An OMG® Structured Assurance Case Metamodel™ Publication

Structured Assurance Case Metamodel (SACM)

Version 2.3

OMG Document Number	ptc/2022-03-02
Release Date	TBD
Normative Reference:	https://www.omg.org/spec/SACM/2.3/PDF
Associated Normative Machine Consumable Files:	
	https://www.omg.org/spec/SACM/202203xx/emof.xml

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page https://www.omg.org, under Documents, https://issues.omg.org/issues/create-new-issue

Table of Contents

Table of Contents

Illustration Index

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at https://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

https://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
9C Medway Road
PMB 274
Milford, MA 01757
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult https://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman/Liberation Serif - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to https://issues.omg.org/issues/create-new-issue.

1 Scope

1.1 General

This specification defines a metamodel for representing structured assurance cases. An Assurance Case is a set of auditable claims, arguments, and evidence created to support the claim that a defined system/service will satisfy the particular requirements. An Assurance Case is a document that facilitates information exchange between various system stakeholder such as suppliers and acquirers, and between the operator and regulator, where the knowledge related to the safety and security of the system is communicated in a clear and defendable way. Each assurance case should communicate the scope of the system, the operational context, the claims, the safety and/or security arguments, along with the corresponding evidence.

Systems Assurance is the process of building clear, comprehensive, and defensible arguments regarding the safety and security properties of systems. The vital element of Systems Assurance is that it makes clear and well-defined claims about the safety and security of systems. Certain claims are supported through reasoning. Reasoning is expressed by explicit annotated links between claims, where one or more claims (called sub-claims) when combined provide inferential support to a larger claim. Certain associations (recorded as assertions) between claims and subclaims can require supporting arguments of their own (e.g., justification of an asserted inference). Claims are propositions which are expressed by statements in some natural language. The degree of precision in formulation of the claims may contribute to the comprehensiveness of an assurance case. The context is important to communicate the scope of the claim, and to clarify the language used by the claim by providing necessary definition and explanations. Context involves assumptions made about the system and its environment. Explicit statement of the assumptions contributes to the comprehensiveness of the argument. Argumentation flow between claims is structured to facilitate communication of the entire assurance case.

1.2 Structured Arguments

Part of this specification defines a metamodel for representing structured arguments. A convincing argument that a system meets its assurance requirements is at the heart of an assurance case, which also may contain extensive references to evidence. The Argumentation Metamodel facilitates projects by allowing them to effectively and succinctly communicate in a structured way how their systems and services are meeting their assurance requirements. The scope of the Argumentation Metamodel is therefore to allow the interchange of structured arguments between diverse tools by different vendors. Each Argumentation Metamodel instance represents the argument that is being asserted by the stakeholder that is offering the argument for consideration.

This specification is designed to stand alone, or may be used in combination with the SACM Artifact Metamodel. The Artifact Metamodel is designed to represent aspects of evidence and properties about evidence in further detail. In the Argumentation Metamodel we have simplified support to model the relation of evidence to a structured argument.

Standardization will ensure that end users are investing not just in individual tools but also rather in a coordinated strategy.

The metamodel for argumentation provides a common structure and interchange format that facilitates the exchange of system assurance arguments contained within individual tool models. The metamodel represents the core concepts for structured argumentation that underlie a number of existing argumentation notations.

1.3 Evidence

Part of this specification provides a metamodel for communicating the way in which evidence artifactartifacts are collected by various participants using techniques, resources and activities. This allows users to build a repository of evidence that communicates its provenance and how it was gathered. This Artifact Metamodel identifies the main elements that determine the evidence collection process: artifacts, participants, resources, activities and techniques. Artifacts may be exchanged as packages or combined into composites.

The SACM Artifact Metamodel defines a catalog of elements for constructing and interchanging packages of evidence that communicate how the evidence was collected.

In conjunction with the Argumentation Metamodel, certain claims may be expressed to be supported by evidence that is within the Artifact Metamodel, to permit the authors of the assurance claims to offer evidentiary support for their positions. Evidence is usually collected by applying systematic methods and procedures and is often collected by automated tools. Evidence is information or objective artifacts, based on established fact or expert judgment, which is presented to show that the claim to which it relates is valid (i.e., true). Various and diverse things may be produced as evidence, such as documents, expert testimony, test results, measurement results, records related to process, product, and people, etc.

1.4 Controlled Vocabulary

Part of this specification provides a metamodel for defining controlled and reusable vocabulary used in the argumentation of an Assurance Case. In the argument of safety and/or security of a system, a set of vocabulary is often referred to repeatedly. Thus, it is important to ensure that the usage of vocabulary refer to the terms with the same semantics. In addition, for model based system assurance, such vocabularies can relate to external heterogeneous models that define the semantics for the terms. Therefore, controlled vocabulary ensures the consistency of the semantics of the terms used in the argumentation, and provides a mean to define the terms used in the argumentation through external models (e.g. standard models).

The SACM Terminology Metamodel defines a number of elements for constructing and interchanging packages of terminologies, to ensure the consistency of semantics.

1.5 History, Motivation, and Rationale

The original Structured Assurance Case Metamodel version 1.0 was the composite of two efforts within the OMG's Systems Assurance Task Force. One effort, the Structured Assurance Evidence Metamodel (SAEM) was created through the OMG Request For Proposal (RFP) approach and the other, the Argumentation Metamodel (ARG) was created through the OMG Request For Comment (RFC) approach. Both were completed in the mid-2010 timeframe and then put into the same Finalization Task Force (FTF) due to the interconnectedness of their topics and concepts. The first version of SACM was eventually produced in the spring of 2012 consisting of a top-level container object joining SAEM and ARG without significantly altering the two original metamodels.

A Revision Task Force (RTF) was convened to drive further integration of the two original parts of SACM into one Metamodel and that effort formulated a set of goals to shape and guide the integration. Basically, the stated goals were:

Improve support for ISO/IEC 15026-2. In order to facilitate the use of structured assurance cases for producing and reviewing ISO/IEC 15026-2 conformant assurance cases, the structured assurance case metamodel needs to more fully support the constructs and entities in ISO/IEC 15026-2.

Improve support for “Goal Structuring Notation.” In order to facilitate the use of structured assurance cases by the existing community of practitioners across the world that are currently using Goal Structuring Notation (GSN) and the specific capabilities in GSN for working with assurance cases, the structured assurance case metamodel needs to more fully support the constructs and entities in GSN.

Harmonization of Parts. In order to facilitate acceptance and successful use of SACM, the argumentation and evidence container metamodels need to be more consistently aligned and integrated. Areas of focus include elimination of overlap, making useful facilities now available on one side generalized to be useful on both sides, achieving uniform terminology and consistency, and using common concepts.

Add initial support for Patterns/Templates. In order to make the use of assurance cases more practical and efficient for users including those that do not have in-depth experience within the assurance case domain

(e.g., acquisition officials, systems integrators, auditors, regulators, and tool vendors), the structured assurance case metamodel needs to support the concept of assurance case patterns and templates. Patterns will provide support to enable reuse and the effective composition of assurance cases along with the underlying argumentation supporting goals. Templates will provide support for defining and describing constraining conventions that a community may require for assurance cases within a particular domain due to regulatory requirements or accepted practices in that domain/industry/community.

Improve the modularity and simplicity of SACM

Provide for future concepts such as structured expressions and other formalisms

The SACM 1.1 was subsequently worked to attempt to meet these goals and a draft metamodel was created during the summer OMG 2013 Berlin meeting. However the magnitude of the changes necessary to actually integrate the two original metamodels into one cohesive approach and achieve some of the other goals turned out to be too big of a change for a point release. The final SACM 1.1, released in July 2015, was scaled back to address some of the issues and it cleaned up some terminology and logical issues but it did not substantially alter the underlying metamodel.

During this same timeframe other efforts in the OMG (the Dependability Assurance Framework for Safety-Sensitive Consumer Devices (DAF)) and in The Open Group (the Dependability Assurance Framework (O-DA), as well as the work of the Food and Drug Administration (FDA) in the U.S. started making use of the assurance case concept and articulated implicit requirements/needs for tools that would work with assurance case models and their exchange.

Additionally, the Open Platform for EvolutioNary Certification of Safety-critical Systems (OPENCOSS) effort in Europe was exploring different uses of assurance cases, including the creation of a Common Certification Language, and the OMG's Architecture Driven Modernization Task Force crafted a Structured Pattern Metamodel Standard (SPMS) that provided a method for describing patterns within models. Together these new needs and the new openly available capabilities represented in OPENCOSS and SPMS offer a way forward.

This version 2.0 of SACM has been created as a major version release since pursuing another point release revision of SACM would appear to be incompatible with achieving the integration and harmonization that is critical to obtain wide-spread adoption and implementation within the tooling market and allow that market to deliver on some of the potential capabilities they could provide to address the emerging and evolving need for assurance case tools, such as:

- Improving the Understandability of an Assurance Case to a 3rd Party
- Improving Rigor of Assurance Cases through Modeling
- Allowing for Reexamination of Assumptions, Argument Structuring, and the Appropriateness of Evidence
- Allowing for Reuse of Sub-Claim/Evidence Constructs That “Work”
- Authoring/Sharing Libraries of Sub-Claims/Supporting Evidence
- Providing for Assurance Case Analytics/Validation
- Providing for Exchange of Assurance Cases (Import/Export)
- Providing for Enforcing Community of Interest Norms of Practice

The resulting metamodel in version 2.0 of SACM came from the ideas in the 2013 Berlin metamodel, along with the approaches utilized for modeling artifact- and process-related concepts in OPENCOSS Common Certification Language and the pattern metamodel and concepts from the SPMS.

In SACM 2.1, the concrete syntax for the Argumentation metamodel was defined. The concrete syntax is designed based on visual notation design theories such as semantic transparency and structure visual inheritance. Furthermore, the existing notations in the domain such as GSN and CAE are also considered in the design process.In SACM 2.2, minor clarifications, expanded explanations for enhanced readability and consistency have been incorporated along with

a new annex to provide additional details about the concepts of package interfaces and bindings and their use within the standard.

In SACM 2.3, a new normative annex and corresponding compliance point to provide a UML profile allowing UML-based tools to work with SACM concepts and the allowing for categories to contain categories as a mechanism for organizing concepts in the Terminology class.

2 Conformance

2.1 Introduction

The Structured Assurance Case Metamodel (SACM) specification defines the following five compliance points:

- Argumentation Model
- Artifact Model
- Assurance Case Model
- Terminology Model
- SACM UML Profile

2.2 Argumentation Model compliance point

Software that conforms to the SACM specification at the Argumentation Model compliance point shall be able to import and export XMI documents that conform with the SACM XML Schema produced by applying XMI rules to the normative MOF metamodel defined in the Argumentation subpackage of the SACM specification, including the common elements defined in the Common and Predefined diagrams of the SACM. The top object of the Argumentation package as a unit of interchange shall be the Argumentation::ArgumentPackage element of the SACM.

Conformance to the Argumentation Model compliance point does not entail support for the Evidence subpackage of SACM, or the terminology sub package of the SACM.

This compliance point facilitates interchange of the structured argumentation documents produced by existing tools supporting existing structured argument notations such as the Goal Structuring Notation (GSN) and the Claims-Arguments-Evidence (CAE) notation which provide their own mapping onto SACM argumentation aspects. Further details of these mappings are given in Annex A.

2.3 Artifact Model compliance point

Software that conforms to the specification at the Artifact Model compliance point shall be able to import and export XMI documents that conform with the SACM XML Schema produced by applying XMI rules to the normative MOF metamodel defined in this Artifact subpackage of the SACM specification, including the common elements defined in the Common and Predefined diagrams of the SACM. The top object of the Evidence package as a unit of interchange shall be the Artifact::ArtifactPackage element of the SACM.

Conformance to the Artifact Model compliance point does not entail support for the Argumentation subpackage of SACM, or the terminology diagram of the SACM. This compliance point facilitates interchange of the packages of evidence. In particular, this compliance point facilitates development of evidence repositories in support of software assurance and regulatory compliance.

2.4 Assurance Case Model compliance point

This compliance point is mandatory. Software that conforms to the specification at the Assurance Case Model compliance point shall be able to import and export XMI documents that conform with the SACM XML Schema produced by applying XMI rules to the normative MOF metamodel defined in this entire specification. The top object of the Assurance Case package as a unit of interchange shall be the SACM::AssuranceCasePackage element.

The Conformance clause identifies which clauses of the specification are mandatory (or conditionally mandatory) and which are optional in order for an implementation to claim conformance to the specification.

2.5 Terminology Model compliance point

Software that conforms to the specification at the Terminology Model compliance point shall be able to import and export XMI documents that conform with the SACM XML Schema produced by applying XMI rules to the normative MOF metamodel defined in this entire specification. The top object of the Terminology package as a unit of interchange shall be the Terminology::TerminologyPackage element.

The Conformance clause identifies which clauses of the specification are mandatory (or conditionally mandatory) and which are optional in order for an implementation to claim conformance to the specification.

2.6 SACM UML Profile compliance point

A tool demonstrating SACM UML Profile interchange conformance can import and export conformant XMI for all valid SACM UML models

3 References

3.1 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

ISO/IEC 15026-1:2019 Systems and software engineering - Systems and software assurance - Part 1: Concepts and vocabulary, 2019. https://www.iso.org/standard/73567.html

ISO/IEC 15026-2: 2011 Systems and software engineering - Systems and software assurance - Part 2: Assurance case, 2011. <https://www.iso.org/iso/catalogue_detail.htm?csnumber=52926>

OMG UML 2.5.1 Infrastructure Specification formal/17-12-05. <https://www.omg.org/spec/UML/>
OMG Meta-Object Facility (MOF) version 2.5.1 formal/19-10-01.
<https://www.omg.org/spec/MOF/>

OMG MOF XML Metadata Interchange (XMI) Specification, version 2.5.1, formal/2015-06-07
<https://www.omg.org/spec/XMI

3.2 Non-normative References

The following non-normative documents contain provisions which, through reference in this text, provide informative context for material in this specification.

Assurance Case Working Group. Goal Structuring Notation Community Standard. Technical Report SCSC-141BA v2.0, Safety Critical Systems Club, 2018. <https://scsc.uk/SCSC-141B> Open Platform for EvolutioNary Certification of Safety-critical Systems (OPENCOSS) WP4: Common Certification Language, 2012-2015. <http://www.opencoss-project.eu/node/7>

Open Platform for EvolutioNary Certification of Safety-critical Systems (OPENCOSS) WP6: Evolutionary Evidential Chain, 2012-2015. <http://www.opencoss-project.eu/node/7>.

Evidence management for compliance of critical systems with safety standards: A survey on the state of practice, Information and Software Technology 60: 1-15, Elsevier (North-Holland) (2015).
<https://www.sciencedirect.com/science/article/pii/S0950584914002560>

OMG Structured Pattern Metamodel Standard (SPMS), 1.2, formal/17-11-01

<https://www.omg.org/spec/SPMS/>

Open Group Dependability Assurance Framework (O-DA), Jul 2013. <https://www2.opengroup.org/ogsys/catalog/C13F>

OMG Dependability Assurance Framework for Safety-Sensitive Consumer Devices (DAF), beta1, May 2015. <https://www.omg.org/spec/DAF/>

Infusion Pumps Total Product Life Cycle Guidance for Industry and FDA Staff, Dec 2014.

<https://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM209337.pdf>

Model Based System Assurance Using the Structured Assurance Case Metamodel. R. Wei, TP. Kelly*, X. Dai, S. Zhao, R. Hawkins. Elsevier Journal of Systems and Software (JSS), 154, 211-233, 2019. <https://www.sciencedirect.com/science/article/pii/S0164121219301062>

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Argument

A body of information presented with the intention to establish one or more claims through the presentation of related supporting claims, evidence, and contextual information.

Assurance Case

A collection of auditable claims, arguments, and evidence created to support the contention that a defined system/service will satisfy its assurance requirements.

Attribute

A feature of a meta-element with a primitive type, such as integer, real, text, boolean, or date.

Claim

A proposition being asserted by the author or utterer that is a true or false statement.

Evidence

Objective artifacts being offered in support of one or more claims.

Evidence Repository

A software service providing access to, and information about, a collection of evidence items, such as records, documents, and other exhibits together with related information that facilitates management of evidence, the interpretation of evidence, and understanding the evidentiary support provided to claims.

Participant Package

A Participant Package is any Package or Package Interface referenced by a Package Binding.

Package Binding

A Package that has references to at least two Participant Packages and to elements of those Participant Packages and defines the relationships among those elements.

Package Interface

A Package that is used to specify the elements of one Package that are declared “public” and available for use by other Packages.

Feature

Super class of attribute and reference.

Reference

A feature of a meta-element whose type is of another meta-element, such as +ParticipantPackage for AssuranceCasePackage

Structured argument

A particular kind of argument where the relationships between the asserted claims, and from the evidence to the claims are explicitly represented.

5 Symbols

In SACM 2.1, a number of symbols (concrete syntax) are defined for the elements in the Argumentation Metamodel, which are detailed in Annex C. The usage of these symbols are illustrated through examples in Annex D. Note: the concrete syntax for other packages are not currently defined.

6 Additional Information

6.1 Changes to Adopted OMG Specifications [optional]

This specification completely replaces the SACM 2.1 specification.

6.2 Acknowledgements

The following companies submitted this specification:

- MITRE Corporation
- Adelard LLP
- KDM Analytics
- Lockheed Martin
- Benchmark Consulting
- Northrop Grumman
- Elparazim
- Fraunhofer Gesellschaft/IESE

The following companies supported this specification:

- University of York
- Dalian University of Technology
- Universidad Carlos III de Madrid
- Carnegie Mellon University
- University of Cambridge

6.3 How to Proceed

The rest of this document contains the technical content of this specification.

