Date: June 2026

Structured Assurance Case Metamodel (SACM)

Version 2.4

OMG Document Number: ptc/2026–06–34

Standard Document URL: https://www.omg.org/spec/SACM /2.4b1

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 Specifications, Report a Bug/Issue.

Table of Contents

Preface

About the Object Management Group

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 Meta-model); 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 Formal Specifications are available from this URL: *https://www.omg.org/spec*

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/IEC standards. Please consult: http://www.iso.org

Issues

The reader is encouraged to report and technical or editing issues/problems with this specification to: https://www.omg.org/report_issue.htm

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 assurance claim (a claim possibly may be a requirement of the system). An Assurance Case is a document that facilitates information exchange between various system stakeholder such as suppliers and acquirers, and between the operator and assessors, where the knowledge related to the assurance 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 assurance arguments, along with the corresponding evidence.

Systems Assurance is the process of building clear, comprehensive, and defensible arguments regarding the assurance of systems. The vital element of Systems Assurance is that it makes clear and well–defined claims about the specific desired assurance of systems. Classically, systems are desired to be assured as secure or safe, but a system may also need to be assured of other critical properties. 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, constrained, or formal 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.

1.1 Structured Arguments

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

The Argument Metamodel is designed to stand alone or may be used in combination with the Artifact and/or Terminology parts of the 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 in a coordinated strategy.

The Argument Metamodel 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.

1.2 Evidence

Part of this specification provides a metamodel for communicating the way in which evidence artifacts 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.

1.3 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 assurance 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.

1.4 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.

In SACM 2.4, updates, restructuring, and additions to the SACM 2.3 model have been made to address noted issues. With these changes SACM 2.4 :

- is a “standard complete” MOF model
- is usable by a wider set of demographics (including System Engineers)
- is usable by people using their favorite tools (e.g. Profiles)
- is embeddable in other modeling languages
- is able to reference other models (and elements within those models)
- is based on UML and KerML (the basis of SysML2) concepts
- provides tighter semantics in the metamodels to replace unneeded OCL statements
- provides strength weights for defining the strength of the relationships between argument elements
- provides various Assurance Case Diagram types
- widens the applicability of various elements in the model (e.g. Properties)
- is more specific about packaging semantics
- now supports all concepts from GSN, CAE, Assurance 2.0
- supports Architecture Views (see GSN)
- supports other types of logics (e.g. Inductive, Deductive, Abductive, other)
- supports (multiple conflicting) Assessments of an Assurance Case
- supports Joins (compare with GSN)

2 Conformance

2.1 Introduction

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

- Argument Model
- Artifact Model
- Packaging Model
- Terminology Model
- SACM UML Profile
- Join compliance point
- Advanced Argument compliance point
- SACMView compliance point
- Assessment compliance point
- DataTimeDuration compliance point
- NamedValue compliance point

2.2 Argument Model compliance point

Software that conforms to the SACM specification at the Argument 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 Argument subpackage of the SACM specification, including the common elements defined in the Common and Predefined diagrams of the SACM. The top object of the Argument package as a unit of interchange shall be the Argument :: ArgumentPackage element of the SACM.

Conformance to the Argument 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. Mappings have been defined in Annex A to help explain the connections of SACM to GSN and CAE.

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 Packaging Model compliance point

This compliance point is mandatory. Software that conforms to the specification at the Packaging 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.

This conformance point also requires strict adherence to the restrictions imposed on various kinds of package ownership. these include :

- AssuranceCasePackage can own subpackages of type: AssuranceCasePackage, ArgumentPackage, ArtifactPackage, TerminologyPackage, and BindingPackage
- Section Packages (e.g. ArgumentPackage) can own another section package, or a section interface package (e.g. ArgumentInterfacePackage)
- Section Interface Package can own another Section Interface Package
- Section Packages and Interface Packages can own their Section Assets
- Any Package can own BaseElements
- There is only one BindingPackage rather than one for each section, and a BindingPackage can own any SACMElement

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.

6.6 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. Note : In the SACM metamodel there is a Foundation section that defines the semantics of various elements in SACM. This set of foundation concepts are already resident in the UML base classes that are extended by the SACM Profile elements, so there is no need for any representation of the Foundation in the Profile.

6.7 Join compliance point

Software that conforms to the SACM specification at the Join 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 non–normative MOF metamodel defined in the informative Join subpackage section of the specification.

6.8 Advanced Argument compliance point

Software that conforms to the SACM specification at the Advanced Argument 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 non–normative MOF metamodel defined in the informative AdvancedArgument subpackage section of the specification.

6.9 SACMView compliance point

Software that conforms to the SACM specification at the SACMView 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 non–normative MOF metamodel defined in the informative SACMView subpackage section of the specification.

6.10 Assessment compliance point

Software that conforms to the SACM specification at the Assessment 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 non normative MOF metamodel defined in the informative Assessment subpackage section of the specification.

6.11 DateTimeDuration compliance point

Software that conforms to the SACM specification at the DateTypeDuration 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 non–normative MOF metamodel defined in the informative DateTypeDuration subpackage section of the specification.

6.12 NamedValue Types compliance point

Software that conforms to the SACM specification at the NamedValue Types 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 non–normative MOF metamodel defined in the informative NamedValueTypes subpackage section of the specification.

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

6.13 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 2.4, 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 is illustrated through examples in Annex D. Note : the concrete syntax for other packages is not currently defined.

6 Additional Information

6.1 Changes to adapted OMG Specifications

This specification completely replaces the SACM 2.3 specification.

6.2 How to read this specification

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

Clause 7. Specification overview - Provides design rationale for the SACM Argument 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. Foundation classes define the foundational classes to support the Structured Assurance Case Metamodel.

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

Clause 10. SACMPakage defines the common package class for the Structured Assurance Case Metamodel.

Clause 11. Structured Assurance Case Packages defines the common packages of the Structured Assurance Case Metamodel.

Clause 12. SACM Terminology defines the common terminology classes of the Structured Assurance Case Metamodel.

Part 2 of the specification defines the SACM Argument metamodel. The Argument 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 Argument and describes how SACM Argument 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 13. The SACM Argument Metamodel – Provides the details of the Argument 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 14 defines the key elements of the Artifact Metamodel.

Part 4 of the specification defines the non–normative Advanced SACM Capabilities Classes. It defines the catalog of elements for constructing and interchanging precise statements involved in SACM Join, Advanced Arguments, SACM View, Assessment, NamedValue Type, and Argument Context capabilities. This part includes a single clause.

Material in this part of the specification is related to the non–normative advanced assurance capabilities and is not required for the other compliance points.

Clause 15 defines the key elements of the non–normative Advanced Assurance Capabilities.

6.3 Acknowledgements

The following companies submitted this specification :

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

The following companies supported this specification :

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

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 A 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

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

So, Conclusion

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

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 Foundation Classes, Base Classes, the Packaging Classes including the Structured Assurance Case Package, and the Terminology Classes.

Subsequent parts define the Argument Metamodel, Artifact Metamodel, and the non–normative Advanced SACM Capabilities Classes.

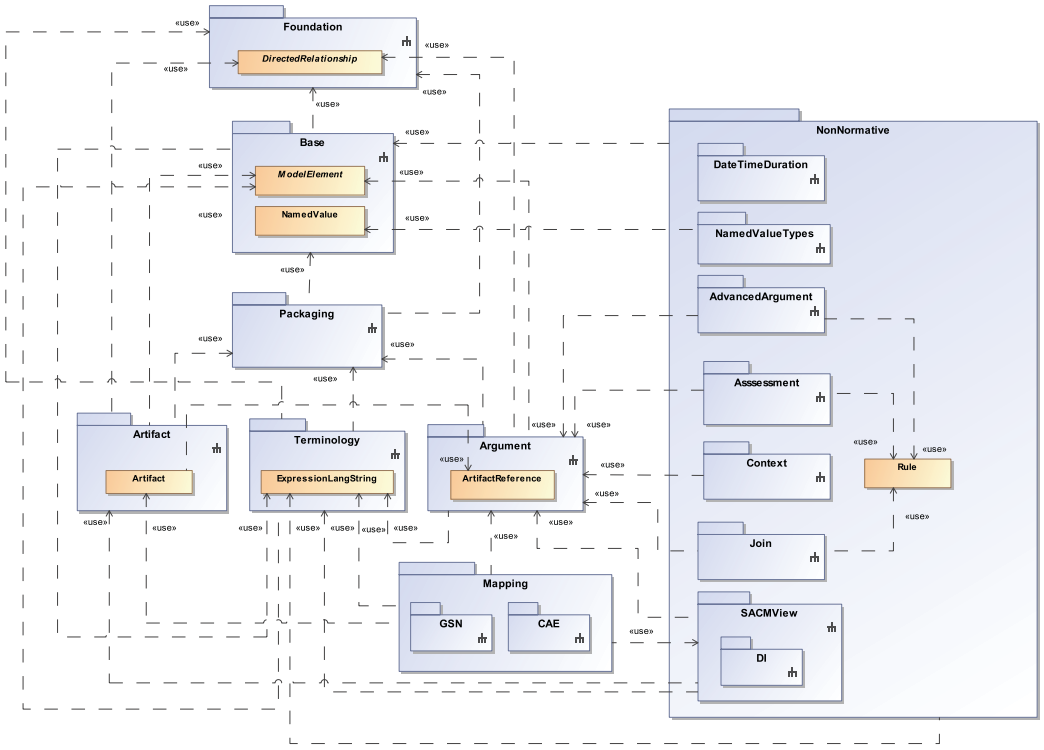

Figure 7.2 - Overall SACM Class Diagram

Clause 8, Foundation Classes.

Clause 9, Base Classes.

Clause 10 and 11, Packaging Classes which includes the Structured Assurance Case Packages.

Clause 12, Terminology Classes.

Clause 13, Argument Metamodel.

Clause 14, Artifact Metamodel.

Clause 15, Advanced SACM Capabilities Classes (informative).

This page intentionally left blank.

8 Foundation Class

8.1 General

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

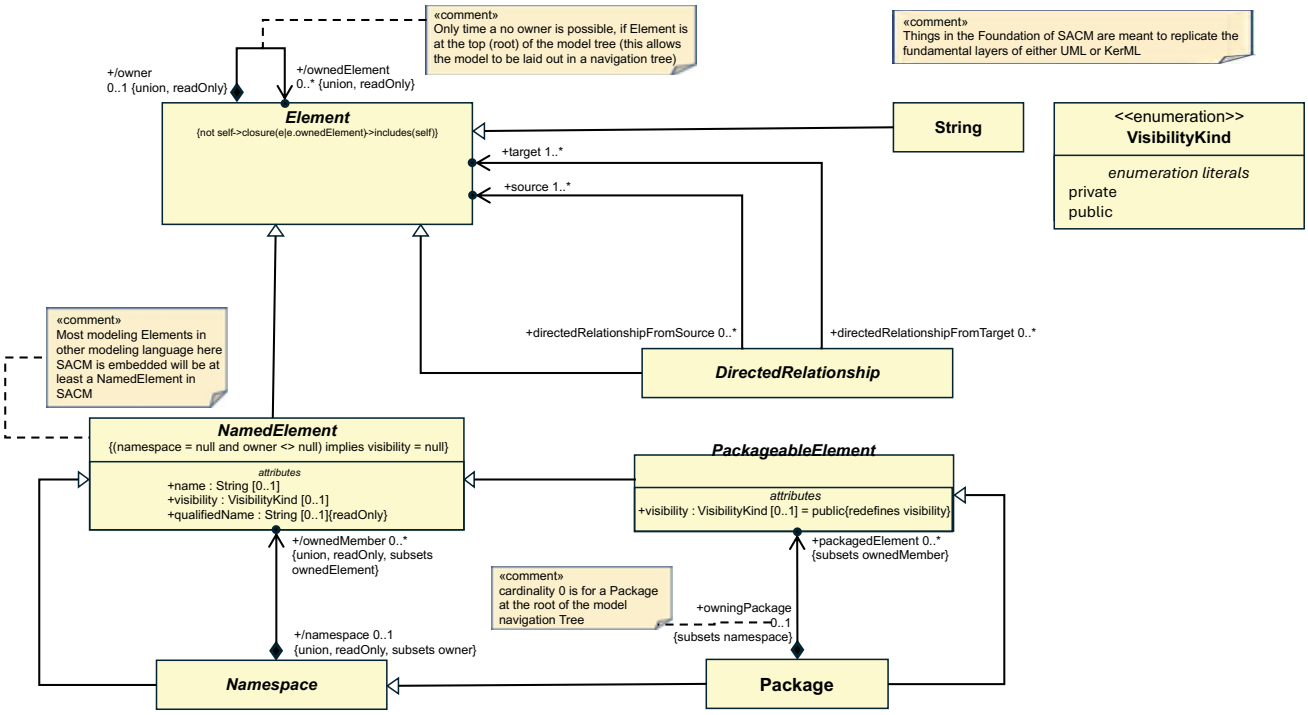

Figure 8.1 – Foundation Class

8.2 DirectedRelalationship [Abstract Class]

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

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

AssociationEnds

source : Element [1..*] – Specifies the source Element of the DirectedRelationship.

target : Element [1..*] – Specifies the target Element of the DirectedRelationship.

8.3 Element [Abstract Class]

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

AssociationEnds

/ownedElement : Element [0..*] {union, readOnly} – The Elements owned by this Element.

Constraints

not_own_self

An element may not directly or indirectly own itself.

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

8.4 NamedElement [Abstract Class]

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

Attributes

name : String [0..1] – The name of the NamedElement.

/qualifiedName : String [0..1] {readOnly} – A name that allows the NamedElement to be iden–fied within a hierarchy of nested Namespaces. It is constructed from the names of the containing Namespaces star–ng at the root of the hierarchy and ending with the name of the NamedElement itself. (Separator of names is " :: ".)

visibility : VisibilityKind [0..1] – Determines whether and how the NamedElement is visible outside its owning Namespace.

Constraints

visibility_needs_ownership

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

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

8.5 Namespace [Abstract Class]

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

AssociationEnds

/ownedMember : NamedElement [0..*] {union, readOnly, subsets ownedElement} – A collec–on of NamedElements owned by the Namespace.

8.6 Package [Class]

A package can have one or more profile applica2ons to indicate which profiles have been applied. Because a profile is a package, it is possible to apply a profile not only to packages, but also to profiles.

Atributes

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

Specifies the packageable elements that are owned by this Package.

8.7 PackageableElement [Abstract Class]

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

Atributes

visibility : VisibilityKind [0..1] = public {redefines visibility}– A PackageableElement must have a visibility specified if it is owned by a Namespace. The default visibility is public.

8.8 VisibilityKind [Enumeration]

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

Literals

public

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

private

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

9 Structured Assurance Case Base Classes

9.1 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.

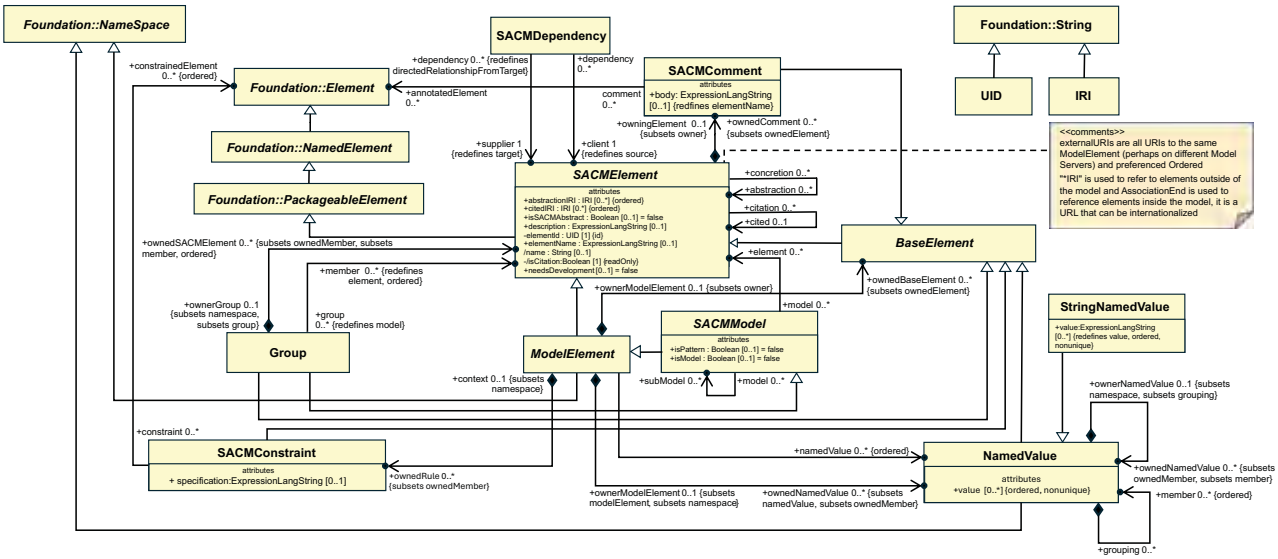

Figure 9.1 – Structured Assurance Case Base Classes Diagram

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

9.2 SACMElement (abstract)

SACMElement is the base class for SACM.

Superclass

Foundation :: PackagableElement

Attributes

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

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

isSACMAbstract : Boolean [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.

needsDevelopment [0..1] = false – indicates that this SACMElement still needs development. Application of this on certain elements can carry other particular semantics, such as Claims need to be supported by evidence or further argumentation.

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

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

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

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

AssociationEnds

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

abstraction : SACMElement [0..1] – a reference to another abstract SACMElement to which this concrete SACMElement conforms.

ownedComment : SACMComment [0..*] {subsets ownedElement} – SACMComments owned by this SACMElement.

Semantics

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

Constraints

AbstractionIsAbstract

An abstraction for a SACMElement must have isSACMAbstract=true.

inv : abstraction–>notEmpty() implies abstraction.isSACMAbstract=true

CitationKindOfCited

A citation must be of the same kind as the cited.

inv : cited–>notEmpty() implies oclKindOf(cited)

DeriveCitation

isCitation is true if cited or citationIRI is not empty.

inv : isCitation = citedIRI–>notEmpty() or cited–>notEmpty()

DerivedName

If there is an elementName and it has content, then name is the first of that content.

inv : elementName–>notEmpty() and elementName.content–>notEmpty() implies name=elementName.content–>first()

IfCitedAbstractCitationAbstract

If cited and cited is abstract, then citation is abstract.

inv : cited–>notEmpty() and cited.isSACMAbstract=true implies isSACMAbstract=true

9.3 ModelElement (abstract)

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

Superclass

SACMElement, Foundation :: NameSpace

AssociationEnds

ownedRule : SACMConstraint [0..*] {subsets ownedMember} – SACMConstraints owned by this ModelElement.

namedValue : NamedValue [0..*] (ordered) – a collection of NamedValues, NamedValues can be used to describe additional features of a ModelElement.

ownedNamedValue : namedValue [0..*] – {subsets namedValue, subsets ownedMember} – NamedValues owned by this ModelElement

Semantics

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

9.4 BaseElement (abstract)

BaseElement is the base element for a number of auxiliary elements which can be used in any diagram type.

Superclass

SACMElement

9.5 SACMContraint

SACMConstraint specifies details of any constraints that must be satisfied.

Superclass

BaseElement

Attributes

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

AssociationEnds

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

9.6 SACMComment

This class specifies a generic note that may be associated with an Element. For example, a SACMComment may include a number of explanatory comments.

Superclass

BaseElement

Attributes

body : ExpressionLangString [0..1] {redefines elementName} – The body (i.e. content) of the comment.

AssociationEnds

annotatedElement : Element [0..*] – Elements to which the SACMComment applies.

Semantics

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

9.7 NamedValue

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

Superclass

BaseElement, Foundation :: Namespace

Attributes

value [0..1] {ordered, nonunique} – the key of the NamedValue.

AssociationEnds

member : NamedValue [0..*] {ordered} – NamedValues owned by this NamedValue

ownedNamedValue [0..*] {subsets ownedMember} – NamedValues owned by this NamedValue

Semantics

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

9.8 Group

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

Superclass

BaseElement, SACMModel

AssociationEnds

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

ownedSACMElement [0..*] {subsets ownedMember, subsets member} – SACMElements that are owned by this Group

9.9 IRI

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

Superclass

Foundation :: String

9.10 SACMModel

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

Superclass

ModelElement

Attributes

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

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

AssociationEnds

+subModel : SACMModel [0..*] {ordered} – subModel is a recursive containment of elements in the model.

+elements : SACMElement [0..*] – elements are items in the model.

9.11 SACMDependency

SACMDependency provides a dependency relationship between two SACMElements. The source of the dependency is called the client and the target is the supplier. The base semantics of the relationship is that if the supplier changes, the client may need to change.

Superclass

Foundation :: DirectedRelationship

AssociationEnds

supplier : SACMElement [1] {redefines target}

client : SACMElement [1] {redefines source}

9.12 StringNamedValue

StringNamedValue is a NamedValue whose value is of type String

Superclass

NamedValue

Attributes

value : ExpressionLangString [0..*] {redefines value, ordered, nonunique}

This page intentionally left blank.

10 SACM Package

10.1 General

This chapter presents the SACM Package information for what is allowed in each package and what are their containment restrictions.

The top level package is a SACMAssuranceCasePackage which allows containment of other SACMAssuranceCasePackages and of any domain packages (i.e. TerminologyPackage, ArtifactPackage, and/or ArgumentPackage). In addition, SACMAssuranceCasePackage can contain BaseElements (i.e. Group, NamedValue and its specializations, SACMComment, SACMConstraint, SACMDependency). In addition, any package of any type can contain a SACMBindingPackage.

Each domain Package can contain any domain Concept (e.g. a TerminologyPackage can contain another TerminologyPackage, a TerminologyInterfacePackage, a TerminologyAsset, which are the other elements in the Terminology domain). A domain InterfacePackage can contain another domain InterfacePackage (e.g. a TerminologyInterfacePackage can contain another TerminologyInterfacePackage).

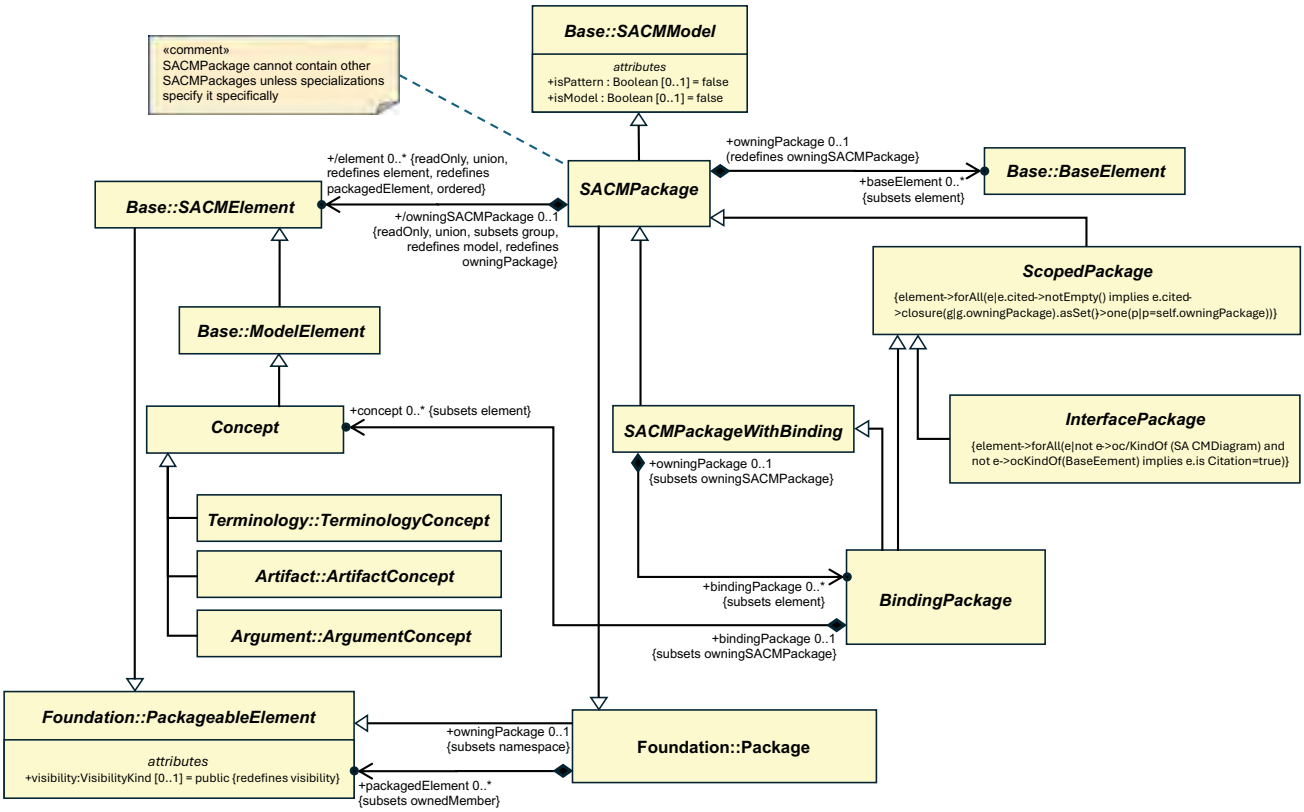

Figure 10.1 – SACM Package Class Diagram

10.2 SACMPackage (abstract)

Abstraction for all the SACM specific packages. SACMPackage constrains what types of SACMElements specializations can have and requires that all specializations allow for containment of Base :: BaseElement.

Superclass

Foundation :: Package, Base :: SACMModel

AssociationEnds

+/element : SACMElement [0..*] {union,readOnly,ordered,redefines element, redefines packagedElement} – elements in this package

+baseElement : BaseElement [0..*] {subsets element} – BaseElement subset of element

10.3 Concept (abstract)

Abstraction for TerminologyConcept, ArtifactConcept, and ArugmnentConcept. A Concept includes assets and elements (packaging) from the various different domains.

Superclass

Base :: ModelElement

10.4 ScopedPackage (abstract)

Cited elements in a scoped package are restricted to be either siblings to the scoped package (i.e. has the same parent package) or one of their parents (recursively) has the same parent package.

Superclass

SACMPackage

Constraints

CitedElementsAreScoped

If elements in the package are citations, then the cited must be members (recursively) in the same package structure as the ScopePackage.

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

10.5 BindingPackage

A BindingPackage contains cited elements and additional elements that allow the stitching together or other domain packages.

Superclass

SACMPackageWithBinding, ScopedPackage

AssociationEnds

+concept : Concept [0..*] {subsets element} – concepts contained in this binding package.

10.6 SACMPackageWithBinding (abstract)

An abstract of packages which can contain a binding package.

Superclass

SACMPackage

AssociationEnds

+bindingPackage : BindingPackage [0..*] {subsets element} – binding packages contained in this package

10.7 InterfacePackage (abstract)

Abstraction for TerminologyInterfacePackage, ArtifactInterfacePackage, and ArugmnentInterfacePackage. Interface packages can have BaseElements and Diagrams as well as their respective domain concepts, but those domain concepts must be citations.

Superclass

ScopedPackage

Constraints

MustBeCited

All elements in an interface except BaseElement and Diagrams must be citations.

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

11 Structured Assurance Case Packages

11.1 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.

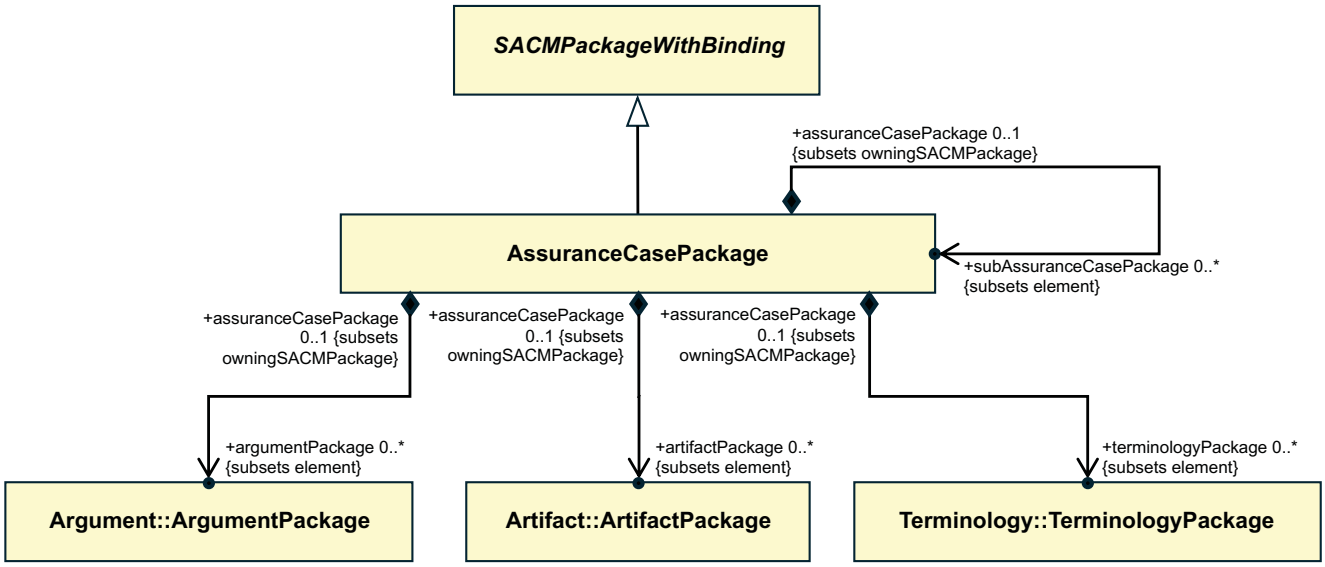

Figure 11.1 – Structured Assurance Case Packages Class Diagram

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, which are specializations of SACMModel, can set isModel=true to indicate that what is contain in the Package should be considered a completed model. If isPattern=true, then what is in the package should be considered a pattern. 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).

11.2 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.

Superclass

SACMPackageWithBinding

AssociationEnds

subAssuranceCasePackage : AssuranceCasePackage [0..*] {subsets element} – a collection of optional sub–packages

artifactPackage : ArtifactPackage [0..*] {subsets element} – a number of optional artifact sub–packages
terminologyPackage : TerminologyPackage [0..*] {subsets element} – a number of optional terminology sub–packages
argumentPackage : Argument :: ArgumentPackage [0..*] {subsets element} – a number of optional argument packages.

Semantics

AssuranceCasePackage is the root class for creating structured assurance cases.

11.3 BindingPackage

Sub–packages within any SACMPackage types can be bound together by means of a BindingPackage. Elements within the recursive sibling Packages of the BindingPackage can be "bound" together using relationships (which are already defined in the SACM language). The Elements from those recursive sibling Packages must have isCitation=true. Other new elements may be introduced and contained within the BindingPackage for the purposes of binding.

Superclass

ModelElement

This page intentionally left blank.

12 Structured Assurance Case Terminology Classes

12.1 General

TerminologyElement is an abstract class that serves as a parent class for all packaging in Terminology (TerminologyPackage and TerminologyInterfacePackage).

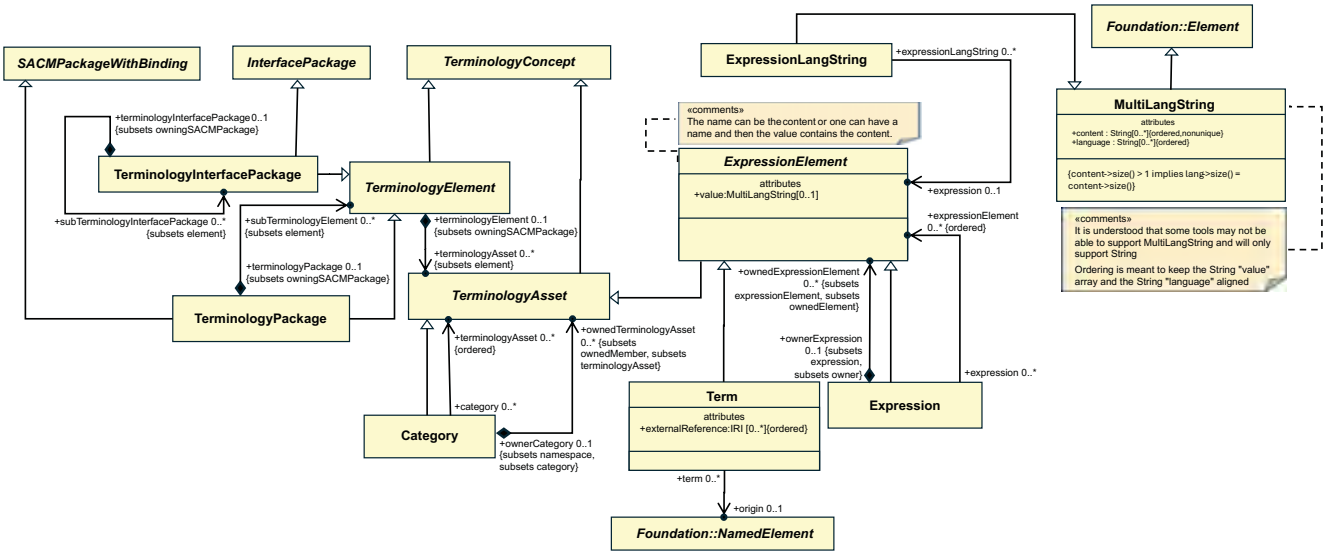

Figure 12.1 – Terminology Class Diagram

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.

12.2 TerminologyElement (abstract)

TerminologyElement is an abstract class that serves as a parent class for all packaging in Terminology (TerminologyPackage and TerminologyInterfacePackage).

Superclass

TerminologyConcept

AssociationEnds

terminologyAsset [0..*] {subsets element} – TerminologyAssets that are contained in this TerminologyElement

12.3 TerminologyPackage

The TerminologyPackage is the container element for SACM terminology assets.

Superclass

TerminologyElement

AssociationEnds

subTerminologyElement : TerminologyElement [0..*] {subsets element} – subTerminologyElement contained in the TerminologyPackage.

Semantics

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

12.4TerminologyInterface

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

AssociationEnds

subTerminologyInterfacePackage : TerminologyInterfacePackage [0..*] {subsets element} – TerminologyInterfacePackages owned by this TerminologyInterfacePackage.

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.

12.5TerminologyAsset (abstract)

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

Superclass

TerminologyConcept

Association

ownedMember : TerminologyAsset [0..*] {subsets ownedMember} – TerminologyAssets contained in other TerminologyAssets.

Semantics

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

12.6 Category

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

Superclass

TerminologyAsset

AssociationEnds

terminologyAsset : TerminologyAsset [0..*] {ordered} – elements contained in the Category

+ownedTerminologyAsset 0..* {subsets ownedMember, subsets terminologyAsset} – TerminologyAssets owned by this 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.

12.7 ExpressionElement (abstract)

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

Superclass

TerminologyAsset

Attributes

value : MultiLangString [0..1] – the value of the expression.

