Relationship of GSN to SACM

The Structured Assurance Case Meta-model (SACM) has been defined by the Object Management Group [1] to improve standardisation and interoperability for assurance cases. In this section we specify the relationships between GSN and SACM. These relationships enable translations from GSN arguments to SACM models and vice-versa. Similar translations can be defined between SACM and other argumentation approaches, thus facilitating translation between notations.

Figure 1 defines the relationship of GSN elements to elements of the SACM Argumentation Package. These relationships are also described below. A more formal description of the relationships between GSN and SACM elements is specified using the Epsilon Transformation Language [2] [3] in Annex A.

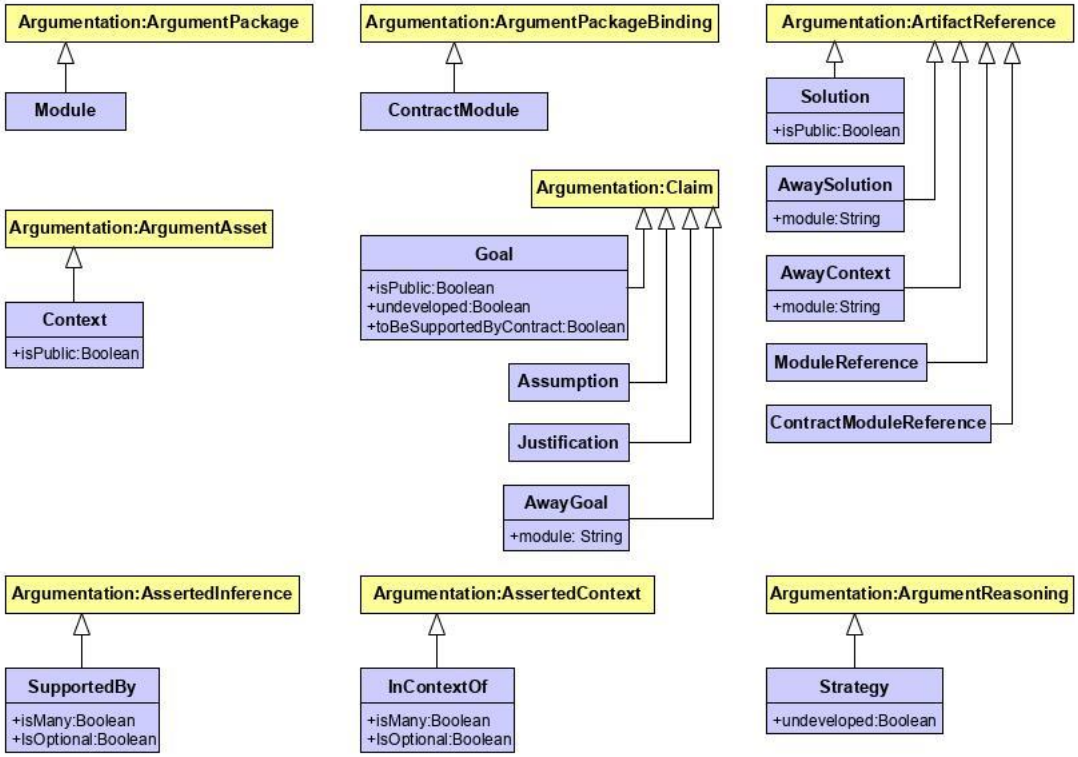

Figure 1: Relationship of GSN elements to elements of SACM

GSN element *Goal* extends SACM's *Claim*. In addition to the attributes of *Claim*, *Goal* has three additional attributes:

- +isPublic - indicating if the *Goal* is public (meaning it can be referred to by *AwayGoal*s residing in *Module*s other than the *Module* in which the *Goal* resides)
- +undeveloped - indicating if the *Goal* is undeveloped
- +toBeSupportedByContract indicating if the *Goal* requires support by a *ContractModule*.

GSN element *Assumption* extends SACM's Claim. Specifically, a GSN *Assumption* transforms to a SACM *Claim* with its +assertionDeclaration set to 'assumed'.

GSN element *Justification* extends SACM's Claim. Specifically, a GSN *Justification* transforms to a SACM *Claim* with its +assertionDeclaration set to 'axiomatic'.

GSN element *Context* extends SACM's *ArugmentAsset*. In GSN, a *Context* can have two functions:

- explanatory – providing only a description stating the context of an element. Explanatory context would be represented in SACM as an Axiomatic Claim.
- referential – referring to an engineering artifact for contextual information. Referential context would be represented in SACM using an ArtefactReference.