Clause 7. Specification overview - Provides design rationale for the SACM Argumentation Metamodel specification.

Part 1 of the specification defines the normative common elements. This part includes three clauses. Material in this part of the specification is related to all compliance points.

Clause 8. SACM Base classes define the common base classes of the Structured Assurance Case Metamodel. Clause 9. SACM Packages define the common packages of the Structured Assurance Case Metamodel.

Clause 10. SACM Terminology defines the common terminology classes of the Structured Assurance Case Metamodel.

Part 2 of the specification defines the SACM Argumentation metamodel. The Argumentation Metamodel defines the catalog of elements for constructing and interchanging structured statements describing argumentations. Material in this part of the specification is related to the Assurance Case and Argumentation compliance points, and is not required for the Evidence Container compliance point. This part includes a single clause. The non-normative Annex B contains some examples of the SACM XML interchange format for Argumentation, and describes how SACM Argumentation is related to existing graphical notations for describing structured arguments, such as the Goal Structuring Notation (GSN) and the Claims-Arguments-Evidence (CAE) notation.

Clause 11. The SACM Argumentation Metamodel - Provides the details of the Argumentation Metamodel specification.

Part 3 of the specification defines the SACM Artifact Metamodel. The Artifact Metamodel defines the catalog of elements for constructing and interchanging precise statements involved in evidence-related efforts. This part includes a single clause.

Material in this part of the specification is related to the Assurance Case and the Evidence Container compliance points, and it is not required for the Argumentation compliance point.

Clause 12 defines the key elements of the Artifact Metamodel.

This page intentionally left blank.

7 Background and Rationale

7.1 The Need for Assurance Cases

All sectors of society are placing growing reliance on software-enabled and connected systems, both information systems and embedded systems. Adequate functioning of many of these systems is critical to the well-being of organizations and society. Today, these numerous, large, complex systems provide increased benefits by connecting with others and often directly or indirectly to the Internet.

However the societal and individual risks posed by attacks on, or in the maladaptive behavior of such systems, are significant enough to warrant a pro-active technology adoption approach whereby the emergent risks can be analyzed, explored, communicated, and ultimately accepted by those responsible for the assurance.

Thus, system suppliers face the task of engineering their products and services to meet these challenges and threats in such a way that users and other stakeholders can rationally possess the needed confidence in them – or at least judge their level of risk. This means that suppliers must not only ensure their delivery of adequate systems, but acquirers and users require the explicit, valid, well-reasoned, and evidence-supported grounds[1] for their confidence and decision making including related engineering conclusions and their uncertainty.

Historically assurance cases covering safety and security requirements for systems have been seen as an important tool for the interchange of assurance information.

To make system assurance more practical, automation and meaningful exchange of this assurance-related information is needed. System suppliers, tool vendors, acquirers, users, and others would benefit from a flexible and extensible means for its representation and exchange.

The concept of an assurance case is one that provides a framework for analyzing and communicating the assurance arguments and evidence that relates to a system under consideration. Suppliers and customers can see how the system lifecycle products (system requirements, design, testing, field experience, etc.) relate to and satisfy the assurance requirements, enabling sufficient confidence to be gained in the behavior and integration of the system within its operational context.

Simply put, an assurance case comprises the arguments and evidence that a system will meet its assurance requirements over its lifecycle.

7.2 Structured Arguments

Arguments have always been used - albeit informally - to communicate and persuade stakeholders that sufficient confidence can be had in a particular system. However these arguments are often spread over a range of system and management documentation, and it is difficult to see the argument as a whole in a clear way.

In the assurance domain an ‘argument’ is defined as “a connected series of statements or reasons intended to establish a position...; a process of reasoning”[2]. In attempting to persuade others of a position, we cite reasons why a claim should be accepted as true. These reasons are described as the premises of the argument, and the claim they support as its conclusion. These terms can be used to define the ‘normal form’ of an argument as:

Premise
Premise
Premise
So, Conclusion

This form reduces argument to its most primitive building blocks, for example:

[1] Suppliers also need the same or similar case to justify release and deployment.
[2] Shorter Oxford English Dictionary, 6th Edition (2007).

Premise: All complex systems are susceptible to failure.
Premise: Failures can lead to accidents.
Therefore,
Conclusion: Accidents can occur in complex safety-critical systems.

The terms ‘premise’ and ‘conclusion’ are relative. The premise of one reasoning step (e.g., that “All complex systems are susceptible to failure”) may itself need further reasoning support and will become the conclusion of a subsequent supporting argument. This gives rise to hierarchical argument structures (‘chains of reasoning’) in which arguments are established by the composition of a number of (premise-conclusion) reasoning steps in order to support an overall conclusion, as illustrated in Figure 7.1.

Structured arguments are therefore one way to allow the communication of how a series of claims can establish a conclusion.

7.3 Arguments as asserted positions

It is important to note that the representation of an argument is not the same as a valid argument. The process of argument representation and communication is separate from that of argument evaluation. For example, an argument may include invalid reasoning, or may have a reliance on irrelevant or false information.

Therefore representations of arguments should be seen as positions that are effectively asserted by the authors or organizations that are putting forward the argument.

Clearly professional ethics require that assurance stakeholders should present arguments that they believe to be correct, valid, and relevant.

A key concept is that structured arguments allow users to express and declare what they consider the argument to be.

7.4 Structured Arguments in SACM

SACM contains those elements presented as fundamental to the expression and exchange of structured arguments.

As noted above, a typical natural language dictionary definition of an argument is that an argument comprises a series of linked premises (propositions), leading to a conclusion. From this we can derive a set of practical modeling approaches that allow users to link together propositions (claims) and to communicate how they consider that higher level claims be supported or derived from the lower level claims. Since a claim can be used to support one or more other claims, the general form of a directed graph emerges.

SACM aims to provide a modeling framework to allow users to express and exchange their argument structures. The representation of an argument in SACM does not imply that the argument is complete, valid, or correct. Similarly, the evaluation or acceptance of an argument by a separate party is not covered by the SACM. In the SACM model, structured arguments comprise argument elements (primarily claims) that are being asserted by the author of the argument, together with relationships that are asserted to hold between those elements.

7.5 Precise statements related to evidence

In the simplest form, evidence consists of a collection of documents, records or artifacts that provide evidentiary support to a set of claims.

Artifacts may be structured together into composite artifacts or collections. For higher degrees of assurance it is pertinent to know how these artifacts have been created and managed over their lifecycle, and what techniques and resources were used in their generation – i.e., the provenance of the artifact.

The Artifact Metamodel defines the vocabulary for constructing and interchanging precise statements describing evidence- related efforts, including:

Describing artifacts and their properties and associated events

Collection and management of evidence by participants, using resources, techniques, and activities, by describing the relationships between them

Structuring of artifacts – e.g., as composite artifacts or collections

An extensible approach is presented whereby users of an Artifact Model may specify the relationships that hold between the artifact assets. If necessary a terminology package may be used to reuse common relationships.

Describing artifacts – artifacts have properties and associated events. An artifact event can be used to communicate, for example, the review date or release date for the artifact.

Collection and Management of Evidence – can be described by means of an extensible set of relationships between participants, activities, resources and the associated evidence artifacts.

Structuring of artifacts – Artifacts may be part of a larger composite by means of artifact to artifact relationships, or within a common artifact package.

This page intentionally left blank.

Part I - Common Elements

The first part of the specification defines the common elements of the Structured Assurance Case Metamodel, including the Base Classes, the Structured Assurance Case Terminology Classes, and the Structured Assurance Case Packages. Subsequent parts define the Argumentation Metamodel and the Artifact Metamodel.

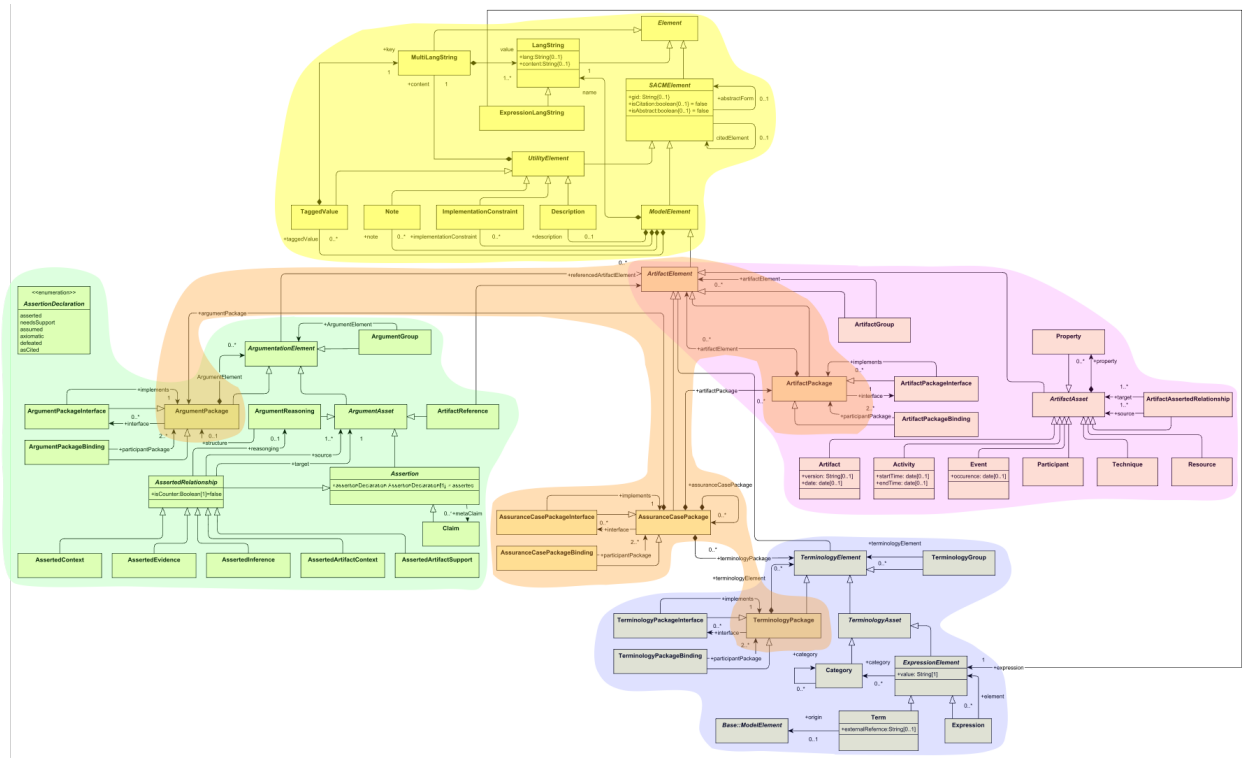


Figure 7.2 - Overall SACM Class Diagram

Yellow denotes items covered in Clause 8, Structured Assurance Case Base Classes.

Orange denotes items covered in Clause 9, Structured Assurance Case Packages.

Blue denotes items covered in Clause 10, Structured Assurance Case Terminology Classes.

Green denotes items covered in Clause 11, Argumentation Metamodel.

Purple denotes items covered in Clause 12, Artifact Metamodel.

This page intentionally left blank.

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.

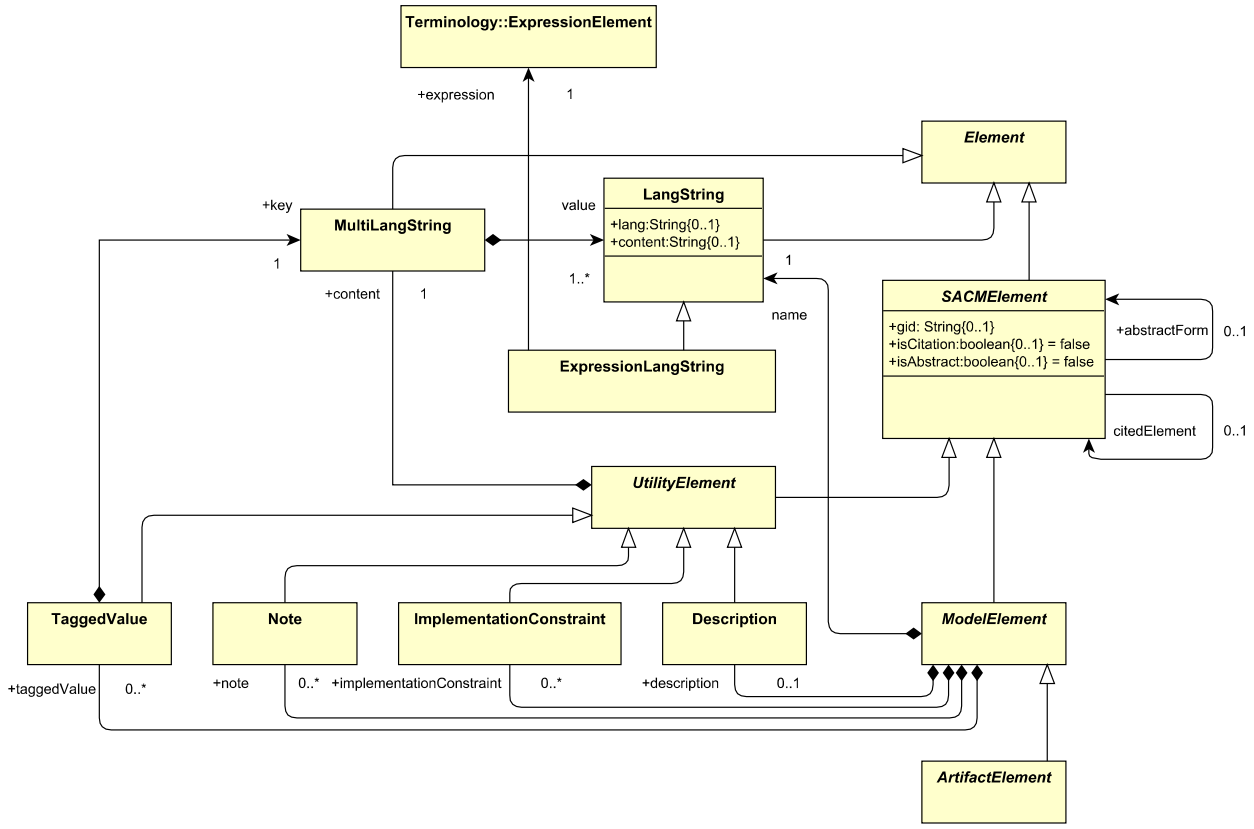


Figure 8.1 - Structured Assurance Case Base Classes Diagram

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.

8.2 SACMElement (abstract)

SACMElement is the base class for SACM.

Superclass

MOF:Element

Attributes

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

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

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:

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

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

Semantics

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

Constraints:

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.

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.

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

LangString

Attributes

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.

8.5 MultiLangString

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

Superclass

Element

Associatiions

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

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

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

8.6 ModelElement (abstract)

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

Superclass

SACMElement

Associations

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

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

8.7 UtilityElement (abstract)

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

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.

8.8 ImplementationConstraint

ImpementationConstraint 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).

8.9 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.

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.

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.

8.11 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.

Superclass

UtilityElement

Semantics

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.

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.

Superclass

UtilityElement

Associations

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

Semantics

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

This page intentionally left blank.

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.