AssociationEnds

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

Semantics

ExpressionElements are used to model (potentially structured) expressions in SACM. The name can be the content or one can have a name and then the value contains the content.

12.8 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

AssociationEndss

espressionElement : ExpressionElement [0..*] {ordered} – an optional reference to other ExpressionElements forming part of the structured Expression.

ownedExpressionElement [0..*] {subsets expressionElement, subsets ownedElement} – ExpressionElements owned by this 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 :

IfConcreteEverythingConcrete

If Expression is not abstract, then none of its ExpressionElements can be abstract.

inv : isSACMAbstract = false implies expressionElement–>forall(e|e.isSACMAbstract = false)

12.9Term

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

Superclass

ExpressionElement

Attributes

externalReference : IRI [0..*] {ordered} – an attribute recording an external reference (e.g., URI) to the object referred to by the Term

AssociationEnds

origin：Foundation :: NamedElement [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.

Constraints

OriginHasName

If a Term has an origin, then that origin must have a name.

inv : origin–>notEmpty() implies origin.name–>notEmpty()

OriginEmptyNameNotEmpty

If the origin is empty, then the Term must have a name.

inv : origin–>isEmpty() implies name–>notEmpty()

12.10 TerminologyConcept (abstract)

TerminologyConcept is an abstraction of all the packaging and elements in the terminology domain.

Superclass

Packaging :: Concept

This page intentionally left blank.

Part II – Argument Metamodel

This part of the specification defines the Argument Metamodel.

This page intentionally left blank.

13 SACM Argumentation Metamodel

13.1 General

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

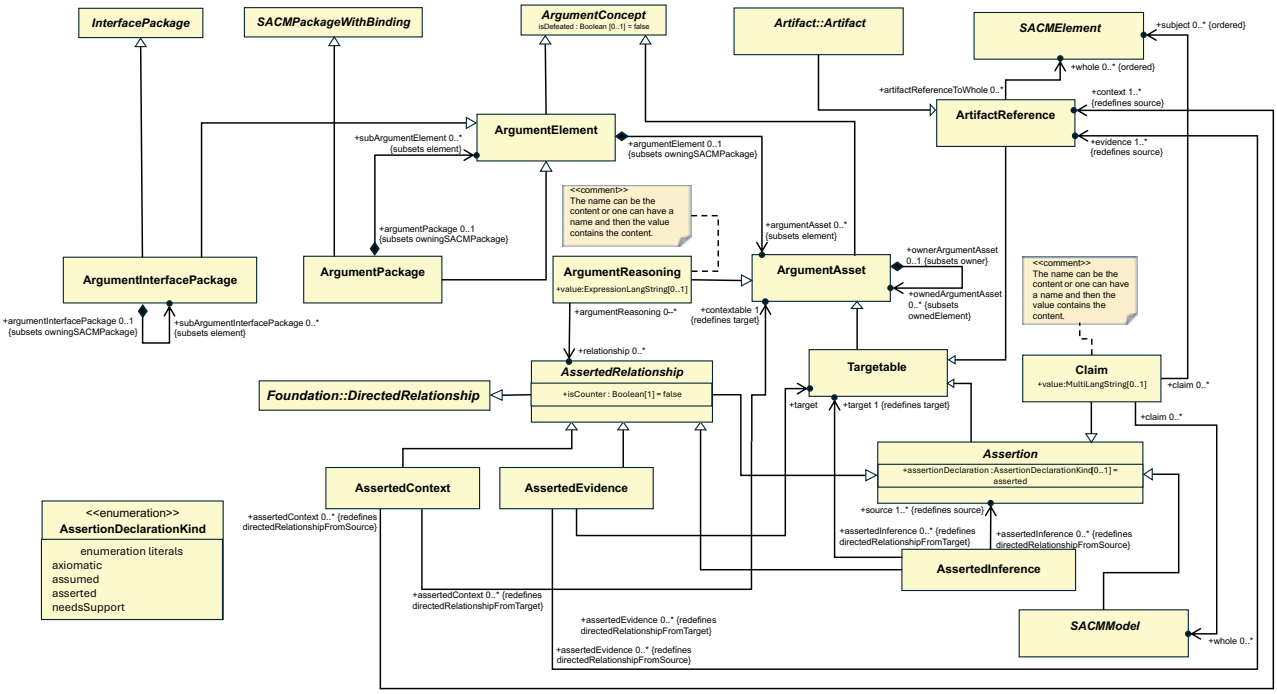

Figure 13.1 – Argumentation Package Diagram

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 ArgumentInterfacePackage. Within an ArgumentInterfacePackage, users create citations of the argument elements they select to disclose to external parties. Users are able to integrate ArgumentPackages through the use of ArgumentBindingPackage. An ArgumentBindingPackage binds ArgumentPackages together by including the declared ArgumentInterfacePackages 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).

13.2 ArgumentElement (abstract)

ArgumentElement is an abstract class that serves as a parent class for all packaging in Argument (ArgumentPackage and ArgumentInterfacePackage)..

Superclass

ArgumentConcept

AssociationEnds

argumentAsset [0..*] {subsets element} – ArgumentAssets contained in this ArgumentElement.

13.3 ArgumentPackage Class

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

Superclass

ArgumentElement

AssociationEnds

argumentElement : ArgumentElement [0..*] (composition) – a collection of ArgumentElements forming a structured argument

subArgumentElement [0..*] {subsets element} – ArgumentElements contained in this ArgumentPackage

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.

13.4 ArgumentInterfacePackage

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

Superclass

ArgumentElement

AssociationEnds

+asset : ArgumentAsset [0..*] – a reference to the ArgumentPackage which the ArgumentInterfacePackage declares.

Semantics

ArgumentInterfacePackages 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

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

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

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

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

13.5 ArgumentAsset (abstract)

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

Superclass

ArgumentConcept

AssociationEnds

ownedArgumentAsset : ArgumentAsset [0..*] {subsets ownedElement} – ArgumentAssets owned by this ArgumentAsset.

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.

13.6 AssertionDeclarationKinds (Enumeration)

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

Superclass

N/A

EnumerationLiterals

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

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

asserted – (default) indicating that an Assertion is asserted.

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

Semantics

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

13.7 ArtifactReference

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

Superclass

Targetable

AssociatonEnds

reference : SACMElement [0..*] {ordered} – artifactReference to the SACMElement.

Semantics

It is necessary to be able to reference SACMElement(s) to 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.

13.8 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

Targetable

Attributes

assertionDeclaration : AssertionDeclarationKind [0..1] = asserted – the declaration indicating the state of the Assertion.

Semantics

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

13.9 Claim

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

Superclass

Assertion

Attributes

value : MultiLangString [0..1] – The content for this Claim. The name can be the content or one can have a name and then the value contains the content.

AssociationEnds

whole : SACMModel [0..*] – the whole structure that this Claim is about

subject : SACMElement [0..*] {ordered} – the subject(s) this Claim

Semantics

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

13.10 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

Attributes

value : ExpressionLangString [0..1] - The content for this ArgumentReasoning. The name can be the content or one can have a name and then the value contains the content.

AssociationEnds

relationship : AssertedRelationship [0..*] – The AssertedRelationships to which this ArgumentReasoning applies.

Semantics

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

13.11 AssertedRelationship (abstract)

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

Superclass

Assertion, Foundation :: DirectedRelationship

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).

Semantics

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

13.12 AssertedInference

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

Superclass

AssertedRelationship

AssociationEnds

+source : Assertion [1..*] {redfines source} – the source must be undefeated in order for the target to be undefeated

+target : Targetable [1] {redefines target} – the source must be undefeated in order for the target to be undefeated

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. If AssertedInference has isCounter=false, then it should be interpreted as "if the source is undefeated then the target is undefeated". If AssertedInference has isCounter=true, then it should be interpreted as "If the source is undefeated then the target is defeated".

13.13 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

AssociationEnds

+evidence : ArtifactReference [0..*] {redefines source} – the evidence for the AssertedEvidence

+target : Targetable [1] {redefines target} – the target of the AssertedEvidence

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. If AssertedEvidence has isCounter=false, then it should be interpreted as "the evidence supports the Assertion". If an AssertedEvidence has isCounter=true, then it should be interpreted as "the evidence defeats the Assertion".

Constraints

- The source of AssertedEvidence relationships must be ArtifactReference.

OCL :

self.source–>forall(s|s.oclIsTypeOf(ArtifactReference))

13.14 AssertedContext

AssertedContext can be used to declare that the Artifact or ArtifactReference provides the context for the interpretation and scoping of an ArgumentAsset (ArtifactReference, Artifact, Claim, ArgumentReasoning, or any AssertedRelationship specialization element).

Superclass

AssertedRelationship

AssociationEnds

+context : ArtifactReference [1..*] {redefines source} – the context provides further clarification or constraint of the contextable

+contextable : ArgumentAsset [1] {redefines target} – the context provides further clarification or constraint of the contextable

Semantics

Contextual information often needs to be cited in order to make clear the interpretation and scope of an ArgumentAsset. 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”). If AssertedContext has isCounter=false, then it should be interpreted as "the contextable must be interpreted in the context". If AssertedContext has isCounter=true, then it should be interpreted as "the contextable must not be interpreted in the context".

13.15 ArgumentConcept (abstract)

ArgumentConcept is an abstraction of all the packaging and elements in the argument domain.

Superclass

Packaging :: Concept

Attributes

isDefeated : Boolean [0..1] = false – specified whether this ArgumentConcept is defeated or not (or with cardinality 0, has not been determined yet).

13.16 Targetable (abstract)

Targetable is an abstraction to support non–assertions as targets of assertable relationships.

Superclass

ArgumentAsset

Part III – Artifact Metamodel

This part of the specification defines the Artifact Metamodel.

This page intentionally left blank.

14 Artifact Classes

14.1 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.

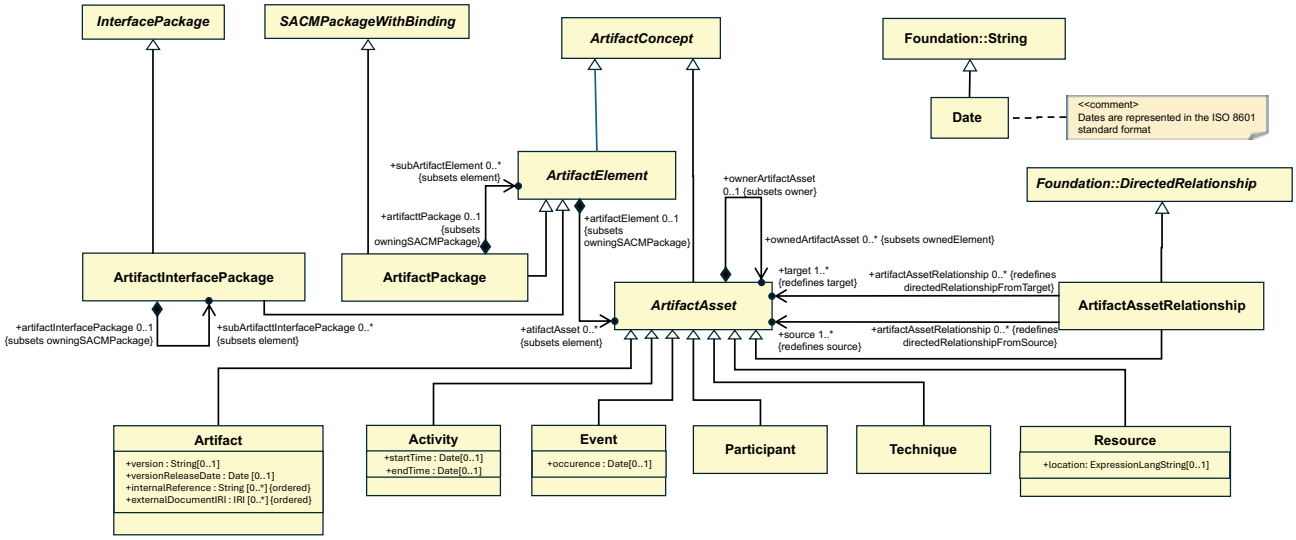

Figure 14.1 – Artifact Package Diagram

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 ‘isSACMAbstract’. 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.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.

14.2 ArtifactPackage

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

Superclass

ArtifactConcept

AssociationEnds

subArtifactElement [0..*] {subsets element} – a collection of ArtifactElements forming an artifact package in a structured assurance case.

14.3 ArtifactInterfacePackage

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

Superclass

ArtifactElement, Package :: InterfacePackage

AssociationEnds

subArtifactInterfacePackage [0..1] {subsets owningSACMPackage} – ArtifactInterfacePackages that are contained in this ArtifactInterfacePackage

Semantics

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

Constraints

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

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

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

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

14.4 ArtifactAsset (abstract)

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

Superclass

ArtifactConcept

AssociationEnds

ownedArtifactAsset : ArtifactAsset [0..*] {subsets ownedElement} – ArtifactAssets owned by this 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.

14.5 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

versionReleaseDate : Date [0..1] – the date on which the artifact was created.

internalReference : String [0..*] {ordered} – refence can be further restricted to some internal part for what the Artifact represents.

externalDocumentIRI : IRI [0..*] {ordered} – IRI to the external document that this Artifact element represents.

IRIs are ordered to allow prefered references appearing earlier in the ordering. All references should be to the same external document.

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 (AssertedEvidence 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.

Constraints

OwnerIsArtifactConcept

Artifacts can only be owned in namespaces that are ArtifactConcepts.

inv: namespace–>oclTypeOf(ArtifactConcept)

14.6 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.

14.7 Resource

Resource corresponds to the tangible objects representing an Artifact.

Superclass

ArtifactAsset

Attributes

location : ExpressionLangString [0..1] – 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.

14.8 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.

14.9 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.

14.10 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.

14.11 ArtifactAssetRelationship

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

Superclass

ArtifactAsset, Foundation::DirectedRelationship

AssociationEnds

source : ArtifactAsset [1..*] {redefines source} – the source of the ArtifactAssetRelationship

target : ArtifactAsset [1..*] {redefines target} – 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.

14.12 Date

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

14.13 ArtifactElement (abstract

ArtifactElement is a common class for the packages in the Artifact domain.

Superclass

ArtifactConcept

AssociationEnds

artifactAsset:Artifact[0..*] {subsets element} - ArtifactAssets contained in this ArtifactElement.

14.14 ArtifactConcept (abstract)

ArtifactConcept is an abstraction of all the packaging and elements in the artifact domain.

Superclass

Packaging::Concept

Part IV – Advanced Argument Classes (informative)

This part of the specification defines the non–normative Advanced Argument Classes.

This page intentionally left blank.

15 Advanced SACM Capabilities with Examples (informative

The following seven SACM Capabilities, 15.1-15.7, are built upon the materials in the previous sections with new items shown as well as additions to the existing items. "[Additions to xxxx]" are used to indicate the previous definitions of the item "xxxx" that the additions are applied to.

15.1SACM Join

Joins can be used in SACM to support both Assurance Case patterns and various logical uses.

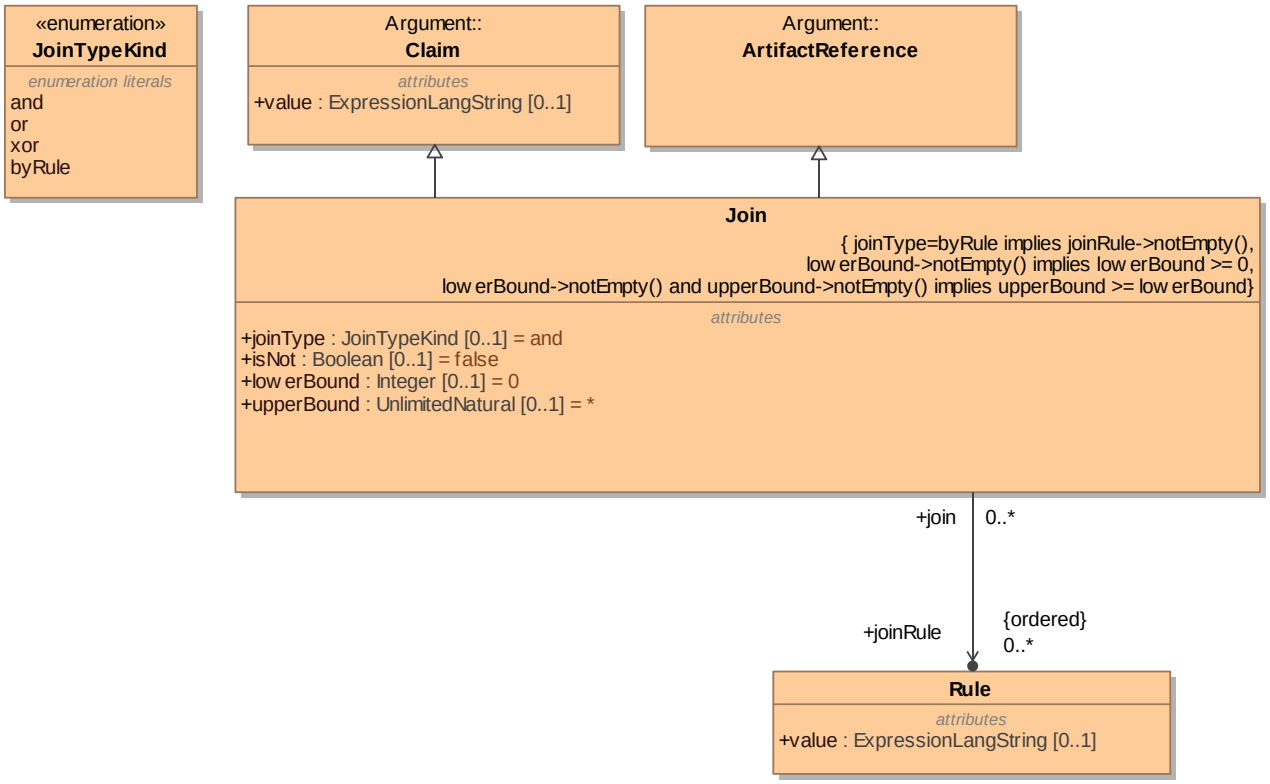

Figure 15.1.1 – SACM Join Diagram

15.1.1 Join

Join is a "logical" connector of multiple sources of relationships (whose targets end on the Join). For Patterns, the Join specifies a set of choices one can make when reifying this pattern. In some logics, this allows the logical collection (e.g. anding) of the sources of a relationship whose targets are this Join. The Join also can be a source of a relationship to some other final target SACMElement. This would be the equivalent of the original source to final target relationship with possible added semantics to the sources of those relationships.

Superclass

Argument::Claim, Argument::ArtifactReference

Attributes

joinType : joinTypeKind [0..1] = and - Defines the JoinTypeKind for this Join

isNot : Boolean [0..1] = false - Is the whole Join a Not (e.g. joinType=and and isNot=true, makes it a NAND operation).

lowerBound : Integer [0..1] = 0 - LowerBound for this Join, this is the least number of sources that can chosen for this Join (Join used for a pattern). This value is 0 or greater. If lowerBound is given and no upperBound is given, upperBound is assumed to be unlimited ("*").

upperBound : UnlimitedNatural [0..1] = * - UpperBound for this Join, this is the greatest number of sources that can be chosen for this Join (Join used for a pattern). If upperBound is given and no lowerBound, then assumed to be a single number specified by the upperBound (i.e. the LowerBound is equal to the UpperBound).

AssociationEnds

joinRule : Rule [0..*] {ordered} - if joinType=byRule, then this defines the Rule that joins the sources.

Constraints

UpperBoundGreaterThanOrEqualToLowerBound

If lowerBound and upperBound exist, then upperBound must be greater than or equal to lowerBound.

inv: lowerBound->notEmpty() and upperBound->notEmpty() implies upperBound >= lowerBound

LowerBoundGreaterThanOrEqualToZero

If lowerBound exists, then lowerBound is greater than or equal to zero.

inv: lowerBound->notEmpty() implies lowerBound >= 0

ByRuleMustHaveRule

If joinType=byRule, then joinRule must have a reference to a Rule.

inv: joinType=byRule implies joinRule->notEmpty()

15.1.2 JoinTypeKind

JoinTypeKind defines the relationships between the sources of the relationships that have targets that end on the Join.

EnumerationLiterals

and - "and" between the sources on the end of relationships

or - "or" between the sources on the end of relationships

xor - "xor" between the sources on the end of relationships

byRule - "byRule" defined relationship between the sources on the end of relationships

combine - is a combination of sources without any logical implications. These are treated as if the relationship that is sourced on the Join is coming directly to the sources.

15.1.3 Examples

The Join in a Template Example shows how a Join can be used in a pattern to define the various options allowed in the template.

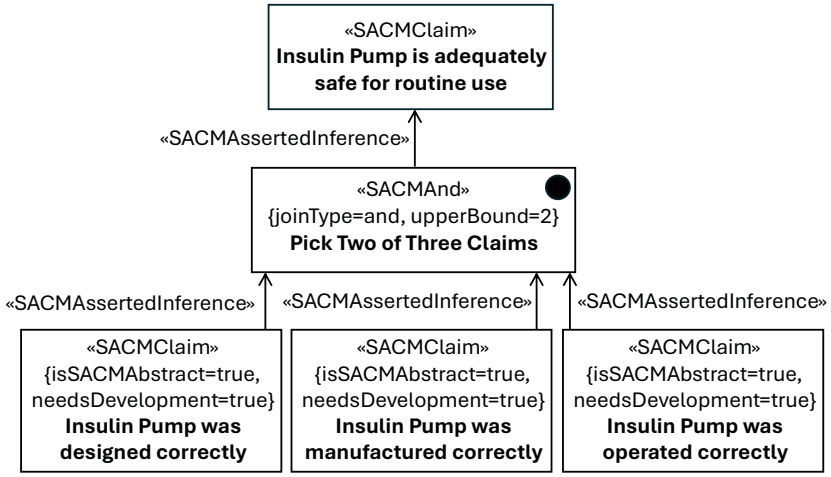

Figure 15.1.2.1 – Join in a Template Example

For a Deductive Argument, Claim B being true or Claim C being True (or Both), is enough to Specify that Claim A is true.

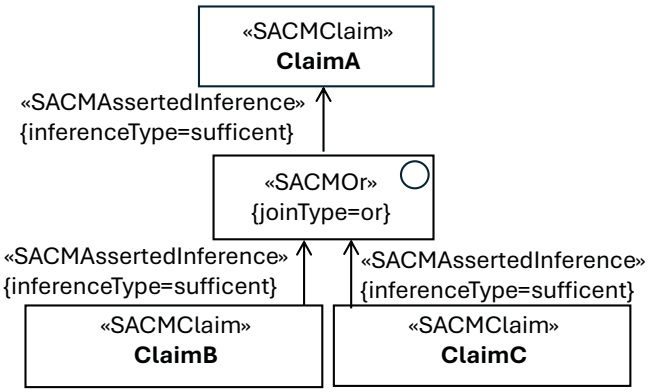

Figure 15.1.2.2 – Join in Deductive Logic Example

Join (in the profile), can be used in a structure like a Strategy (from GSN) or an Argument (from CAE). The Join (by either name or value tagged value) is the statement of the ArgumentReasoning,and required tagged values joinType=combine and a relationship with references to the AssertedRelationships.

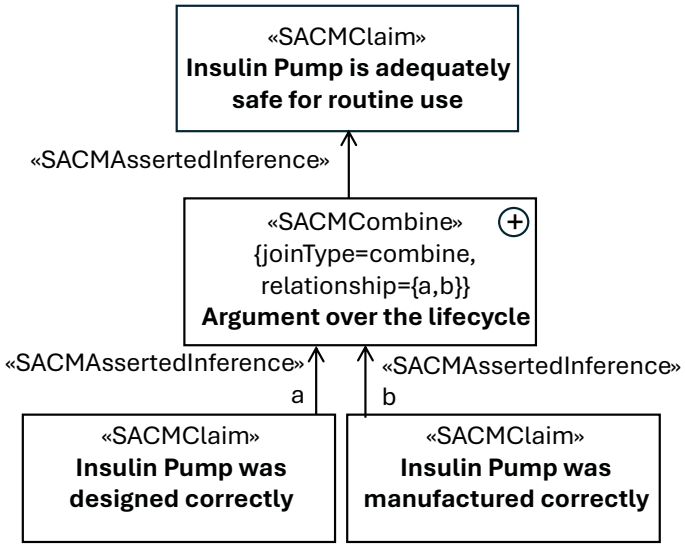

Figure 15.1.2.3 – Join as an ArgumentReasoning (Like a Strategy) Example

15.2 Advanced Arguments

Advanced Arguments adds in additional features to support

- Strengths - to allow the inclusion of strength of belief in an argument
- Additional AssertionDeclarations - an additional AssertionDeclarationKind EnumerationLiterals ("byRule") and an assertionDeclartionRule AssociationEnd to allow the extension of AssertionDeclarations to include user-defined specifications
- Additional InferenceTypes - a new Enumeration InferenceTypeKind that adds in support for inductive (default), deductive, abductive and user-defined specifications (through "byRule" EnumerationLiterals and inferenceRule AssociationEnd) on AssertedInferenceRelation
- Explicit Truth values (through isTrue Attribute on Assertion), or
- (possible Multi-valued) assertionValues and varied logic (e.g. Three-valued logic) using assertionValue AssociationEnd on Assertion

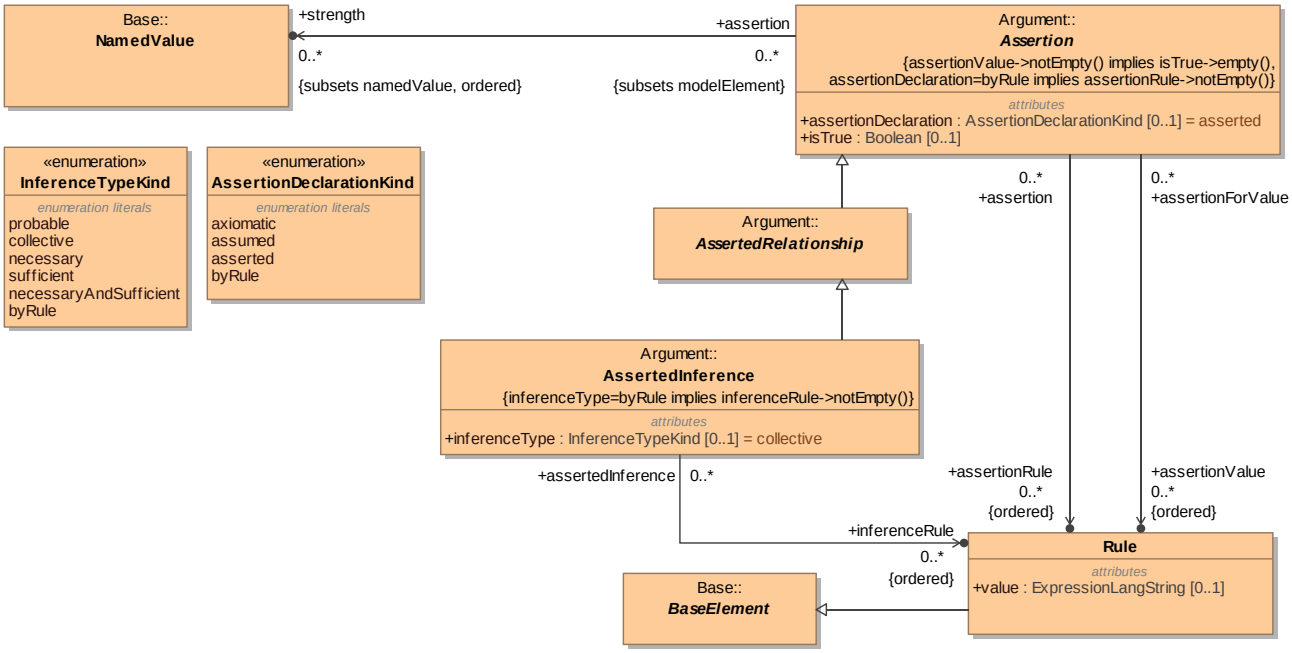

Figure 15.2.1 – Advanced Arguments Diagram

15.2.1 AssertionDeclarationKind [Additions to 13.7]

EnumerationLiterals

- byRule - allows the specification of an extension to AssertionDeclarationKind (e.g. which might allow for Modal logics)

15.2.2 Assertion (abstract) [Additions to 13.9]

Attributes

isTrue : Boolean [0..1] - whether an Assertion has a Boolean value or not. No value (cardinality of 0) can be used to show undecidable things.

AssociationEnds

strength : NamedValue [0..*] {subsets namedValue, ordered} - Strength values for the assertion

assertionDeclarationRule : Rule [0..*] {ordered} - Allows specification of assertionDeclaration “byRule”

assertionValue : Rule [0..*] {ordered} - Allows for specification of assertion value (e.g. Three-valued logic value)

Semantics

SACMDependencies can be defined between strengths, and SACMConstraints can be used to calculate values of strengths based on other strengths.

Constraints

ByRuleDeclarationRequiresRule

If assertionDeclaration is byRule then assertionRule must reference at least one Rule

inv: assertionDeclaration=byRule implies assertionRule->notEmpty()

AssertionValueImpliesNoIsTrue

If there is an assertionValue then isTrue must be empty

inv: assertionValue->notEmpty() implies isTrue->empty()

15.2.3 Rule

Rules allow extension of Argumentation for other kinds of logics and reasoning.

Superclass

Base::BaseElement

Attributes

value : ExpressionLangString [0..1] - Value for the rule. The name can be the content or one can have a name and then the value contains the content.

Semantics

When multiple rules are referenced, lower index rules take precedence over higher index rules.

15.2.4 InferenceTypeKind

InferenceTypeKind enumerates inference types which control the semantics of the AssertedInferenceRelationship.

EnumerationLiterals

- probable - (P) infers that the source make the target more probable (usually shown with strengths)
- collective - (C) [default] infers that all sources are Necessary for the target, taken together all sources together are Sufficient for the target (meant for inductive arguments)
- necessary - (N) infers that the source is Necessary for the target

- sufficient - (S) infers that the source is sufficient for the target
- necessaryAndSufficient - (NS) infers that the source is necessary and sufficient for the target
- byRule - (R) infers that the source has some inference to the target as specified by the Rule in the inferenceRule

Note: these can be counter-inferences by setting isCounter=true.

15.2.5 AssertedInference [Additions to 13.13]

Attributes

inferenceType : InferenceTypeKind [0..1] = collective - specifies the inference type for this inference

AssociationEnds

inferenceRule : Rule [0..*] {ordered} – if inferenceType=byRule, this specifies the ordered set of Rules for the inference type

Constraints

ByRuleMustHaveRule

If inferenceType is byRule then inferenceRule must reference at least one Rule.

inv: inferenceType=byRule implies inferenceRule->notEmpty()

15.2.6 Examples

SACM provides the ability to argue using Deductive, Inductive, Abductive, and user-defined byRule inferencing. The various logics inferencing can be controlled by setting inferenceType using the InferenceTypeKind Enumeration.

In the Abduction Example, there is an argument that Bill is more likely the Killer (raises the probability strength to .6) because Bill has a Yellow Coat. This inference is not defeated because The Killer wore a yellow coat. If the Killer did not wear a Yellow Coat (i.e. that claim is defeated) then the AssertedInference aaa (default inferenceType=collective, i.e. arguing defeation rather than truth value) would defeat the probable inference.

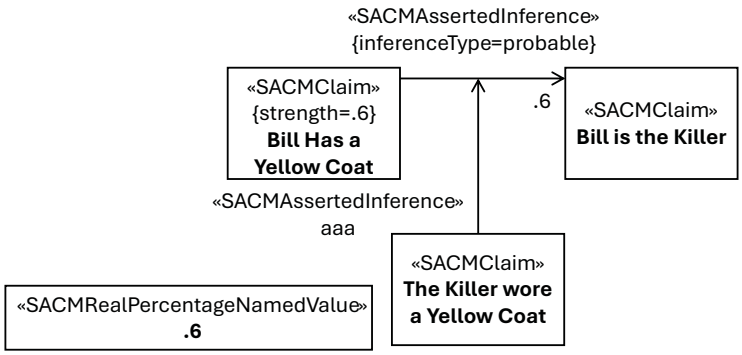

Figure 15.2.6.1 – Abduction Example

The byRule (user-defined) example, ClaimA using TriLogicInference infers ClaimB (which is asserted to be Possible) and has value of MidValue. This facility in SACM allows for user-defined (e.g. Modal) logic to be defined within the SACM model.

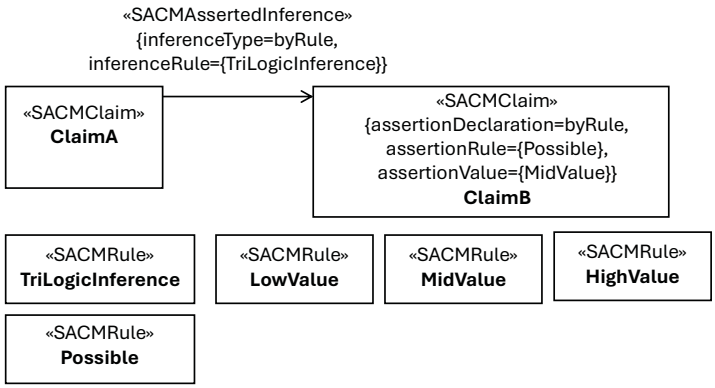

Figure 15.2.6.2 – byRule Example

15.3 SACM View

SACMView allows for the display of Assurance Cases in some readable forms for people. These forms usually take the form of Graphical or Tabular form. SACMView defines the structure of those readable forms so that they can be serialized with the model (e.g. in XMI).

15.3.1 OMG Diagram Definition and Interchange

DI (Data Interchange) is a specification out of the DD (Diagram Definition) standard [https://www.omg.org/spec/DD] which specifies how to interchange Diagrams in general for languages especially ones that are built from MOF. The SACMDiagram exchange is based on this standard whose diagram is reproduced below [from fig. 9.2 of the DD standard], Diagram Definition Specification Version 1.1.