Context has an additional attribute:

- +isPublic - indicating if the *Context* is public (meaning it can be referred to by *AwayContext* references from *Module*s other than the *Module* in which the *Context* resides)

GSN element *Solution* extends SACM's ArtifactReference. *ArtifactReference* refers to an SACM *Artifact* residing in an *ArtifactPackage* (N.B. an SACM-compliant GSN model has access to all other SACM packages, i.e. Artifact, Terminology, Base and AssuranceCase). *Solution* has an additional attribute:

- +isPublic - indicating if the *Solution* is public (meaning it can be referred to by *AwaySolution* references from *Module*s other than the *Module* in which the *Solution* resides)

GSN element *SupportedBy* extends SACM's *AssertedInference*. In addition to the attributes of *AssertedInference*, *SupportedBy* defines two additional attributes:

- +isMany – used for GSN patterns to define multiplicity relationships between GSN elements.
- +isOptional – used for GSN patterns to define optional relationships between GSN elements.

N.B. the +source and +target attributes of GSN *SupportedBy* and SACM *AssertedInference* are reversed. This reflects the fact that the semantics of *SupportedBy* are that 'A is supported by B' (from A to B), and the semantics of *AssertedInference* are that 'B infers A' (from B to A).

GSN element *InContextOf* extends SACM's *AssertedContext*. In addition to the attributes of *AssertedContext*, *InContextOf* defines two additional attributes:

- +isMany – used for GSN patterns to define multiplicity relationships between GSN elements.
- +isOptional – used for GSN patterns to define optional relationships between GSN elements.

N.B. the +source and +target attributes of GSN *InContextOf* and SACM *AssertedContext* are reversed. This reflects the fact that the semantics of *InContextOf* are that 'A is in the context of B' (from A to B), and the semantics of *AssertedContext* are that 'B provides context for A' (from B to A).

GSN element *Stragety* extends SACM's *ArgumentReasoning*. An *ArgumentReasoning* is a node associated to *AssertedRelationship*s, to provide reasons for the *AssertedRelationship*s. Unlike *Strategy*, *ArgumentReasoning* does not connect directly to *Claim*s and *ArtifactReference*s, special care is needed when transforming GSN from SACM. *Strategy* has an additional attribute: +undeveloped indicating whether it needs further development.

GSN element *Module* extends SACM's *ArgumentPackage*. Both of these contain argumentation elements within them.

GSN element *AwayGoal* extends SACM's Claim. In SACM, *Claim*s can 'cite' other *Claim*s, enabling the referential function of *AwayGoal*. *AwayGoal* defines an additional attribute:

- +module – used to specify the module in which the goal being referenced resides.

GSN elements *AwaySolution*, *AwayContext*, *ModuleReference* and *ContractModuleReference* all extend SACM's *ArtifactReference*, as they *refer* to other elements. *AwaySolution* and *AwayContext* both define an additional attribute:

- +module – used to specify the module in which the element being referenced resides.

GSN element *ContractModule* extends SACM's *ArgumentPackageBinding*. In GSN the binding of packages is done using *Away elements*. In SACM the binding of packages is done through reference to elements provided in *ArgumentPackageInterface*s. The semantics of GSN *ContractModules* and SACM *ArgumentPackageBinding* are however equivalent.

References

[1] OMG, 2020. Structured Assurance Case Metamodel (SACM) (formal/20-04-01). https://www.omg.org/spec/SACM/2.1/PDF
[2] The Epsilon Transformation Language (ETL), https://www.eclipse.org/epsilon/doc/etl/
[3] Kolovos, D.S., Paige, R.F. and Polack, F.A., 2008, July. The epsilon transformation language. In International Conference on Theory and Practice of Model Transformations (pp. 46-60). Springer, Berlin, Heidelberg.