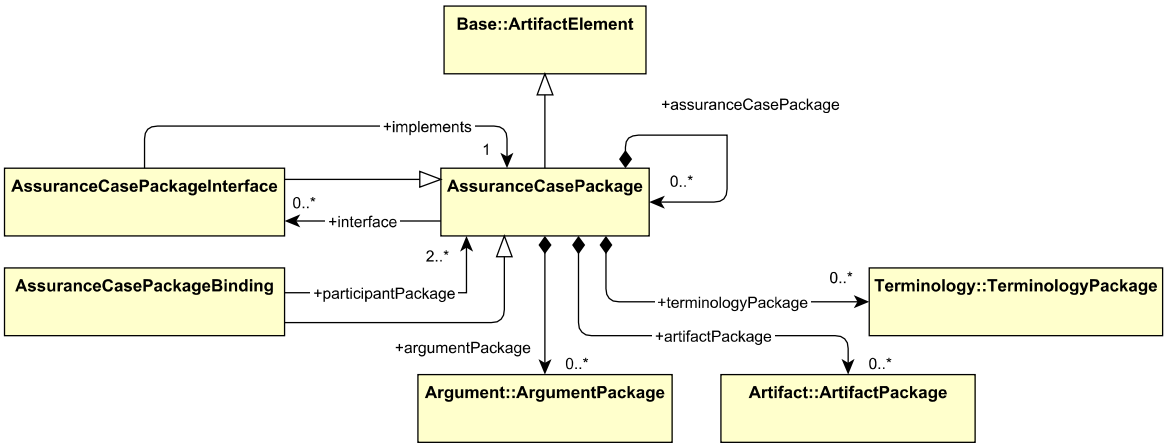


Figure 9.1 - Structured Assurance Case Packages Class Diagram

In SACM, the parent container element is AssuranceCasePackage. AssurancesCasePackages 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 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::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