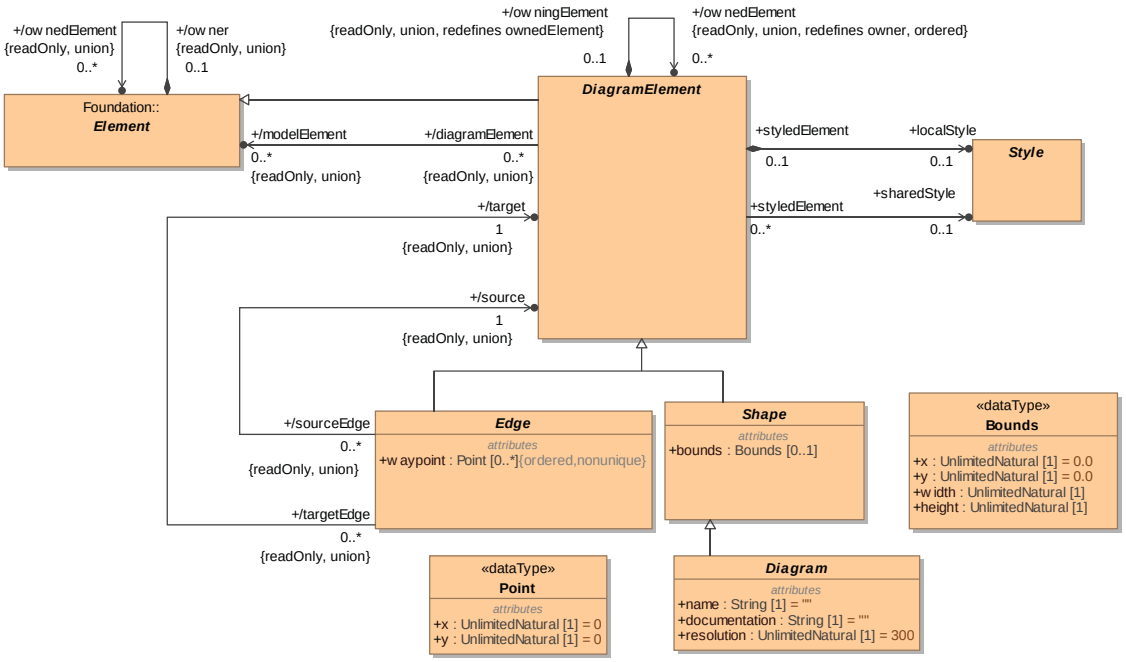

Figure 15.3.1 – OMG's DI Standard Diagram

15.3.2 SACMView Model

SACMView allows for the specification of multiple types of Views that can be used to represent an AssuranceCase. SACMView itself is an abstract metaclass which can be extended to provide different modeling artifacts (e.g. Diagrams, Tables). SACMDiagram (also abstract) allows for the representation of a Diagram with SACMDiagramElements. There are specific concrete specializations of SACMDiagram which allow for various parts of Structured Assurance Case Diagrams to be separated on different diagrams.

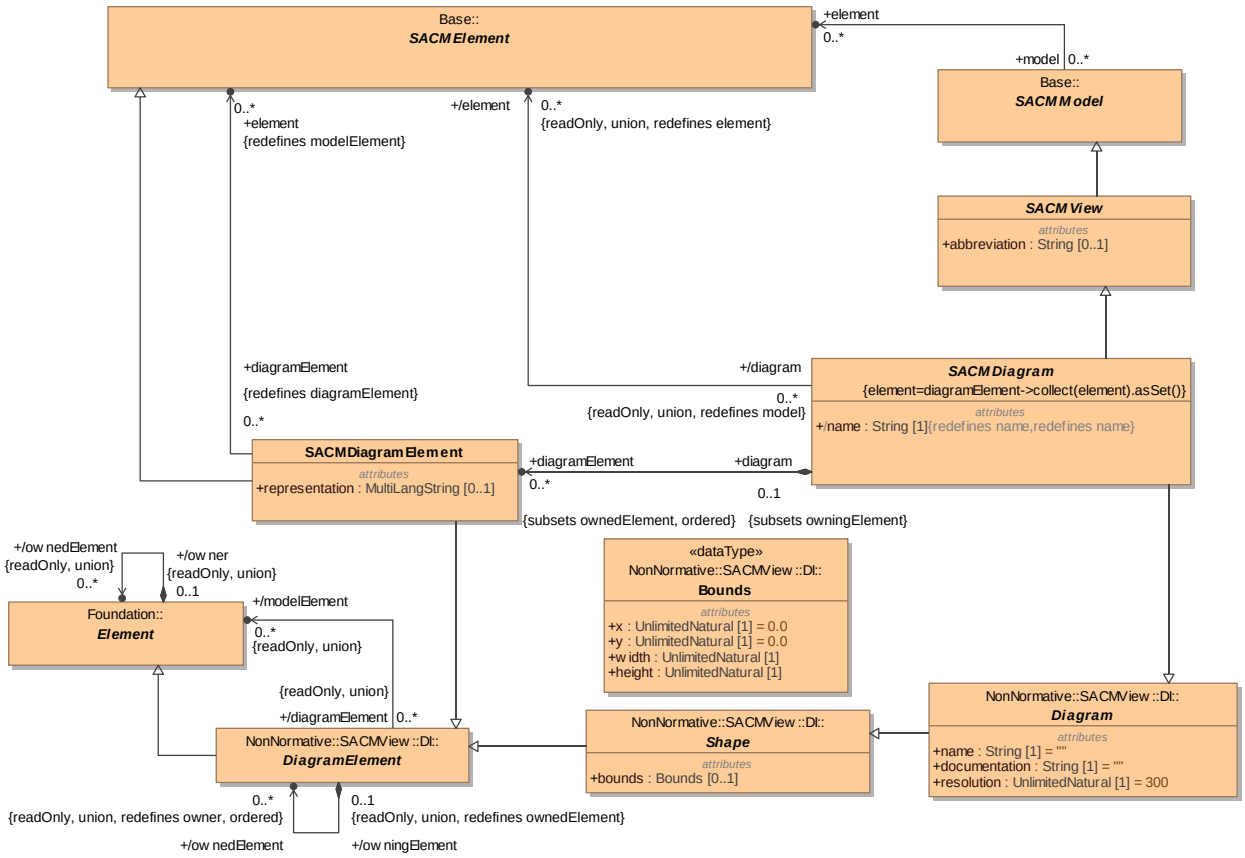

Figure 15.3.2 – SACMDiagram

15.3.3 SACMView (abstract)

SACMView is an abstract metaclass that can be specialized for various view of an Assurance Case (e.g. Diagrams, Catalogs, Matrices, Tables, etc.).

Superclass

Base:SACMModel

Attributes

abbreviation : String [0..1] - abbreviation that can be used in the header of a view to describe this view

15.3.4 SACMDiagram (abstract)

SACMDiagram is the abstract metaclass for all diagram types in SACM.

Superclass

SACMView

Attributes

/name : String [1] {redefines name, redefines name} - name of the diagram.

AssociationEnds

/element : SACMElement [0..*] {readonly, union, redefines element} - derived SACMElements in a diagram through SACMDiagramElement.

diagramElement : SACMDiagramElement [0..*] {subsets ownedElement, ordered} - SACMDiagramElements in this SACMDiagram.

/baseElement : BaseElement [0..*] {subsets element} - derived SACMElements that are BaseElements.

Constraints

deriveElement

element collection is all the SACMElements that are represented by all of the diagramElements.

inv: element=diagramElement->collect(element).asSet()

15.3.5 SACMDiagramElement

SACMDiagramElements are the visible DiagramElements within a SACMDiagram.

Superclass

DiagramElement

Attributes

representation : MultiLangString [0..1] - representation (e.g. SVG) of the DiagramElement within the SACMDiagram. Language of representation should be specified in the MultiLangString.

AssociatonEnds

element : SACMElement [0..*] {redefines modelElement} - SACMElements that this SACMDiagramElement represents in the SACMDiagram.

15.3.6 SACMDiagram Types

Four separate specific diagrams are defined that allows a general SACM diagram (StructuredAssuranceCaseDiagram) as well as three specific domain diagrams (StructuredAssuranceTerminologyDiagram, StructuredAssuranceArtifactDiagram, StructuredAssuranceArgumentDiagram). Each of these diagrams can have BaseElements on them, as well as specific

concepts from each of the three domains in their respective diagrams.

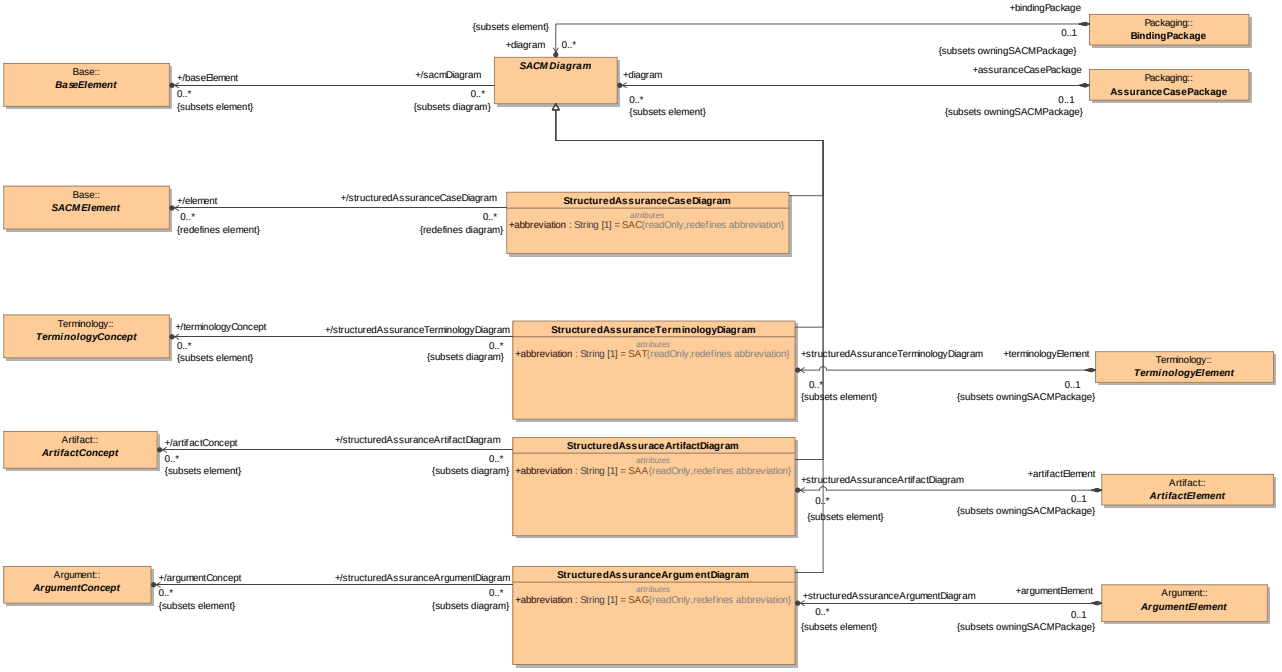

Figure 15.3.4 – SACMDiagramTypes

15.3.7 StructuredAssuranceCaseDiagram

StructuredAssuranceCaseDiagram allows any SACMElement to be represented on them.

Superclass

SACMDiagram

Attributes

abbreviation : String [1] = SAC {readOnly, redefines abbreviation}

AssociationEnds

/element : SACMElement [0..*] {redefines element} - Any SACMElement can be represented by SACMDiagramElements on a StructuredAssuranceCaseDiagram

15.3.8 StructuredAssuranceTerminologyDiagram

StructuredAssuranceTerminologyDiagram allows any BaseElement or TerminologyConcept to be represented on them.

Superclass

SACMDiagram

Attributes

abbreviation : String [1] = SAT {readOnly, redefines abbreviation}

AssociationEnds

/terminologyConcept : TerminologyConcept [0..*] {subsets element} - only TerminologyConcepts (along with BaseElements) should be represented by SACMDiagramElements on a StructuredAssuranceTerminologyDiagram

15.3.9 StructuredAssuranceArtifactDiagram

StructuredAssuranceArtifactDiagram allows any BaseElement or ArtifactConcept to be represented on them.

Superclass

SACMDiagram

Attributes

abbreviation : String [1] = SAA {readOnly, redefines abbreviation}

AssociationEnds

/artifactConcept : ArtifactConcept [0..*] {subsets element} - only ArtifactConcepts (along with BaseElements) should be represented by SACMDiagramElements on a StructuredAssuranceArtifactDiagram

15.3.10 StructuredAssuranceArgumentDiagram

StructuredAssuranceArgumentDiagram allows any BaseElement or ArgumentConcept to be represented on them.

Superclass

SACMDiagram

Attributes

abbreviation : String [1] = SAG {readOnly, redefines abbreviation}

AssociationEnds

/argumentConcept : ArgumentConcept [0..*] {subsets element} - only ArgumentConcepts (along with BaseElements) should be represented by SACMDiagramElements on a StructuredAssuranceArgumentDiagram

15.3.11 AssuranceCasePackage [Additions to 11.2]

AssociationEnds

diagram: SACMDiagram [0..*] {subsets element} - any SACMDiagram can be contained within an AssuranceCasePackage

15.3.12 BindingPackage [Additions to 10.3]

AssociationEnds

diagram: SACMDiagram [0..*] {subsets element} - any SACMDiagram can be contained within a BindingPackage

15.3.13 TerminologyElement [Additions to 12.2]

AssociationEnds

structuredAssuranceTerminologyDiagram : StructuredAssuranceTerminologyDiagram [0..*] {subsets element} - Only StructuredAssuranceTerminologyDiagrams can be contained in a TerminologyElement.

ArtifactElement [Additions to 9.7]

AssociationEnds

structuredAssuranceArtifactDiagram : StructuredAssuranceArtifactDiagram [0..*] {subsets element} - Only StructuredAssuranceArtifactDiagrams can be contained in a ArtifactElement.

15.3.14 ArgumentElement [Additions to 13.2]

AssociationEnds

structuredAssuranceArgumentDiagram : StructuredAssuranceArgumentDiagram [0..*] {subsets element} - Only StructuredAssuranceTArgumentDiagrams can be contained in a ArgumentElement.

15.4Assessment

Assurance Cases, in general, can be assessed by multiple subject matter experts (e.g. an Assessor) who may have differing ideas about the correctness of the argumentation or support of that argumentation with evidence. The first "assessment" made on the Assurance Cases comes from the author of the Assurance Case who puts their best effort forward in their description of the model. But, this "assessment" may be further augmented by another Assessor. This section describes additional parts to Assurance Cases to support multiple Assessments and changes to the model to support those assessments. Also, Assurance Cases may go through many changes from it beginning to the point of publication of the "final" model, and a record of those changes may want to be kept (e.g. so wrong avenues of thinking that have been explored and found to be unwanted can be documented in a tool). These changes can be kept through this mechanism as well.

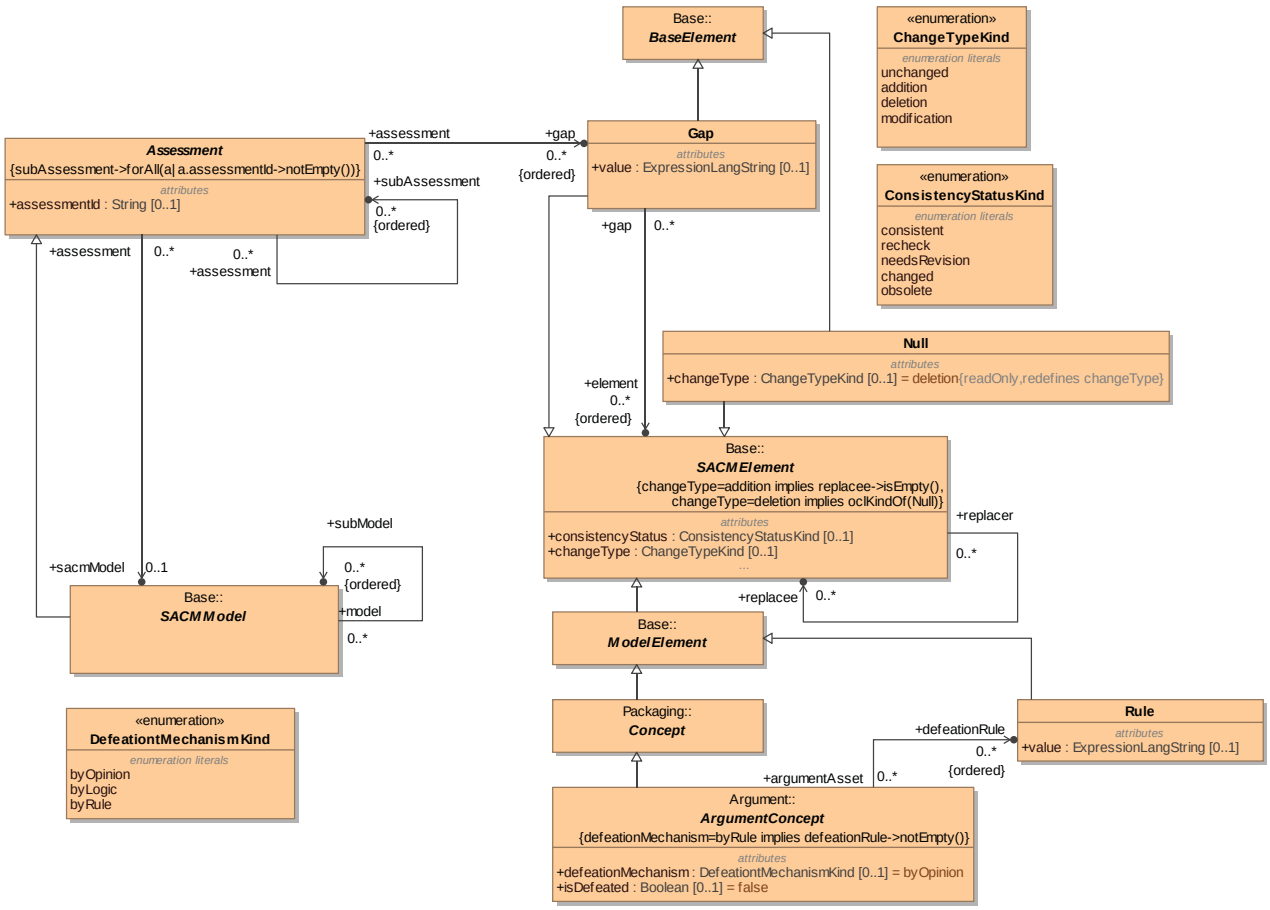

Figure 15.4.1 – SACM Assessments

15.4.1 Assessment (abstract)

Assessment is an additional metaclass generalization for SACMModel to allow a SACMModel to be defined as an Assessment.

Attributes

assessmentId : String [0..1] - an assessmentId that could be used as a unique key in a database that defines the nature of this assessment (e.g. who the Assessor was). Any assessmentId (not necessarily unique) means that the concrete SACMModel is considered an Assessment. No assessmentId means that this SACMModel is not considered an Assessment.

AssociationEnds

subAssessment : Assessment [0..*] {ordered} - A sub assessment (e.g. a previous assessment) of this Assessment. Direct judgements on an Assessment override any judgement in a subAssessment. Judgements in earlier assessments are overridden by judgements in later Assessments in the order list of subAssessments.

Constraints

SubAssessmentAreAssessments

subAssessment must be Assessments (i.e. assessmentId have a value).

inv: subAssessment->forAll(a| a.assessmentId->notEmpty())

15.4.2 Gap

A Gap can be specified by an Assessor as something missing from the Assurance Case. Gaps can be further detailed by specifying specific replacer SACMElements.

Superclass

Base::BaseElement

Attributes

value : ExpressionLangString [0..1] - statement of the Gap. The name can be the content or one can have a name and then the value contains the content.

AssociationEnds

element : SACMElement [0..*] {ordered} - Replacer SACMElements that are related to this Gap.

15.4.3 Null

Null is a replacer which allows an Assessor to remove any replacee SACMElement in an Assurance Case in their Assessment.

Superclass

Base::BaseElement, Base::SACMElement

Attributes

changeType : ChangeTypeKind [0..1] = deletion {readOnly, redefines changeType}

15.4.4 DefeationMechanismKind

DefeationMechanismKind defines how a defeation (or no defeation) has been decided.

EnumerationLiterals

- byOpinion - state of the defeation is by the opinion of the Assessor
- byLogic - state of the defeation is by the understood logic of the inferences
- byRule - state of the defeation is by the Rules specified

15.4.5 ChangeTypeKind

ChangeTypeKind defines how a replacer SACMElement changes in this assessment from its replacee.

EnumerationLiterals

- unchanged - The replacer SACMElement does not change from the replacee.
- addition - The replacer SACMElement is an addition (there will be no replacee).
- deletion - The replacer Null is a deletion of the replacee.
- modification - The replacer SACMElement modifies the replacee.

15.4.6 ConsistencyStatusKind

ConsistencyStatusKind allows marking of replacer SACMElements during the process and at the end of the process of an Assessment.

EnumerationLiterals

- consistent - the replacer SACMElement is consistent with what is needed to fix the Assurance Case.
- recheck - the replacer SACMElement is considered consistent now, but want to recheck it later.
- needsRevision - the replacer SACMElement needs revision, but it has not been changed as yet.
- changed - the replacer SACMElement has been changed to fix the Assurance Case.
- obsolete - The replacer SACMElement is no longer needed to fix the Assurance Case.

Note: ChangeTypeKind and ConsistencyStatusKind are based on Figure 6.3 from the whitepaper: Checkable Safety Arguments – A Modeling Framework Supporting the Maintenance of Safety Arguments Consistent with System Development Artifacts, Carmen Cârlan, 2024.

15.4.7 SACMElement [Additions to 9.2]

Attributes

consistencyStatus : ConsistencyStatusKind [0..1] - Defines the consistency for the SACMElement on this particular Assessment.

changeType : ChangeTypeKind [0..1] - Defines what the change type is for this SACMElement is in the Assessment.

AssociationEnds

replacee : SACMElement [0..*] - Specifies the SACMElement which this SACMElement replaces in this Assessment.

Constraints

DeletionsAreNulls

If changeType=deletion then the replacer must be of type Null (or one of its specializations).

inv: changeType=deletion implies oclKindOf(Null)

AdditionHasNoReplacee

If changeType=addition then the replacee cardinality is 0.

inv: changeType=addition implies replacee->isEmpty()

15.4.8 ArgumentConcept [Additions to 13.18]

Attributes

defeationMechanism : DefeationMechanismKind [0..1] = byOpinion - Defines the defeation mechanism used on this ArgumentConcept.

AssociationEnds

defeationRule : Rule [0..*] {ordered} - if defeationMechanism=byRule, then these are the (possible multiple) Rule(s) that define the reason why defeation (or no defeation) happened.

Constraints

ByRuleImpliesDefeationRule

If defeationMechanism=byRule then defeationRule must have at least one reference.

inv: defeationMechanism=byRule implies defeationRule->notEmpty()

15.4.9 Examples

Assessment is not normally modeled in graphical form, but does need to be represented in the model. In this Assessment Example, Assessment1 with Gap1, replaced AssertedInference p with Null and ClaimB with ClaimC. In Addition, Null was marked as consistent, but ClaimC is marked to be rechecked.

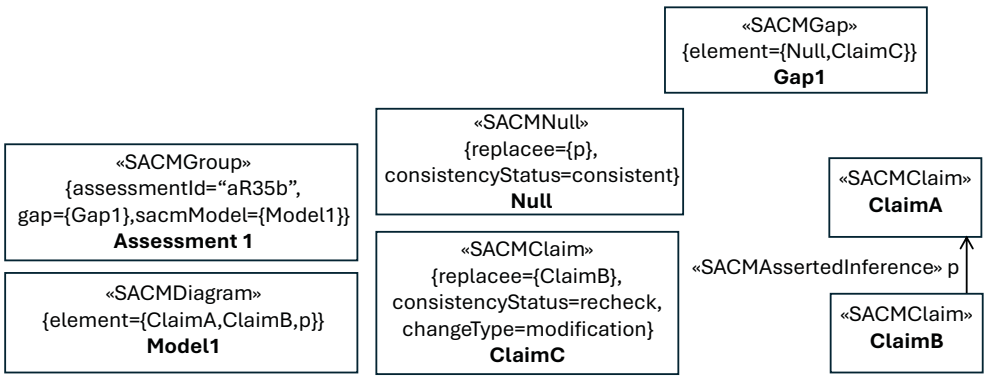

Figure 15.4.9 – Assessment Example

15.5Date Time Duration

To support ISO 15026, SACM allows SACMElements in an AssuranceCase to provide a validity time.

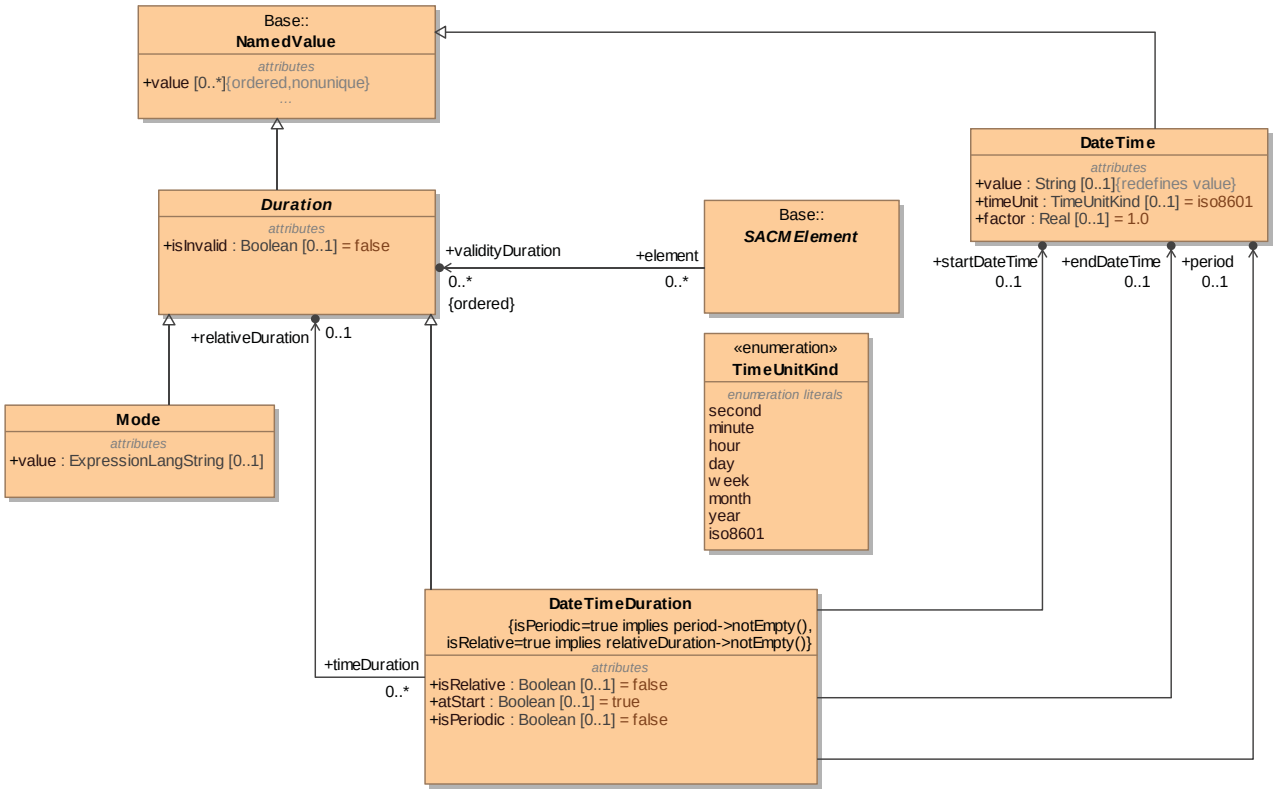

Figure 15.5.1 – Data Time Duration Diagram

15.5.1 SACMElement (abstract) [Additions to 9.2]

AssociationEnds

validityDuration : Duration [0..*] {ordered} - Specifies the time during which the SACMElement is valid (or invalid). If no validityDuration is specified then assume that the duration is somehow explicitly understood or it is valid over all time. Durations of lower index take precedence over higher index Durations. An invalid Duration, in order to invalidate a portion of a valid Duration must be of lower ordering.

15.5.2 DateTime

Specifies a particular DateTime.

Superclass

Base::NamedValue

Attributes

value : String [0..1] {redefines value} - value for the DateTime.

timeUnit: TimeUnitKind [0..1] = iso8601

factor : Real [0..1] = 1.0

15.5.3 Duration (abstract)

Duration specifies the valid (or invalid) times for the SACMElement.

Superclass

Base::NamedValue

Attributes

isInvalid : Boolean [0..1] = false - Can Specify when something is not a valid SACMDuration.

15.5.4 Mode

Used to define a Duration when something is in a particular Mode. Modes may be Terms in a terminology (e.g. DevelopmentPhaseTime) or computed by a Constraint (e.g. when a=6).

Superclass

Duration

Attributes

value : ExpressionLangString [0..1] - Specification of the duration. The name can be the content or one can have a name and then the value contains the content.

15.5.5 TimeUnitKind

Specifies the units that time is defined.

EnumerationLiterals

- second
- minute
- hour
- day
- week
- month
- year
- iso8601 - [default] a string based on ISO 8601

15.5.6 DateTimeDuration

DateTimeDuration

Duration based on DateTime. Allows Absolute as well as Relative durations, and periodic durations.

Superclasss

Duration

Attributes

isRelative : Boolean [0..1] = false - is this duration relative or not

atStart : Boolean [0..1] = true - atStart is only valid when isRelative=true, if atStart=true then the relative time starts at the beginning of another Duration, otherwise it starts at the end of the other Duration

isPeriodic : Boolean [0..1] = false - is this duration period or not

AssociationEnds

startDateTime : DateTime [0..1] - start of duration

endDateTime : DateTime [0..1] - end of duration

period : DateTime [0..1] - period of duration (how often it repeats). If periodic, startDateTime and endDateTime specify its start and end time for a single period.

Semantics

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

Constraints

PeriodicImpliesPeriod

If isPeriod=true then period must reference a DateTime.

isPeriodic=true implies period->notEmpty()

RelativeImpliesRelativeDuration

if isRelative=true then relativeDurantion must reference a Duration.

isRelative=true implies relativeDuration->notEmpty()

15.5.7 AdvancedArtifact

Certain ArtifactAssets have Time and Duration attributes to define them. If one has TimeDate and TimeDateDuration one has a more power mechanism to define Time and Duration in model elements. This section describes the additional AssociationEnds to achieve that end.

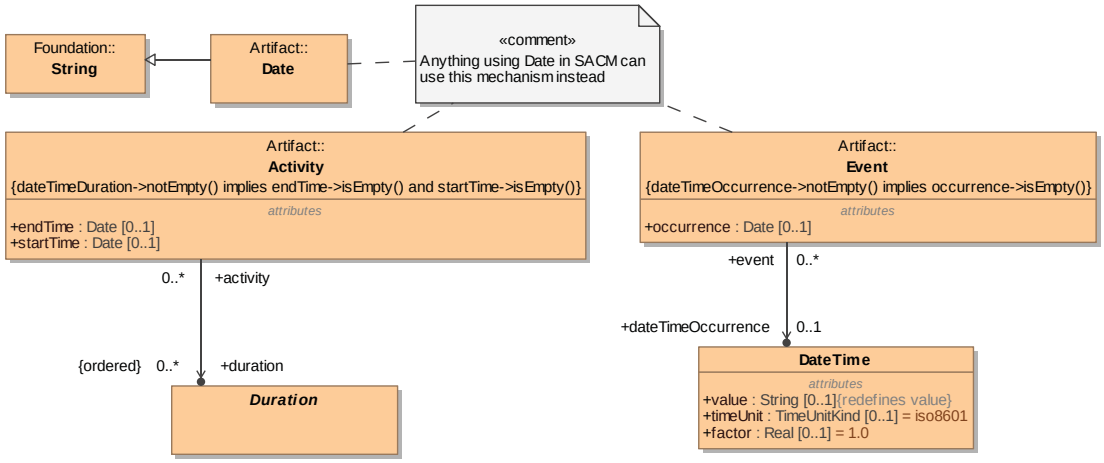

Figure 15.5.2 – Advanced Artifact Diagram

15.5.8 Activity [Addition to 14.9]

AssociationEnd

duration : Duration [0..*] {ordered} - duration of the Activity.

15.5.9 Event [Addition to 14.7]

AssociationEnd

dateTimeOccurrence : DateTime [0..1] - DateTime for the occurrence of the Event.

15.5.10 Examples

ISO 15026 allows Argument elements to have time durations associated with them. In the DateTimeDuration Example, Bill having a Yellow Coat during his employment should overlap with the DateTimeDuration "Yellow Coat Observed At Crime", in order for the probable inference to succeed.

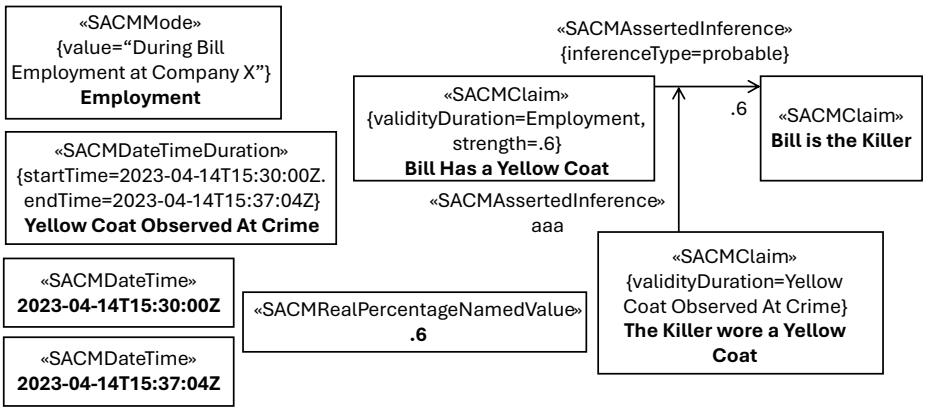

Figure 15.5.8 – DateTimeDuration Examples

15.6 NamedValue Type

NamedValue is extended to allow for defaultValues, Literals (to make Enumerations), Typing (using isSACMAbstract=true) and subclasses that support different primitive types (Strings, Booleans, Integers, and Reals, including Percentage Reals).