Annex A: ETL Translation between GSN and SACM models

This annex provides a description of the translations between GSN models and SACM models using the Epsilon Translation Language (ETL) [2] [3].

```
pre {
    "Transformation Begins...".println();
    var aeToDiscard = new Sequence;
}

post {
    "Transformation is Completed.".println();
    for(ae in aeToDiscard) {
        delete ae;
    }
}

@lazy
//transforms LangString to LangString
rule LangString2LangString
    transform l1: GSN!LangString
    to l2: SACM!LangString {
        "Langstring: it's working...".println();
        l2.lang = l1.lang;
        l2.content = l1.content;
}

@lazy
//transforms from MultiLangString to MultiLangString
rule MultiLantString2MultiLangString
    transform m1: GSN!MultiLangString
    to m2: SACM!MultiLangString {
        m2.value.addAll(m1.value.equivalent());
}


@lazy
//transforms from ExpressionLangString to ExpressionLangString
rule ExpressionLangString2ExpressionLangString
    transform els1: GSN!ExpressionLangString
    to els2: SACM!ExpressionLangString {
        "Expression: it's working...".println();
        els2.lang = els1.lang;
        els2.content = els1.content;
```

```
        if(els1.expression.isDefined()) {
            els2.expression = els1.expression.equivalent();
        }
}


@lazy
//transforms from Expression to Expression
rule Expression2Expression
    transform e1: GSN!Expression
    to e2: SACM!Expression {
        e2.value = e1.value;
        e2.element.addAll(e1.element.equivalent());
}


@lazy
//transforms from Term to Term
rule Term2Term
    transform t1: GSN!Term
    to t2: SACM!Term {
        t2.value = t1.value;
        t2.externalReference = t1.externalReference;
        t2.origin = t1.origin.equivalent();
}

@lazy
//transforms Description to Description
rule Description2Description
    transform d1: GSN!Description
    to d2: SACM!Description {
        d2.content = d1.content.equivalent();
}

//rule to transform module to ArgumentPackage
rule Module2ArgumentPackage
    transform m:GSN!Module
    to ap: SACM!ArgumentPackage{
        //transform name
        ap.name = m.name.equivalent();
        ap.description = m.description.equivalent();
        //transform argumentationElement
        ap.argumentationElement.addAll(m.argumentationElement.equivalent());
}
```

```
rule Contract2ArgumentPackageBinding
    transform c: GSN!ContractModule
    to  apb: SACM!ArgumentPackageBinding {
        apb.name = c.name.equivalent();
        apb.description = c.description.equivalent();
        //transform argumentationElement
        apb.argumentationElement.addAll(c.argumentationElement.equivalent());
}

@lazy
//transforms Goal to Claim
rule Goal2Claim
    transform g: GSN!Goal
    to c: SACM!Claim {

        g.name.equivalent().println();
        c.name = g.name.equivalent().println();

        c.description = g.description.equivalent();

        //normal goal
        if(g.undeveloped = false and g.toBeSupportedByContract = false and
g.uninstantiated = false) {
            c.assertionDeclaration = SACM!AssertionDeclaration#asserted;
        }
        //Undeveloped Goal
        else if(g.undeveloped = true and g.toBeSupportedByContract = false
and g.uninstantiated = false) {
            c.assertionDeclaration = SACM!AssertionDeclaration#needsSupport;
        }
        //Uninstantiated Goal
        else if(g.undeveloped = false and g.toBeSupportedByContract = false
and g.uninstantiated = true) {
            c.isAbstract = true;
        }
        //ToBeSupportedByContract Goal
        else if(g.undeveloped = false and g.toBeSupportedByContract = true
and g.uninstantiated = false) {
            c.assertionDeclaration = SACM!AssertionDeclaration#needsSupport;
            c.taggedValue.add(createTaggedValue("toBeSupportedByContract"));
        }
        //to be supported and uninstantiated
        else if(g.undeveloped = false and g.toBeSupportedByContract = true
```

```
and g.uninstantiated = true) {
            c.isAbstract = true;
            c.assertionDeclaration = SACM!AssertionDeclaration#needsSupport;
            c.taggedValue.add(createTaggedValue("toBeSupportedByContract"));
        }

        //if public then add TaggedValue
        if(g.isPublic = true) {
            c.taggedValue.add(createTaggedValue("public"));
        }

        ("GSN Goal: " + g.name.content + " is transformed.").println();
}

@lazy
rule Justification2Claim
    transform j: GSN!Justification
    to c: SACM!Claim {
        if(j.uninstantiated = true) {
            c.isAbstract = true;
        }
        c.assertionDeclaration = SACM!AssertionDeclaration#axiomatic;
        c.name = j.name.equivalent();
        c.description = j.description.equivalent();
}

@lazy
rule Assumption2Claim
    transform a: GSN!Assumption
    to c: SACM!Claim {
        if(a.uninstantiated = true) {
            c.isAbstract = true;
        }
        c.assertionDeclaration = SACM!AssertionDeclaration#assumed;
        c.name = a.name.equivalent();
        c.description = a.description.equivalent();
}

@lazy
//transforms from Solution to ArtifactReference