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))
```

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 Package 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.

Superclass

AssuranceCasePackage

Associations

participantPackage:AssuranceCasePackage[2..*] – references to AssuranceCasePackages which the AssuranceCasePackageBinding binds together. The Participant Package provides access to its elements through a Package Interface, if a Package Interface is named in the Package Binding, or to all elements if the Package is named in the Package Binding.

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. ArgumentPackageBindings, ArtifactPackageBindings and TerminologyPackageBindings) contained in the AssuranceCasePackages, together with an optional ArgumentationPackage that asserts the relationship between +participantPackage. The AssuranceCasePackageBinding includes a reference to each participant Package which can be either an AssuranceCasePackage or an AssuranceCasePackageInterface and that are external to Packages to the AssuranceCasePackage that contains this AssuranceCasePackageBinding. Within the AssuranCasePackageBinding, there might be zero or more ArgumentPackageBinding, ArtifactPackageBinding, or TerminologyPackageBinding. Each of these that exist would refer to at least two participant packages that could be a Package or PackageInterface of the same type.

Constraints

The participantPackages should be either AssuranceCasePackage or AssuranceCasePackageInterfaces.

OCL:

self.participantPackage->forall(pp|pp.oclIsTypeOf(AssuranceCase::AssuranceCasePackage) or pp.oclIsTypeOf(AssuranceCase::AssuranceCasePackageInterface))

This page intentionally left blank.

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

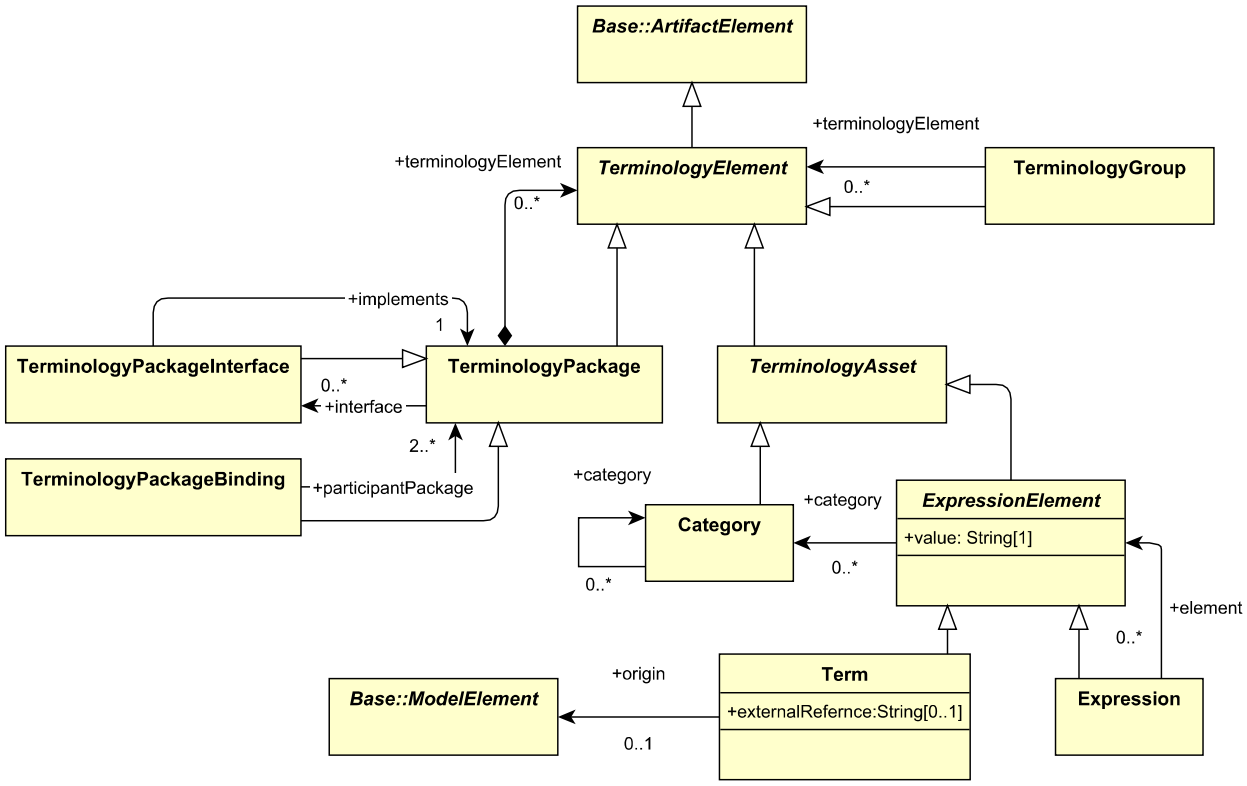


Figure 10.1 - Terminology Class Diagram

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).

10.3 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).

10.4 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.

10.5 TerminologyPackageInterface

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

Associations

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.

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.oclIsKindOf(Terminology::TerminologyPackage))

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, 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).

10.11 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

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.

This page intentionally left blank.

Part II - Argumentation Metamodel

This part of the specification defines the Argumentation Metamodel.

This page intentionally left blank.

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.

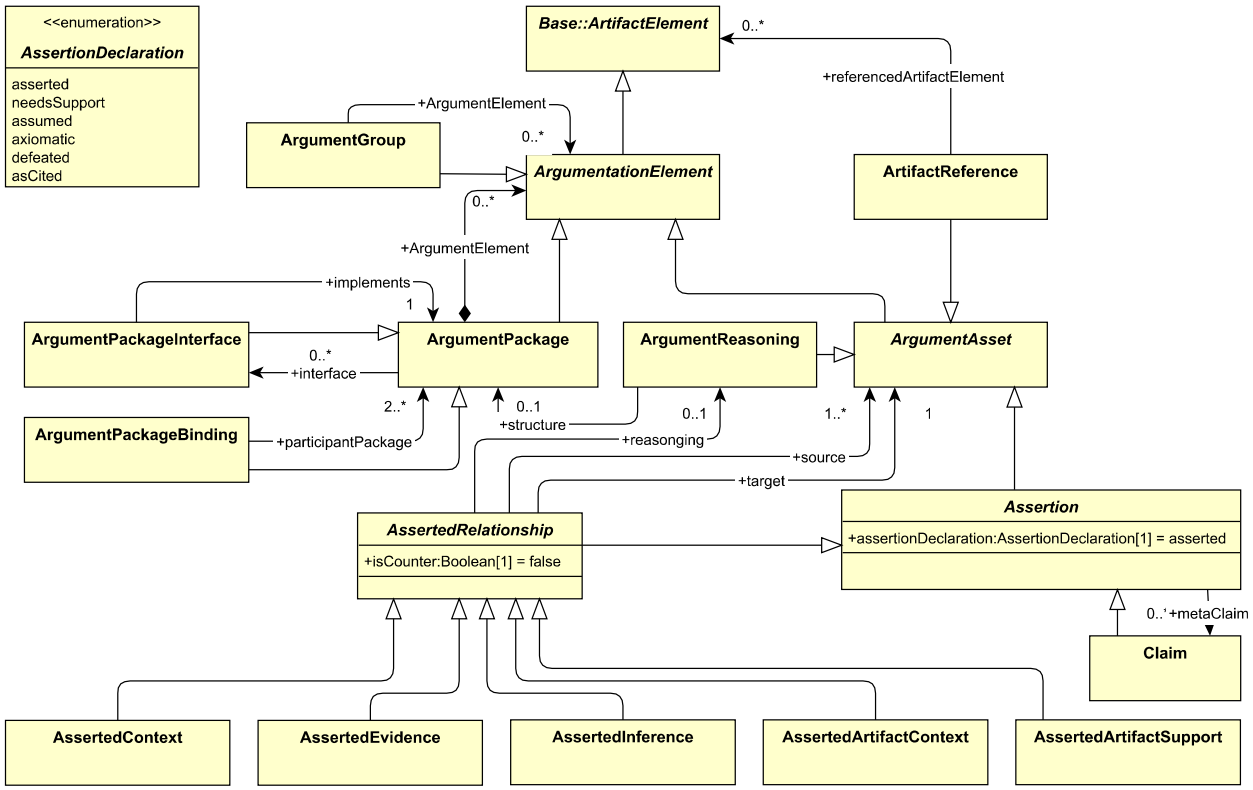


Figure 11.1 - Argumentation Package Diagram

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).

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

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 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 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.

11.5 ArgumentPackageBinding

ArgumentElement within the ArgumentPackage can be bound together by means of ArgumentPackageBinding. An ArgumentPackages 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., ArguementPackageInterfaces) 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 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 ArgumentatonElement used in its argument structure that relates ArgumentationElements from different ArgumentPackages.

Constraints

The participantPackages should be only ArgumentPackages

OCL:

self.participantPackage->forall(pp|pp.oclIsTypeOf(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.

11.6 ArgumentPackageInterface

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

implements:ArgumentPackage[1] – a reference to the ArgumentPackage which the ArgumentPackageInterface declares.

Semantics

ArgumentPackageInterfaces 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 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 insided 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.

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.

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 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.

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

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 (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”.

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

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

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.

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 ArtifactReferemce) may provide evidence for more than one Claim.

Superclass

AssertedRelationship

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.oclIsTypeOf(ArtifactReference))

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

Semantics

Contextual information often needs to be cited in order to make clear the interpretation and scope of an Assertion and supporting arguementation. 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”.

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.

Part III - Artifact Metamodel

This part of the specification defines the Artifact Metamodel.

This page intentionally left blank.

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.

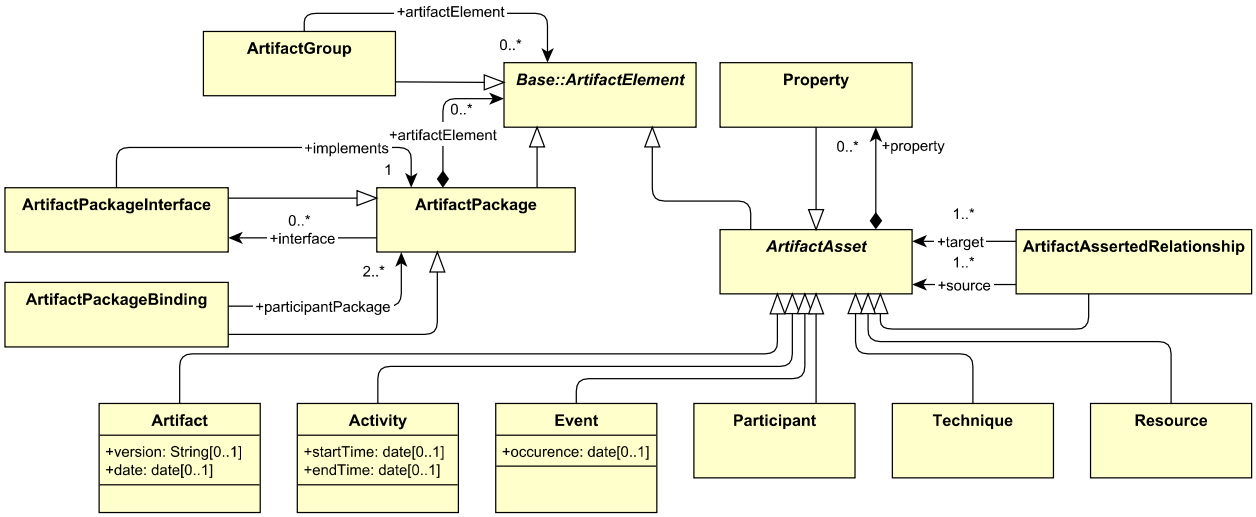


Figure 12.1 - Artifact Package Diagram

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

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.

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).

12.4 ArtifactPackageBinding

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

Superclass

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

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

12.5 ArtifactPackageInterface

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

Superclass

ArtifactPackage

Associations

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 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 Propert(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

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

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.

12.8 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).

12.9 Event

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

Superclass

ArtifactAsset

Attributes

date: date[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.

12.10 Resource

Resource corresponds to the tangible objects representing an Artifact.

Superclass

ArtifactAsset

Attributes

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.

12.11 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.

12.12 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.

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.

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

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.

Annex A: Mappings from existing industrial notations for assurance cases

(informative)

A.1 Goal Structuring Notation (GSN)

Details of the of the mapping between GSN elements and SACM, and the available relevant tool support, are maintained at the following URL:

http://www.goalstructuringnotation.info/gsn-metamodel

A.2 Claims, Arguments, Evidence (CAE)

Details of the mapping between CAE elements and SACM, and the available relevant tool support, are maintained at the following URL:

https://www.adelard.com/asce/choosing-asce/standardisation.html

This page intentionally left blank.

Annex B: Examples of Assurance Cases in SACM 2.0 XMI

(informative)

B.1 Example Assurance Cases

Examples of SACM 2.0 Assurance Cases with HTML renderings, graphical depictions, and machine readable XMI are maintained at the following URL:

http://www.goalstructuringnotation.info/sacm-examples

This page intentionally left blank.

Annex C: Concrete Syntax (Graphical Notations) for the Argumentation Metamodel

(informative)

C.1 ArgumentPackage