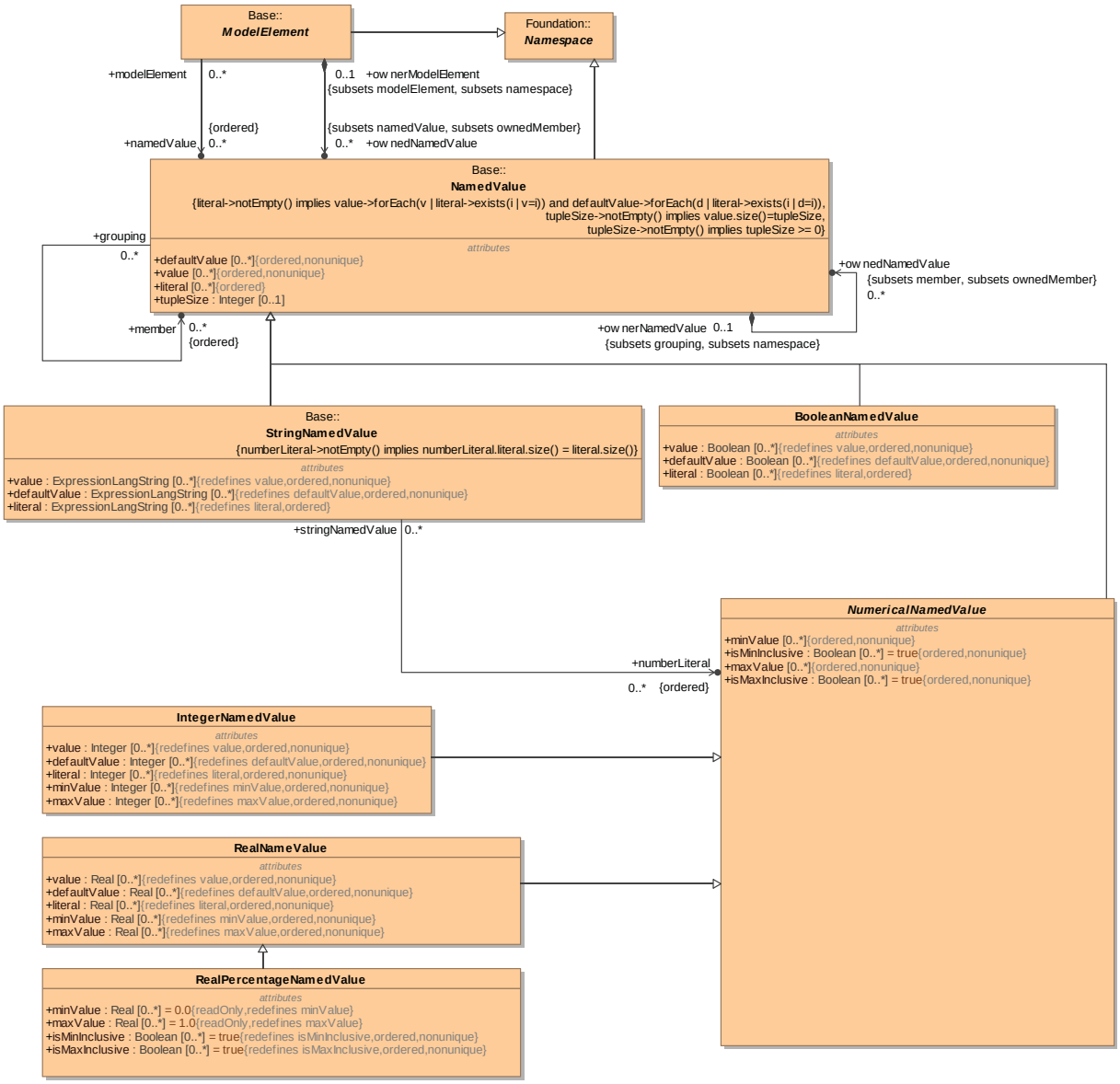

Figure 15.6.1 – NamedValue Type Diagram

15.6.1 NamedValue [Additions to xxx]

Attributes

defaultValue [0..*] {ordered, nonunique} - defaultValue for a value if not set. Can be a tuple of defaultValues. Need not be a literal value if it exists.

literal [0..*] {ordered} - set of literals (enumerated values) for values. If it exists, then values must have one of the literal values.

tupleSize : Integer [0..1] - constraint for the size of the value tuple. If cardinality is 0, then there is no constraint.

Semantics

- value, defaultValue, and literal are all untyped which means they can support any type.
- The name can be the content or one can have a name and then the value contains the content.
- Specializations of NamedValue support other primitive types
- If SACM is embedded in a language with Attributes that is available to be used, this can be used as well
- If defaultValue.size() = 1 then that defaultValue applies to all values
- isSACMAbstract=true can be used to define a type
- A SACMConstraint can be used to define the calculated value of the depender based on the dependees (which can be named through a SACMDependency

Constraints

IfLiteralValuesMustBeOne

If literal exists, then value and defaultValue must be one of these literals.

inv: literal->notEmpty() implies value->forEach(v | literal->exists(i | v=i)) and defaultValue->forEach(d | literal->exists(i | d=i))

TupleSizeConstraint

If tupleSize exists, then value's size must be equal to tupleSize.

inv: tupleSize->notEmpty() implies value.size()=tupleSize

TupleSizeGreaterEqual0

If tupleSize exists, then tupleSize must be greater than or equal to zero.

tupleSize->notEmpty() implies tupleSize >= 0

15.6.2 StringNamedValue

StringNamedValue is a specialization of NamedValue whose value, defaultValue and literal are typed by String.

Superclass

NamedValue

Attributes

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

defaultValue : String [0..*] {redefines defaultValue, ordered, nonunique}

literal : String [0..*] {redefines literal, ordered}

AssociationEnds

numberLiteral : NumericalNamedValue [0..*] {ordered} - this is the corresponding number associated with the String literal.

Constraints

NumberLiteralSize

If the numberLiteral exists, then its size is the same as the String's literal size.

inv: numberLiteral->notEmpty() implies numberLiteral.literal.size() = literal.size()

15.6.3 BooleanNamedValue

BooleanNamedValue is a specialization of NamedValue whose value, defaultValue and literal are typed by Boolean.

Superclass

NamedValue

Attributes

value : Boolean [0..*] {redefines value, ordered, nonunique}

defaultValue : Boolean [0..*] {redefines defaultValue, ordered, nonunique}

literal : Boolean [0..*] {redefines literal, ordered}

15.6.4 NumericalNamedValue (abstract)

NumericalNamedValue is an abstract specialization of NamedValue for Integers and Reals.

Superclass

NamedValue

Attributes

minValue [0..*] {ordered, nonunique} - defines the minimum bound of the value.

isMinInclusive: Boolean [0..*] {ordered, nonunique} - defines whether the value can be set to the minValue specifically.

maxValue [0..*] {ordered, nonunique} - defines the maximum bound of the value.

isMaxInclusive: Boolean [0..*] {ordered, nonunique} - defines whether the value can be set to the maxValue specifically.

Semantics

- If minValue, maxValue, isMinInclusive, isMaxInclusive have only one value then that value is used for all
- If minValue, maxValue, isMinInclusive, isMaxInclusive are not specified then there is no constraint

15.6.5 IntegerNamedValue

IntegerNamedValue is a specialization of NamedValue whose value, defaultValue and literal are typed by Integers.

Superclass

NumericalNamedValue

Attributes

value : Integer [0..*] {redefines value, ordered, nonunique}

defaultValue : Integer [0..*] {redefines defaultValue, ordered, nonunique}

literal : Integer [0..*] {redefines literal, ordered}

minValue : Integer [0..*] {redefines minValue, ordered, nonunique}

maxValue : Integer [0..*] {redefines maxValue, ordered, nonunique}

15.6.6 RealNamedValue

RealNamedValue is a specialization of NamedValue whose value, defaultValue and literal are typed by Reals.

Superclass

NumericalNamedValue

Attributes

value : Real [0..*] {redefines value, ordered, nonunique}

defaultValue : Real [0..*] {redefines defaultValue, ordered, nonunique}

literal : Real [0..*] {redefines literal, ordered}

minValue : Real [0..*] {redefines minValue, ordered, nonunique}

maxValue : Real [0..*] {redefines maxValue, ordered, nonunique}

15.6.7 RealPercentageNamedValue

RealPercentageNamedValue is a specialization of NamedValue whose value, defaultValue and literal are typed by Reals and are values from 0.0 to 1.0.

Superclass

RealNamedValue

Attributes

minValue : Real [0..*] = 0.0 {redefines minValue, ordered, nonunique}

isMinInclusive: Boolean [0..*] = true {ordered, nonunique}

maxValue : Real [0..*] = 1.0 {redefines maxValue, ordered, nonunique}

isMaxInclusive: Boolean [0..*] = true {ordered, nonunique}

15.6.8 NamedValue Type Examples

NamedValue Type Example shows a Risk type which has members of Effect and Frequency (which are owned by Risk). A specific Risk instance is High Risk which has a Critical Effect and a Frequent Frequency (which are owned by High Risk).

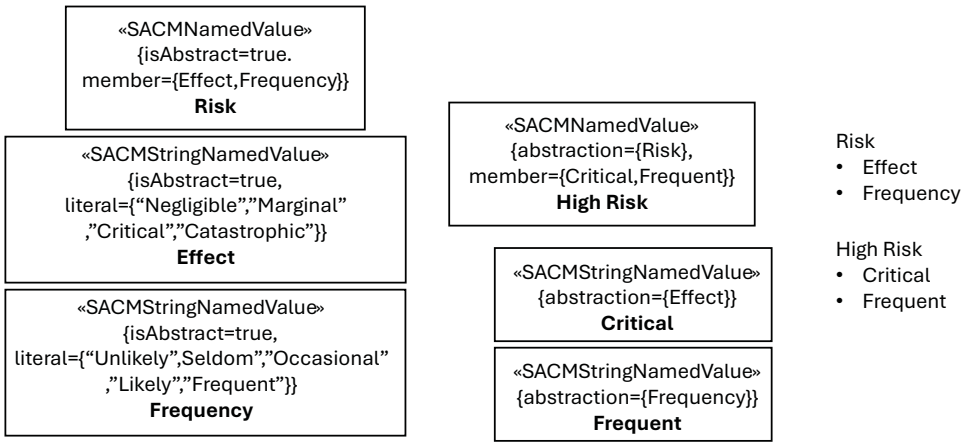

Figure 15.7.2 NamedValue Type Example

15.7 Argument Context

Context comes from a best practice whitepaper “Towards a Clearer Understanding of Context and Its Role in Assurance Argument Confidence” by Patrick John Graydon in 2014 specifying how one should define Context in an Assurance Case. This is an element representing that Context idea.

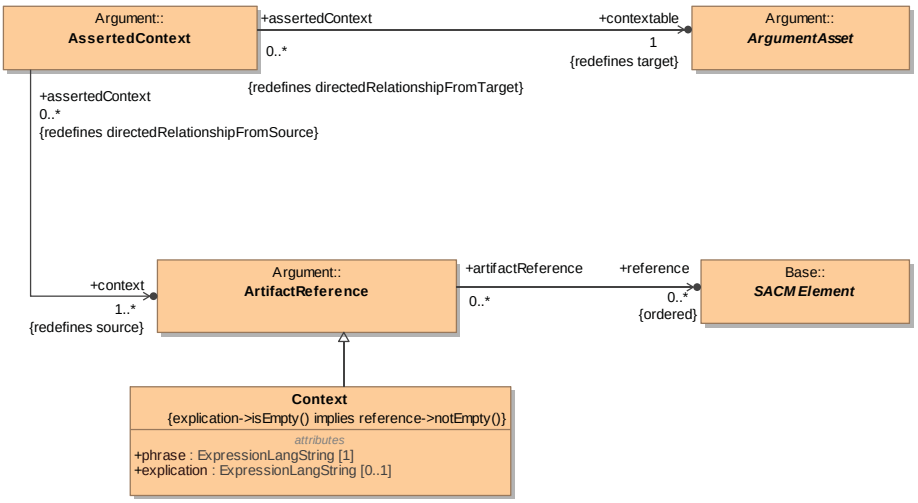

Figure 15.7.1 – Argument Context Diagram

15.7.1 Context

Context is "PHRASE is interpreted as EXPLICATION" or if EXPLICATION is empty, then "PHRASE is interpreted in the context of the referenced SACMElement (perhaps an Artifact)"

Superclass

Argument::ArtifactReference

Attributes

phrase : ExpressionLangString [1] - the phrase targeted by this Context

explication : ExpressionLangString[0..1] - the explication to interpret the phrase as

The explication applies to
(i) the element e at which the context c is asserted,
(ii) any ArgumentConcept in the same argument module that directly or indirectly supports e through AssertedRelationships, and

(iii) any justification, assertion, or confidence element in the same module asserted as context to an element to which c applies as per rules (i) and (ii).

Semantics

- Non-circularity: Context may not have any phrase that is directly or indirectly explicated in terms of itself
- Uniqueness: May not assert two explications for the same phrase that apply to the same element

- Loaded language: Arguers should not use phrase-context to phrase arguments in terms whose plain-language meaning might cause misunderstanding of the argument
- UI Note: Explicated phrases should be visually distinguished from non-explicated text and Context has the graphical form "phrase:explication"

Constraints

ExplicationIsEmpty

If Explication is empty then there must be a reference to a SACMElement.

inv: explication->isEmpty() implies reference->notEmpty()

15.7.2 Argument Context Example

Context1 provides an explication for Claim1's use of the phrase "routine" by its "use in compliance with Insulin Pump User Guide" and provides a reference to the Artifact in the Assurance Case defining that Insulin Pump's User Guide.

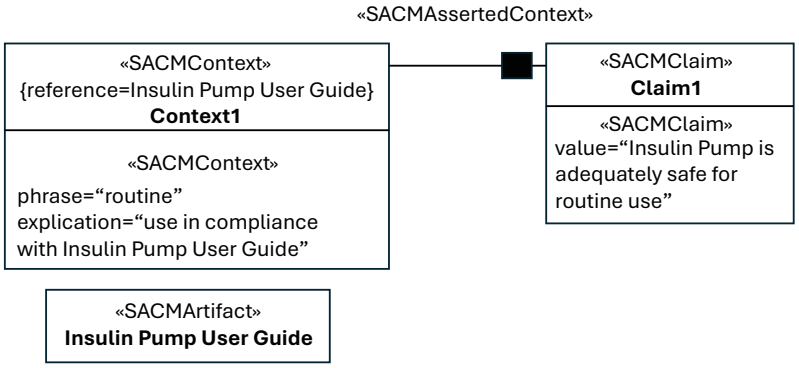

Figure 15.7.2 Argument Context Example

Annex A: Mappings from existing industrial notations for assurance cases

(informative)

This appendix supplies a Metamodel and Profile mapping of the two most popular Assurance Case Notations: namely GSN (Goal Structure Notation) and CAE (Claims, Arguments, and Evidence).

The Profiles for GSN and CAE have green colored elements which define elements that are concrete (instantiable) and provide a layer on top of SACM to do GSN and CAE.

A.1 Goal Structuring Notation (GSN)

GSN (Goal Structuring Notation) is one of the most commonly used notations for Assurance Cases. GSN is a notation but not a modeling language, even though it has some language-like features such as stated restrictions on its use of elements (like which elements one can connect with which relationship). SACM has been built taking GSN into account such that every concept in GSN can be captured in SACM and even greater nuances than GSN can be represented. There is not a one-to-one correspondence with GSN, but this section has a mapping between the two. Some of the concepts in GSN are captured in SACM through other mechanisms. This standard would suggest that Assurance Cases be represented directly in SACM, but GSN can be used (with a backing of SACM) as a notation on top of SACM.

The GSN standard can be found at https://scsc.uk/gsn-standard.

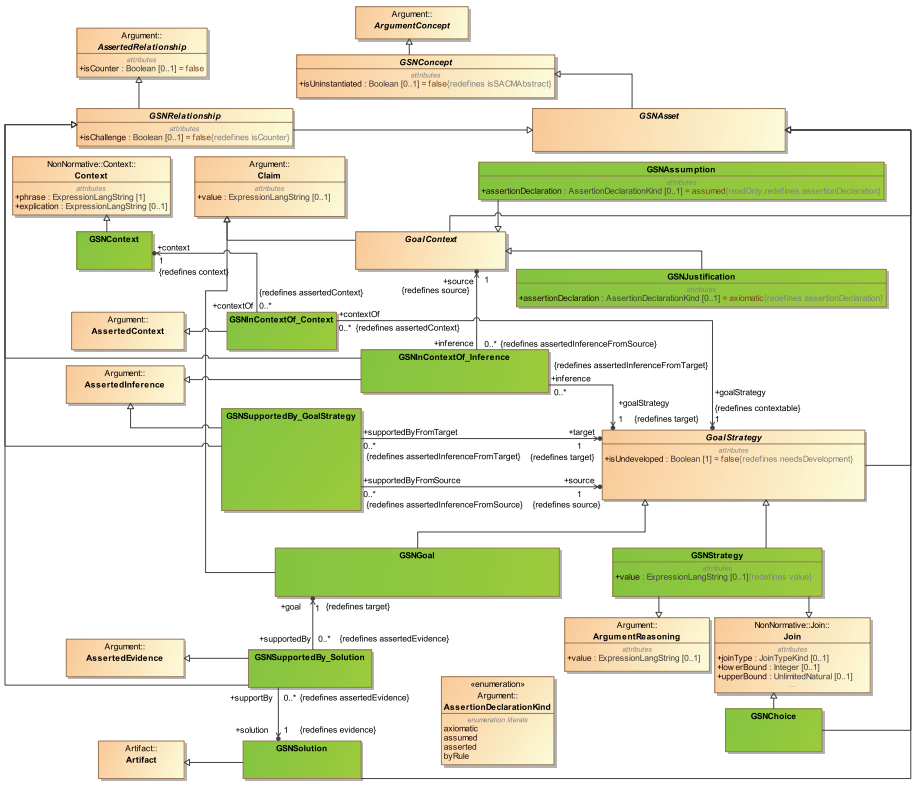

Figure A.1 – GSN Diagram

A.1.1 GSNConcept (abstract)

GSNConcept is the most abstract metaclass for all GSN concepts including GSNRelationship and GSNElement.

Superclass

Argument::ArgumentConcept

Attributes

isUninstantiated : Boolean [0..1] = false {redefines isSACMAbstract} - SACM's term is Abstract for uninstantiated, one can define a concrete element in SACM

A.1.2 GSNAsset (abstract)

GSNElement is the most abstract metaclass for all GSN non-relationships.

Superclass

GSNConcept

A.1.3 GSNRelationship (abstract)

GSNRelationship is the most abstract metaclass for all GSN relationships.

Superclass

GSNAsset, Argument::AssertedRelationship

Attributes

isChallenge : Boolean [0..1] = false {redefines isCounter} - This is a counter argument called a Challenge in GSN

Semantics

Note that each of the relationships in SACM has targets and sources reversed in their metamodels and notations.

All Relationships are binary in GSN.

A.1.4 GoalStrategy (abstract)

GoalStrategy is the abstract metaclass for GSNGoal and GSNStrategy.

Superclass

GSNElement

Attributes

isUndeveloped : Boolean [1] = false {redefines needsDevelopment} - GSN term is isUndeveloped, SACM term is needsDevelopment

A.1.5 GSNGoal

GSNGoal represents the Goal in GSN.

Superclass

GoalStrategy

A.1.6 GSNStrategy

GSNStrategy represents the Strategy in GSN.

Superclass

GoalStrategy

Attributes

value : ExpressionLangString [0..1] {redefines value} - Content of Strategy statement

Semantics

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

A.1.7 GoalContext (abstract)

GoalContext is an abstract class for GSNJustification and GSNAssumption.

Superclass

Argument::Claim, GSNElement

Semantics

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

A.1.8 GSNJustification

GSNJustification is a Justification in GSN.

Superclass

GoalContext

Attributes

assertionDeclaration : AssertionDeclarationKind [0..1] = axiomatic {redefines assertionDeclaration}

A.1.9 GSNAssumption

GSNAssumption is an Assumption in GSN.

Superclass

GoalContext

Attributes

assertionDeclaration : AssertedDeclarationKind [0..1[= assumed {redefines assertionDeclaration}

A.1.10 GSNSolution

GSNSolution is a Solution in GSN.

Superclass

GSNAsset,Artifact::Artifact

A.1.11 GSNContext

GSNContext is a Context in GSN.

Superclass

GSNAsset, Artifact::Artifact

A.1.12 GSNChoice

GSNChoice is the one element needed to support all of the GSN Extensions for Structural Abstract (options, multiple instantiation, and choice).

Superclass

GSNAsset, NonNormative::Join::Join

A.1.13 GSNInContextOf_Context

GSNInContextOf_Context is the GSN InContextOf relationship that is supported by AssertedContext relationship in SACM.

Superclass

GSNRelationship, Argument::AssertedContext

AssociationEnds

context : GSNContext [1] {redefines context} - the context part (source in SACM, target in GSN) of this relationship

goalStrategy : GoalStrategy [1] {redefines contextable} - the contextable part (target in SACM, source in GSN) of this relationship

A.1.14 GSNInContextOf_Inference

GSNInContextOf_Inference is the GSN InContextOf relationship that is supported by AssertedInference relationship in SACM.

Superclasses

GSNRelationship, Argument::AssertedInference

AssociationEnds

source : GoalContext [1] {redefines source} - the inferrer of this relationship

goalStrategy : GoalStrategy [1] {redefines target} - the inferred of this relationship

A.1.15 GSNSupportedBy_GoalStrategy

GSNSupportedBy_GoalStrategy is the GSN SupportedBy relationship that is supported by AssertedInference relationship in SACM.

Superclass

GSNRelationship, Argument::AssertedInference

AssociationEnds

source : GoalStrategy [1] {redefines source} - the supporting element of the relationship

target : GoalStrategy [1] {redefines target} - the supported element of the relationship

A.1.16 GSNSupportedBy_Solution

GSNSupportedBy_Solution is the GSN SupportedBy relationship that is supported by AssertedEvidence relationship in SACM.

Superclass

GSNRelationship, Argument::AssertedEvidence

AssociationEnds

solution : GSNSolution [1] {redefines evidence} - the supporting element of the relationship

goal : GSNGoal [1] {redefines target} - the supported element of the relationship.

A.2 GSN Package

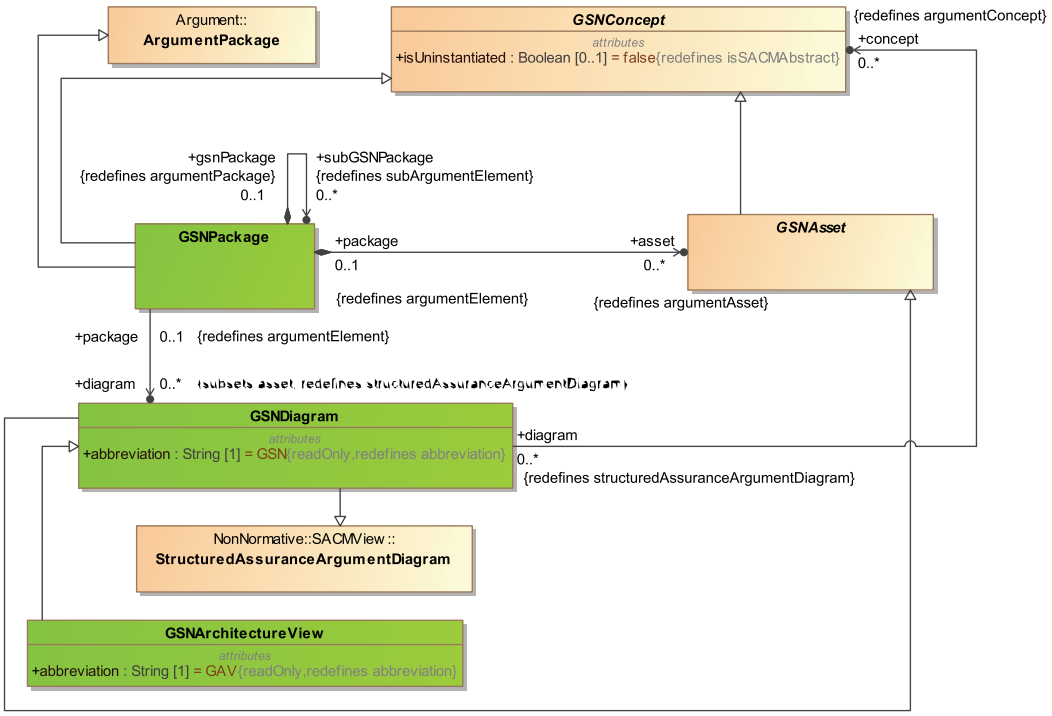

Figure A.2 – GSN Package Diagram

A.2.1 GSNPackage

GSNPackage is a Package that may contain any GSN Assets as well as BaseElements, and can contain another GSNPackage.

Superclass

GSNConcept, Argument::ArgumentPackage

AssociationEnds

asset : GSNAsset [0..*] {redefines argumentAsset} - GSNAsset contained in this package

diagram : GSNDiagram [0..*] {subsets asset, redefines structuredAssuranceArgumentDiagram} - GSNDiagrams contained in this package

subGSNPackage : GSNPackage [0..*] {redefines subArgumentElement}

A.2.2 GSNDiagram

GSNDiagram is a Diagram that can have any GSN concept as well as BaseElements.

Superclass

GSNAsset, NonNormative::SACMView::StructuredAssuranceArgumentDiagram

Attributes

abbreviation : String [1] = GSN {readOnly, refines abbreviation}

AssociationEnds

concept : GSNConcept [0..*] {redefines argumentConcept} - concepts that are shown on the GSNDiagram.

A.2.3 GSNArchitectureView

GSNArchitectureView is a GSN Diagram that has high level connections among GSN Modules.

Superclass

GSNDiagram

Attributes

abbreviation : String [1] = GAV {readOnly, refines abbreviation}

A.2.4 Away Elements

There are many elements that are prefaced by "Away" in GSN. "Away" is modeled in SACM as cited Elements in SACM. In SACM, an "Away" element is created in the model and they can cite

either a local (in the same model) element using cited AssociationEnd or if it is not local, then citedIRI can be used to provide an IRI for a non-local reference.

A.2.5 Modular Packaging

There are many concepts that are in GSN to support modularization and contracts. These can be modeled in SACM with Packages, InterfacePackages, and BindingPackages.

A.3 GSN Profiles

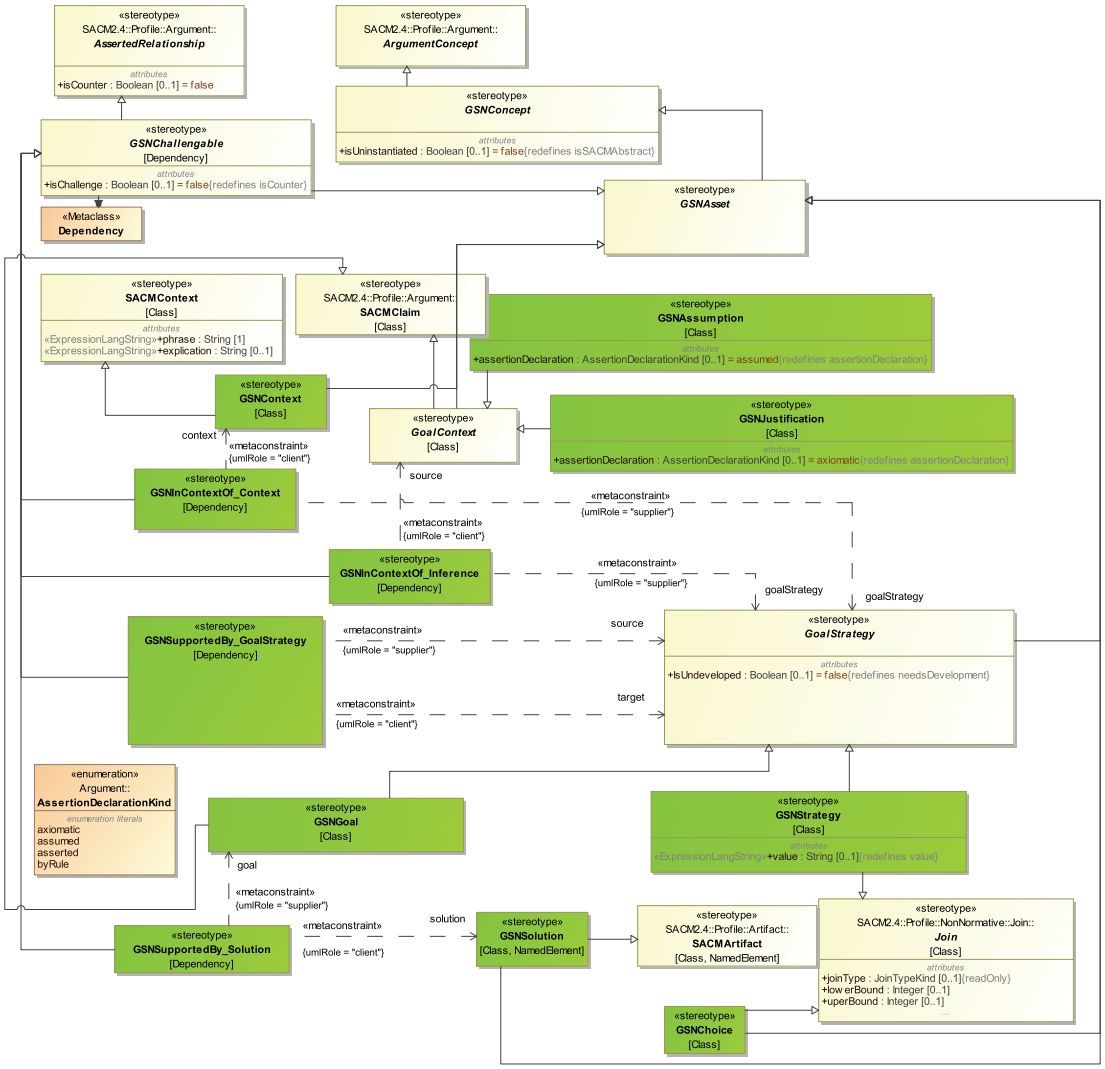

Figure A.3.1 – GSN Profile Diagram

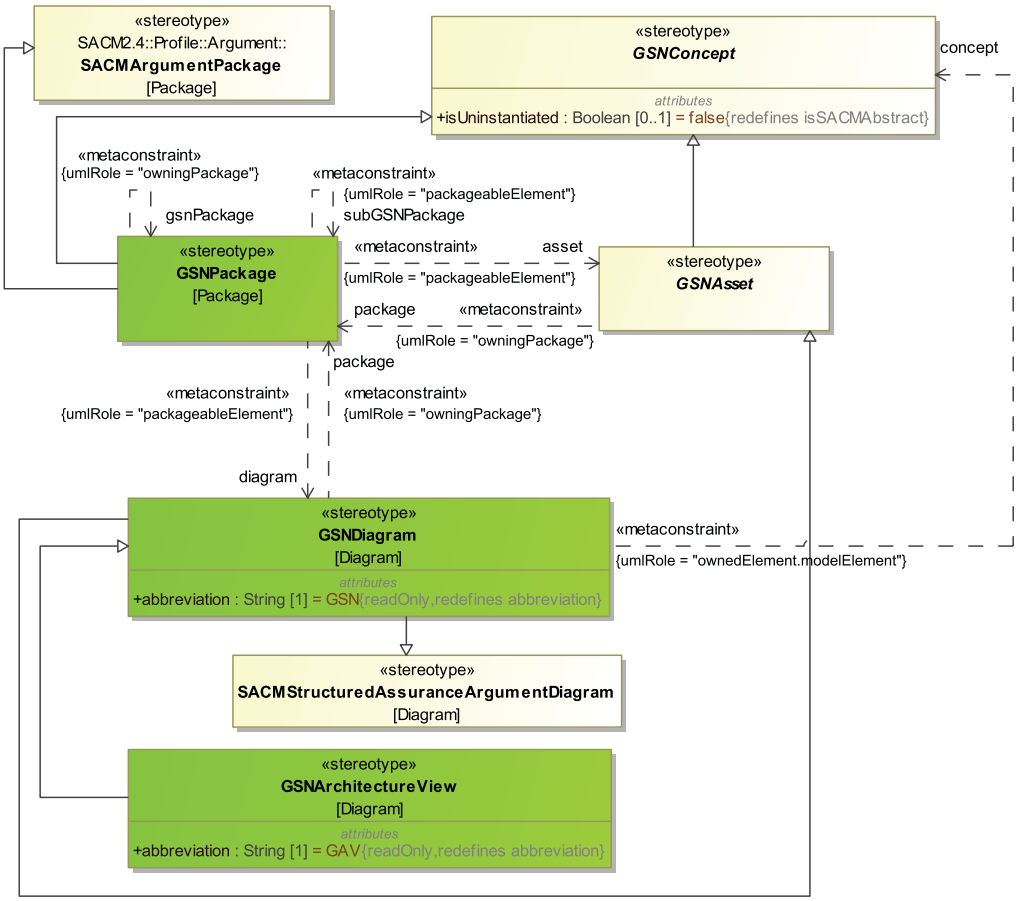

Figure A.3.2 – GSN Package Profile Diagram

A.4 Claims, Arguments, Evidence (CAE)

CAE (Claims, Arguments, and Evidence) is a standard way of defining Assurance Cases using a minimal set of elements.

CAE documentation can be found at https://claimsargumentsevidence.org/notations/claims-arguments-evidence-cae.

For "Other" (hexagon) Notation in CAE, which is used to capture various supporting information (e.g. constraints, explanations, context, additional details about the case, etc.), that help in understanding the case but do not form part of the logical argument, use specific SACM BaseElements such as SACMComment, SACMConstraint, Group, and SACMDependency.

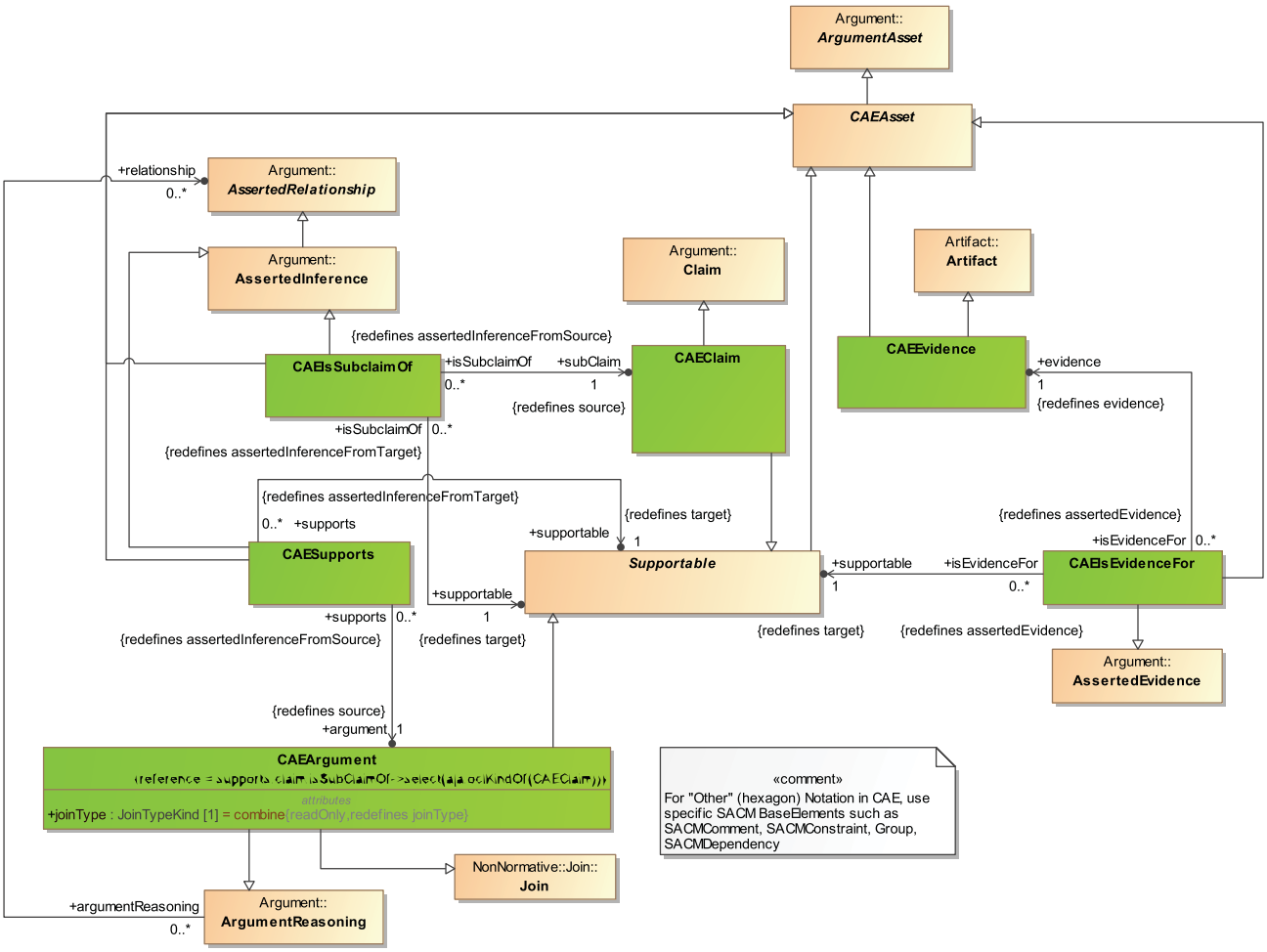

Figure A.4 – CAE Diagram

A.4.1 CAEClaim

A claim is a true/false statement about a property of a particular object. A claim is just what you might consider it to be from common usage of the term; an idea that someone is trying to convince somebody else is true. An example claim could be made on a train, “The train is safe”.

