ArgumentPackage can be bound together by means of ArgumentPackageBinding. ArgumentPackageBinding bind the participant packages by means of argument elements that connect the cited elements of the participant packages.
Superclass
ArgumentPackage

Associations
participantPackage:ArgumentPackageInterface[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 linking ArgumentElements that cite the claims in question.
ArgumentPackageBinding is a sub type of ArgumentPackage, it is used to record the argument that connects the arguments of two or more ArgumentPackages.

Constraints
The participantPackage should be only ArgumentPackage
OCL: self.participantPackage->forall(pp|pp.oclIsTypeOf(Argument::ArgumentPackage))
The ArgumentElements contained by an ArgumentPackageBinding must be ArgumentElement citations to ArgumentElements contained within the ArgumentPackages associated by the participantPackage association.

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

Superclass
ArgumentPackage

Associations

Semantics
ArgumentPackageInterfaces can be used to declare (by means of containing ArgumentAssetCitations) 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 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.

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

11.7 ArgumentAsset Class (abstract)
The ArgumentAsset Class is the abstract class 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.

11.8 Assertion Class (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.

11.9 ArtifactReference
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, 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.
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.
11.10 ArtifactElementCitation Class

The ArtifactElementCitation Class enables the citation of an artifact that relates to the structured argument.

**Superclass**

ArgumentAsset

**Attributes**

- externalReference: String An attribute recording a URL to external evidence.

**Associations**

- citedArtifact:ArtifactElement[0..1]

The ArtifactElements cited by the current ArtifactElementCitation object.

**Semantics**

It is necessary to be able to cite artifacts that provide supporting evidence, context, or additional description for the core reasoning of the recorded argument. ArtifactElementCitations 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.

The externalReference attribute can be used when wishing to cite an Artifact not being modeled by an SACM ArtifactElement.

11.11 ArgumentAssetCitation Class

The ArgumentAssetCitation cites an ArgumentAsset within another ArgumentPackage, for use within the current ArgumentPackage.

**Superclass**

ArgumentAsset

**Associations**

- citedAsset:ArgumentAsset[0..*]

References an ArgumentAsset within another ArgumentPackage.

**Semantics**

Within an ArgumentPackage it can be useful to be able to cite elements of another ArgumentPackage (i.e., ArgumentAssets) to act as explicit proxies for those elements acting within the argumentation structure. For example, in supporting a Claim it may be useful to cite a Claim contained within another ArgumentPackage.

**Constraints**

The citedAsset referred to by an ArgumentAssetCitation must be outside of the containment hierarchy containing the citation.

11.12 Claim Class

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**

- assumed: Boolean
An attribute recording whether the claim being made is declared as being assumed to be true rather than being supported by further reasoning.

**toBeSupported**: Boolean

An attribute recording whether further reasoning has yet to be provided to support the Claim (e.g., further evidence to be cited).

**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).

A Claim that is intentionally declared without any supporting evidence or argumentation can be declared as being assumed to be true. It is an assumption. However, it should be noted that a Claim that is not ‘assumed’ (i.e., assumed = false) is not being declared as false.

A Claim that is intentionally declared as requiring further evidence or argumentation can be denoted by setting toBeSupported to be true.

**Constraints**

Self.assumed and self.toBeSupported cannot both be true simultaneously.

### 11.12 ArgumentReasoning Class

ArgumentReasoning can be used to provide additional description or explanation of the asserted inference or challenge that connects one or more Claims (premises) to another Claim (conclusion). ArgumentReasoning elements are therefore related to AssertedInferences and AssertedChallenges. 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.

**Superclass**

ReasoningElement

**Attributes**

- structure:ArgumentPackage[0..1] – optional
- Optional reference to another 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 ArtifactElementCitation to a Claim, may not always be obvious. In such cases ArgumentReasoning can be used to provide further description of the reasoning involved.

### 11.14 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

**Attributes**

- isCounter:Boolean[1] = false – a flag indicating that 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.

11.15 **AssertedInference Class**

The AssertedInference association class 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 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.

**Constraints**
The source of AssertedInference relationships must be Claims, or ArgumentElementCitations that cite a Claim.
The target of AssertedInference relationships must be Assertions, or ArgumentElementCitations that cite an Assertion.

11.16 **AssertedEvidence Class**

The AssertedEvidence association class records the declaration that one or more artifacts of Evidence (cited by ArtifactElementCitations) 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 ArtifactElementCitation) may provide evidence for more than one Claim.