The concrete syntax for ArgumentPackage is defined in Figure C1.

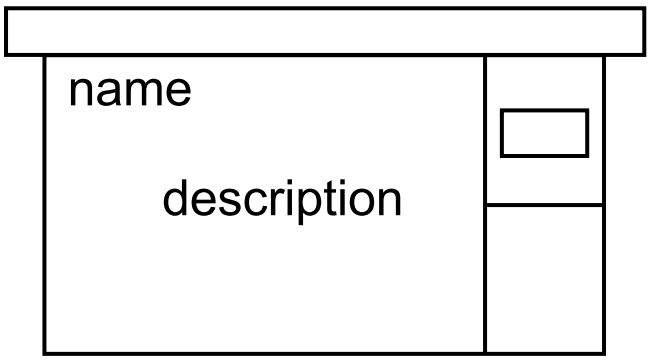


Figure C1 - Concrete Syntax for ArgumentPackage

C.2 ArgumentPackageInterface

The concrete syntax for ArgumentPackageInterface is defined in Figure C2.

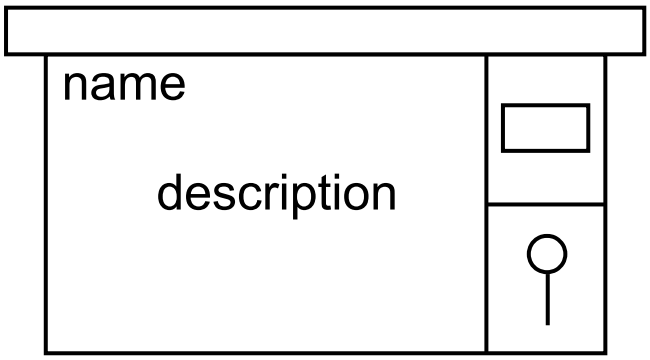


Figure C2 - Concrete Syntax for ArgumentPackageInterface

C.3 ArgumentPackageBinding

The concrete syntax for ArgumentPackageBinding is defined in Figure C3.

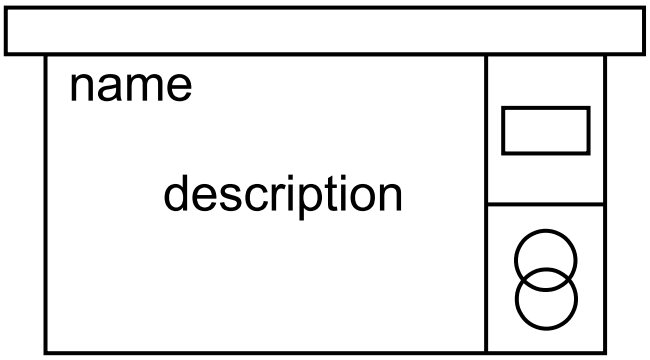


Figure C3 - Concrete Syntax for ArgumentPackageBinding

C.4 ArtifactReference

The concrete syntax for ArtifactReference is defined in Figure C4.

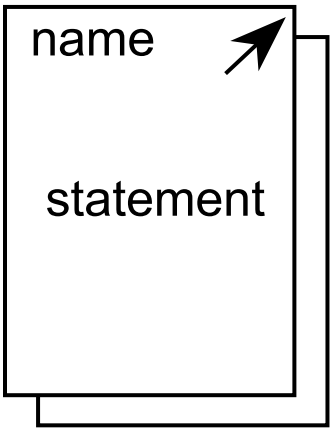


Figure C4 - Concrete Syntax for ArtifactReference

C.5 The +metaClaim reference

+metaClaim can be used as references Claims concerning (i.e., about) the Assertion (e.g., regarding the confidence in the Assertion). The concrete syntax for the +metaClaim reference is defined in Figure C5.

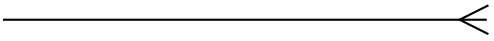


Figure C5 – Concrete Syntax for +metaClaim reference

Examples of using the +metaClaim reference can be found in Appendix D.

C.6 Claim

By default the AssertionDeclaration of a Claim is set to asserted, the concrete syntax for an asserted Claim is defined in Figure C6.

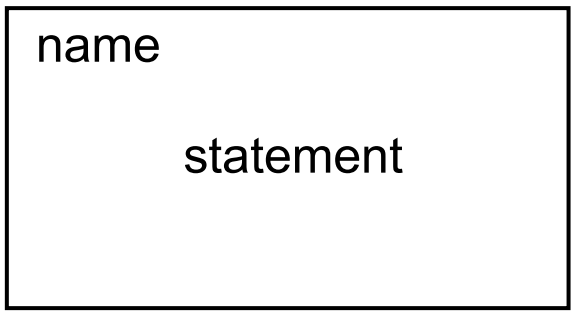


Figure C6 - Concrete Syntax for asserted Claim

An assumed Claim indicates that an assumption is declared without any supporting evidence or argumentation. The concrete syntax for an assumed Claim is defined in Figure C7.

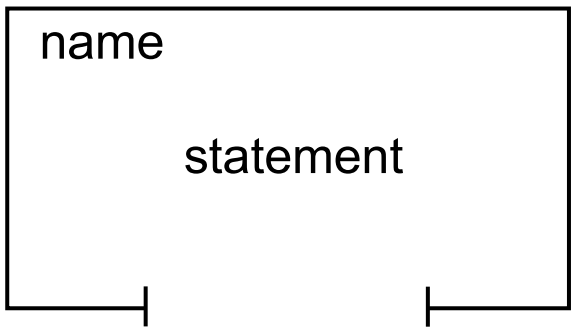


Figure C7 - Concrete Syntax for assumed Claim

A needsSupport Claim indicates that a Claim is declared as requiring further evidence or argumentation. The concrete syntax for a needsSupport Claim is defined in Figure C8.

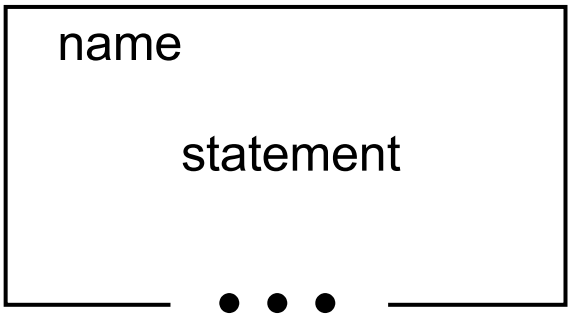


Figure C8 - Concrete Syntax for needsSupport Claim

An axiomatic Claim indicates that a Claim is intentionally declared to be axiomatically true. The concrete syntax of an axiomatic Claim is defined in Figure C9.

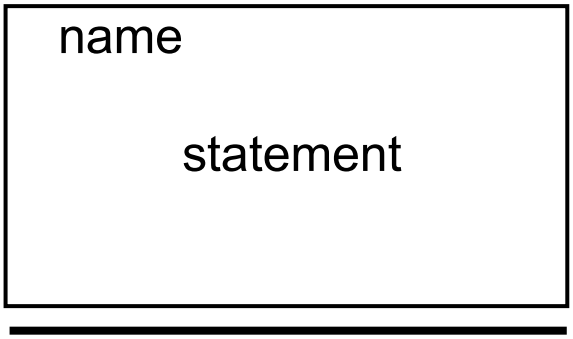


Figure C9 - Concrete Syntax for axiomatic Claim

A defeated Claim indicates that a Claim is defeated by counter-evidence. The concrete syntax of a defeated Claim is defined in Figure C10.

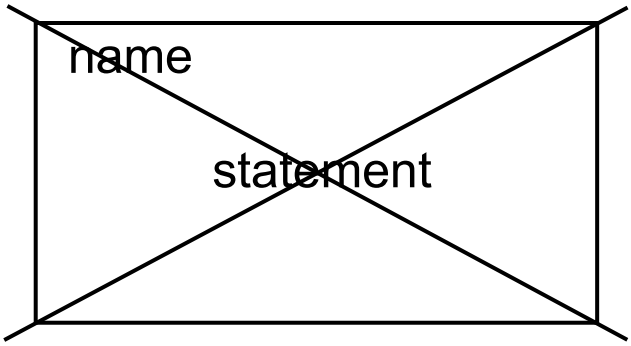


Figure C10 - Concrete Syntax for defeated Claim

An asCited Claim indicates that a Claim cites another claim and is hence supported by the cited Claim. The concrete syntax of an asCIted Claim is defined in Figure C11.

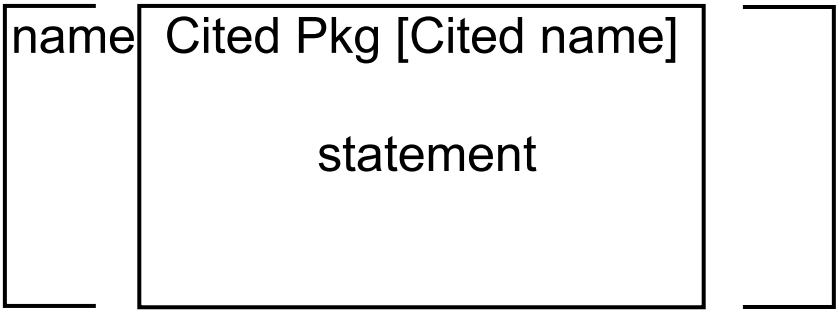


Figure C11 - Concrete Syntax for asCited Claim

An abstract Claim indicates that a Claim is part of a pattern or a template. The concrete syntax for an Abstract Claim is to render the Claim with dash lines, Figure C12 is an example of an abstract asserted Claim.

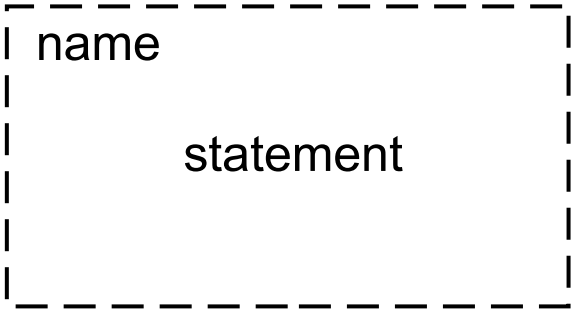


Figure C12 - Concrete Syntax for abstract asserted Claim

For other types of Claims, they should be rendered in dash lines, should their +isAbstract attribute is true.

C.7 ArgumentReasoning

The concrete syntax of ArgumentReasoning is defined in C13 (note: the right hand side of the notation).

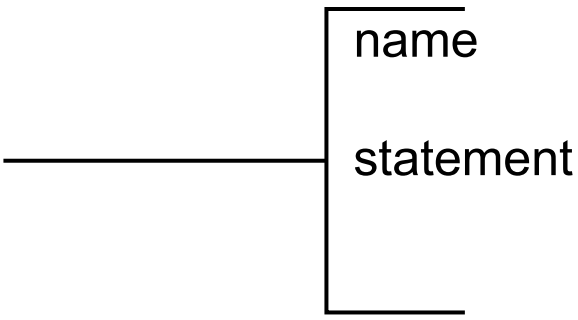


Figure C13 - Concrete Syntax for ArgumentReasoning

C.8 AssertedInference

The concrete syntax of AssertedInference is defined in Figure C14, where the dot represents the AssertedInference instance, the edge without an arrow represents the +source reference of the AssertedInference, and the edge with an arrow represents the +target reference of the AssertedInference.

Figure C14 - Concrete Syntax for asserted AssertedInference

An assumed AssertedInference indicates that the inference is assumed without any supporting evidence or argumentation. The concrete syntax of an assumed AssertedInference is defined in Figure C15 (note: the change is applied to the +target reference edge of an AssertedInference).

Figure C15 - Concrete Syntax for assumed AssertedInference

A needsSupport AssertedInference indicates that the inference is declared as requiring further evidence or argumentation. The concrete syntax of a needsSupport AssertedInference is defined in Figure 16 (note: the change is applied to the +target reference edge of an AssertedInference).

Figure C16 - Concrete Syntax for needsSupport AssertedInference

An axiomatic AssertedInference indicates that the inference is declared to be axiomatically true. The concrete syntax of

an axiomatic AssertedInference is defined in Figure C17 (note: the change is applied to the +target reference edge of an AssertedInference).

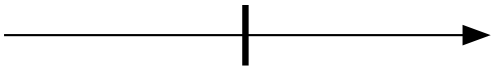


Figure C17 - Concrete Syntax for axiomatic AssertedInference

A defeated AssertedInference indicates that the inference is defeated by counter-evidence. The concrete syntax of a defeated AssertedInference is defined in Figure C18 (note: the change is applied to the +target reference edge of an AssertedInference).

Figure C18 - Concrete Syntax for defeated AssertedInference

An asCited AssertedInference indicates that the inference cites another AssertedInference and is hence supported by the cited AssertedInference. The concrete syntax of an asCited AssertedInference is defined in Figure C19 (note: the change is applied to the +target reference edge of an AssertedInference).

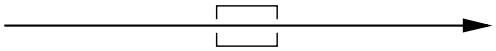


Figure C19 - Concrete Syntax for asCited AssertedInference

An abstract AssertedInference indicates that the inference is part of a pattern or template. The concrete syntax of an abstract AssertedInference is defined in Figure C20 (note: the change is applied to the +target reference edge of an AssertedInference).

Figure C20 - Concrete Syntax for abstract asserted AssertedInference

For other types of AssertedInference, they should be rendered in dash lines, should their +isAbstract attribute is true.

An isCounter AssertedInference indicates that the inference counters its declared purposes. The concrete syntax of an isCounter AssertedInference is defined in Figure C21 (note: the change is applied to the +target reference edge of an AssertedInference).

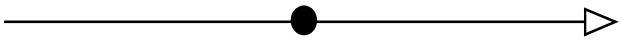


Figure C21 - Concrete Syntax for counter asserted AssertedInference

C.9 AssertedEvidence

The concrete syntax of AssertedEvidence is defined in Figure C22, where the dot represents the AssertedEvidence instance, the edge without an arrow represents the +source reference of the AssertedEvidence, and the edge with an arrow represents the +target reference of the AssertedEvidence.

Figure C22 - Concrete Syntax for asserted AssertedEvidence

An assumed AssertedEvidence indicates that the inference is assumed without any supporting evidence or argumentation. The concrete syntax of an assumed AssertedEvidence is defined in Figure C23 (note: the change is applied to the +target reference edge of an AssertedEvidence).

Figure C23 - Concrete Syntax for assumed AssertedEvidence

A needsSupport AssertedEvidence indicates that the inference is declared as requiring further evidence or argumentation. The concrete syntax of a needsSupport AssertedEvidence is defined in Figure C24 (note: the change is applied to the +target reference edge of an AssertedEvidence).

Figure C24 - Concrete Syntax for needsSupport AssertedEvidence

An axiomatic AssertedEvidence indicates that the inference is declared to be axiomatically true. The concrete syntax of an axiomatic AssertedEvidence is defined in Figure C25 (note: the change is applied to the +target reference edge of an AssertedEvidence).

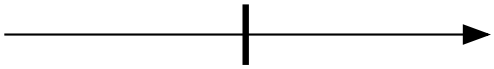


Figure C25 - Concrete Syntax for axiomatic AssertedEvidence

A defeated AssertedEvidence indicates that the inference is defeated by counter-evidence. The concrete syntax of a defeated AssertedEvidence is defined in Figure C26 (note: the change is applied to the +target reference edge of an AssertedEvidence).

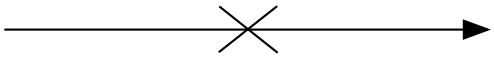


Figure C26 - Concrete Syntax for defeated AssertedEvidence