Superclass

Argument::Claim, CAEAsset

A.4.2 CAEArgument

An argument is a rule that provides the bridge between what we know or are assuming (sub-claims, evidence) and the claim we are investigating. The argument used depends on the type, trustworthiness and extent of available evidence and the nature of the claim. Note that "argument" is an overloaded word and we are using with a specific meaning here.

Superclass

NonNormative::Join::Join, Argument::ArgumentReasoning, Supportable

Attributes

joinType : JoinTypeKind [1] = combine {readOnly,redefines joinType}

Constraints

DeriveReference

Reference is set to be the CAEClaims that are subclaims of this CAEArgument.

inv: reference = supports.claim.isSubClaimOf->select(a|a.oclKindOf(CAEClaim))

A.4.3 CAEEvidence

Evidence is an artefact that establishes facts that can be trusted and lead directly to a claim. In projects there can be many sources of information, but what makes this evidence is the support or rebuttal it gives to a claim.

Superclass

Artifact::Artifact, CAEAsset

A.4.4 CAEIsSubclaimOf

CAEIsSubclaimOf connects a source CAEClaim to either another CAEClaim (indicating that the source claim is a subclaim of the target) or to a CAEArgument (indicates that the source claim is a subclaim of the CAESupports target connected to the CAEArgument).

Superclass

Argument::AssertedInference, CAEAsset

AssociationEnds

subClaim : CAEClaim [1] {redefines source} - subclaim of this relationship

supportable : Supportable [1] {redefines target} - CAEClaim or CAEArgument target of this relationship

A.4.5 CAESupports

CAESupports connects CAEArgument source to a CAEClaim target indicating that the CAEArgument (and everything connected to it) support the CAEClaim.

Superclass

Argument::AssertedInference, CAEAsset

AssociationEnds

claim : CAEClaim [1] {redefines target} - target CAEClaim supported by the CAEArgument

argument : CAEArgument [1] {redefines source} - CAEArgument that supports the CAEClaim

supportable : Supportable [1] {redefines target} - target CAEClaim (or other CAEArgument) supported by the CAEArgument

argument : CAEArgument [1] {redefines source} - CAEArgument that supports the CAEClaim

A.4.6 CAEIsEvidenceFor

CAEIsEvidenceFor ties CAEEvidence to what it provides evidence for (CAEClaims or CAEArgument).

Superclass

Argument::AssertedEvidence, CAEAsset

AssociationEnds

supportable : Supportable [1] {redefines target} - target CAEClaim or CAEArgument for this relationship

evidence : CAEEvidence [1] {redefines evidence} - source evidence for this relationship

A.4.7 CAEAsset (abstract)

CAEAsset is an abstract class for all of the CAE elements.

Superclass

Argument::ArgumentAsset

A.4.8 Supportable (abstract)

Supportable is an abstract class that combines specializations for CAEArgument and CAEClaim.

Superclass

CAEAsset

A.5 CAE Package

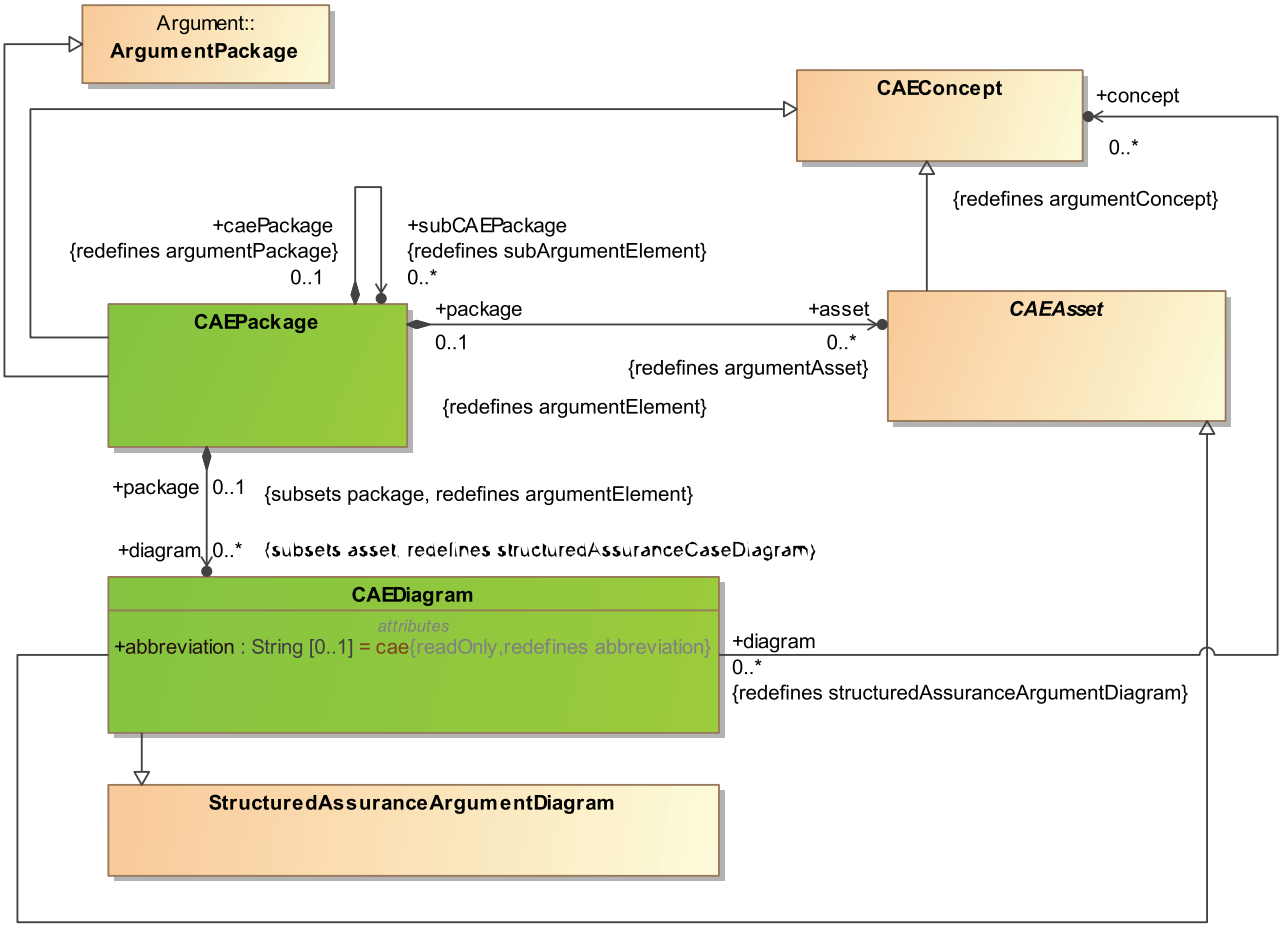

Figure A.5 – CAE Package Diagram

A.5.1 CAEPackage

CAEPackage allows for the ownership of CAEConcepts (including CAEDiagrams) and BaseElements.

Superclass

Argument::ArgumentPackage

Attributes

caeConcept : CAEConcept [0..*] {redefines argumentAsset} - concepts that can be included in this package along with BaseElements

caeDiagram : CAEDiagram [0..*] {subsets caeConcept, redefines structuredAssuranceCaseDiagram} - CAEDiagrams in this package

AssociationEnds

asset : CAEAsset [0..*] {redefines argumentAsset}

diagram : CAEDiagram [0..*] {subsets asset, redefines structuredAssuranceCaseDiagram}

subCAEPackage : CAEPackage [0..*] {redefines subArgumentElement}

A.5.2 CAEDiagram

CAEDiagram is the diagram type for CAE Assurance Cases that include elements of type CAEConcepts and BaseElements

Superclass

NonNormative::SACMView::SACMDiagram

Attributes

caeConcept : CAEConcept [0..*] {redefines argumentAsset} - concepts that can be included in this package along with BaseElements

A.6 CAE Profiles

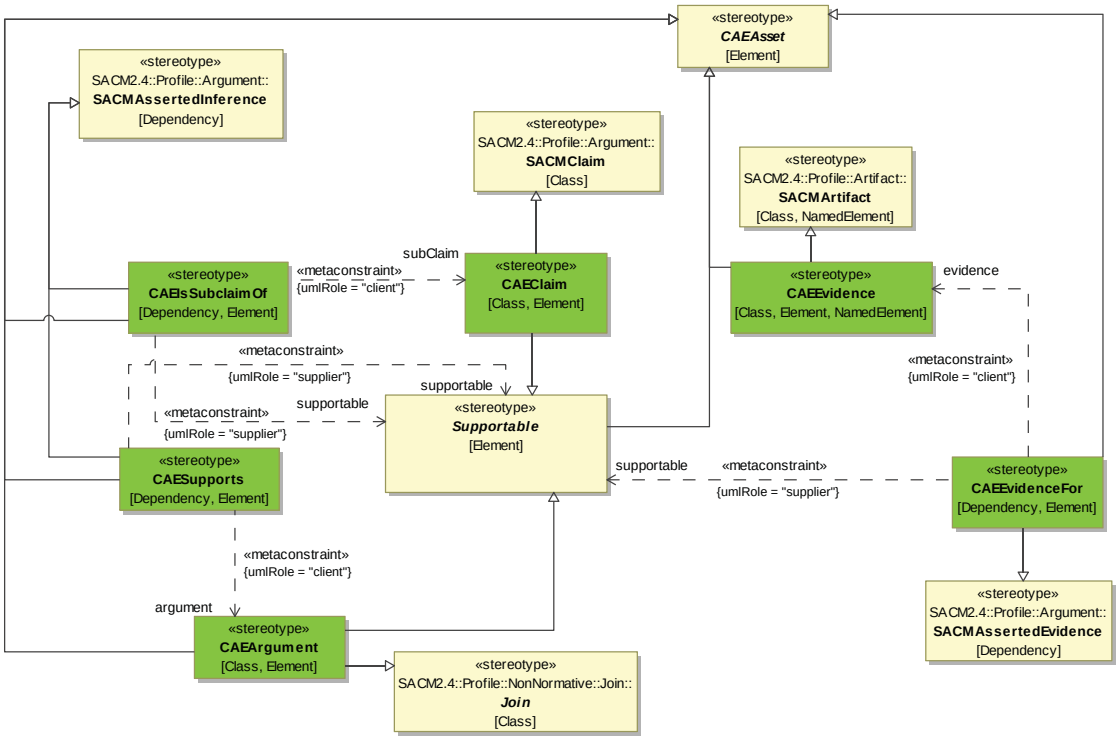

Figure A.6.1 – CAE Profile Diagram

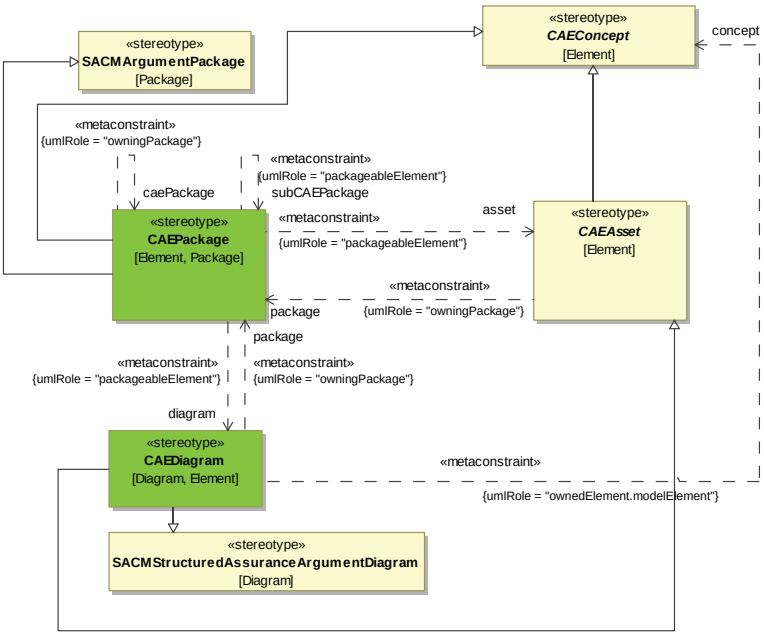

Figure A.6.2 – CAE Package Profile Diagram

This page intentionally left blank.

Annex B: Examples of Assurance Cases in SACM 2.0 XMI

(informative)

B.1 SACM Example

The Example given is a typical example of a Claim supported by two subclaims each supported by separate evidence. This model example is using the Profile for the language as given in the standard.

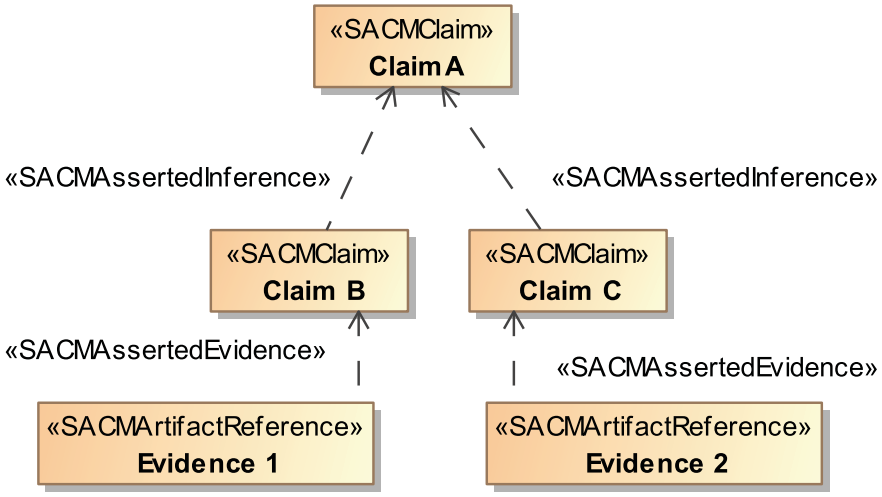

Figure B.1 – SACM Example 1 Assurance Case Structure Diagram

In addition, the ownership of the model elements in the Assurance Case model are shown below.