**Superclass**
AssertedRelationship

**Semantics**
Where evidence (cited by ArtifactElementCitation) 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 ArtifactElementCitation 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 ArtifactElementCitation.
The target of AssertedEvidence relationships must be Assertions, or ArgumentElementCitations that cite an Assertion.

11.17 **AssertedContext Class**

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

**Superclass**
AssertedRelationship

**Semantics**
Contextual information often needs to be cited in order to make clear the interpretation and scope of a Claim or ArgumentReasoning description. 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 information cited by Artifact B" or conversely "InformationItem B is the asserted context for Claim A"). A declaration (AssertedContext) of context (ArtifactElementCitation B) for a ReasoningElement A records that B is asserted to be contextual information required for the interpretation and scoping of A (i.e., B defines the context where the reasoning presented by A is asserted as true).
Contextual Claims often need to be cited as preconditions for a Claim or ArgumentReasoning. 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”. Similarly, a description of ArgumentReasoning A may only be considered true in a context where Claim B is true”.

Constraints
The source of AssertedContext relationships must be ArtifactElementCitations or Claims.
The target of AssertedContext relationships must be Assertions, ArgumentElementCitations that cite an Assertion, “ArgumentReasoning” elements or ArgumentElementCitations that cite ArgumentReasoning elements.

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.

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

Superclass
AssertedRelationship

Semantics
One or more other artifacts provide the necessary context in which the assertions associated with another artifact should be understood. 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 AssertedContext between Claims drawn out from the ArtifactReference.

Constraints
The source and target of AssertedArtifactContext must be of type ArtifactReference.
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.

![Artifact Package Diagram](image_url)

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’ (ModelElement). 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 (ArtifactProperty) 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...
12.5 ArtifactPackageInterface
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**
ArtifactPackage

**Associations**

**Semantics**
ArtifactPackageInterface enables the declaration of the elements of an ArtifactPackage that might be referred to (cited) in another ArtifactPackage.

**Constraints**
ArtifactPackageInterfaces are only allowed to contain Artifacts with `+isCitation=true` citing ArtifactAssets within the ArtifactPackage with which this ArtifactPackageInterface is associated.

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

**Superclass**
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.

12.7 Artifact class
The Artifact class represents the distinguishable units of data used in an assurance case.

**Superclass**
ArtifactAsset

**Attributes**
- `version: String [0..1]` – the version of the Artifact.
- `date: Date [0..1]` – the date on which the artifact was created.

**Associations**
- `artifactProperty::ArtifactProperty[0..*]` – The set of ArtifactProperties that correspond to the specific characteristics of an Artifact.
- `artifactEvent::ArtifactEvent[0..*]` – The set of ArtifactEvents that represent the lifecycle of an Artifact.

**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). 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.

### 12.8 ArtifactProperty class

The ArtifactProperty class enables the specification of the characteristics of an Artifact.

**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).

### 12.9 ArtifactEvent class

The ArtifactEvent class enables the specification of the events in the lifecycle of an Artifact.

**Attributes**

- **date**: Date
  - The date on which the ArtifactEvent occurred.

**Semantics**

Artifacts change during their lifecycle, and different types of happenings can occur at different moments: creation, modification, revocation... ArtifactEvents 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.

### 12.10 Resource class

The Resource class corresponds to the tangible objects representing an Artifact.

**Superclass**

ArtifactAsset

**Attributes**

- **location**: String
  - The path or URL specifying the location of the Resource.

**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.

### 12.11 Activity class

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

**Superclass**

ArtifactAsset

**Attributes**

- **startTime**: Date
  - Time when the Activity started.
- **endTime**: Date
  - 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.

### 12.12 Technique class

The Technique class 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.

### 12.13 Participant class

The Participant class 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.

### 12.14 ArtifactAssetRelationship class

The ArtifactAssetRelationship class enables the ArtifactAssets of an 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

**Associations**

- source:ArtifactAsset[1..*] - the source of the ArtifactRelationship
- target:ArtifactAsset[1..*] - the target of the ArtifactRelationship

**Constraints**

The source or target of an ArtifactAssetRelationship cannot be another ArtifactAssetRelationship.

**Semantics**

An ArtifactAsset can be related to other ArtifactAssets. This kind of information is specified by means of ArtifactAssetRelationships, which can also have a specific type depending on the ArtifactAssets being linked together. Name and description of the ArtifactAssetRelationship can be used to describe the semantics of the ArtifactAssetRelationship.