An asCited AssertedEvidence indicates that the inference cites another AssertedEvidence and is hence supported by the cited AssertedEvidence. The concrete syntax of an asCited AssertedEvidence is defined in Figure C27 (note: the change is applied to the +target reference edge of an AssertedEvidence).

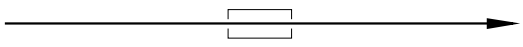


Figure C27 - Concrete Syntax for asCited AssertedEvidence

An abstract AssertedEvidence indicates that the inference is part of a pattern or template. The concrete syntax of an abstract AssertedEvidence is defined in Figure C28 (note: the change is applied to the +target reference edge of an AssertedEvidence).

Figure C28 - Concrete Syntax for abstract asserted AssertedEvidence

For other types of AssertedEvidence, they should be rendered in dash lines, should their +isAbstract attribute is true.

An isCounter AssertedEvidence indicates that the inference counters its declared purposes. The concrete syntax of an isCounter AssertedEvidence is defined in Figure C29 (note: the change is applied to the +target reference edge of an AssertedEvidence).

Figure C29 - Concrete Syntax for counter asserted AssertedEvidence

C.10 AssertedContext

The concrete syntax of AssertedContext is defined in Figure C30, where the dot represents the AssertedContext instance, the edge without an arrow represents the +source reference of the AssertedContext, and the edge with an arrow represents the +target reference of the AssertedContext.

Figure C30 - Concrete Syntax for asserted AssertedContext

An assumed AssertedContext indicates that the inference is assumed without any supporting evidence or argumentation. The concrete syntax of an assumed AssertedContext is defined in Figure C31 (note: the change is applied to the +target reference edge of an AssertedContext).

Figure C31 - Concrete Syntax for assumed AssertedContext

A needsSupport AssertedContext indicates that the inference is declared as requiring further evidence or argumentation. The concrete syntax of a needsSupport AssertedContext is defined in Figure C32 (note: the change is applied to the +target reference edge of an AssertedContext).

Figure C32 - Concrete Syntax for needsSupport AssertedContext

An axiomatic AssertedContext indicates that the inference is declared to be axiomatically true. The concrete syntax of an axiomatic AssertedContext is defined in Figure C33 (note: the change is applied to the +target reference edge of an AssertedContext).

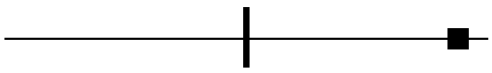


Figure C33 - Concrete Syntax for axiomatic AssertedContext

A defeated AssertedContext indicates that the inference is defeated by counter-evidence. The concrete syntax of a defeated AssertedContext is defined in Figure C34 (note: the change is applied to the +target reference edge of an AssertedContext).

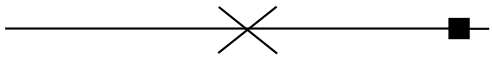


Figure C34 - Concrete Syntax for defeated AssertedContext

An asCited AssertedContext indicates that the inference cites another AssertedContext and is hence supported by the cited AssertedContext. The concrete syntax of a defeated AssertedInference is defined in Figure C35 (note: the change is applied to the +target reference edge of an AssertedContext).

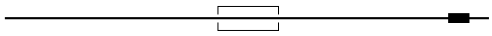


Figure C35 - Concrete Syntax for asCited AssertedContext

An abstract AssertedContext indicates that the inference is part of a pattern or template. The concrete syntax of a defeated AssertedContext is defined in Figure C36 (note: the change is applied to the +target reference edge of an AssertedContext).

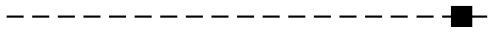


Figure C36 - Concrete Syntax for abstract asserted AssertedContext

For other types of AssertedContext, they should be rendered in dash lines, should their +isAbstract attribute is true.

An isCounter AssertedContext indicates that the inference counters its declared purposes. The concrete syntax of an isCounter AssertedContext is defined in Figure C37 (note: the change is applied to the +target reference edge of an AssertedContext).

Figure C37 - Concrete Syntax for counter asserted AssertedContext

C.11 AssertedArtifactSupport

The concrete syntax of AssertedArtifactSupport is defined in Figure C38, where the dot represents the AssertedArtifactSupport instance, the edge without an arrow represents the +source reference of the AssertedArtifactSupport, and the edge with an arrow represents the +target reference of the AssertedArtifactSupport.

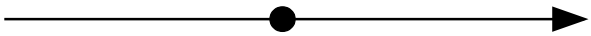


Figure C38 - Concrete Syntax for asserted AssertedArtifactSupport

An assumed AssertedArtifactSupport indicates that the inference is assumed without any supporting evidence or argumentation. The concrete syntax of an assumed AssertedArtifactSupport is defined in Figure C39 (note: the change is applied to the +target reference edge of an AssertedArtifactSupport).

Figure C39 - Concrete Syntax for assumed AssertedArtifactSupport

A needsSupport AssertedArtifactSupport indicates that the inference is declared as requiring further evidence or argumentation. The concrete syntax of a needsSupport AssertedArtifactSupport is defined in Figure C40 (note: the change is applied to the +target reference edge of an AssertedArtifactSupport).

Figure C40 - Concrete Syntax for needsSupport AssertedArtifactSupport

An axiomatic AssertedArtifactSupport indicates that the inference is declared to be axiomatically true. The concrete syntax of an axiomatic AssertedArtifactSupport is defined in Figure C41 (note: the change is applied to the +target reference edge of an AssertedArtifactSupport).

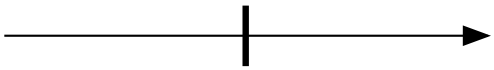


Figure C41 - Concrete Syntax for axiomatic AssertedArtifactSupport

A defeated AssertedArtifactSupport indicates that the inference is defeated by counter-evidence. The concrete syntax of a defeated AssertedArtifactSupport is defined in Figure C42 (note: the change is applied to the +target reference edge of an AssertedArtifactSupport).

Figure C42 - Concrete Syntax for defeated AssertedArtifactSupport

An asCited AssertedArtifactSupport indicates that the inference cites another AssertedArtifactSupport and is hence supported by the cited AssertedArtifactSupport. The concrete syntax of a defeated AssertedInference is defined in Figure C43 (note: the change is applied to the +target reference edge of an AssertedArtifactSupport).

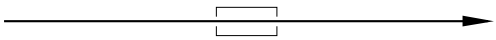


Figure C43 - Concrete Syntax for asCited AssertedArtifactSupport

An abstract AssertedArtifactSupport indicates that the inference is part of a pattern or template. The concrete syntax of a defeated AssertedArtifactSupport is defined in Figure C44 (note: the change is applied to the +target reference edge of an AssertedArtifactSupport).

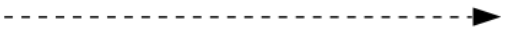


Figure C44 - Concrete Syntax for abstract asserted AssertedArtifactSupport

For other types of AssertedArtifactSupport, they should be rendered in dash lines, should their +isAbstract attribute is true.

An isCounter AssertedArtifactSupport indicates that the inference counters its declared purposes. The concrete syntax of an isCounter AssertedArtifactSupport is defined in Figure C45 (note: the change is applied to the +target reference edge of an AssertedArtifactSupport).

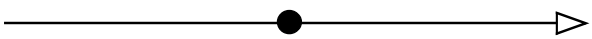


Figure C45 - Concrete Syntax for counter asserted AssertedArtifactSupport

Note: although the graphical notation of AssertedArtifactSupport is similar to AssertedInference/AssertedEvidence, they are distinguishable through the types of elements that the +source and +target references connect to.

C.12 AssertedArtifactContext

The concrete syntax of AssertedArtifactContext is defined in Figure C46, where the dot represents the AssertedArtifactContext instance, the edge without an arrow represents the +source reference of the AssertedArtifactContext, and the edge with an arrow represents the +target reference of the AssertedArtifactContext.

Figure C46 - Concrete Syntax for asserted AssertedArtifactContext