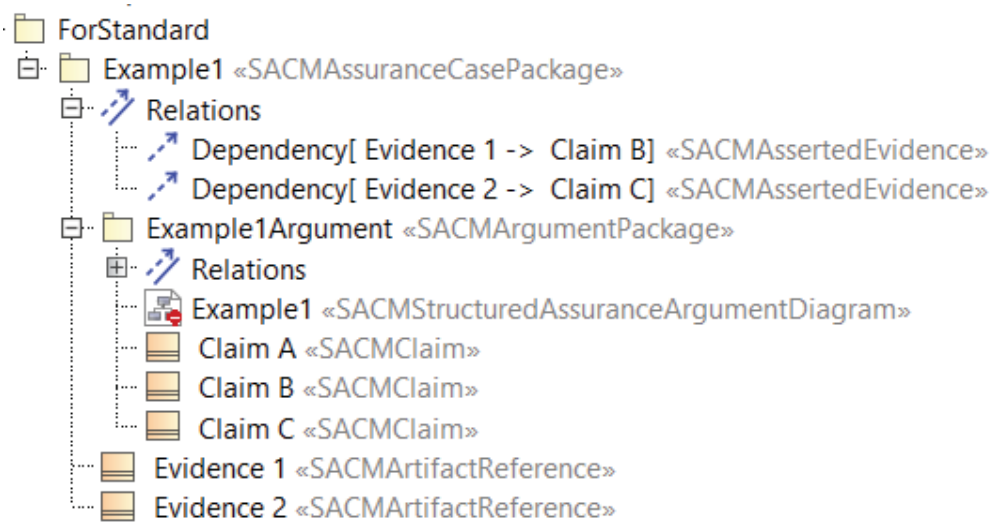

Figure B.2 – SACM Example Model Element Ownership Diagram

B.2 SACM Example Model XMI Listing

The XMI output for this example is given below and specifies the MOF XML (i.e. XMI) serialization of the model into a file. This XMI file is a way to send the model to various other tools that support the XMI standard and may also allow long term storage of the model in a vendor-independent format.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmlns:uml="http://www.omg.org/spec/UML/20131001"
         xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
         xmlns:Profile="http://www.magicdraw.com/schemas/Profile.xmi">
  <uml:Model xmi:type="uml:Model"
    xmi:id="_2022x_1_27f60570_1742433912151_479101_1669" name="Model">
    <packagedElement xmi:type="uml:Package"
      xmi:id="_2024x_3_109d038b_1776881572636_117251_3950" name="ForStandard">
      <packagedElement xmi:type="uml:Package"
        xmi:id="_2024x_3_109d038b_1776881829831_783055_4162" name="Example1">
        <packagedElement xmi:type="uml:Package"
          xmi:id="_2024x_3_109d038b_1776881842552_656603_4171" name="Example1Argument">
          <packagedElement xmi:type="uml:Class"
            xmi:id="_2024x_3_109d038b_1776881866164_204035_4207" name="Claim A"/>
          <packagedElement xmi:type="uml:Class"
            xmi:id="_2024x_3_109d038b_1776881886189_485602_4237" name="Claim B"/>
          <packagedElement xmi:type="uml:Class"
            xmi:id="_2024x_3_109d038b_1776881907557_529114_4269" name="Claim C"/>
          <packagedElement xmi:type="uml:Dependency"
            xmi:id="_2024x_3_109d038b_1776881932359_300819_4317">
            <client xmi:idref="_2024x_3_109d038b_1776881886189_485602_4237"/>
            <supplier xmi:idref="_2024x_3_109d038b_1776881866164_204035_4207"/>
          </packagedElement>
          <packagedElement xmi:type="uml:Dependency"
            xmi:id="_2024x_3_109d038b_1776881950333_530939_4350">
            <client xmi:idref="_2024x_3_109d038b_1776881907557_529114_4269"/>
            <supplier xmi:idref="_2024x_3_109d038b_1776881866164_204035_4207"/>
          </packagedElement>
        </packagedElement>
        <packagedElement xmi:type="uml:Class"
          xmi:id="_2024x_3_109d038b_1776882014653_554511_4424" name="Evidence 1"/>
        <packagedElement xmi:type="uml:Class"
          xmi:id="_2024x_3_109d038b_1776882044621_607136_4464" name="Evidence 2"/>
        <packagedElement xmi:type="uml:Dependency"
          xmi:id="_2024x_3_109d038b_1776882094604_823018_4508">
          <client xmi:idref="_2024x_3_109d038b_1776882014653_554511_4424"/>
          <supplier xmi:idref="_2024x_3_109d038b_1776881886189_485602_4237"/>
        </packagedElement>
        <packagedElement xmi:type="uml:Dependency"
          xmi:id="_2024x_3_109d038b_1776882136646_385464_4568">
          <client xmi:idref="_2024x_3_109d038b_1776882044621_607136_4464"/>
          <supplier xmi:idref="_2024x_3_109d038b_1776881907557_529114_4269"/>
        </packagedElement>
      </packagedElement>
    </packagedElement>
  </uml:Model>
  <Profile:SACMAssuranceCasePackage xmi:id="_2024x_3_109d038b_1776881829831_783055_4162_application"
    base_Package="_2024x_3_109d038b_1776881829831_783055_4162"/>
  <Profile:SACMArgumentPackage xmi:id="_2024x_3_109d038b_1776881842552_656603_4171_application"
    base_Package="_2024x_3_109d038b_1776881842552_656603_4171"/>
  <Profile:SACMClaim xmi:id="_2024x_3_109d038b_1776881866164_204035_4207_application"
    base_Class="_2024x_3_109d038b_1776881866164_204035_4207"/>
  <Profile:SACMClaim xmi:id="_2024x_3_109d038b_1776881886189_485602_4237_application"
    base_Class="_2024x_3_109d038b_1776881886189_485602_4237"/>
  <Profile:SACMClaim xmi:id="_2024x_3_109d038b_1776881907557_529114_4269_application"
    base_Class="_2024x_3_109d038b_1776881907557_529114_4269"/>
  <Profile:SACMAssertedInference xmi:id="_2024x_3_109d038b_1776881932359_300819_4317_application"
    base_Dependency="_2024x_3_109d038b_1776881932359_300819_4317"/>
  <Profile:SACMAssertedInference xmi:id="_2024x_3_109d038b_1776881950333_530939_4350_application"
    base_Dependency="_2024x_3_109d038b_1776881950333_530939_4350"/>
  <Profile:SACMArtifactReference xmi:id="_2024x_3_109d038b_1776882014653_554511_4424_application"
    base_Class="_2024x_3_109d038b_1776882014653_554511_4424"/>
  <Profile:SACMArtifactReference xmi:id="_2024x_3_109d038b_1776882044621_607136_4464_application"
    base_Class="_2024x_3_109d038b_1776882044621_607136_4464"/>
  <Profile:SACMAssertedEvidence xmi:id="_2024x_3_109d038b_1776882094604_823018_4508_application"
    base_Dependency="_2024x_3_109d038b_1776882094604_823018_4508"/>
  <Profile:SACMAssertedEvidence xmi:id="_2024x_3_109d038b_1776882136646_385464_4568_application"
    base_Dependency="_2024x_3_109d038b_1776882136646_385464_4568"/>
</xmi:XMI>
```

This page intentionally left blank.

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

(informative)

C.1 ArtifactReference

The concrete syntax for ArtifactReference is defined in Figure C4.

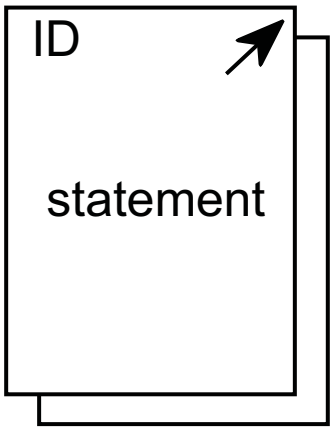

Figure C1 – Concrete Syntax for ArtifactReference

C.2 The Subject reference

Claims can use subject AssociationEnd to reference a SACMElement as the subject of the Claim (e.g., regarding the confidence in the Assertion). The concrete syntax for the Subject reference is defined in Figure C2.

Figure C2 – Concrete Syntax for Subject reference

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

C.3 Claim

By default the AssertionDeclaration of a Claim is set to asserted, the concrete syntax for an asserted Claim is defined in Figure C3.

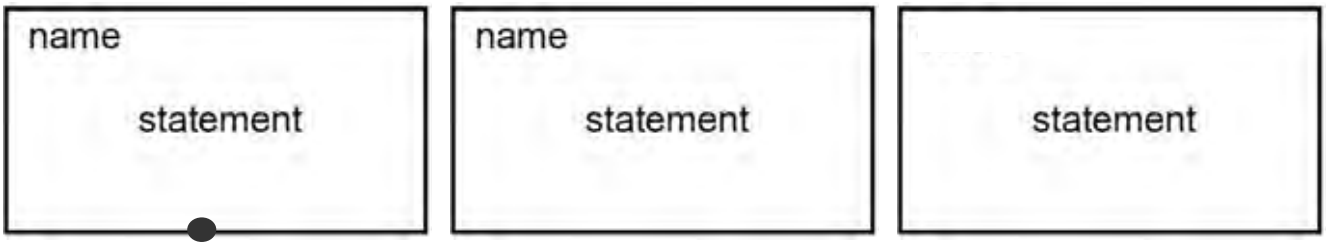

Figure C3 – Concrete Syntax for asserted Claim

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 C4.

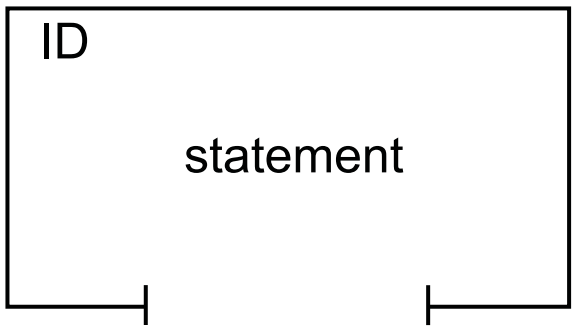

Figure C4 – Concrete Syntax for assumed Claim

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 C5.

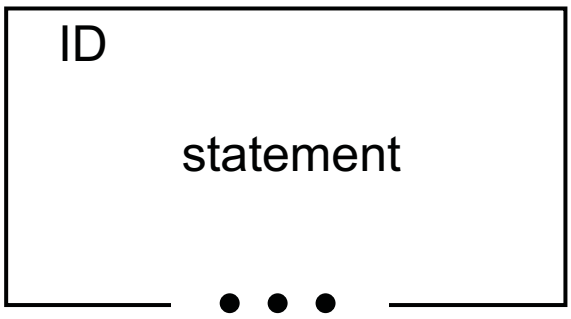

Figure C5 – Concrete Syntax for needsSupport Claim

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 C6.

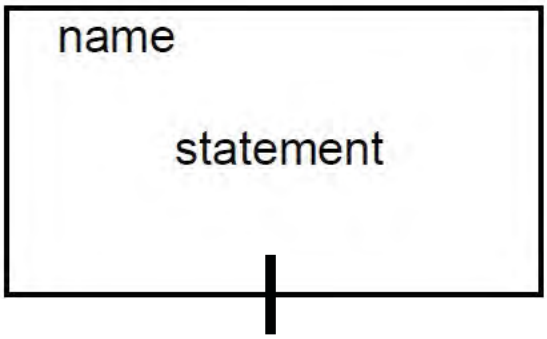

Figure C6 – Concrete Syntax for axiomatic Claim

A defeated Claim indicates that a Claim is defeated by counter–evidence. The concrete syntax of a defeated Claim is defined in Figure C7.

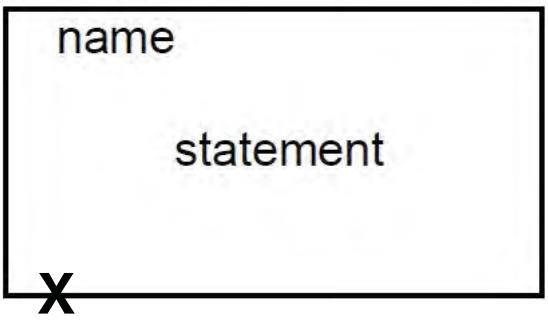

Figure C7 – Concrete Syntax for defeated Claim

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 C8.

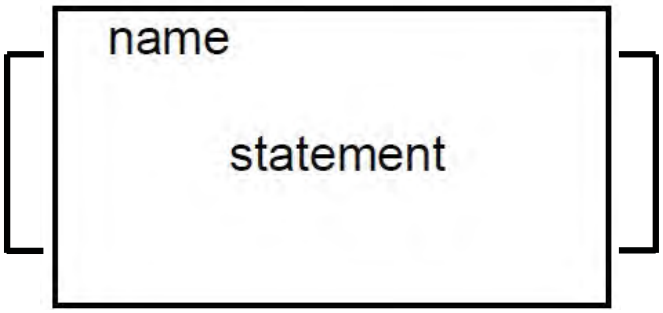

Figure C8 – Concrete Syntax for asCited Claim

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 C9 is an example of an abstract asserted Claim.

Figure C9 – 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.4 ArgumentReasoning

The concrete syntax of ArgumentReasoning is defined in C10 (note : the right hand side of the notation).

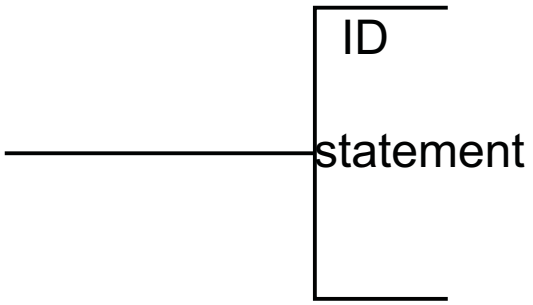

Figure C10 – Concrete Syntax for ArgumentReasoning

C.5 AssertedInference

The concrete syntax of AssertedInference is defined in Figure C11, 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 C11 – 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 C12 (note : the change is applied to the +target reference edge of an AssertedInference).

Figure C12 – 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 13 (note : the change is applied to the +target reference edge of an AssertedInference).

Figure C13 – 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 C14 (note : the change is applied to the +target reference edge of an AssertedInference).

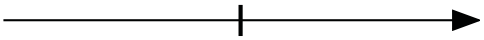

Figure C14 – Concrete Syntax for axiomatic AssertedInference

A defeated AssertedInference indicates that the inference is defeated by counter–evidence. The concrete syntax of a defeated AssertedInference is defined in Figure C15 (note : the change is applied to the +target reference edge of an AssertedInference).

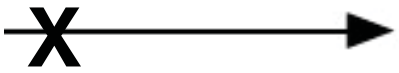

Figure C15 – Concrete Syntax for defeated AssertedInference

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 C16 (note : the change is applied to the +target reference edge of an AssertedInference).

Figure C16 – 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 C17 (note : the change is applied to the +target reference edge of an AssertedInference).

Figure C17 – Concrete Syntax for abstract asserted AssertedInference

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

If an AssertedInference has isCounter=true, then it should be interpreted as "If the source is undefeated then the target is defeated". The concrete syntax of an isCounter AssertedInference is defined in Figure C18 (note : the change is applied to the +target reference edge of an AssertedInference).

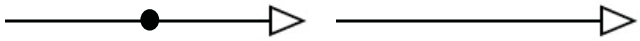

Figure C18 – Concrete Syntax for counter asserted AssertedInference

C.6 AssertedEvidence

