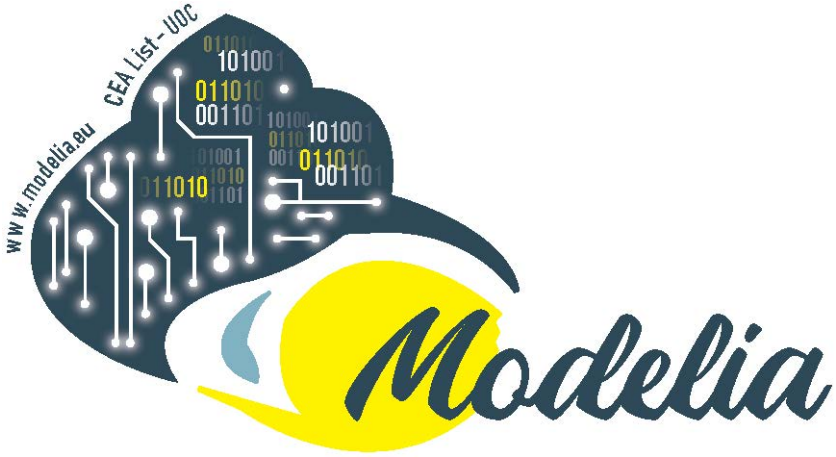
R&I

Universitat Oberta de Catalunya

4TH DELIVERABLE: TRUSTABLE TRACEABILITY FOR SYSMLV2



Edouard R. Batot

SOM Research Lab
Av. Carl Friedrich Gauss, 5. Building B3
08860 Castelldefels

SOM

4th Deliverable:

Trustable Traceability

Contents

List of Acronyms

UOC Fundacio per a la Universitat Oberta de Catalunya

SOM Systems, Software and Models Lab

CEA Commissariat à l'énergie atomique et aux énergie alternatives

SST SysML v2 Submission Team

MDE Model-Driven Engineering

SYSML-V2 Systems Modeling Language (SysML®) v2

KERML Kernel Modeling Language (KerML)

List of Figures

Listings

1 Introduction

This document presents the integration of Trace*a* [4], a language dedicated to traceability into the new release of SysML language. It is the continuation of deliverables 1, 2, and 3 of the Trace*a* project[1]. The first deliverable introduces the conceptualization work (SoA) related to traceability (D1) [1]. The second presents the metamodel of a language dedicated to traceability (D2) that fills the gaps detected in the existing literature during D1 [2]. The third deliverable shows the evaluation and integration of functionalities (confidence and evidence) in a software dedicated to traceability (D3) [3]. This report presents the integration of the concepts related to traceability developed during these first three deliverables within SysMLv2[2].

This document is organized as follows. Section 2, first briefly introduces the language dedicated to traceability that we have developed as well as the KerML / SysMLv2 ecosystem on which we integrate it. This section also defines the high level needs for quality traceability. Section 3, presents which elements of KerML / SysMLv2 languages are of interest and how we have adapted and used them through the different integration strategies considered. We will detail our choices of architecture and implementation and the limits that we encountered in Section 4[3]. Finally, we conclude this document in Section 5 before pointing at the resulting software artefacts and their examples of use in Section 6.

[1] Trace*a* is part of the Modelia intiative which aims at "Bringing artificial intelligence to the modeling world" `https://modelia.eu/projects/`.

[2] We alternately use the names SysML and SysMLv2 to refer to SysMLv2. The distinction between the two as well as all documentation relating to KerML / SysMLv2 is available by following the link: `https://github.com/Systems-Modeling`.

[3] We are integrating Trace*a* into SysMLv2 while it is being formalized by the SST.

2 Background: Trace*a* and SysML

This section introduces existing techniques we build upon (SysML and Trace*a*) then presents high level requirements for trustable traceability.

This section first introduces the Trace*a* metamodel [4] and presents in particular the functionalities for evaluating the confidence of trace links.

Then, an introduction to the KerML / SysMLv2 architecture establishes the preliminary notions necessary for understanding the project. The use of SysML does not require special knowledge of KerML (according to the SysML v2 Submission Team (SST)). However, it seems reductive to understand a language without addressing its foundations. We will see how the integration of Trace*a* can and should also be considered at the KerML level. Finally this section describes the high level requirements for quality traceability as defined in previous deliverables.