An assumed AssertedArtifactContext indicates that the inference is assumed without any supporting evidence or argumentation. The concrete syntax of an assumed AssertedArtifactContext is defined in Figure C47 (note: the change is applied to the +target reference edge of an AssertedArtifactContext).

Figure C47 - Concrete Syntax for assumed AssertedArtifactContext

A needsSupport AssertedArtifactContext indicates that the inference is declared as requiring further evidence or argumentation. The concrete syntax of a needsSupport AssertedArtifactContext is defined in Figure C48 (note: the change is applied to the +target reference edge of an AssertedArtifactContext).

Figure C48 - Concrete Syntax for needsSupport AssertedArtifactContext

An axiomatic AssertedArtifactContext indicates that the inference is declared to be axiomatically true. The concrete syntax of an axiomatic AssertedArtifactContext is defined in FigureC49 (note: the change is applied to the +target reference edge of an AssertedArtifactContext).

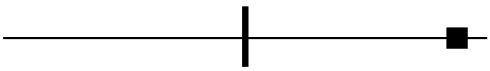


Figure C49 - Concrete Syntax for axiomatic AssertedArtifactContext

A defeated AssertedArtifactContext indicates that the inference is defeated by counter-evidence. The concrete syntax of a defeated AssertedArtifactContext is defined in Figure C50 (note: the change is applied to the +target reference edge of an AssertedArtifactContext).

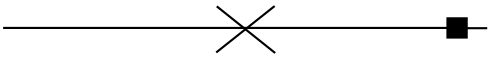


Figure C50 - Concrete Syntax for defeated AssertedArtifactContext

An asCited AssertedArtifactContext indicates that the inference cites another AssertedArtifactContext and is hence supported by the cited AssertedArtifactContext. The concrete syntax of a defeated AssertedInference is defined in Figure C51 (note: the change is applied to the +target reference edge of an AssertedArtifactContext).

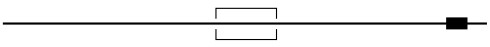


Figure C51 - Concrete Syntax for asCited AssertedArtifactContext

An abstract AssertedArtifactContext indicates that the inference is part of a pattern or template. The concrete syntax of a defeated AssertedArtifactContext is defined in Figure C52 (note: the change is applied to the +target reference edge of an AssertedArtifactContext).

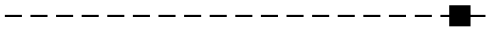


Figure C52 - Concrete Syntax for abstract asserted AssertedArtifactContext

For other types of AssertedArtifactContext, they should be rendered in dash lines, should their +isAbstract attribute is true.

An isCounter AssertedArtifactContext indicates that the inference counters its declared purposes. The concrete syntax of an isCounter AssertedArtifactContext is defined in Figure C53 (note: the change is applied to the +target reference edge of an AssertedArtifactContext).

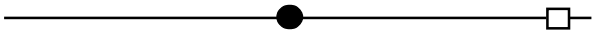


Figure C53 - Concrete Syntax for counter asserted AssertedArtifactContext

Note: although the graphical notation of AssertedArtifactContext is similar to AssertedContext, they are distinguishable through the types of elements that the +source and +target references connect to.

Annex D: Examples of Argumentation Elements

(informative)

D.1 Claims

In some cases, it is necessary to state explicitly the assumption to support the declared Assertion in an argumentation. For example, Claims G2 and G3 are asserted to support Claim G1, the relationship between them is declared using an AssertedInference. In this case, an assumed Claim A1 is declared to explicitly describe the assumption that is being made to support the AssertedInference between Claim G2, G3, and G1.

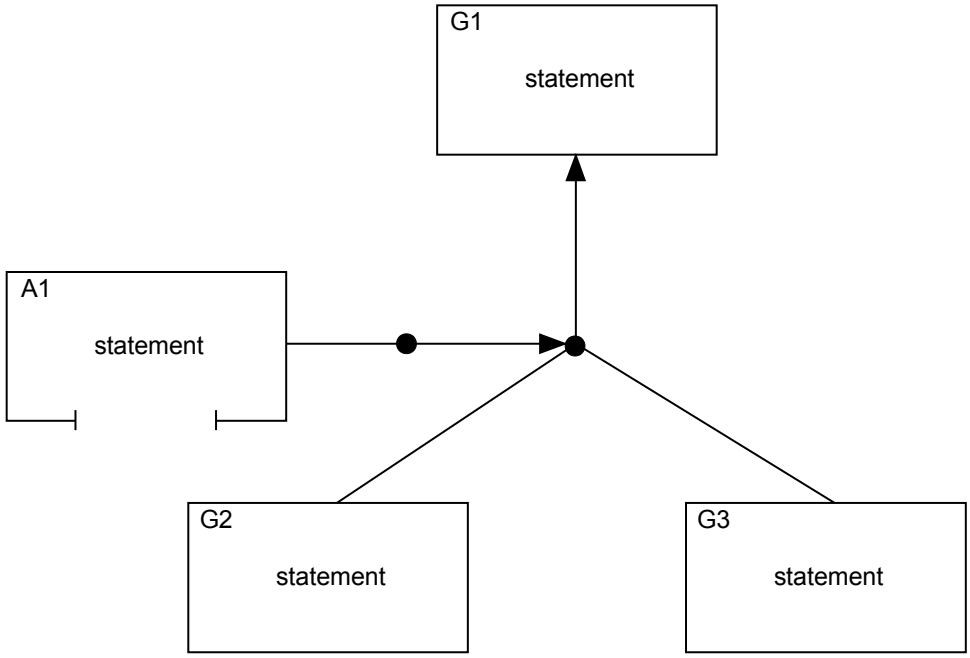


Figure D1 – Example of Claim Assumptions

A needsSuport Claim indicates a Claim is intentionally declared as requiring further evidence or argumentation. For example, Claim G11 is supported by Claim G12 and Claim G13. However, both Claim G12 and Claim 13 is declared as needsSupport Claims, indicates that both Claims required further evidence or argumentation.

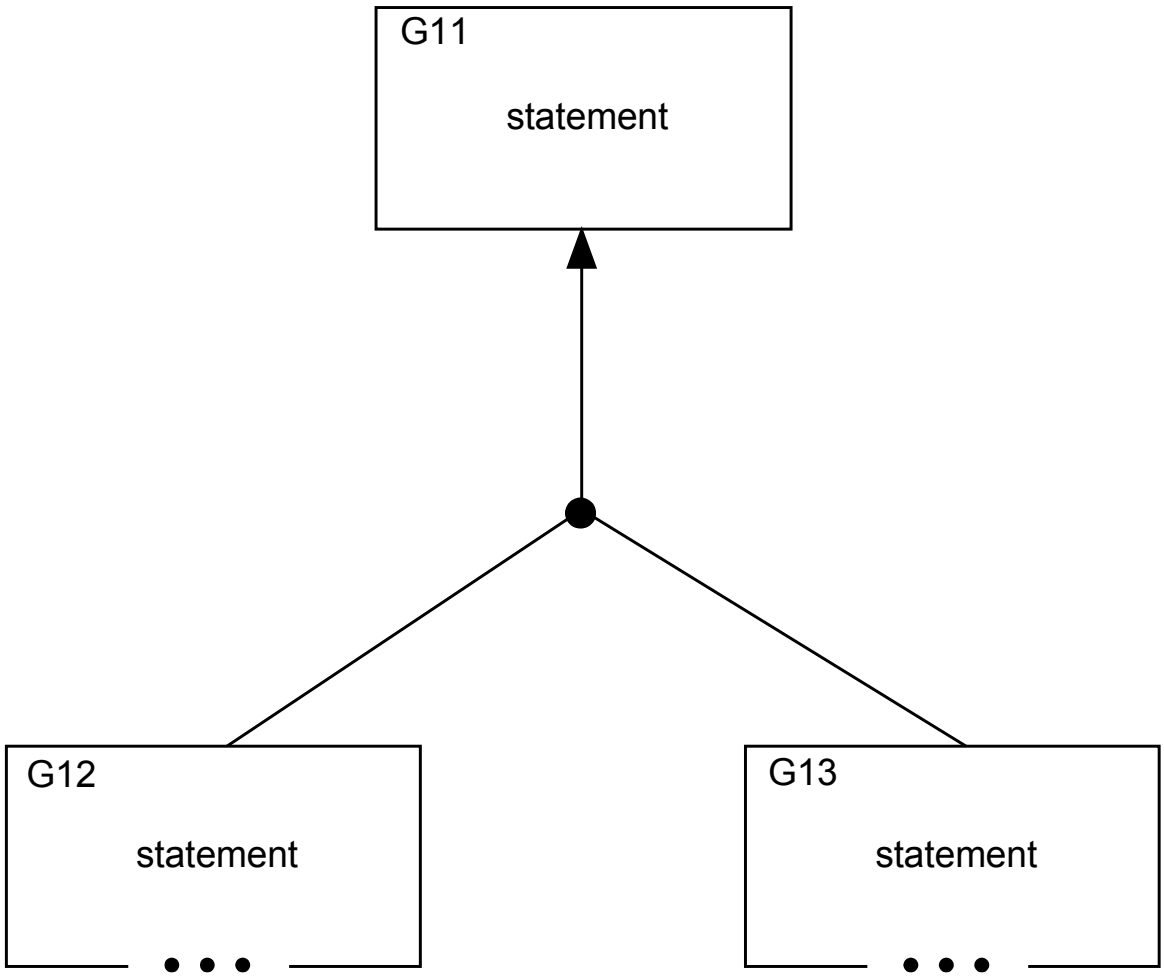


Figure D2 – Example of a Claim needing support

An axiomatic indicates a Claim is intentionally declared as axiomatically true. In some cases, an axiomatic Claim can be used to support the assertion that is made in an argumentation. For example, an axiomatic Claim AC1 is declared to support the inference (using AssertedInference) that is asserted from the Claim G8 and Claim G9 to support Claim G6.

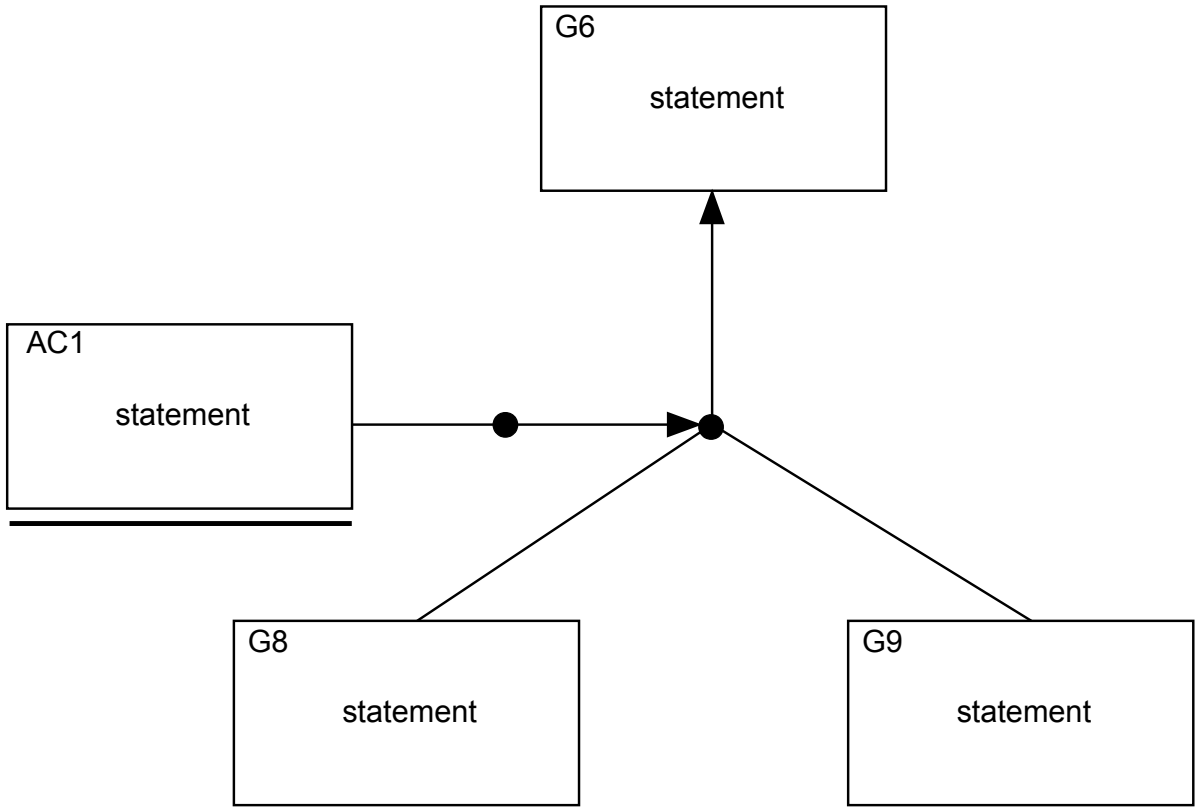


Figure D3 – Example of an Axiomatic Claim

A defeated Claim indicates a Claim is defeated by counter-evidence. For example, Claim G9 is defeated by evidence E3 (cited using ArtifactReference) that is declared using the counter-evidence relationship. Therefore, the Claim G9 is further declared as Defeated Claim.

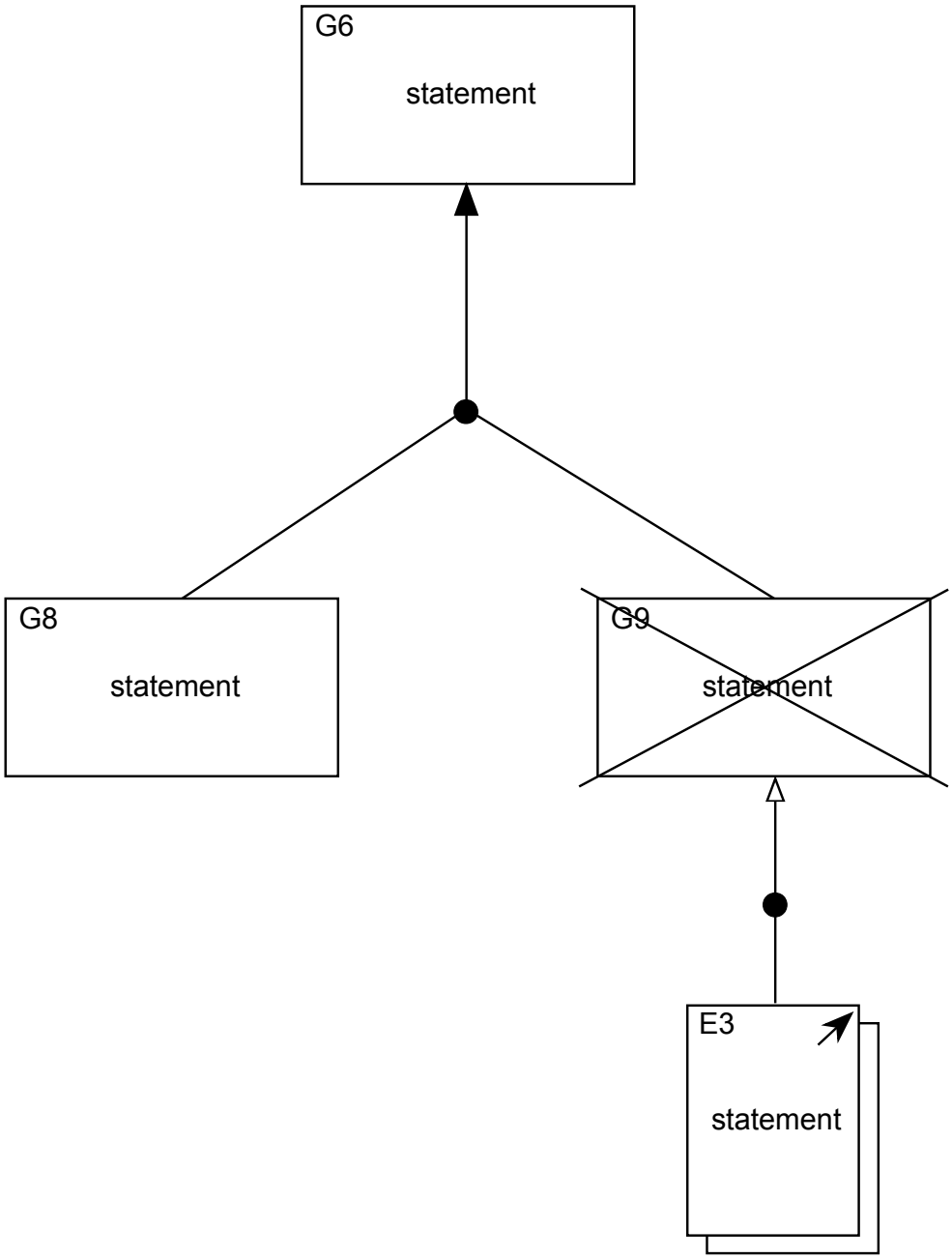


Figure D4 – Example of Defeating a Claim

An asCited Claim indicates a Claim which cites another claim and supported by the cited claim. The identifier of the Claim is placed in the top-left corner of the square brackets. The identifier of the cited Claim is placed in the top-left corner of the cited Claim and is written within a square bracket. An optional identifier of the cited package where the cited claim is located, can be written before the cited claim identifier. For example, Claim G3 is supported by Claim G6 and Claim G7. Claim G7 is declared as asCited Claim that is a Claim that cited another Claim, in this case is Claim G10.

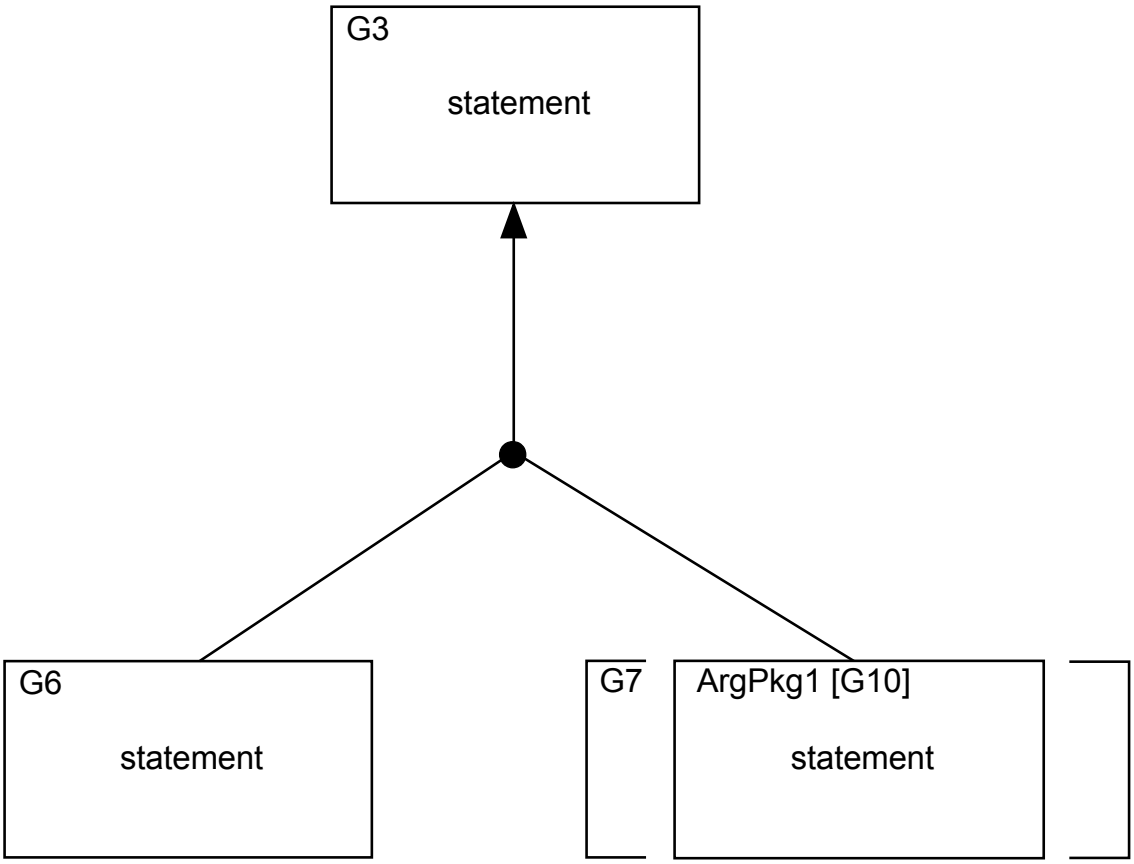


Figure D5 - Example of Claim citation

An abstract Claim indicates a Claim is part of a pattern or template. For example, Claim G1, G2, and G3 are declared as an abstract Claim that indicates that abstract Claim G1, G2, and G3 are part of argument pattern.

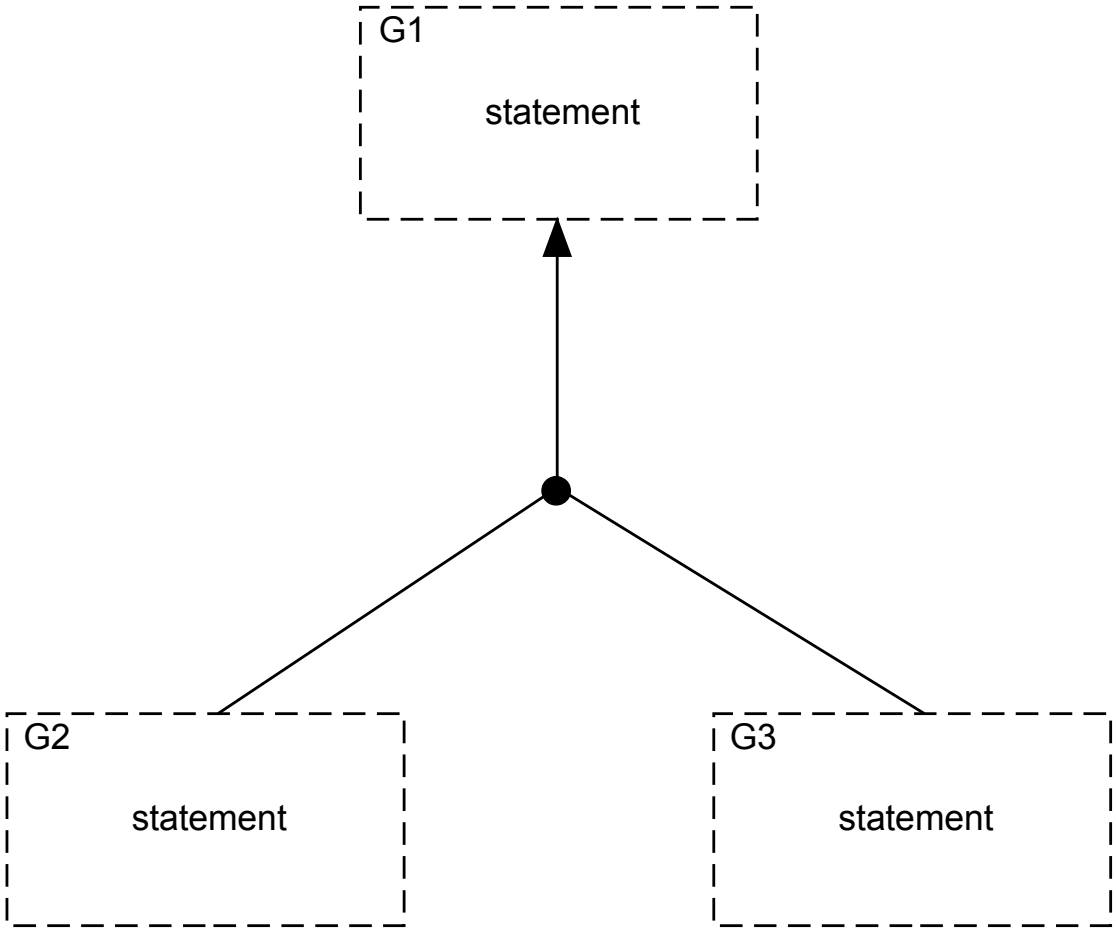


Figure D6 – Example of abstract Claims forming an argument pattern

D.2 MetaClaim

When used in a diagram, the source element of the MetaClaim must be type of Claim and the targeted element can be type of Assertion. The location of the source element of the MetaClaim must be located on the left and right side of the targeted element and the relationship between them is declared using the MetaClaim.

For example, Claim MC1 that is connected to Claim G1. The relationship between MC1 and G1 is declared using the MetaClaim, indicates Claim MC1 is concerning (i.e. about) Claim G1.

Figure D7 – Example of Claim and MetaClaim Relationship

D.3 AssertedInference

One or more assertions (e.g. Claims) can be linked together using the AssertedInference relationship. The direction of the AssertedInference relationship starts from the supporting element to the supported element. When used in a diagram, a connected dot is used as a connection point when more than one AssertedInferences are connected.

For example, Claim G1 is supported by Claim G2 and G3. The direction of the AssertedInference relationship is start from the supporting elements, Claim G2 and G3, to the supported element, Claim G1.

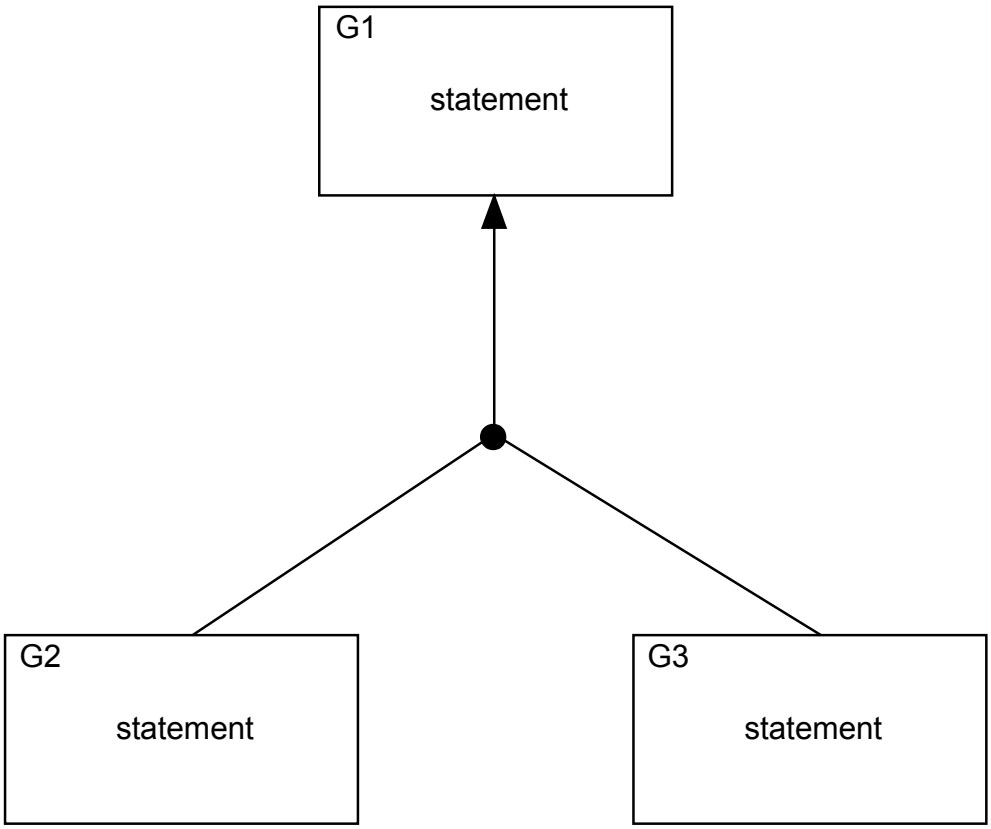


Figure D8 – Example of AssertedInference of Supporting Claims

D.4 ArtifactReference and AssertedEvidence

AssertedEvidence can be used to record the declaration that one or more artifacts of Evidence (cited by ArtifactReference) provide supporting information that helps establish the truth of a Claim. When used in a diagram, the direction of the AssertedEvidence relationship starts from the evidence (cited by ArtifactReference) to the supported element. The position of the ArtifactReference as evidence must be located below the supported element.

For example, Claim G4 is supported by Evidence E1 (cited by ArtifactReference), connected via AssertedEvidence relationship.

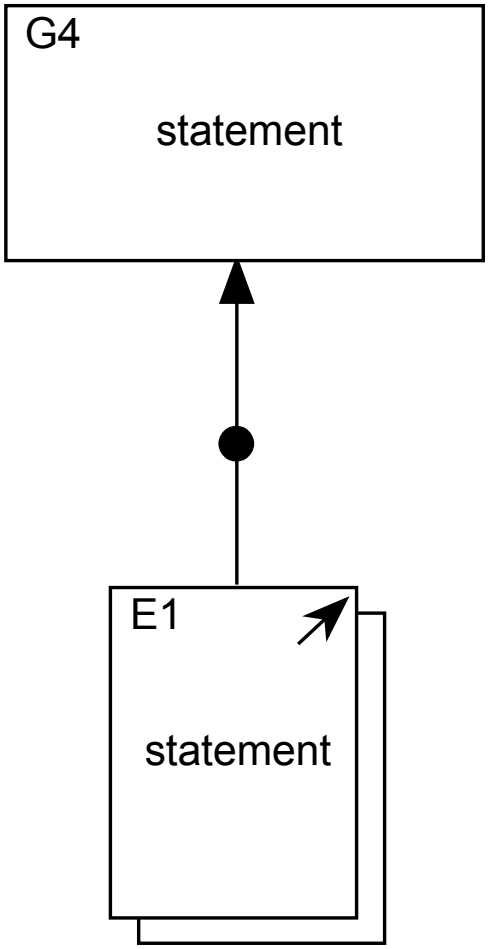


Figure D9 – Example of ArtifactReference Citation via AssertedEvidence

In another case, ArtifactReference as evidence can be used to support another ArtifactReference, for example ArtifactReference as context, to provides evidential information. In this case, ArtifactReference as evidence E2 is declared to support ArtifactReference as context C1. The ArtifactReference as evidence E2 is located below ArtifactReference C1. The relationship between them is declared using the AssertedArtifactSupport.