The concrete syntax of AssertedEvidence is defined in Figure C19, 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 C19 – 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 C20 (note : the change is applied to the +target reference edge of an AssertedEvidence).

Figure C20 – 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 C21 (note : the change is applied to the +target reference edge of an AssertedEvidence).

Figure C21 – 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 C22 (note : the change is applied to the +target reference edge of an AssertedEvidence).

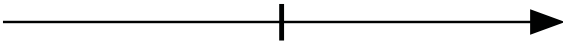

Figure C22 – Concrete Syntax for axiomatic AssertedEvidence

A defeated AssertedEvidence indicates that the inference is defeated by counter–evidence. The concrete syntax of a defeated AssertedEvidence is defined in Figure C23 (note : the change is applied to the +target reference edge of an AssertedEvidence).

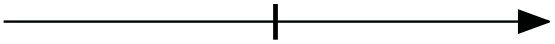

Figure C23 – Concrete Syntax for defeated AssertedEvidence

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 C24 (note : the change is applied to the +target reference edge of an AssertedEvidence).

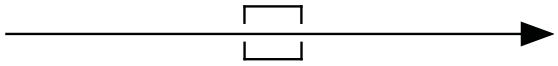

Figure C24 – Concrete Syntax for asCited AssertedEvidence

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 C25 (note : the change is applied to the +target reference edge of an AssertedEvidence).

Figure C25 – Concrete Syntax for abstract asserted AssertedEvidence

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

If an AssertedEvidence has isCounter=true, then it should be interpreted as "the evidence defeats the Assertion". The concrete syntax of an isCounter AssertedEvidence is defined in Figure C26 (note : the change is applied to the +target reference edge of an AssertedEvidence).

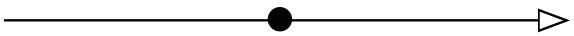

Figure C26 – Concrete Syntax for counter asserted AssertedEvidence

C.7 AssertedContext

The concrete syntax of AssertedContext is defined in Figure C27, 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 C27 – 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 C28 (note : the change is applied to the +target reference edge of an AssertedContext).

Figure C28 – 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 C29 (note : the change is applied to the +target reference edge of an AssertedContext).

Figure C29 – 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 C30 (note : the change is applied to the +target reference edge of an AssertedContext).

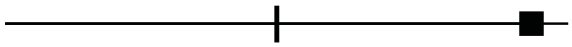

Figure C30 – Concrete Syntax for axiomatic AssertedContext

A defeated AssertedContext indicates that the inference is defeated by counter–evidence. The concrete syntax of a defeated AssertedContext is defined in Figure C31 (note : the change is applied to the +target reference edge of an AssertedContext).

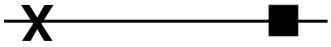

Figure C31 – Concrete Syntax for defeated AssertedContext

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 C32 (note : the change is applied to the +target reference edge of an AssertedContext).

Figure C32 – 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 C33 (note : the change is applied to the +target reference edge of an AssertedContext).

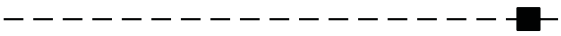

Figure C33 – Concrete Syntax for abstract asserted AssertedContext

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

If an AssertedContext has isCounter=true, then it should be interpreted as "the contextable must not be interpreted in the context". The concrete syntax of an isCounter AssertedContext is defined in Figure C34 (note : the change is applied to the +target reference edge of an AssertedContext).

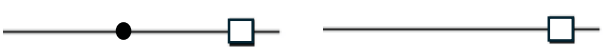

Figure C34 – Concrete Syntax for counter asserted AssertedContext

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.

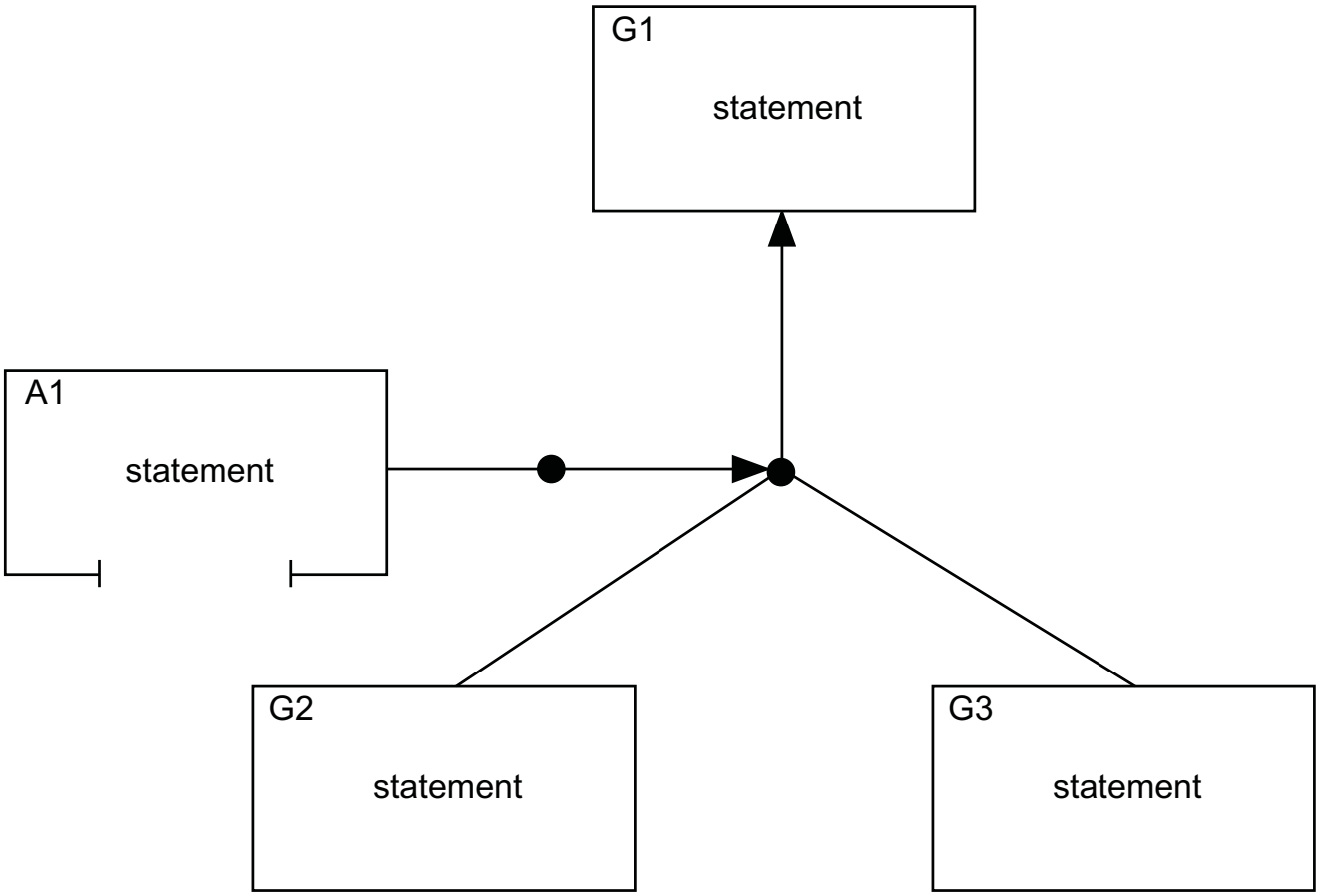

Figure D1 – Example of Claim Assumptions

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.

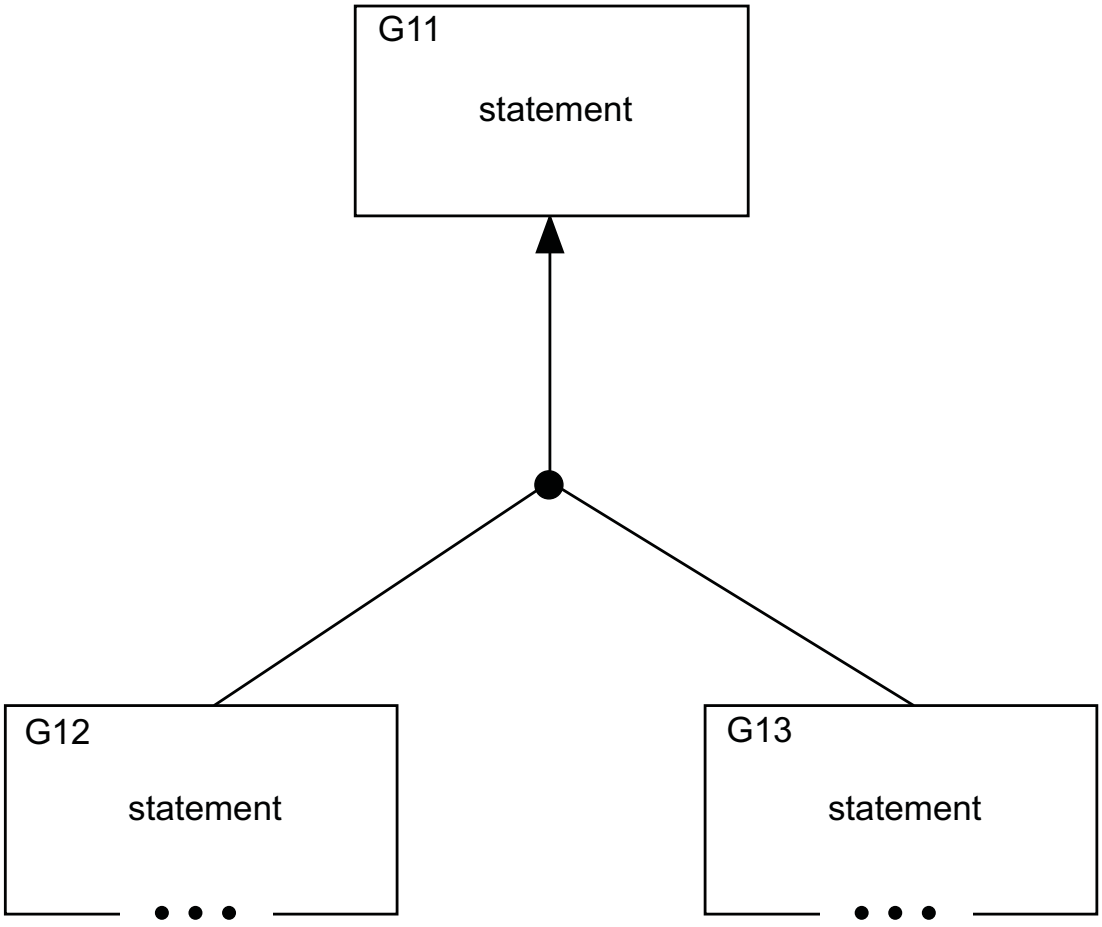

Figure D2 – Example of a Claim needing support

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.

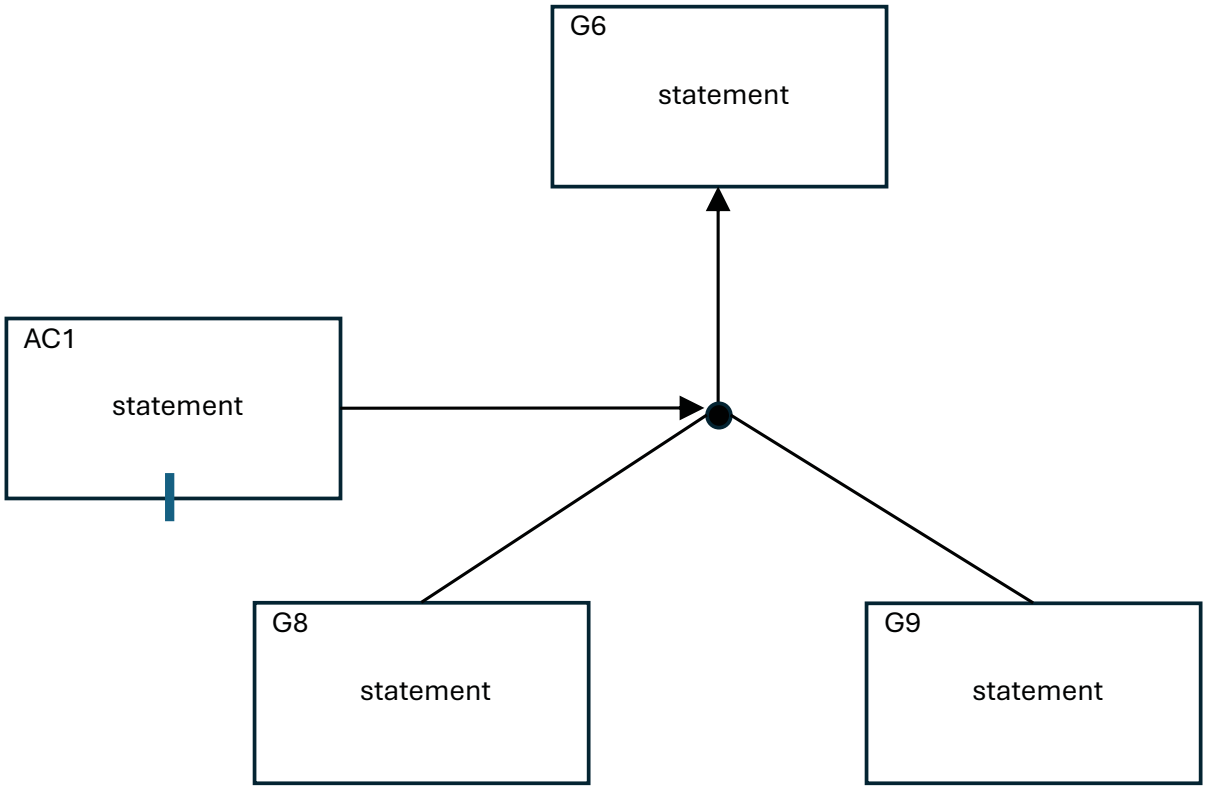

Figure D3 – Example of an Axiomatic Claim

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.

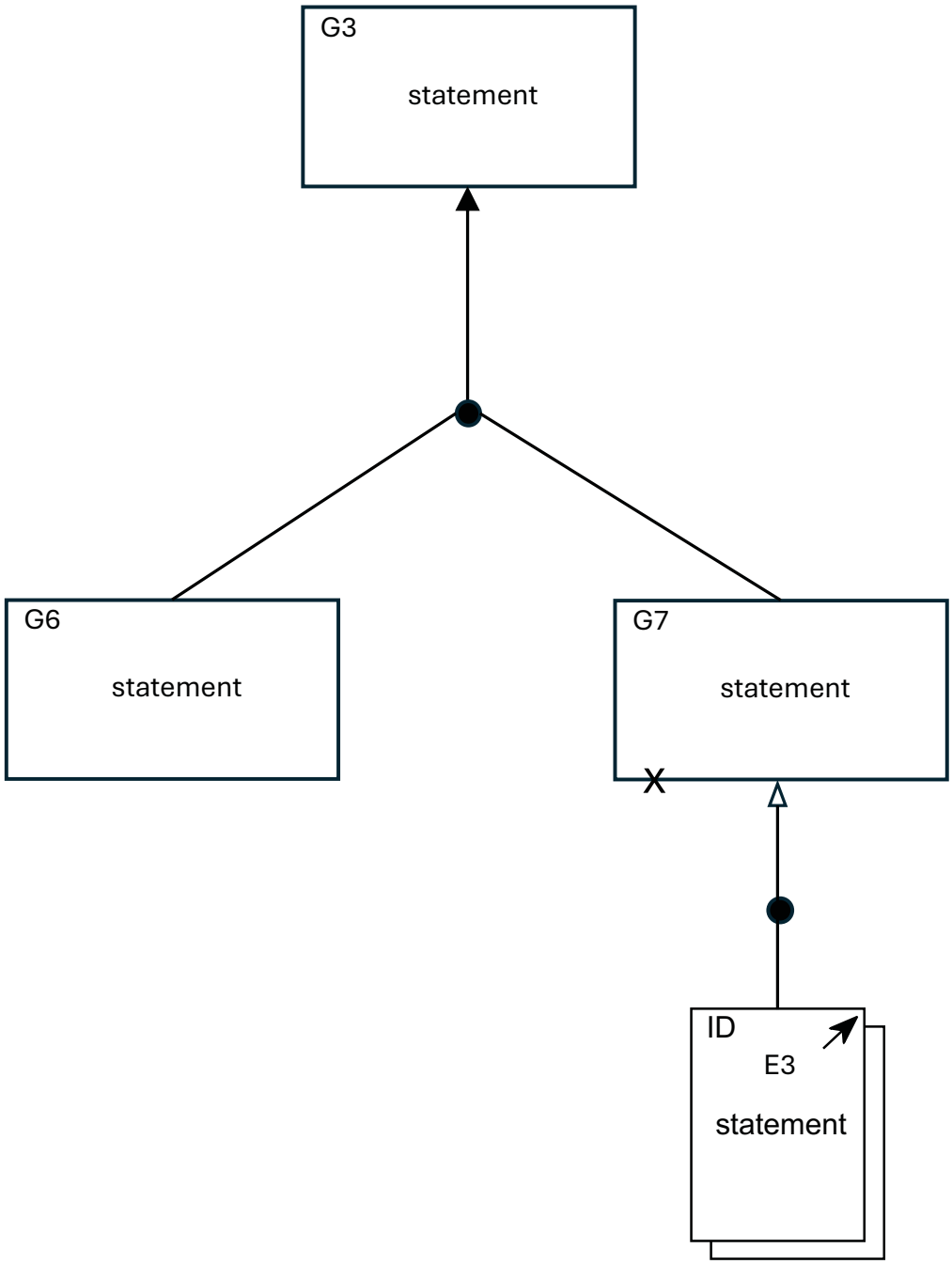

Figure D4 – Example of Defeating a Claim

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.

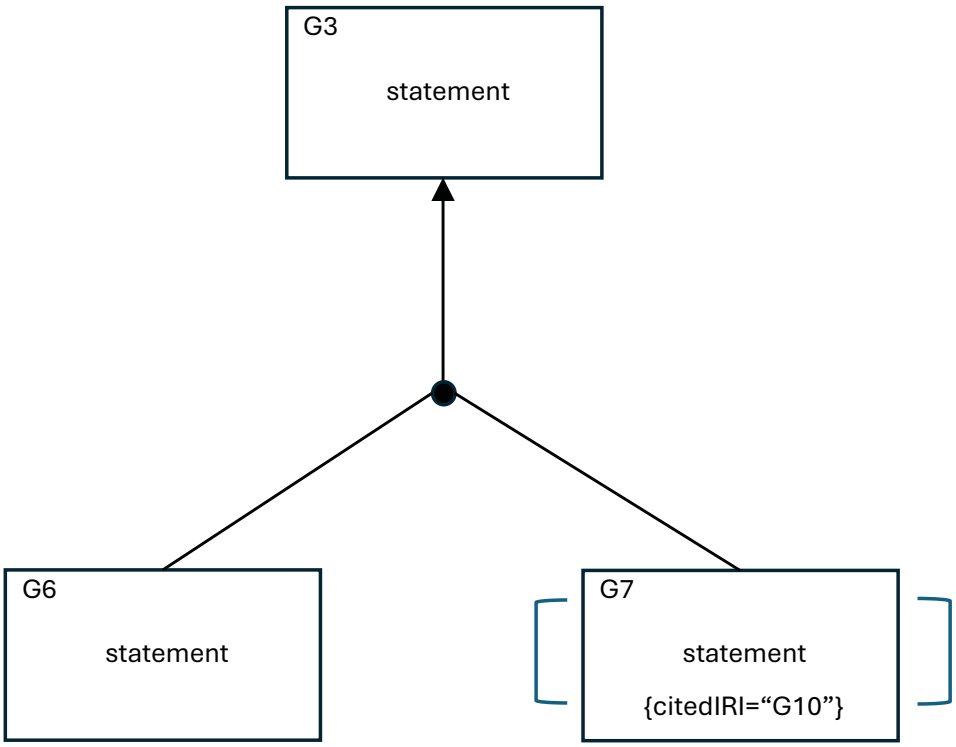

Figure D5 – Example of Claim citation

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.

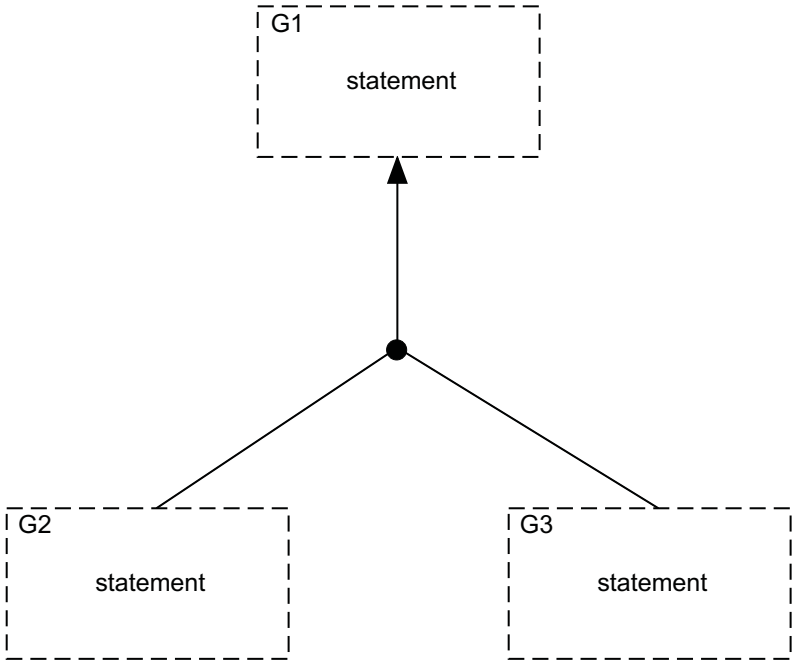

Figure D6 – Example of abstract Claims forming an argument pattern

D.2 Subject

When used in a diagram, the source element Claim and the targeted element can be type of SACMElement..

For example, Claim MC1 that is connected to Claim G1. The relationship between MC1 and G1 is declared using the Subject, indicates Claim MC1 is about Claim G1.

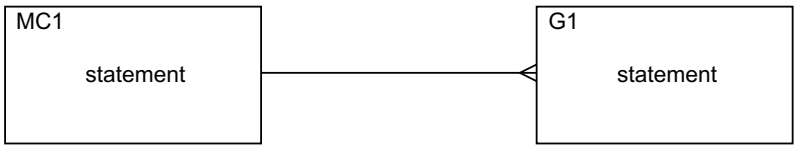

Figure D7 – Example of Claim and Subject Relationship

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.

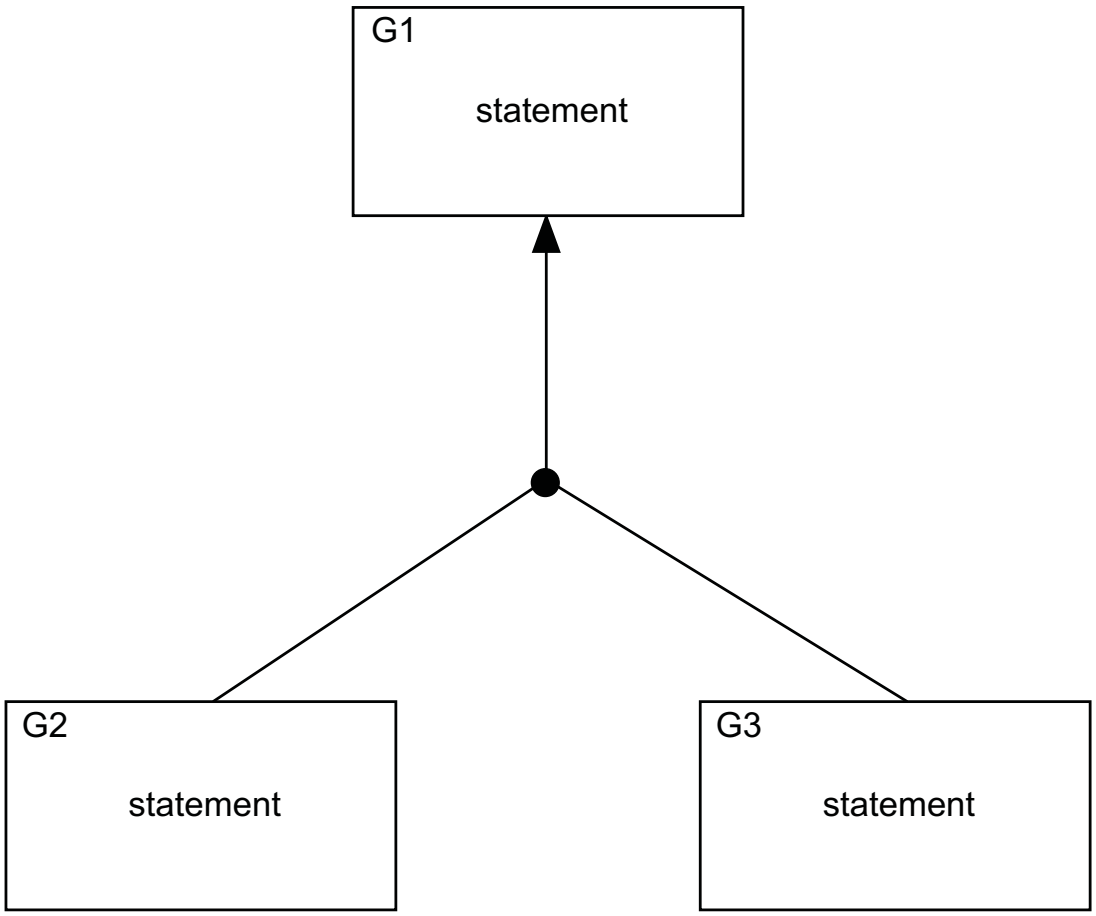

Figure D8 – Example of AssertedInference of Supporting Claims

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.

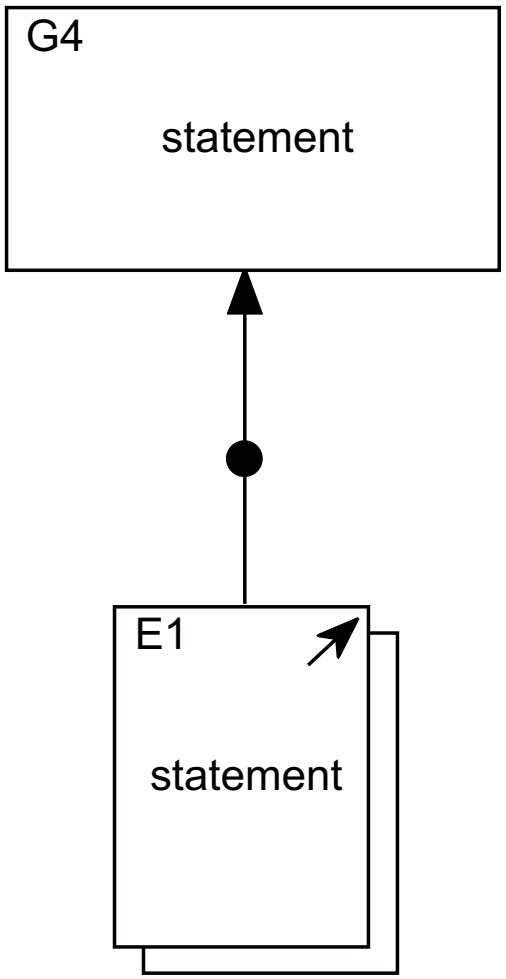

Figure D9 – Example of ArtifactReference Citation via AssertedEvidence

ArtifactReference as evidence can be used to support another ArtifactReference using the AssertedEvidence relationship.

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.

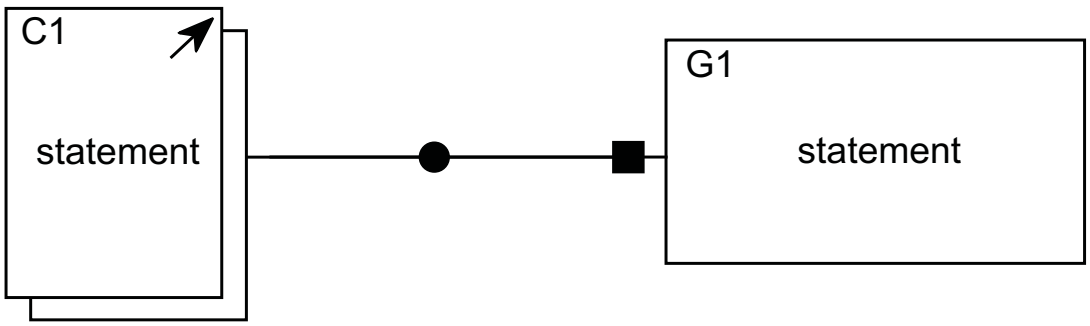

Figure D10 – Example of AssertedContext

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 of the ArtifactReference as evidence E1 are related by the AssertedContext relationship.

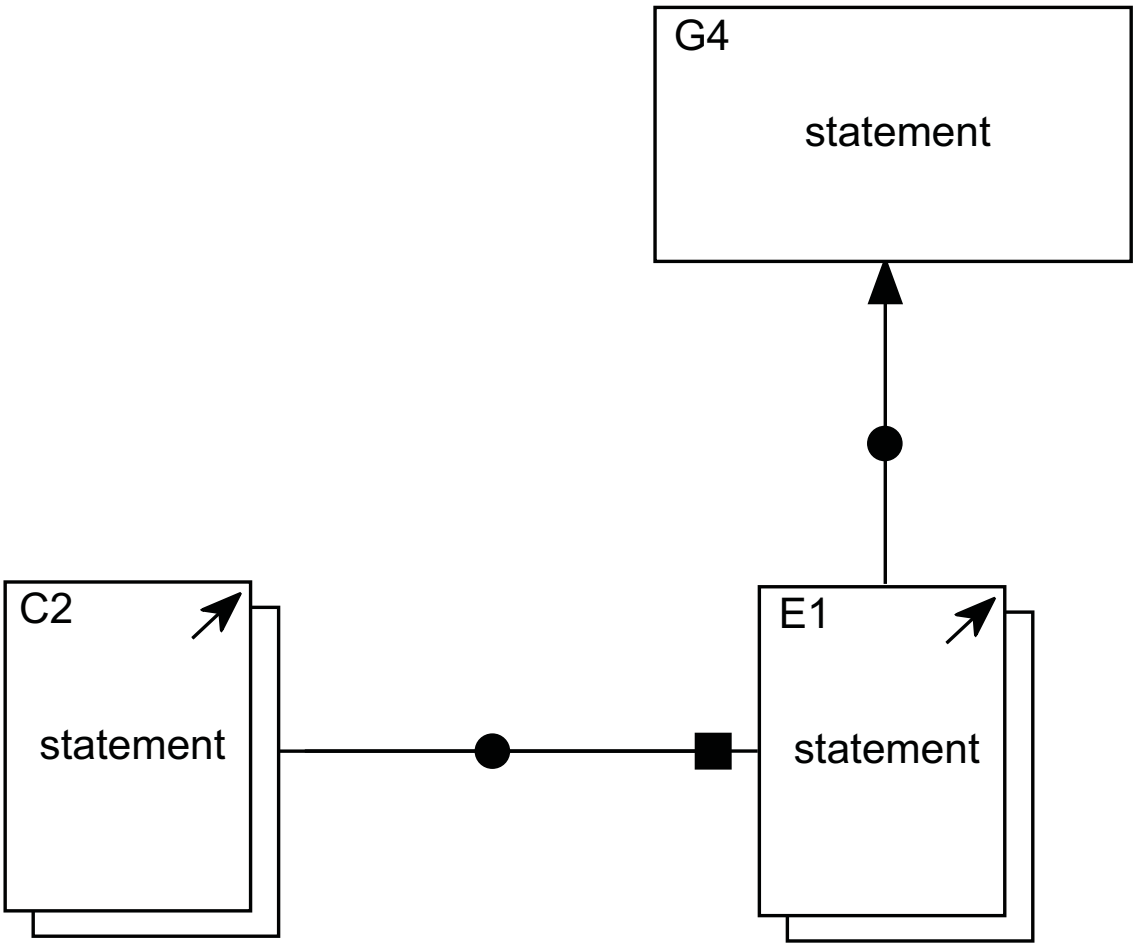

Figure D11 – Example of AssertedContext

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

(informative)

This annex shows howInterfaces and BindingPackages can be used to combine elements of separate Packages. 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).

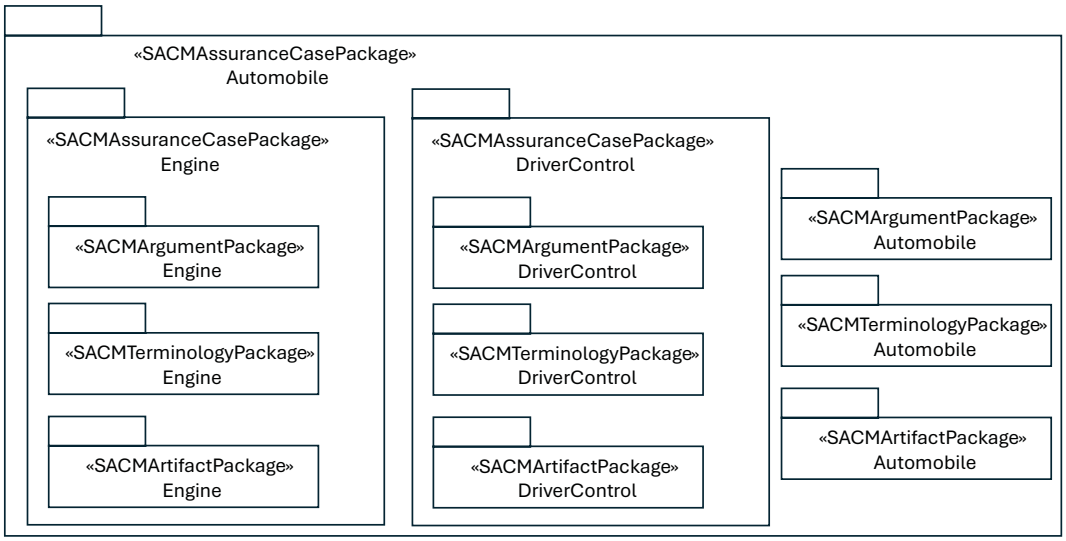

Figure E1 – Automobile Assurance Case Illustration

In Figure E1, the various top level packages that could be filled to define the complete AssuranceCase for Automotive including Engine and DriverControls are shown..

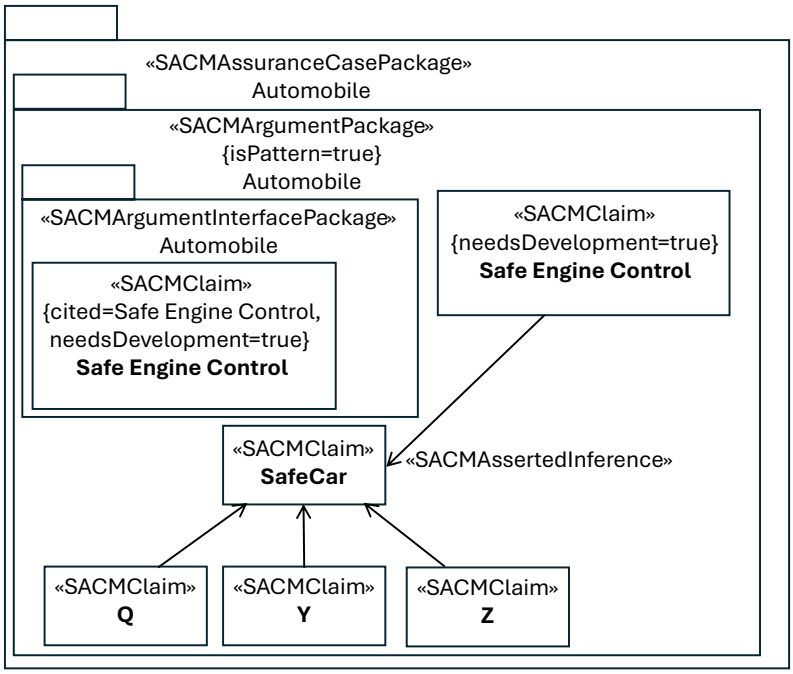

Figure E2 – Automobile Assurance Case Illustration

Figure E2 shows the overall assurance picture for Automobile.

The ArgumentPackageInterface for Automobile shows local claims, Y, Z, and Q, as well as Claim SafeEngineControl for which needsDevelopment=true. This Claim is cited in the Automobile's ArgumentInterfacePackage for use..

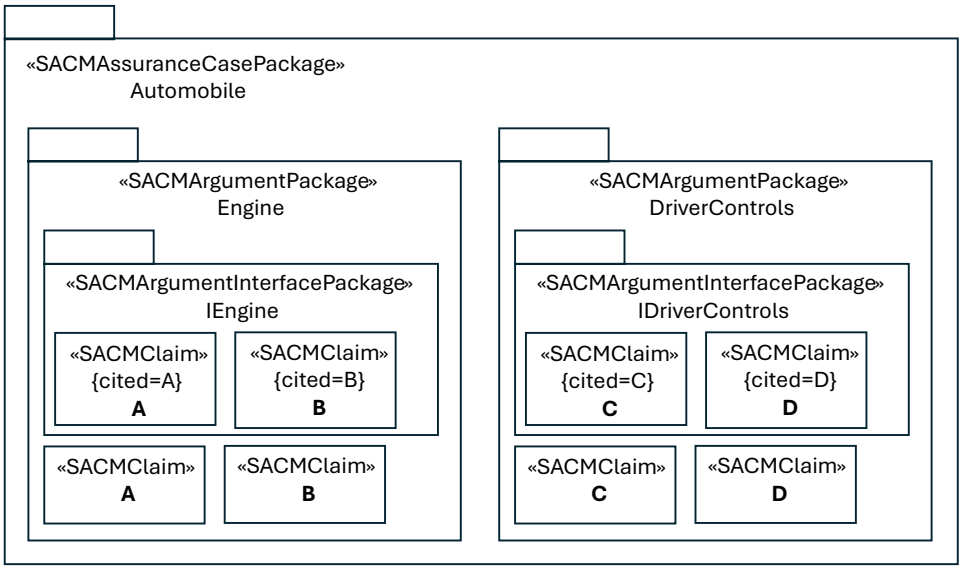

Figure E3 – Assurance Case Bindings Illustration

In Figure E3, each of the ArgumentInterfacePackage for Engine and DriverControls is shown as contained within each of the ArgumentPackage for Engine and DriverControls which are contained within AssuranceCasePackage for Automobile.

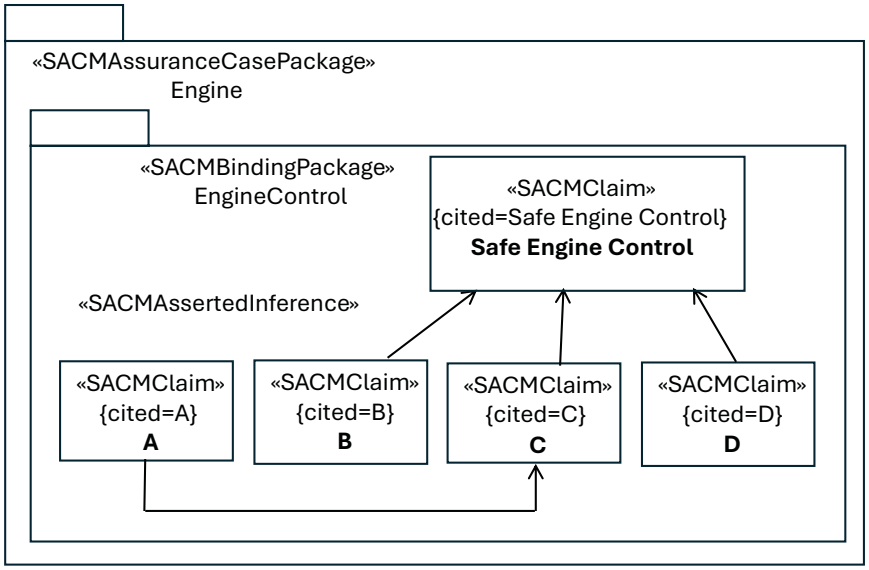

Figure E4 – Automobile Assurance Case Illustration

Figure E4 shows how the BindingPackage 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.

This page intentionally left blank.

Annex F: SACM UML profile

(normative)

The SACM UML profile is created by extending from standard UML

F.1 meta Profile section

The meta Profile section of the SACM profile contains two stereotypes that can be placed on dependencies. These two stereotypes allow for more precise definition of profiles. This technique is also used in OMG's Unified Architecture Framework (UAF), section 2.3.1 Metaconstraint dependency and the examples more fully explain the concept being used.

F.1.1 metaconstraint

A metaconstraint on a dependency defines a UML Role that can be fulfilled between two Base UML Elements that the stereotypes are applied to.

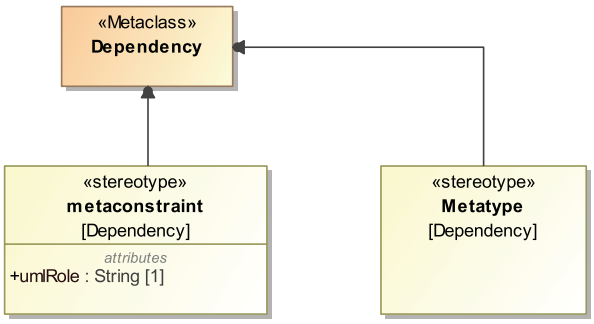

Figure F.1 – SACM metaconstraint

metaconstraint

Extends

Dependency

Attribute

umlRole : String [1]

For example, in Figure F.2, between SACMAssurancCasePackage stereotype on one Package and SACMTerminologyPackage on another Package, there may be a UML Role for "owningPackage" on one side and "packageableElement" on the other side as reflected in the UML metamodel as illustrated in this snippet from UML v. 2.5.1 in Figure F.3 below.

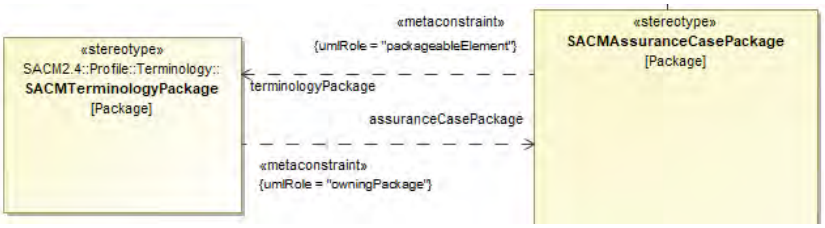

Figure F.2 – UML role in a profile

This can be reflected in a profile as shown in Figure F.3, below.

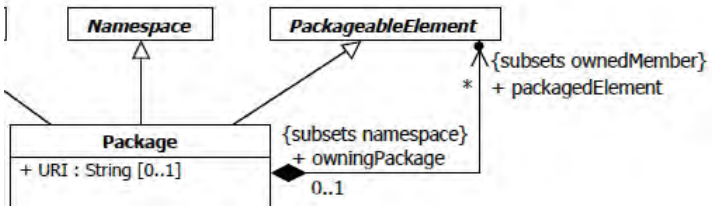

Figure F.3 – UML role snippet from UML v. 2.5.1

F.1.2 Metatype

Metatype stereotype on a dependency specifies what Metaclass should be used as a base when constructing something from the Palette of a Tool. For example, an SACMArtifact can extend any NamedElement in UML (to allow for maximum flexibility) as shown in the profile diagram shown in Figure F.4 below, but if one creates an SACMArtifact from a Palette, then the Metaclass used will be a Class (i.e., something that is concrete and not abstract). Metatype was not named "metaclass" to avoid conflicts with its other uses in the language.