rule Solution2ArtifactReference
    transform s: GSN!Solution
    to ar: SACM!ArtifactReference {
        ar.name = s.name.equivalent();
```

```
        ar.description = s.description.equivalent();

        if(s.uninstantiated = true) {
            ar.isAbstract = true;
        }

        if(s.isPublic = true) {
            ar.taggedValue.add(createTaggedValue("public"));
        }
}

@lazy
rule Context2Claim
    transform context: GSN!Context
    to claim: SACM!Claim {
        guard: context.refersToExternalMaterial = false or
context.refersToExternalMaterial.isUndefined()
        if(context.uninstantiated = true) {
            claim.isAbstract = true;
        }

        claim.name = context.name.equivalent();
        claim.description = context.description.equivalent();

        if(context.isPublic = true) {
            claim.taggedValue.add(createTaggedValue("public"));
        }
}

@lazy
rule Context2ArtifactReference
    transform c: GSN!Context
    to ar: SACM!ArtifactReference {
        guard: c.refersToExternalMaterial = true

        if(c.uninstantiated = true) {
            ar.isAbstract = true;
        }
        ar.name = c.name.equivalent();
        ar.description = c.description.equivalent();

        if(c.isPublic = true) {
            aec.taggedValue.add(createTaggedValue("public"));
        }
```

```
}

@lazy
rule Strategy2ArgumentReasoning
    transform s: GSN!Strategy
    to ai: SACM!AssertedInference, ae: SACM!AssertedEvidence, ar:
SACM!ArgumentReasoning {
        var upperLevel = getSupportedByForStrategyAsTarget(s);
        var lowerLevel = getSupportedByForStrategyAsSource(s);

        if(s.uninstantiated = true) {
            ar.isAbstract = true;
        }
        ar.name = s.name.equivalent();
        ar.description = s.description.equivalent();

        var supportedByConnectingSolutions: Collection =
lowerLevel.select(sb|sb.target.exists(t|t.isTypeOf(GSN!Solution)));
        if(supportedByConnectingSolutions.isEmpty()) {
            aeToDiscard.add(ae);
        }
        else {

    ae.source.addAll(supportedByConnectingSolutions.target.flatten().asSet().
equivalent());

    ae.target.addAll(upperLevel.source.flatten().asSet().equivalent());
            ae.reasoning = ar;
        }
        var suppoortedByConnectingGoals: Collection =
lowerLevel.select(sb|sb.target.exists(t|t.isTypeOf(GSN!Goal)));
        if(suppoortedByConnectingGoals.isEmpty()) {
            aeToDiscard.add(ai);
        }
        else {

    ai.source.addAll(lowerLevel.target.flatten().asSet().equivalent());

    ai.target.addAll(upperLevel.source.flatten().asSet().equivalent());
            ai.reasoning = ar;
        }
}

@lazy
```

```
rule SupportedBy2AssertedEvidence
    transform sb: GSN!SupportedBy
    to ae: SACM!AssertedEvidence {
        guard : sb.source.forAll(s|s.isTypeOf(GSN!Goal)) and
sb.target.forAll(t|t.isTypeOf(GSN!Solution))
        ae.source.addAll(sb.target.equivalent());
        ae.target.addAll(sb.source.equivalent());
}

@lazy
rule SupportedBy2AssertedInference
    transform sb: GSN!SupportedBy
    to ai: SACM!AssertedInference {
        guard : sb.source.forAll(s|s.isTypeOf(GSN!Goal)) and
sb.target.forAll(t|t.isTypeOf(GSN!Goal))
        ai.source.addAll(sb.target.equivalent());
        ai.target.addAll(sb.source.equivalent());
}

@lazy
rule InContextOf2AssertedContext
    transform ico: GSN!InContextOf
    to ac: SACM!AssertedContext {
        ac.source.addAll(ico.target.equivalent());
        ac.target.addAll(ico.source.equivalent());
}

@lazy
rule AwayGoal2ArtifactReference
    transform ag: GSN!AwayGoal
    to c: SACM!Claim {
        c.isCitation = true;
        c.name = ag.name.equivalent();
        c.description = ag.description.equivalent();
        c.citedElement = ag.citedElement.equivalent();
}

@lazy
rule AwaySolution2ArtifactReference
    transform awaySolution: GSN!AwaySolution
    to ar: SACM!ArtifactReference {
        ar.name = awaySolution.name.equivalent();
        ar.description = awaySolution.description.equivalent();
```

```
    ar.referencedArtifactElement.add(awaySolution.referencedArtifactElement.e
quivalent());
}

@lazy
rule AwayContext2ArtifactReference
    transform ac: GSN!AwayContext
    to ar: SACM!ArtifactReference {
        ar.name = ac.name.equivalent();
        ar.description = ac.description.equivalent();
        ar.referencedArtifactElement.add(ac.citedElement.equivalent());
}

operation getSupportedByForStrategyAsSource(strategy: GSN!Strategy) {
    var allSupportedBy = getAllSupportedBy();
    return allSupportedBy.select(sb|sb.source.includes(strategy));
}

operation getSupportedByForStrategyAsTarget(strategy: GSN!Strategy) {
    var allSupportedBy = getAllSupportedBy();
    return allSupportedBy.select(sb|sb.target.includes(strategy));
}

operation getAllSupportedBy() {
    return GSN!SupportedBy.allInstances();
}

operation createTaggedValue(key: String) {
    var taggedValue = new SACM!TaggedValue;
    var _key = new SACM!MultiLangString;
    var value = new SACM!LangString;
    value.content = key;
    taggedValue.key = _key;
    _key.value.add(value);
    return taggedValue;
}
```