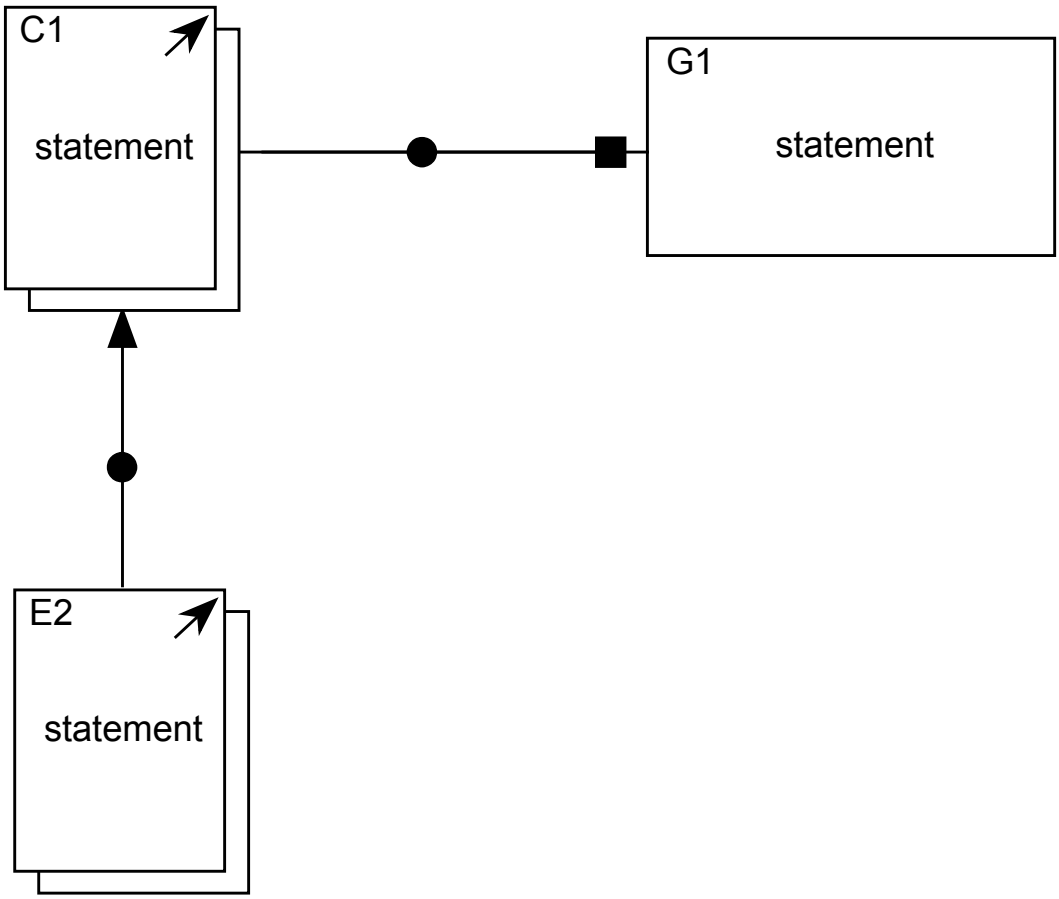


Figure D10 – Example of ArtifactReference Support of Another ArtifactReference

D.5 AssertedContext

AssertedContext can be used to declare that the artifact (cited by an ArtifactReference) provides the context for the interpretation and scoping of a Claim. When used in a diagram, the source element of the AssertedContext must be an ArtifactRefence element, and the targeted element can be the Assertion type element (e.g. Claim). The location of the ArtifactReference as a context must be located on the left and right side of the targeted element.

For example, ArtifactReference C1 as a context provides contextual information to the Claim G1 that is connected using AssertedContext relationship.

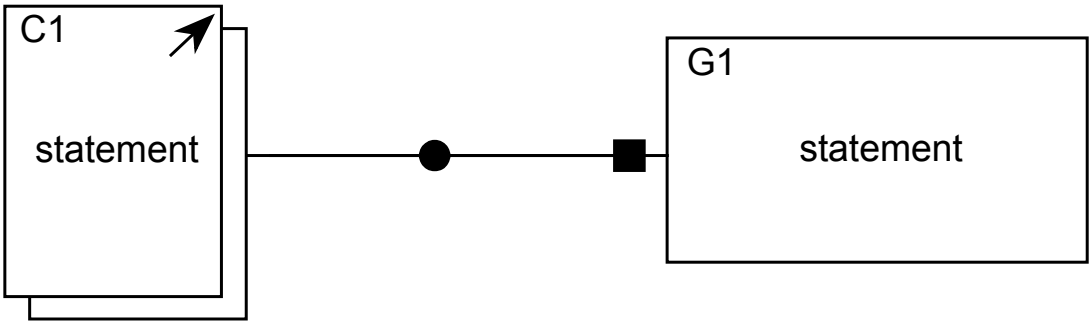


Figure D11 – Example of AssertedContext

In another case, ArtifactReference as context can be used to provides contextual information to another ArtifactReference (as evidence). In this case, ArtifactReference as context C2 is located on the right side of the ArtifactReference as evidence E1. The relationship between them is declared using the AssertedArtifactContext relationship.

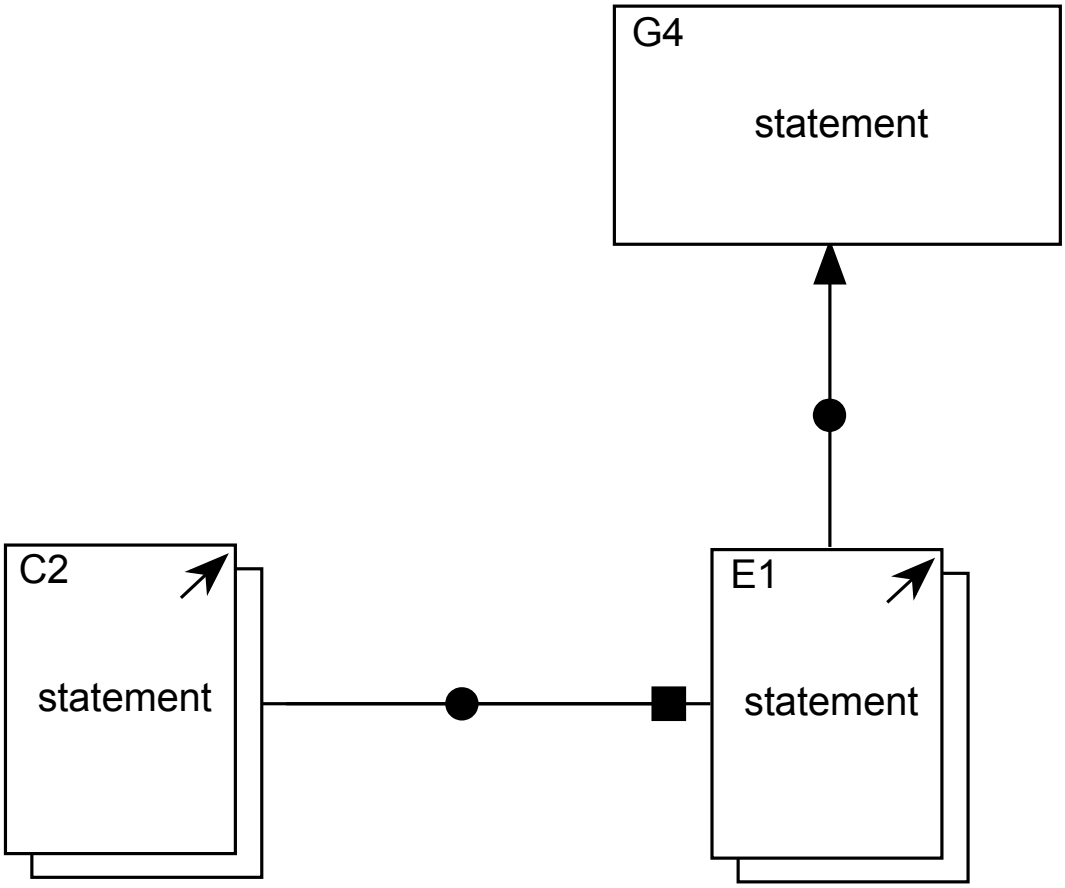


Figure D12 – Example of AssertedArtifactContext

Annex E: Illustration of PackageInterfaces and PackageBindings Usage with Assurance Cases

(informative)

This annex includes a series of illustrations and Venn diagrams that show how PackageInterfaces and PackageBindings can be used to combine elements of separate AssuranceCasePackages. The example uses a trivial number of elements from an assurance case for an engine and from an assurance case for controls for that engine that are combined (presumably along with many other elements) into an assurance case for a complete automobile (along with many other elements not identified).

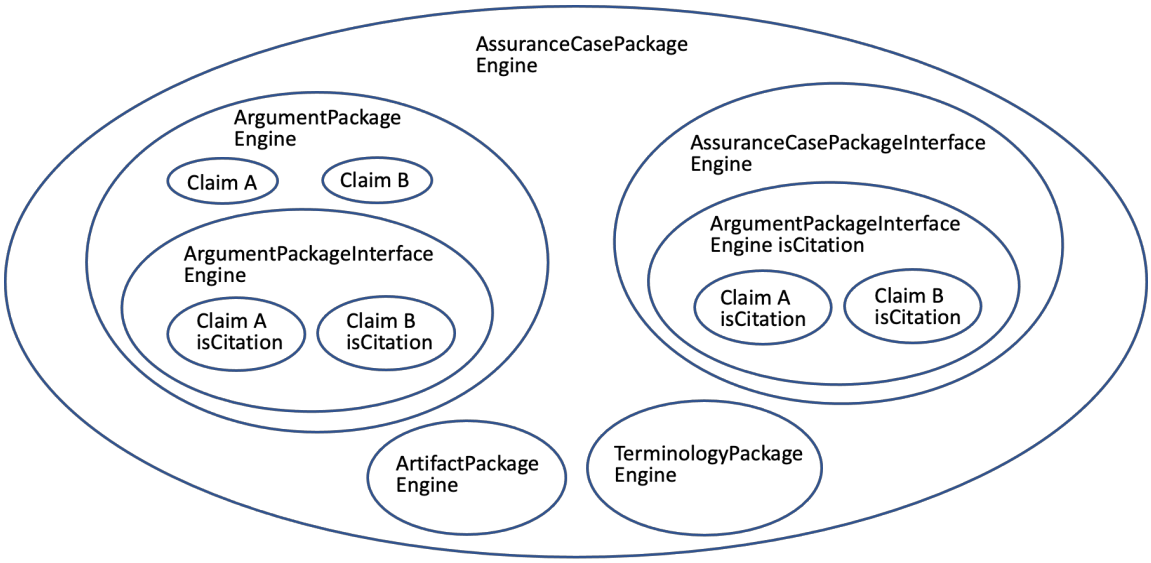


Figure E1 – Engine Assurance Case Illustration

In Figure E1, ArgumentPackageInterface for Engine is shown as contained within the ArgumentPackage for Engine which itself is contained within AssuranceCasePackage for Engine. The ArgumentPackageInterface for Engine is also cited within the AssuranceCasePackageInterface for Engine. Claim A and Claim B are defined within the ArgumentPackage and cited within both the actual and cited ArgumentPackageInterface.

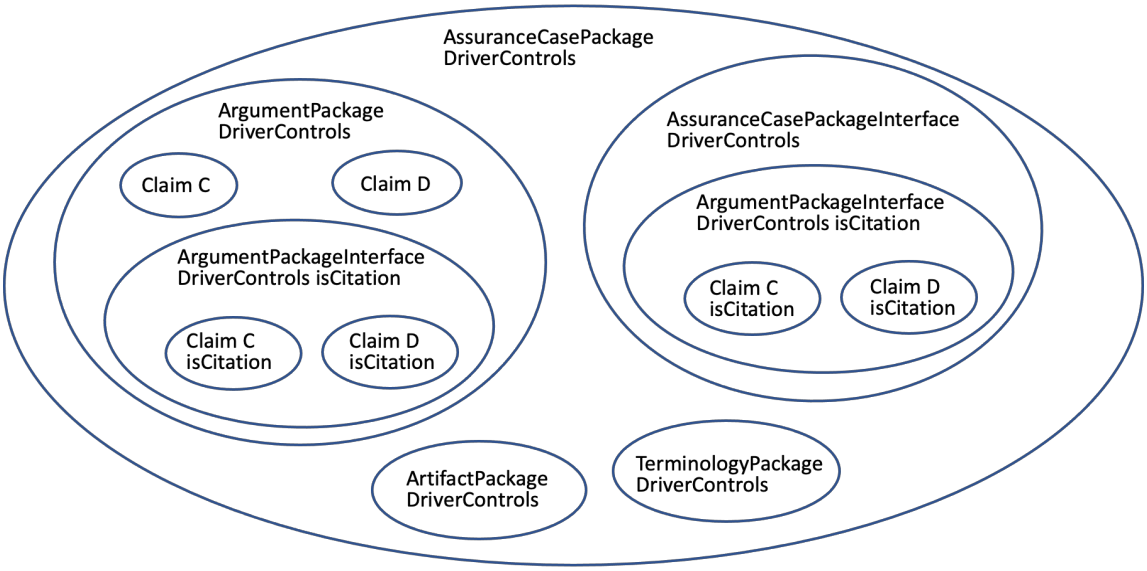


Figure E2 – Driver Controls Assurance Case Illustration

Figure E2 shows how the ArgumentPackageInterface for DriverControls is contained within the ArgumentPackage for DriverControls which itself is contained within AssuranceCasePackage for DriverControls.

The ArgumentPackageInterface for DriverControls is also cited within the AssuranceCasePackageInterface for DriverControls. Claim C and Claim D are defined within the ArgumentPackage and cited within both the actual and cited ArgumentPackageInterface.

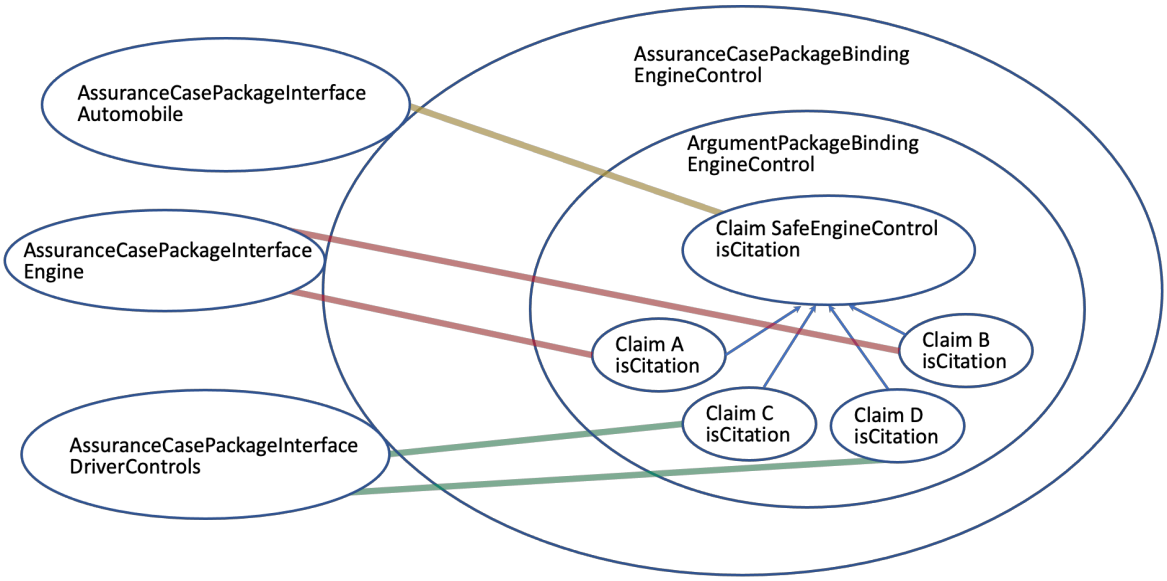


Figure E3 – Assurance Case Bindings Illustration

Figure E3 shows how the AssuranceCasePackageBinding for EngineControl externally references its participant packages, specifically, the AssuranceCasePackageInterface for Automobile, Engine, and DriverControls. It contains the ArgumentPackageBinding for EngineControl. This binding shows Claims A and B from Engine and Claims C and D

from DriverControls, in support of Claim SafeEngineControl from Automobile. The colored lines show how each claim in the ArgumentPackageBinding refers to the appropriate participant package through an AssuranceCasePackageInterface.

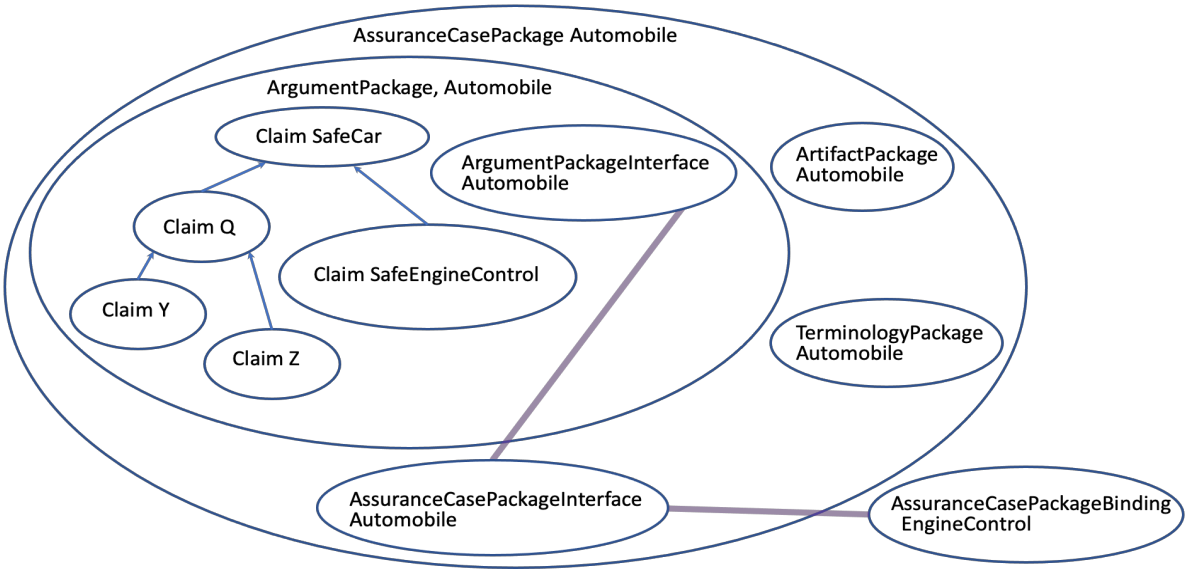


Figure E4 – Automobile Assurance Case Illustration

Figure E4 shows the overall assurance picture for Automobile. AssuranceCasePackage for Engine and for DriverControls, as well as AssuranceCasePackageBinding for EngineControl are expanded from those in the previous figures above. The ArgumentPackage for Automobile shows local claims, Y, Z, and Q, as well as Claim SafeEngineControl, which is supported by claims A, B, C, and D in the Binding. The colored lines show how elements in the ArgumentPackageBinding for EngineControl refer to the AssuranceCasePackageInterface for Automobile, and in turn to the ArgumentPackageInterface for Automobile. Similar relationships might also be needed for ArtifactPackage and/or TerminologyPackage.

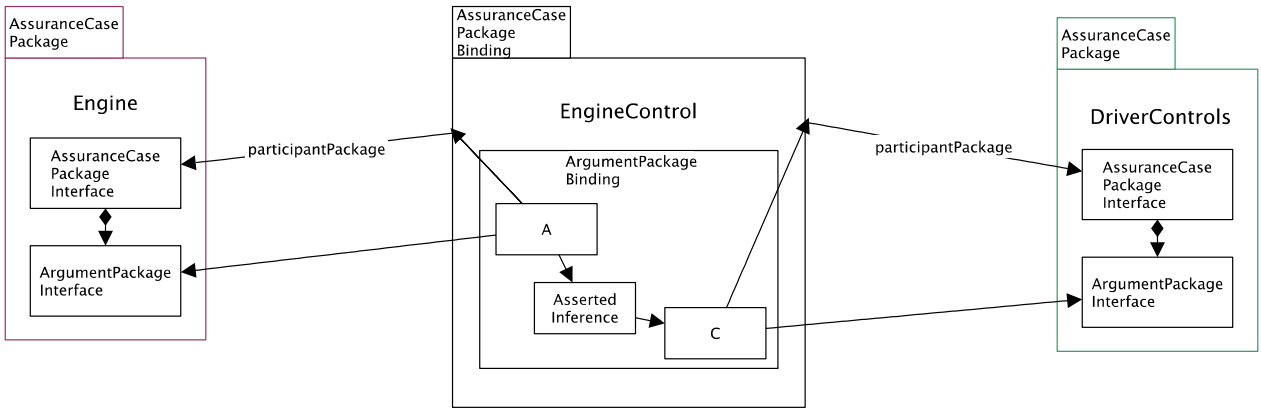


Figure E5 – Logical Relationships between DriverControls, EngineControl, and Engine Assurance Cases Illustration

In Figure E5, above, the logical relationships between DriverControls, EngineControl, and Engine Assurance Cases are shown, where some claims are omitted to improve readability.

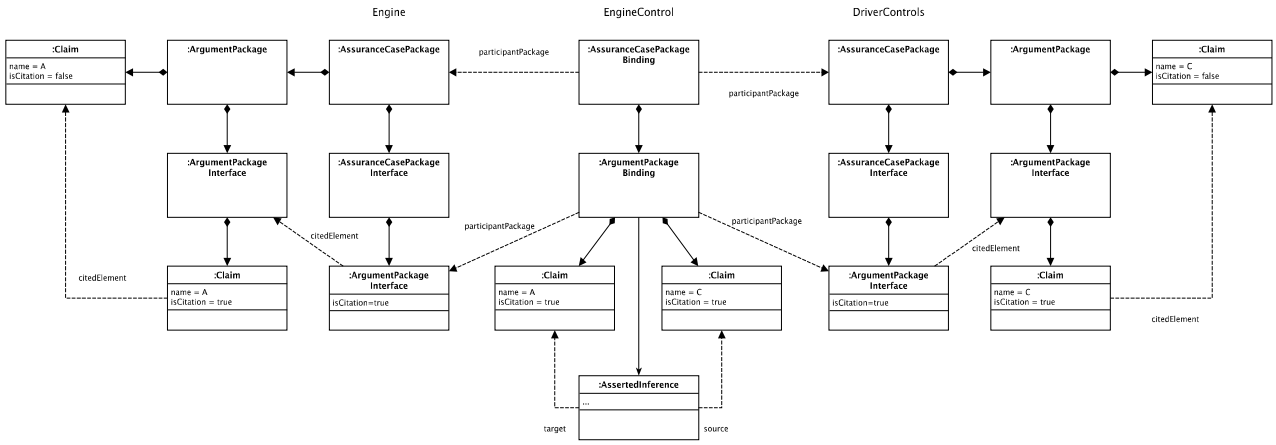


Figure E6 – Specific Relationships between DriverControls, EngineControl, and Engine Assurance Cases Illustration

Finally, in Figure E6 the specific (model element wise) relationships between DriverControls, EngineControl, and Engine Assurance Cases are shown, where some claims are omitted to improve readability.

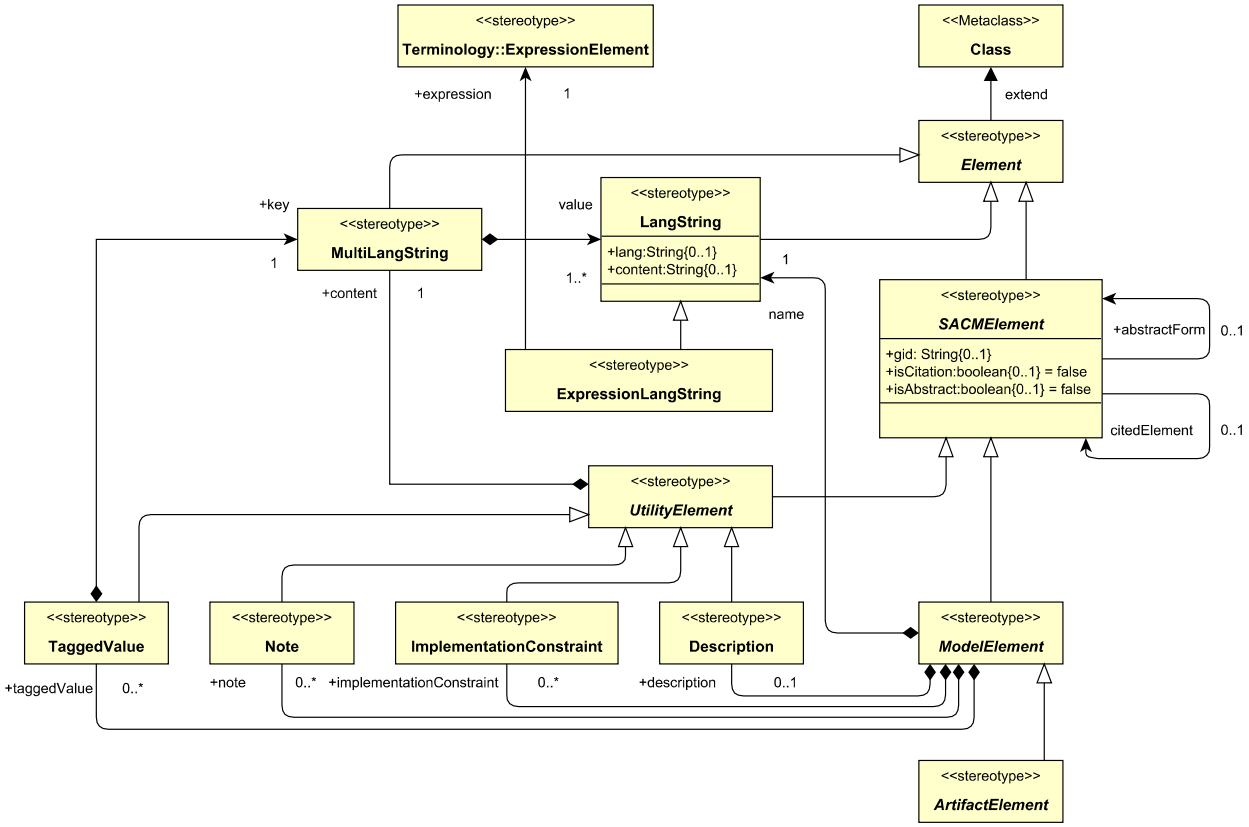
Annex F: SACM UML profile

(normative)

The SACM UML profile is created by extending meta element Class and Association from standard UML.

F.1 Base Component

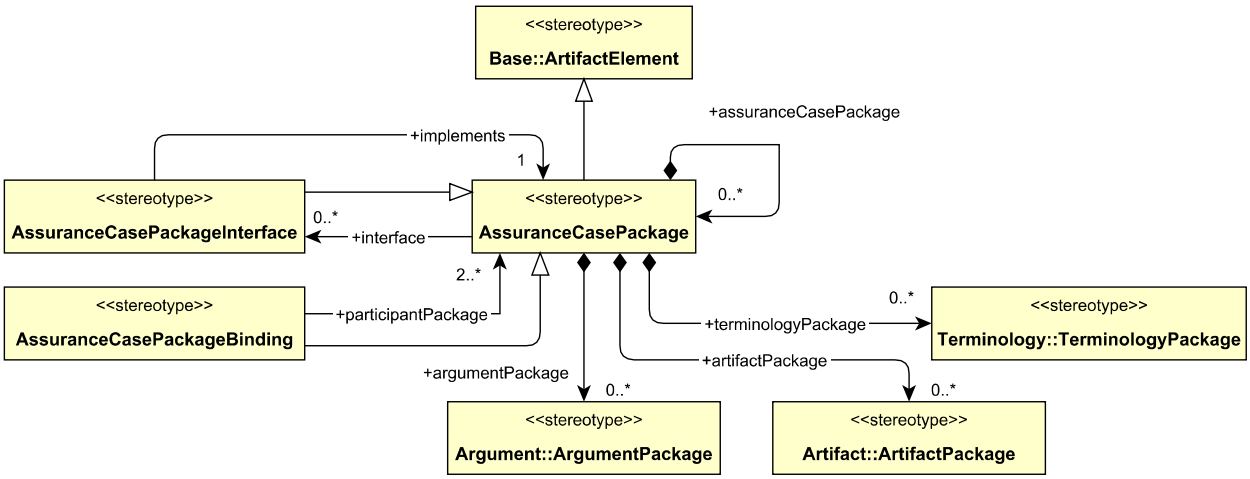
The base component of the SACM profile is Shown in F.1. In this component, the metaclass Class from the standard UML is imported and is generalized by Element of SACM, which is defined as a stereotype. Other elements of SACM are generalizations of Element and are also defined as stereotypes. The descriptions of the elements in SACM can be found in the specification.



F.1 Base component of the SACM profile.

F.2 Assurance case component

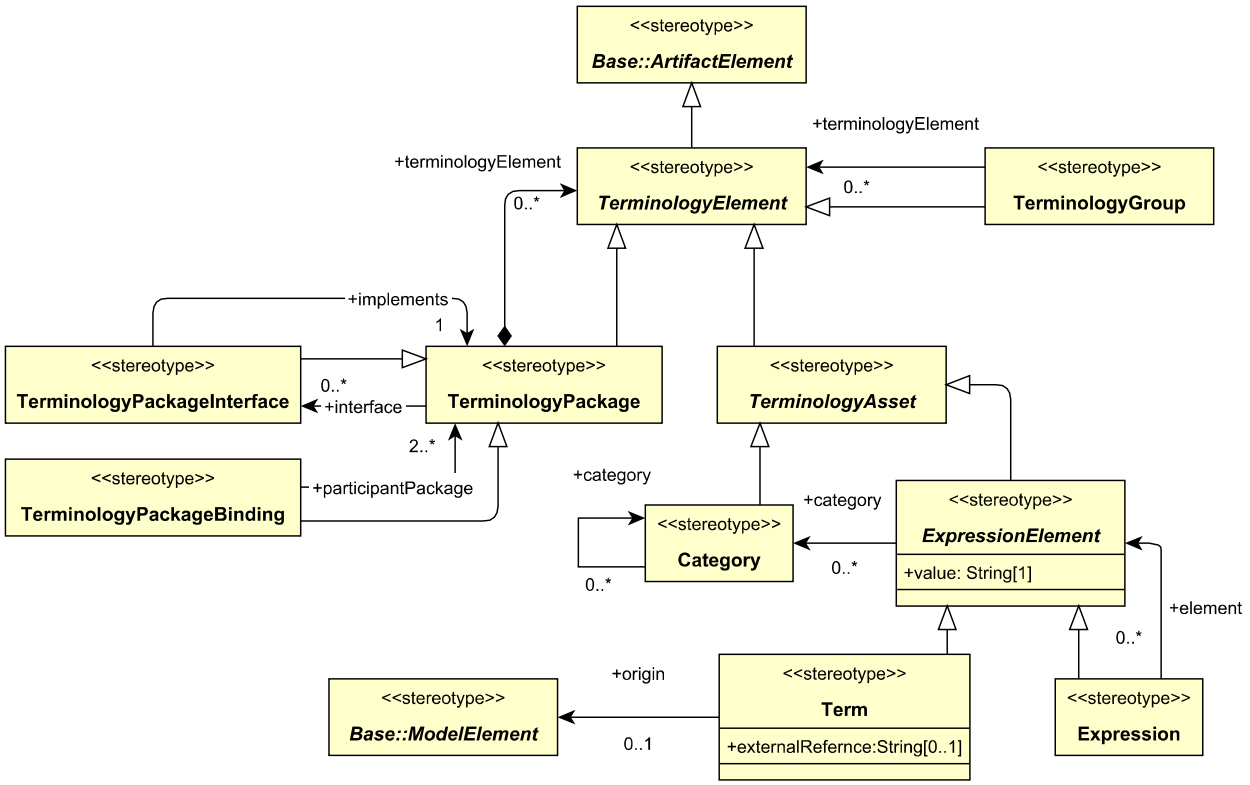
The assurance case component of the SACM profile is Shown in F.2. In this component, all elements are defined as stereotypes, the descriptions of which can be found in the specification. In particular, stereotype *AssuranceCasePackage* is defined as a generalization of stereotype *ArtifactElement*, which is defined in the base component.



F.2 Assurance case component of the SACM profile.

F.3 Terminology component

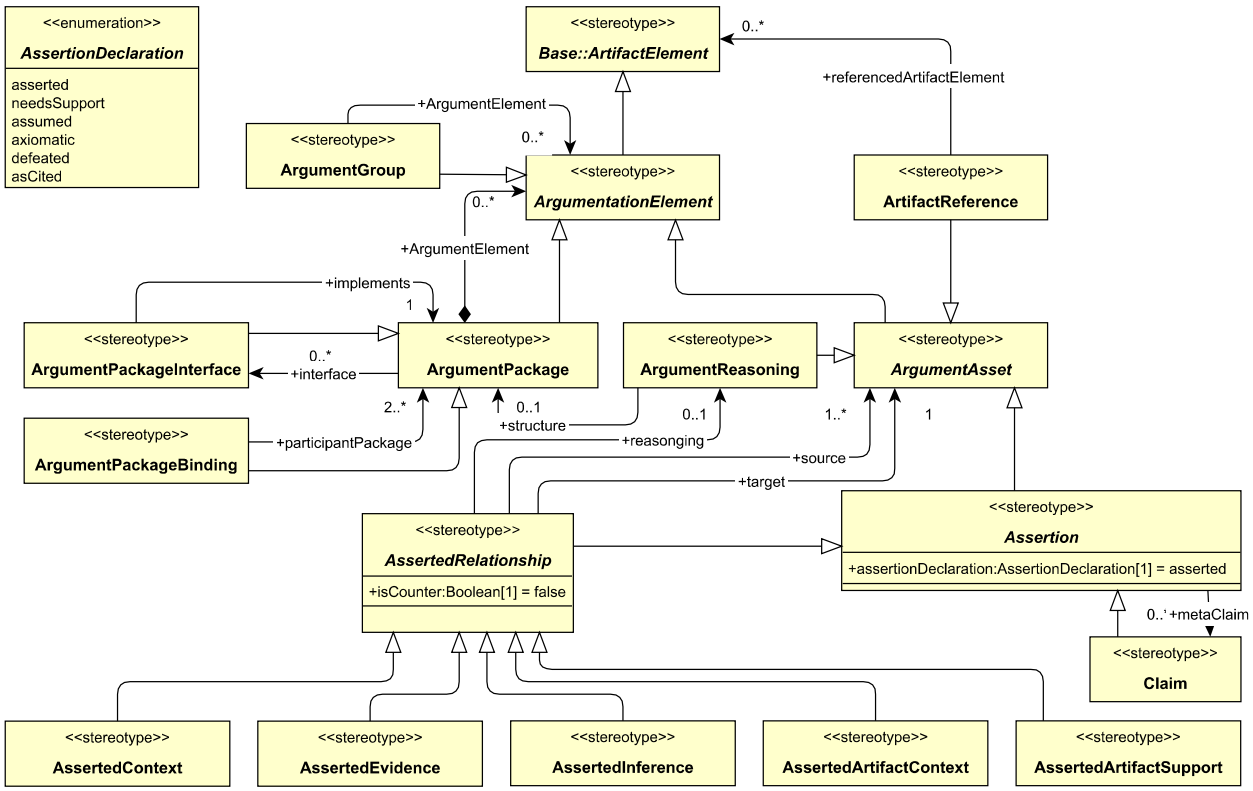
The terminology component of the SACM profile is Shown in F.3. In this component, all elements are defined as stereotypes, the descriptions of which can be found in the specification. In particular, stereotype TerminologyElement is a generalization of stereotype ArtifactElement, which is defined in the base component.



F.3 Terminology component of the SACM profile.

F.4 Argumentation component

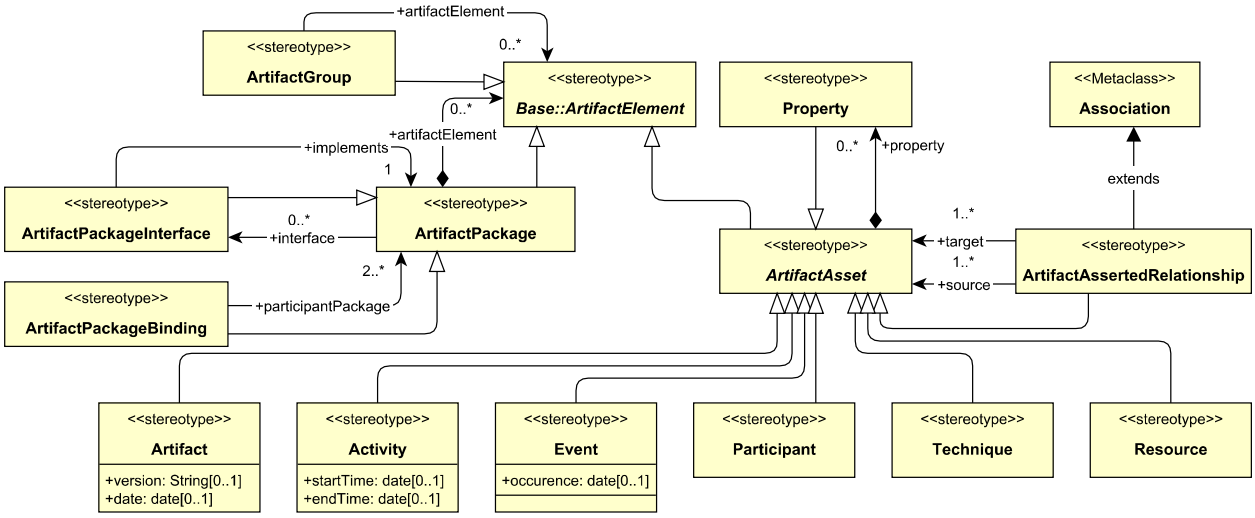
The argumentation component of the SACM profile is Shown in F.4. In this component, all elements are defined as stereotypes, the descriptions of which can be found in the specification. In particular, stereotype ArgumentationElement is a generalization of stereotype ArtifactElement, which is defined in the base component. It is to be noted that stereotype *AssertedRelationship* does not import ***Association*** from the standard UML package, it is due to the fact that *AssertedRelationship* acts as an object in SACM's concrete syntax, although it has a notion of source and target.



F.4 Argumentation component of the SACM profile.

F.5 Artifact component

The artifact component of the SACM profile is Shown in F.5. In this component, all elements are defined as stereotypes, the descriptions of which can be found in the specification. In particular, stereotype ArtifactElement is a generalization of stereotype ArtifactElement, which is defined in the base component. It is to be noted that stereotype *ArtifactAssertedRelationship* extends metaclass ***Association*** from the standard UML package, its defines additional source and target features which are both navigable.



F.5 Artifact component of the SACM profile.