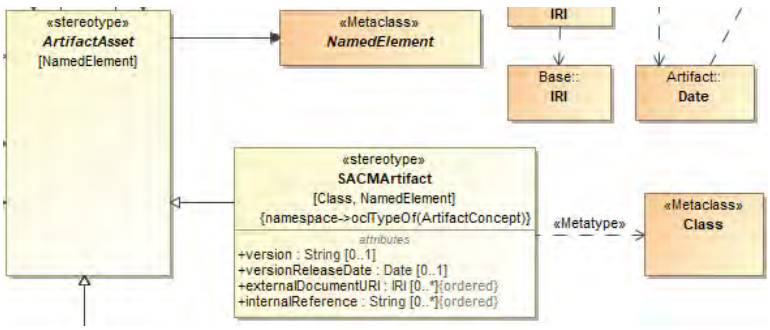

Figure F.4 – Metatype to Class

Metatype

Extends

Dependency

F.2 Base section

The Base section of the SACM profile is Shown in F.5 – F.8. This section includes the abstract SACMElement and all of its structure and ModelElement. In addition, BaseElements which can be used anywhere in AssuranceCases are defined.

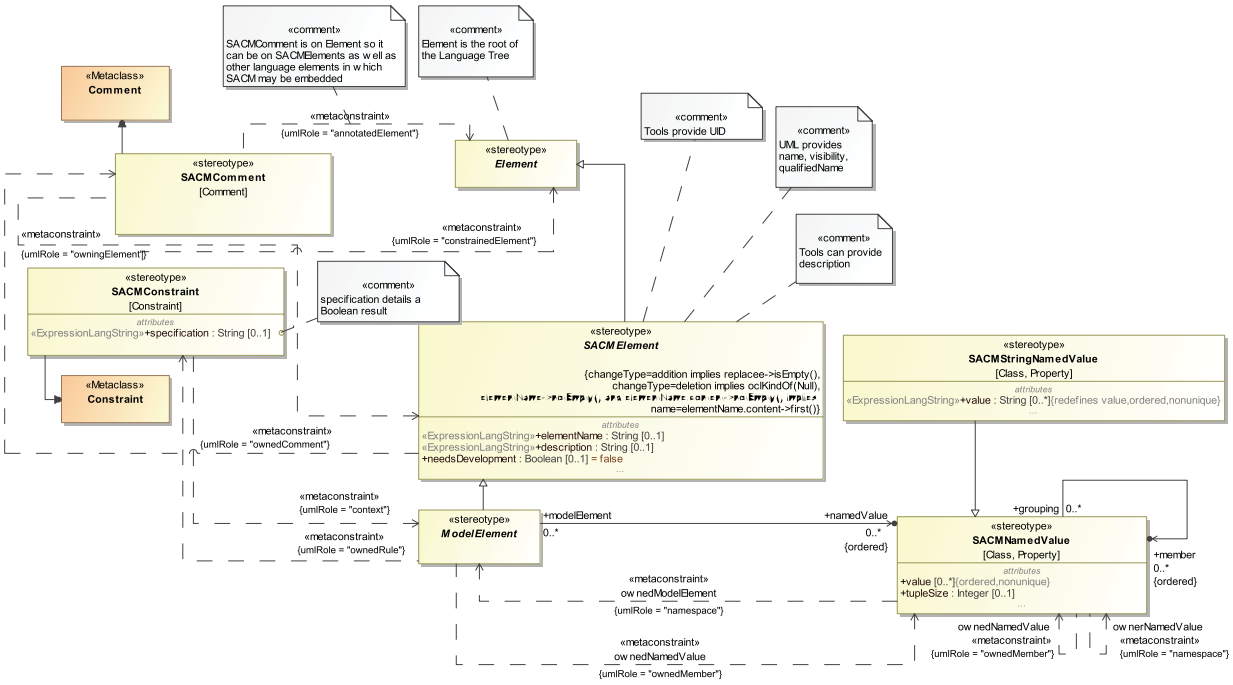

Figure F.5 – Base section of the SACM profile.

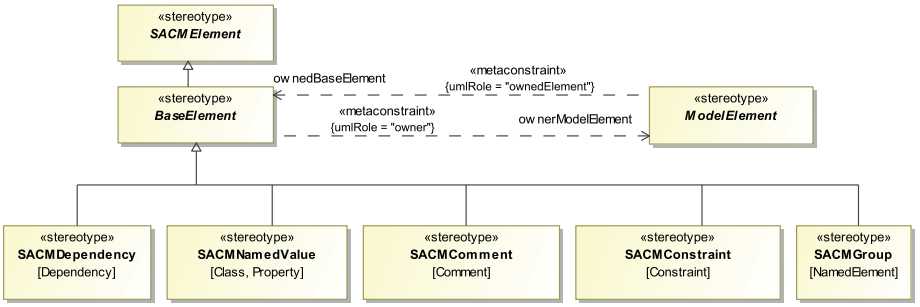

Figure F.6 – BaseElement section of the SACM profile.

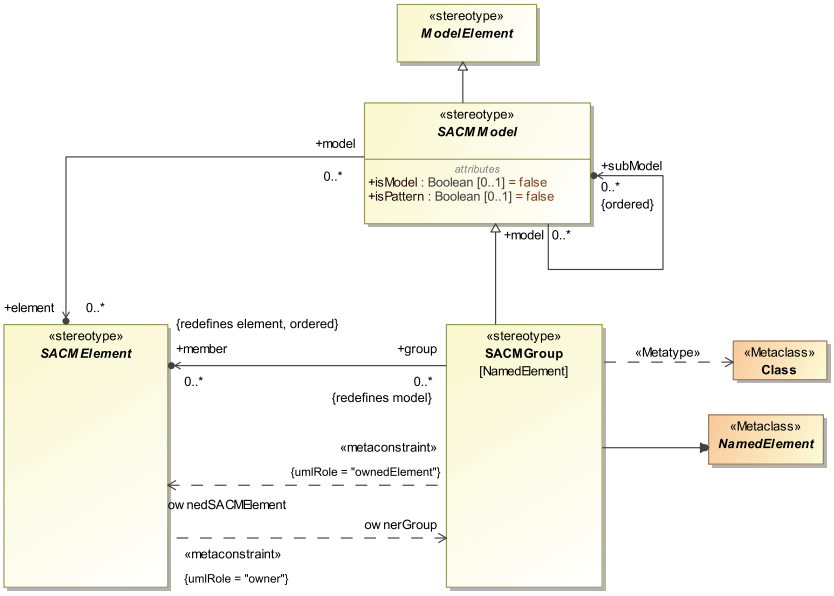

Figure F.7 – Group section of the SACM profile.

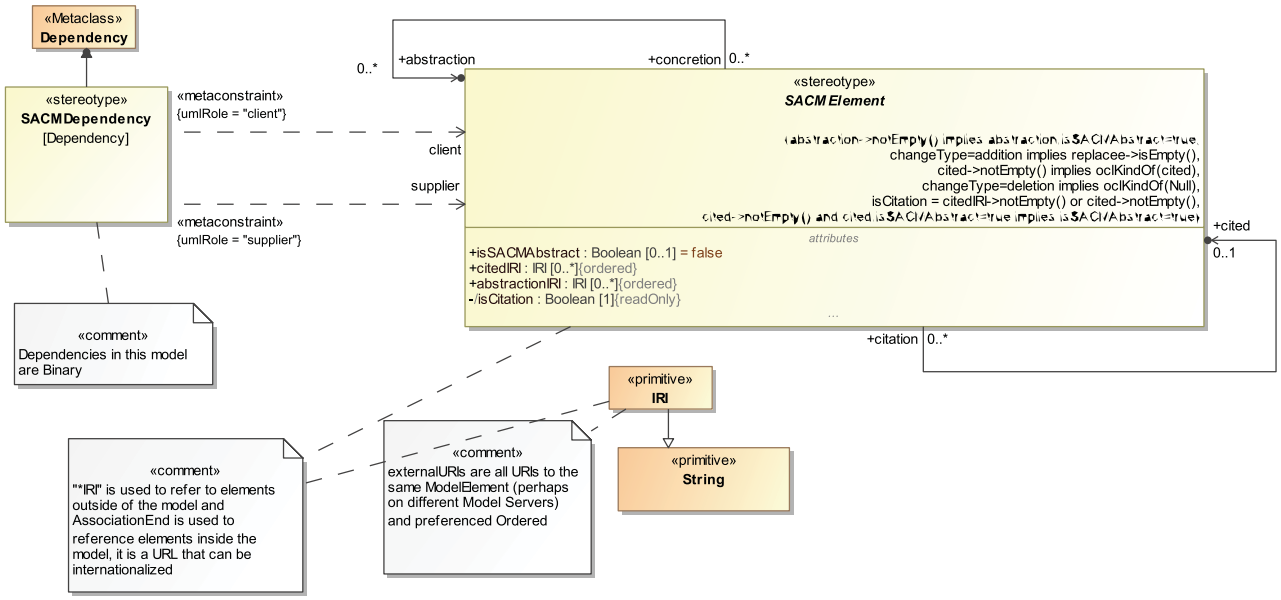

Figure F.8 – References and Dependencies section of the SACM profile.

F.3 Packaging section

The Packaging section of the SACM profile is Shown in F.9 – F.10. This section includes an abstract notion of SACMPackage as well as defining the AssuraneCasePackage and its possible ownership of other Packages.

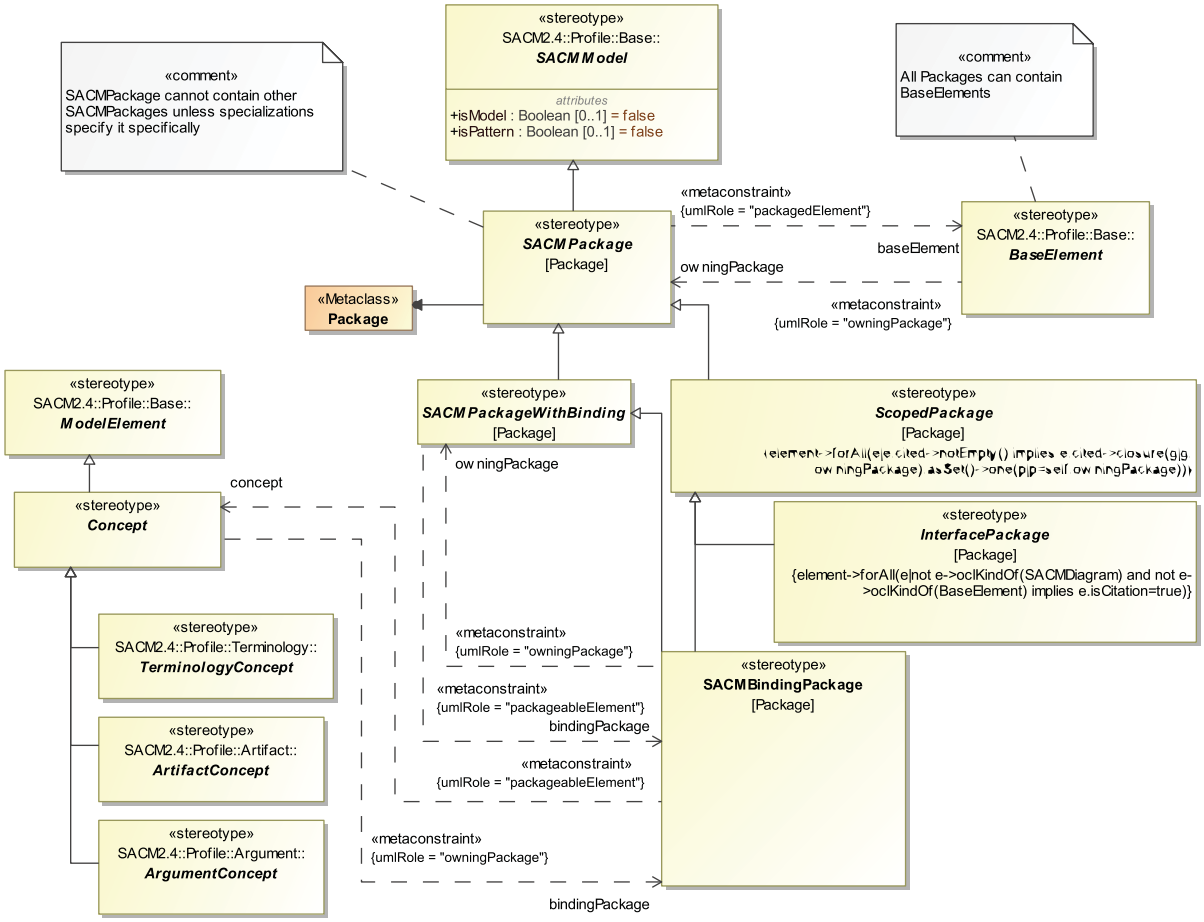

Figure F.9 – SACMPackage section of the SACM profile.

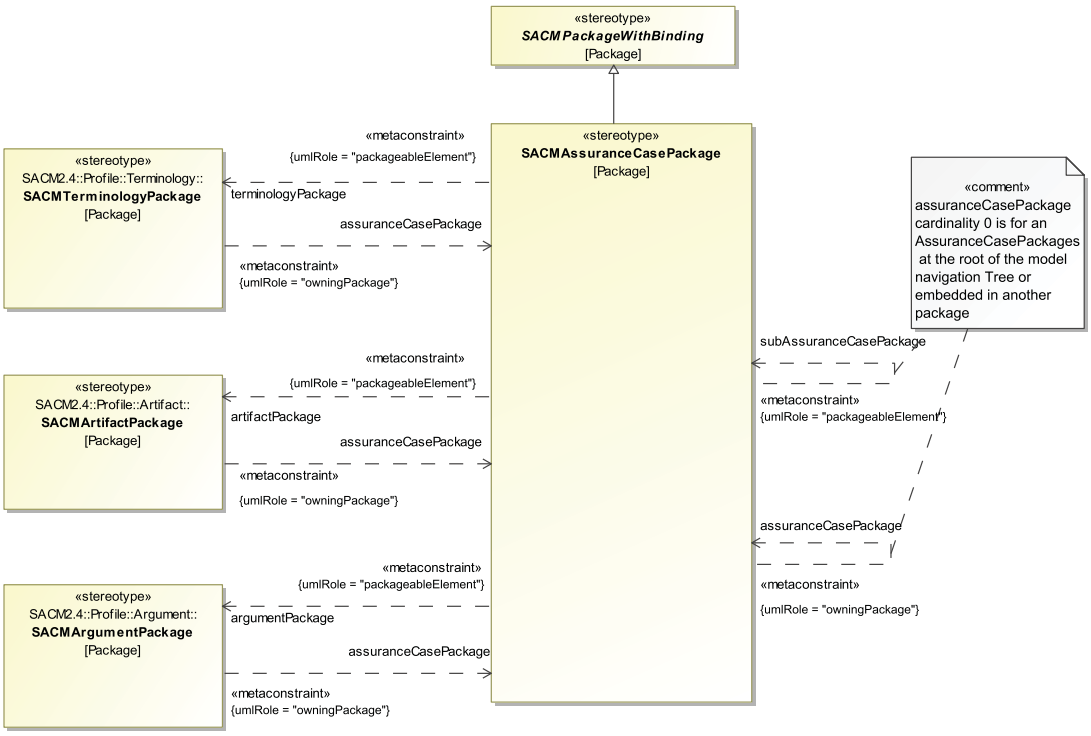

Figure F.10 – SACMAssuranceCasePackage section of the SACM profile.

F.4 Terminology section

The terminology section of the SACM profile is Shown in F.11 - F.12. This section includes the notion of a MultiLangString as well as Category, Expressions, and Terms. In Addition, Terminology Package kinds are defined.

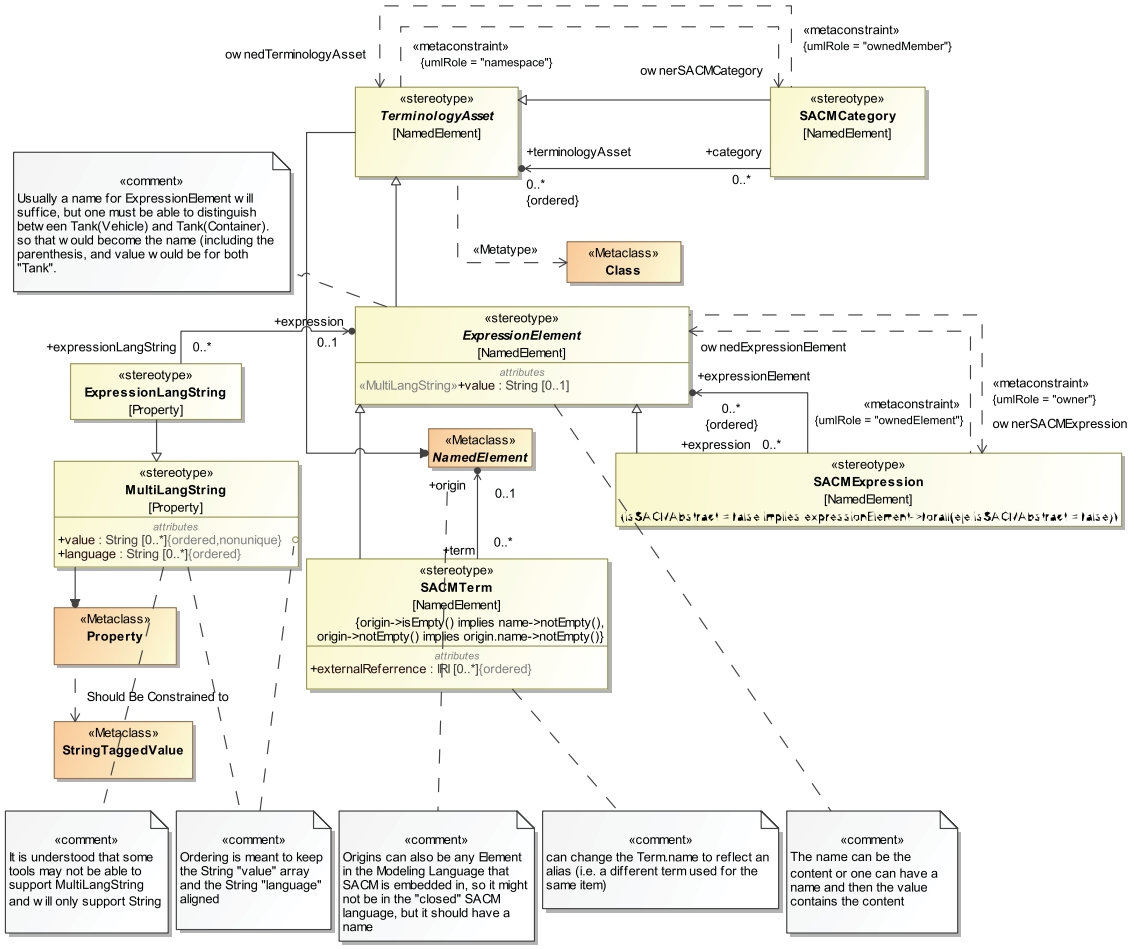

Figure F.11 – Terminology section of the SACM profile.

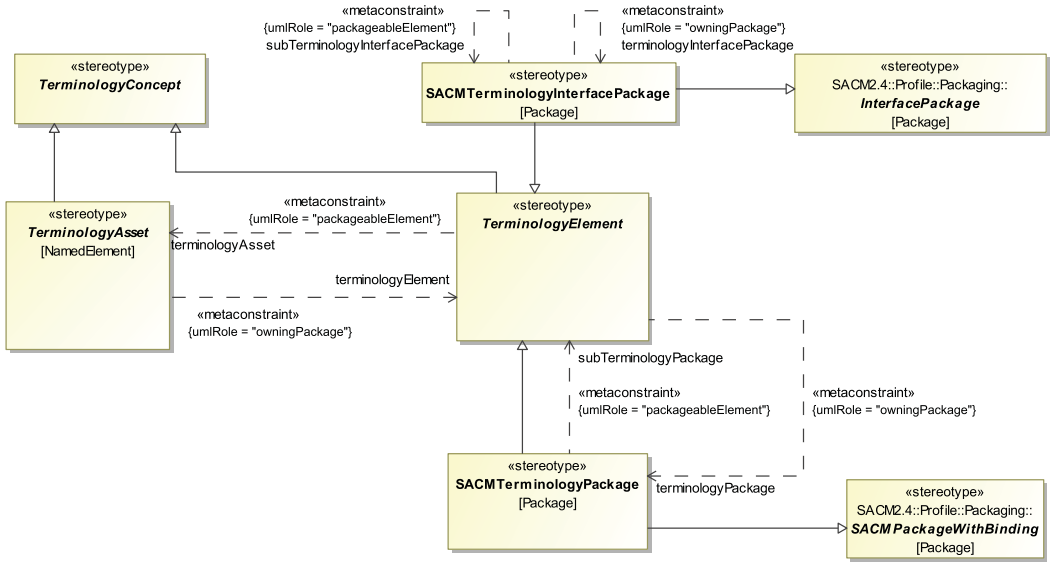

Figure F.12 – TerminologyPackage section of the SACM profile.

F.5 Argument section

The argument section of the SACM profile is Shown in F.13 - F.17. This section includes all the elements that are needed to build an argument in SACM. Included are all the Asserted Relationships used for Argumentation. In addition, Artifact Package kinds are defined.

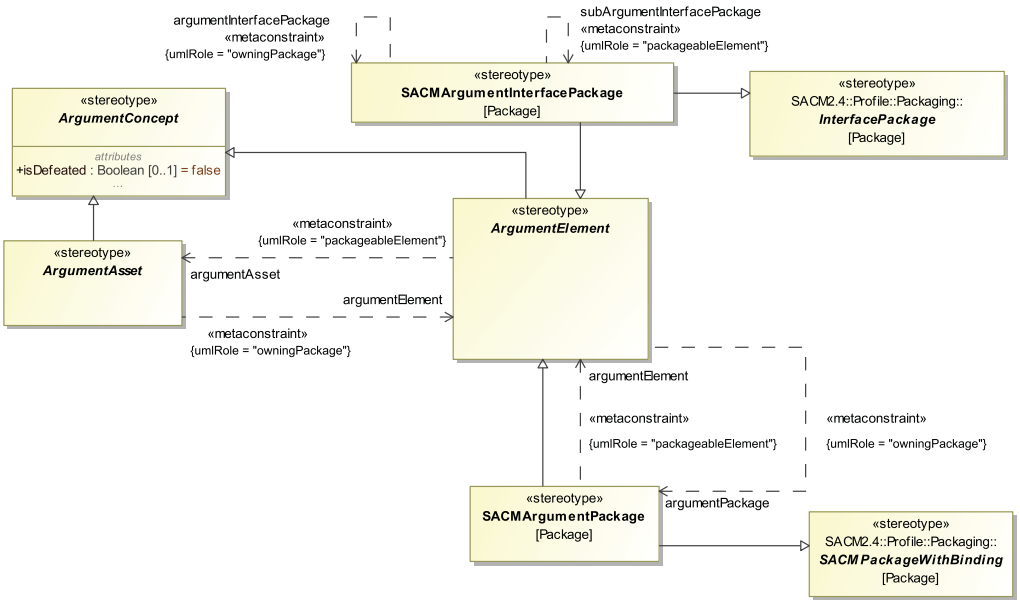

Figure F.13 – ArgumentPackages section of the SACM profile.

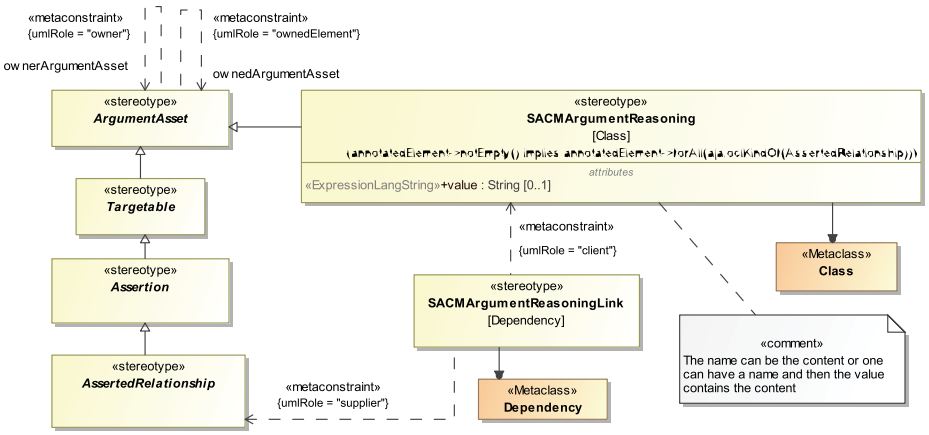

Figure F.14 – ArgumentReasoning section of the SACM profile.

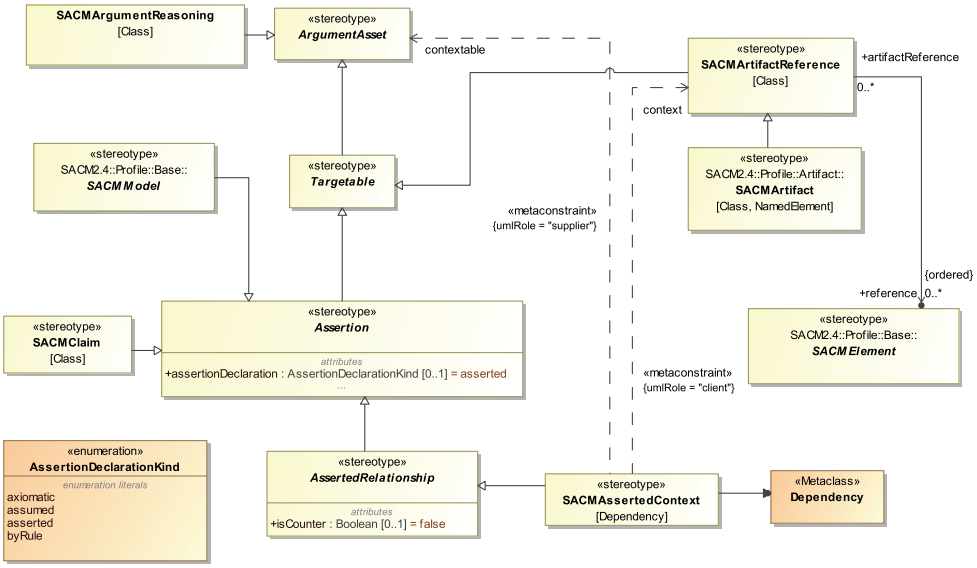

Figure F.15 – AssertedContext section of the SACM profile.

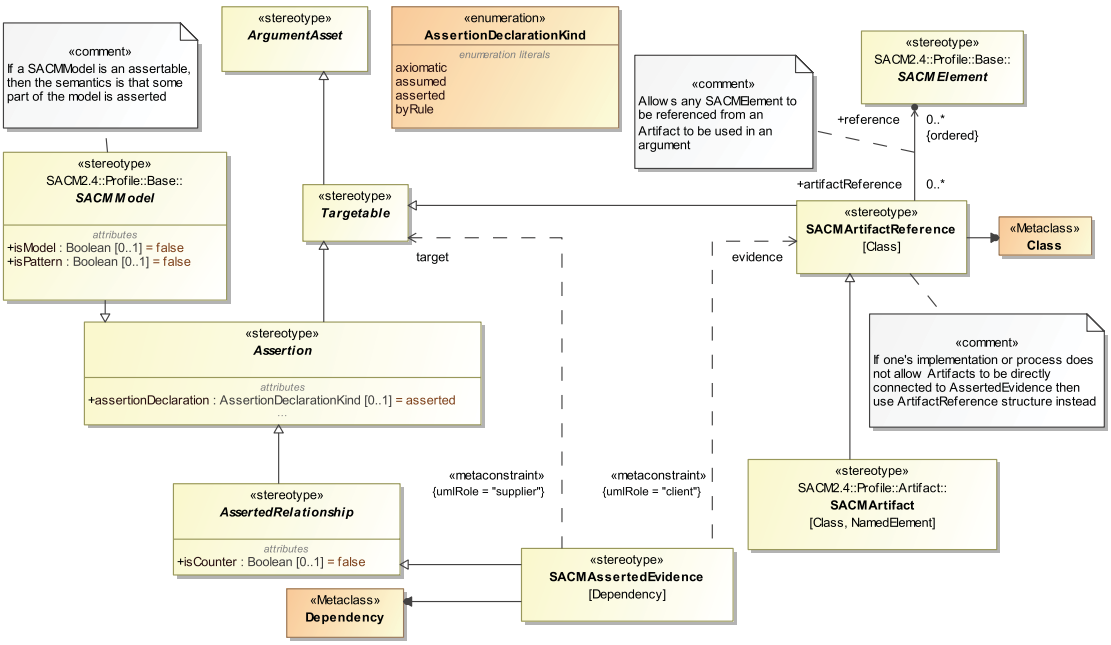

Figure F.16 – AssertedEvidence section of the SACM profile.

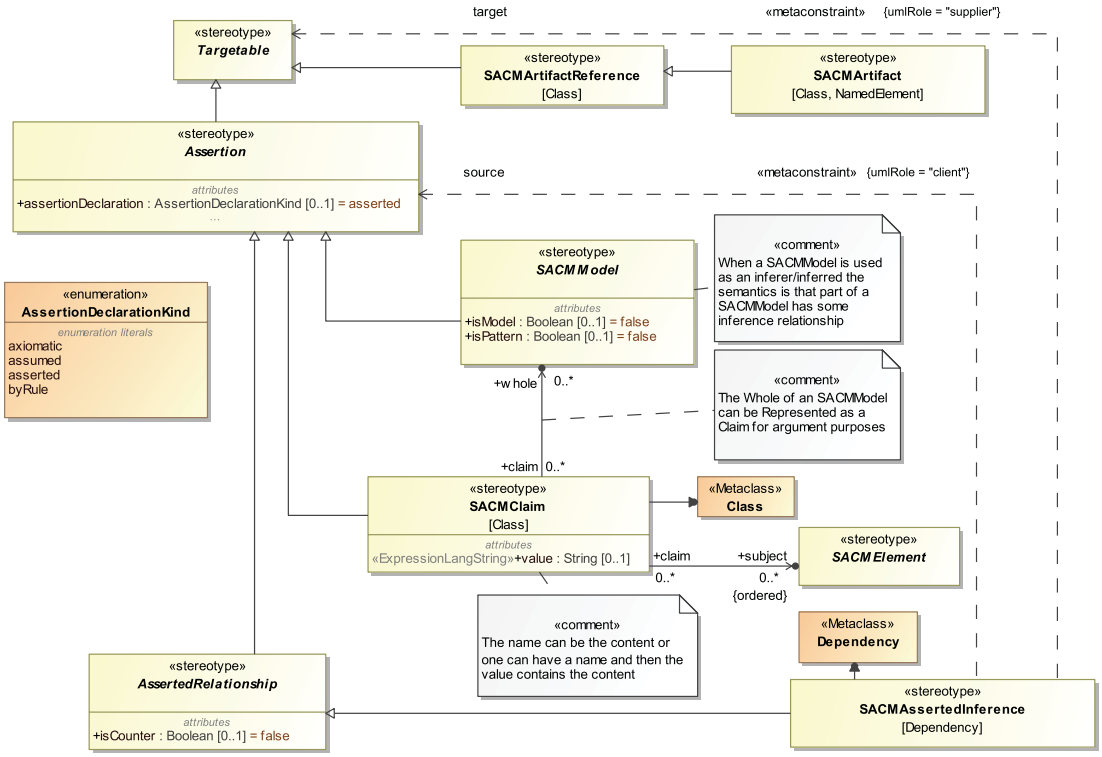

Figure F.17 – AssertedInference section of the SACM profile.

F.6 Artifact section

The artifact component of the SACM profile is Shown in F.18 - F.19. This section includes the elements needed to define the pedigree and provenance of the Artifacts used in arguments. In addition, Artifact Package kinds are defined.

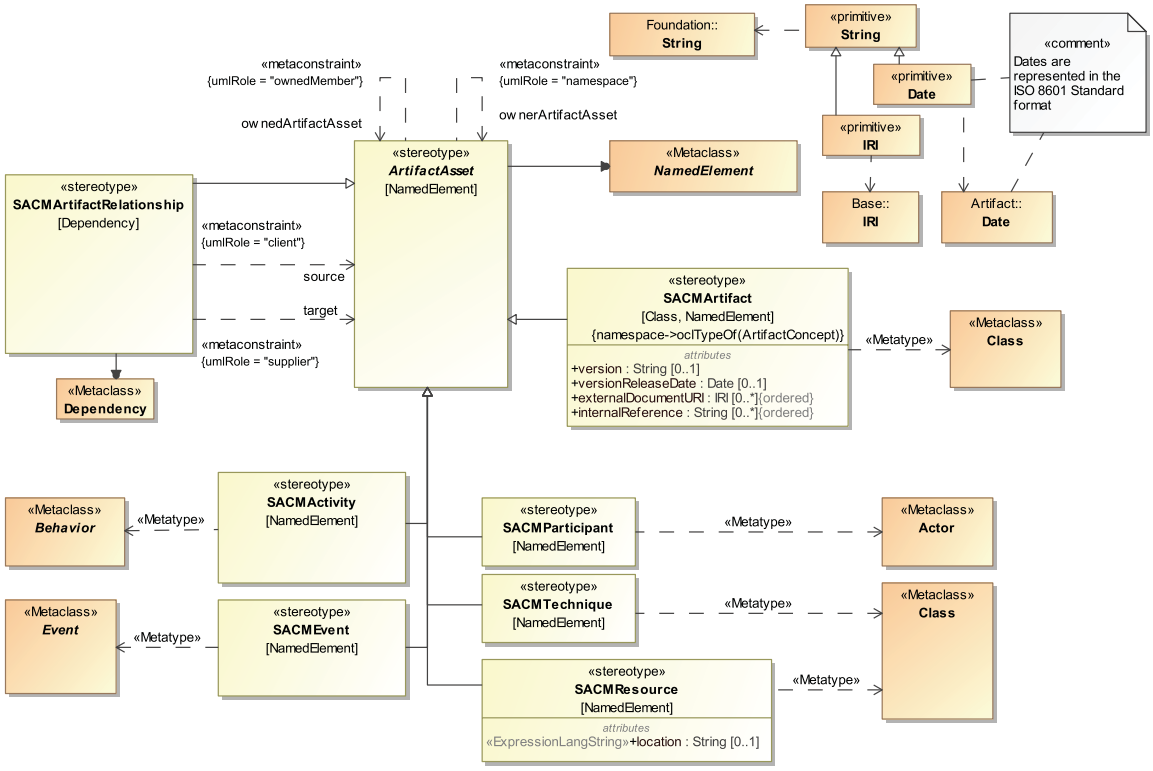

Figure F.18 – Artifact section of the SACM profile.

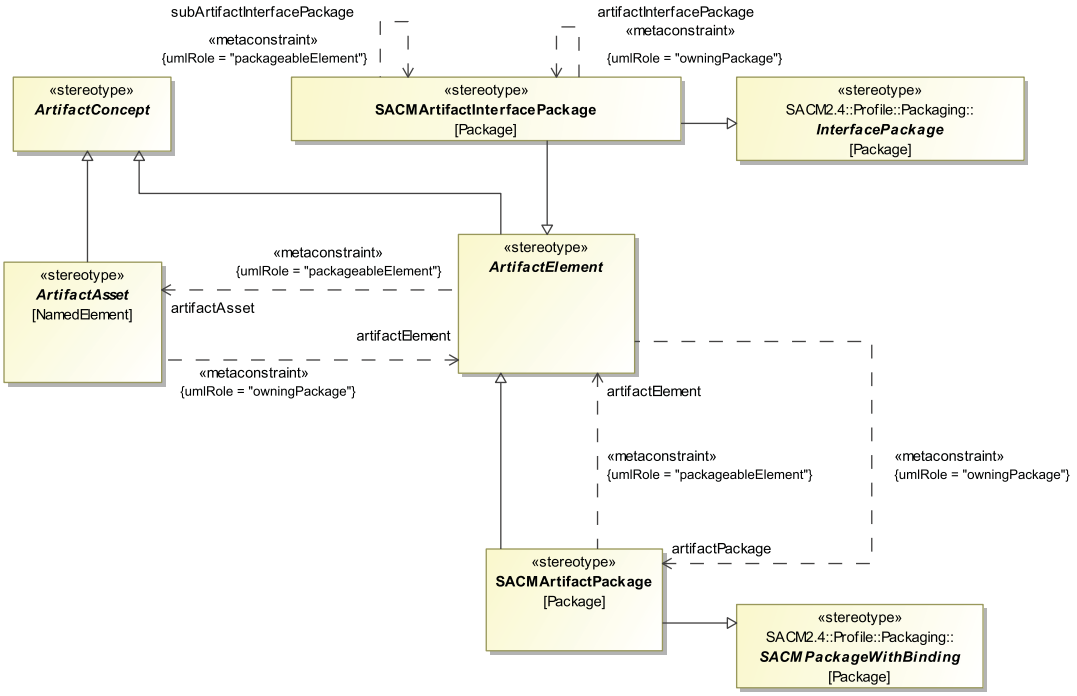

Figure F.19 – ArtifactPackage section of the SACM profile.

F.7 Advanced Argument section (infotmative)

The Non-Normative sections of the SACM profile are Shown in F.20 - F.28. This section includes all of the elements that are non-normative.

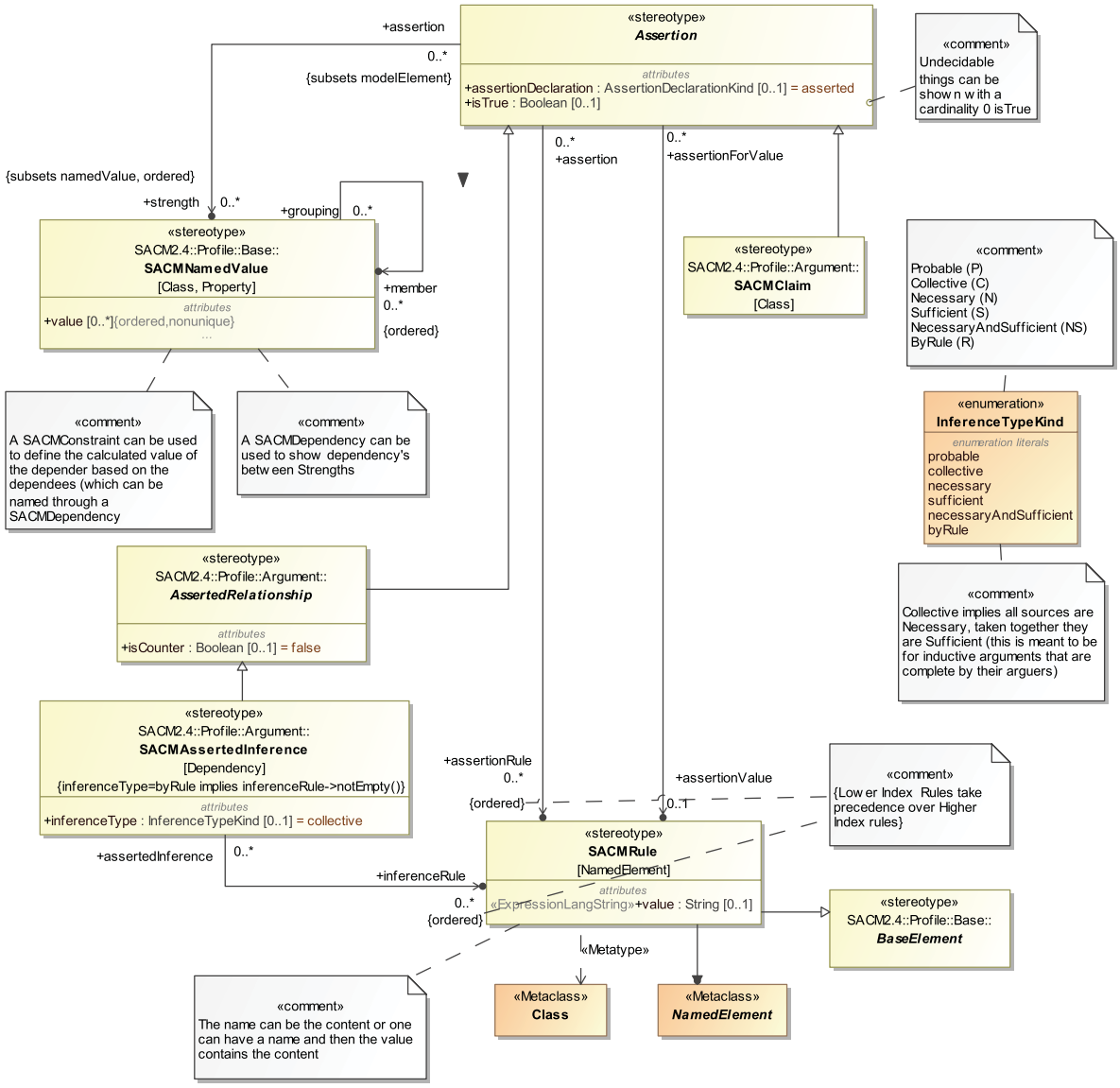

Figure F.20 – AdvancedArgument section of the SACM profile.

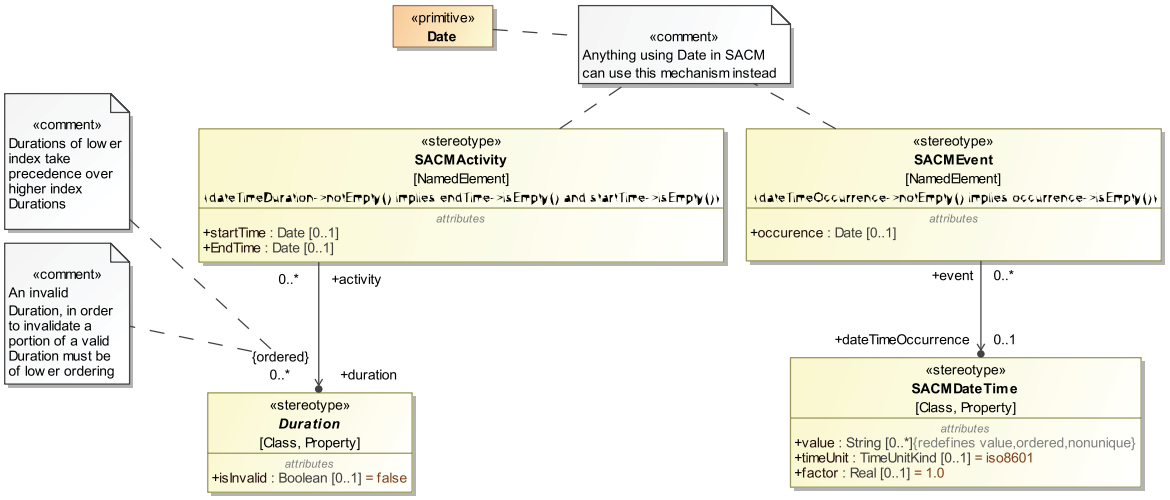

Figure F.21 – AdvancedArtifact section of the SACM profile.

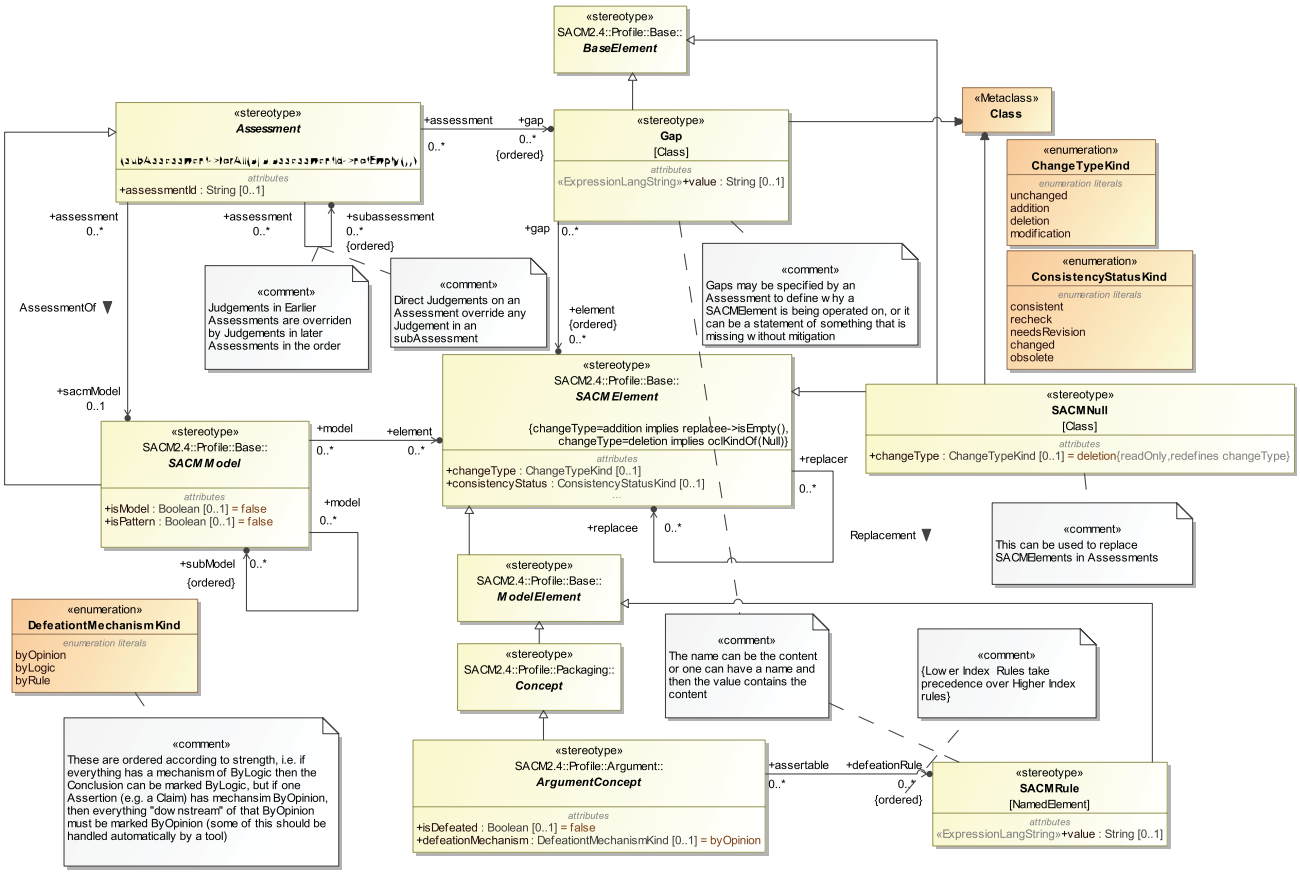

Figure F.22 – Assessment section of the SACM profile.

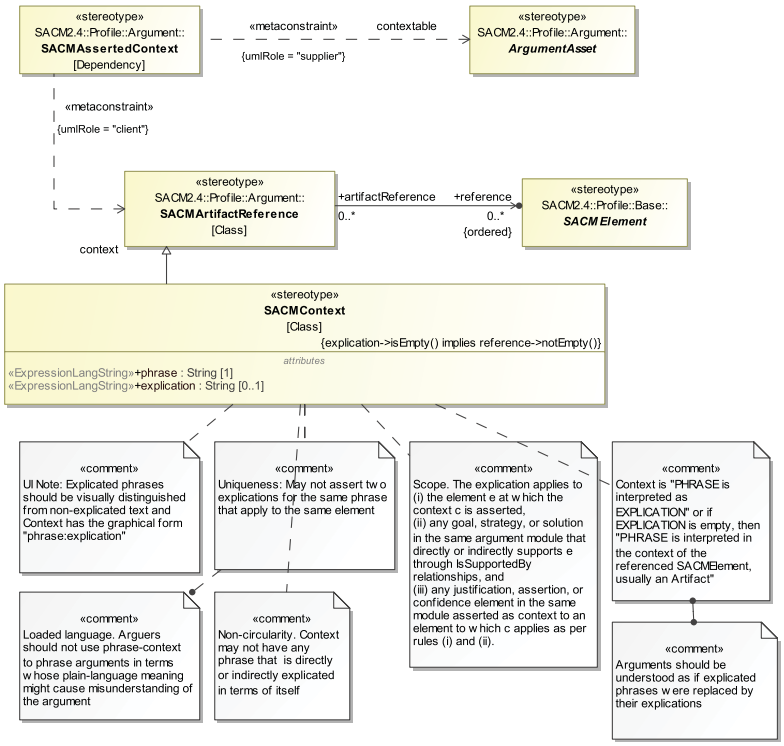

Figure F.23 – Context section of the SACM profile.

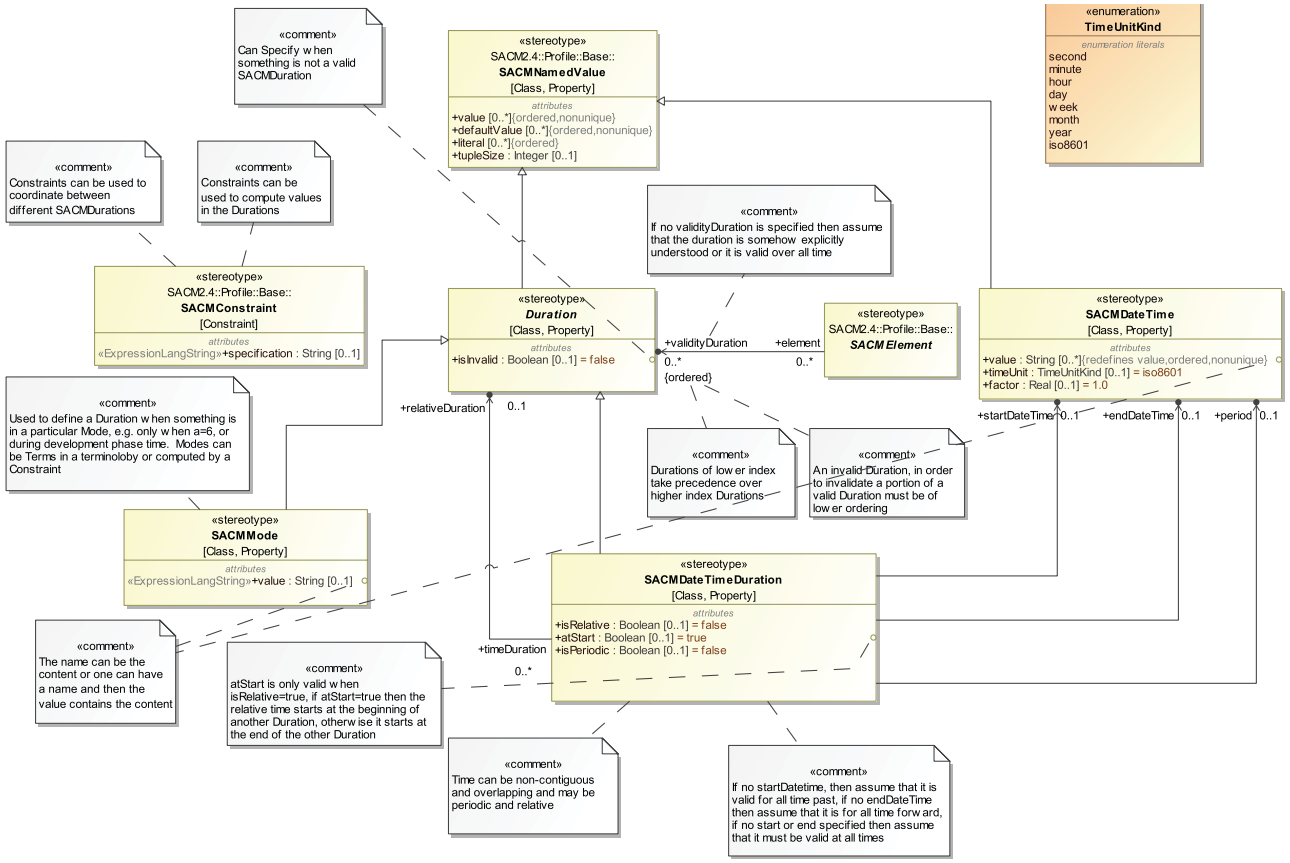

Figure F.24 – DateTime section of the SACM profile.

Figure F.25 – Join section of the SACM profile.

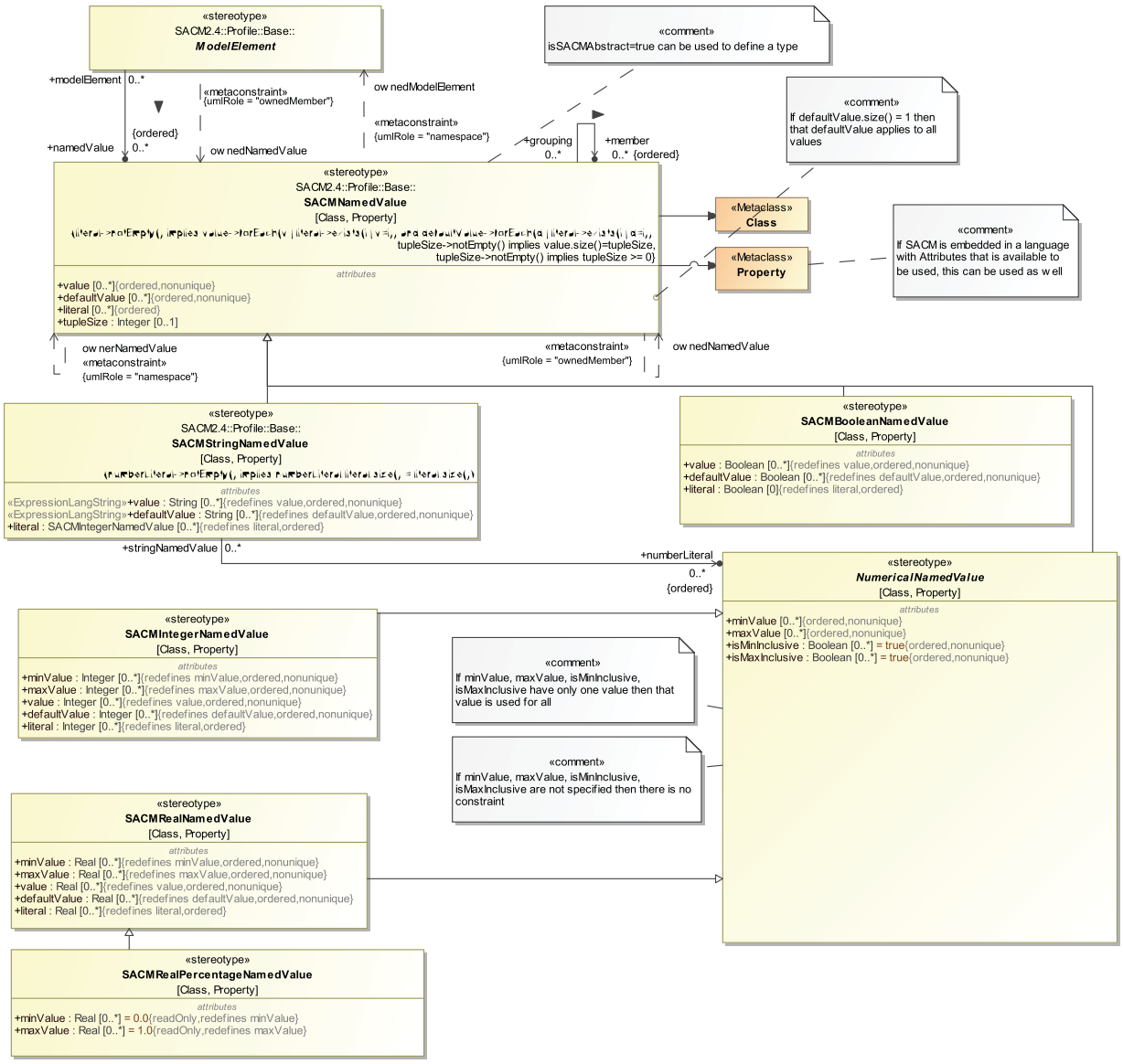

Figure F.26 – NamedValue section of the SACM profile.

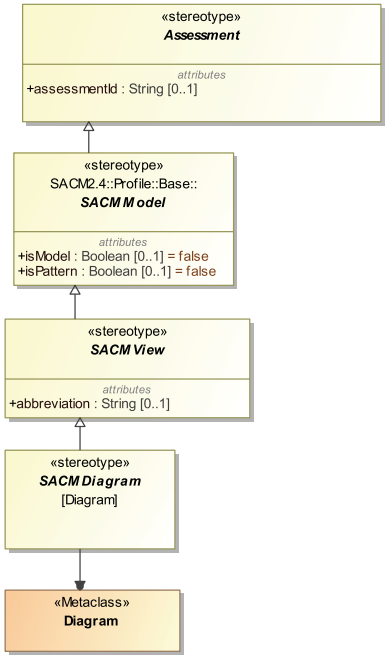

Figure F.27 – SACMDiagram section of the SACM profile.

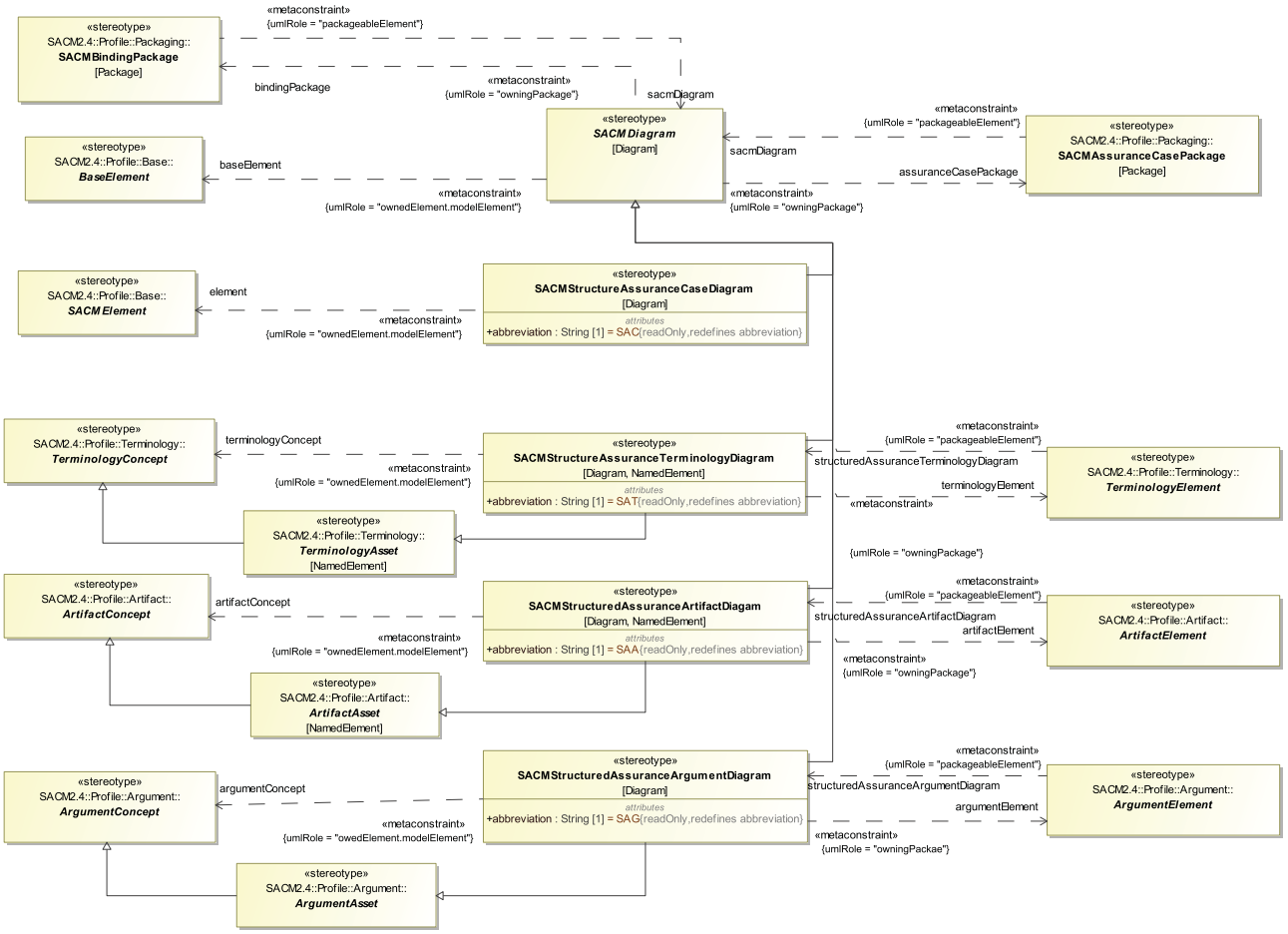

Figure F.28 – SACMDiagramTypes section of the SACM profile.

Annex G: Transformation from SACM 2.3 to SACM 2.4

(normative)

In general, the mapping of models from SACM 2.3 to SACM 2.4 is fairly straightforward and uses the same constructs. There have been some changes of attributes and AssociationEnd names to enhance consistency across the model. In addition, some concepts have been expanded to include a wider application to other elements in the model. For example, ImplementationConstraints were made into Constraints and can now apply to anything in the model..

G.1 Foundation

Foundation was added not as new elements (all of them are abstract) but as an aid to defining semantics of elements already in SACM 2.3. These semantics are nothing new, they come from UML (and KerML) concepts. These are now tied to understood elements (e.g. like UML Packages) so that SACM element meanings are not ambiguous. Include in this section is Element, DirectedRelationship, NamedElement, Namespaced, PackageableElement, and Package.

G.2 Base

1. TagValue is renamed NamedValue since Tags are already used in modeling terminology
2. NamedValues are separate objects from their SACMElements so multiple SACMElements can have reference to it
3. SACMConstraints are now separate objects, and can reference any object (including the language that SACM is embedded in)
4. needsDevelopment can be placed on any SACMElement
5. Visibility is now available on any SACMElement
6. SACMComments are now separate objects, and can reference any object (including the language that SACM is embedded in)
7. BaseElements (SACMComment, SACMConstraint, SACMDependency, NamedValue, Group) can be used with any other elements
8. Citation and Abstraction can be done through reference (to something in the local model) or IRI (to non-local model elements)
9. SACMDependency has been added to allow dependency relationships in Assurance Cases
10. The Concept of a SACMModel, to indicate whether or not something is part a SACMModel or not
11. The Concept of isPattern can be set on specialization of SACMModel, including Groups, SACMDiagrams, and SACMPackages
12. SACMModels can have sub-models defined
13. Model ownership is defined throughout the model, ModelElements can own any BaseElements
14. There is now only one Group element (rather than a Group per section), Groups can contain elements from any section

G.3 Packaging

1. Packages now have a much stricter rules on which kinds of packages can own other packages
2. AssuranceCasePackage can own subpackages of
   type: AssuranceCasePackage, ArgumentPackage, ArtifactPackage, TerminologyPackage, and BindingPackage

3. Section Packages (e.g. ArgumentPackage) can own another section package, or a section interface package (e.g. ArgumentInterfacePackage)
4. Section Interface Package can own another Section Interface Package
5. Section Packages and Interface Packages can own their Section Assets
6. Any Package can own BaseElements
7. There is only one BindingPackage rather than one for each section, and a BindingPackage can own any SACMElement

G.4 Terminology

1. MultiLangString is now the combination of LangString and MultiLangString into one Object
2. MultiLangString has been moved to the Terminology section
3. MultiLangString and ExpressionLangString are now types of content in the model (so things can now point to elements in Terminology)4
4. Categories can own Categories, Expressions, and Terms
5. Expressions can own Expressions and Term

G.5 Artifact

1. Artifact date has been changed to versionReleaseDate
2. Artifact can have a reference to an external document using externalDocumentIRI
3. Artifact can reference an internal part of a document using internalReference
4. Any ArtifactAsset can be owned by any other ArtifactAsset

G.6 Argument

1. Any ArgumentAsset can own any other ArgumentAsset
2. Artifact is a specialization of ArtifactRefrence
3. ArtifactReference references any SACMElement (sometimes an Artifact)
4. ArtifactRefence can represent evidence provided by a whole SACMElement (e.g. Package)
5. A SACMModel (Group, Package, Diagram) can be an Assertion
6. AssertionDeclaration is one of axiomatic, assumed, asserted
7. AssertedContext can tie an ArtifactReference (possibly an Artifact) to an ArgumentAsset
8. AssertedEvidence can tie an ArtifactRefernce (possibly an Artifact) to a Targetable (ArtifactReference or Assertion)
9. AssertedInference can tie an Assertion to a Targetable
10. Claim can refer to a whole SACMModel (e.g. Package)
11. Claim can define its subject(s) SACMElement through its association end subject
12. Any ArgumentConcept can be defeated in an argument
13. AssertedArtifactSupport was removed, In SACM 2.4, One can have an AssertedEvidence between one ArtifactReference and another, and therefore a separate Relationship is not necessary
14. AssertedArtifactContext was removed, In SACM 2.4, One can have an AssertedContext between one ArtifactReference and another, and therefore a separate Relationship is not necessary