2.1 Existing

2.1.1 Trace*a*

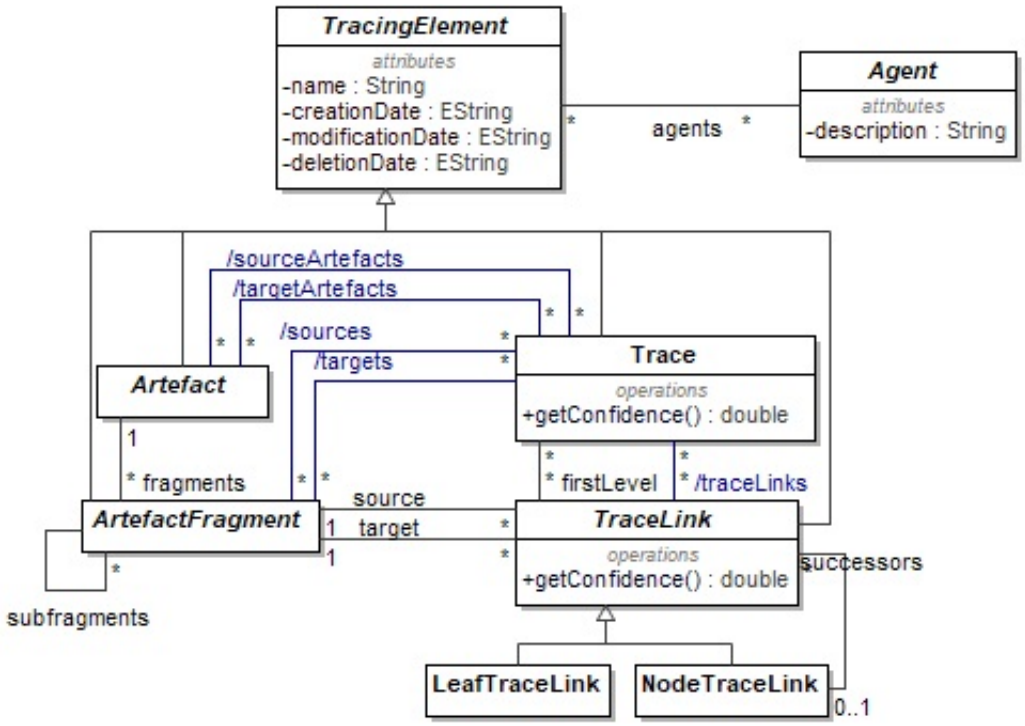


FIGURE 1: Core of Tracea: Artefacts and (Trace)Links.

Fig. 1 presents the core of the Trace*a* language. At its center we find the notions of *Trace*, *link* (TraceLink), *artefact* and *fragment*. Links make up the traces and connect *fragments* of the system to each other. We also notice the abstract nature of the Artefacts and Fragments. These are by definition volatile and must be redefined for each project or company to be best suited to concrete tracing needs.

Fig. 2 shows an extract from Trace*a* specialized in the qualification of traces and links. The concepts of confidence (applied to Trace and TraceLink), as well as the creation, modification and deletion dates of the trace elements are present. These concepts make it possible to put in contact the evolution of the elements, their agent in charge (human or not) and the level of confidence linked to the traces and links.

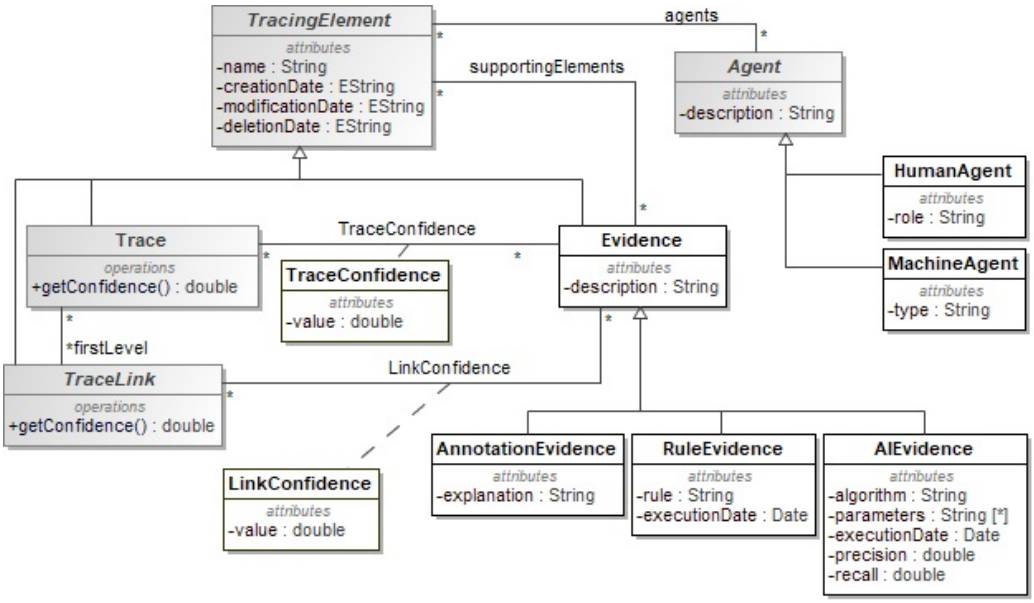


FIGURE 2: Excerpt of Tracea dedicated to the quality of trace and link.

Moreover, Trace*a* allows the attribution to the traces of information (evidences) allowing their confidence values to be justified. Depending on the nature of the means of identification employed, these are an identification rule, a simple textual annotation, or the details of the training configuration and execution of a learning algorithm (classes at the bottom right corner). The execution parameters of the automated identification means (rules or learning algorithms) are thus available for consultation after their execution. The evidence may also point to supporting elements implicated in the calculation and justification of the level of confidence.

2.1.2 The KerML/SysMLv2 ecosystem.

Fig. 3 shows the architecture of the KerML / SysML ecosystem. The Kernel Modeling Language (KerML) formally defines a set of concepts allowing the development of languages specific to a given activity. KerML has language elements defined at the syntax core level, themselves reusing the root language elements to allow the use of constructs specific to system modeling. These classes are redefined to establish the precise elements of modeling: relative to the construction of sequence diagrams, classes, activity, and the whole panoply of UML diagrams, among others. (For more details, see the KerML metamodel specification document [5]).

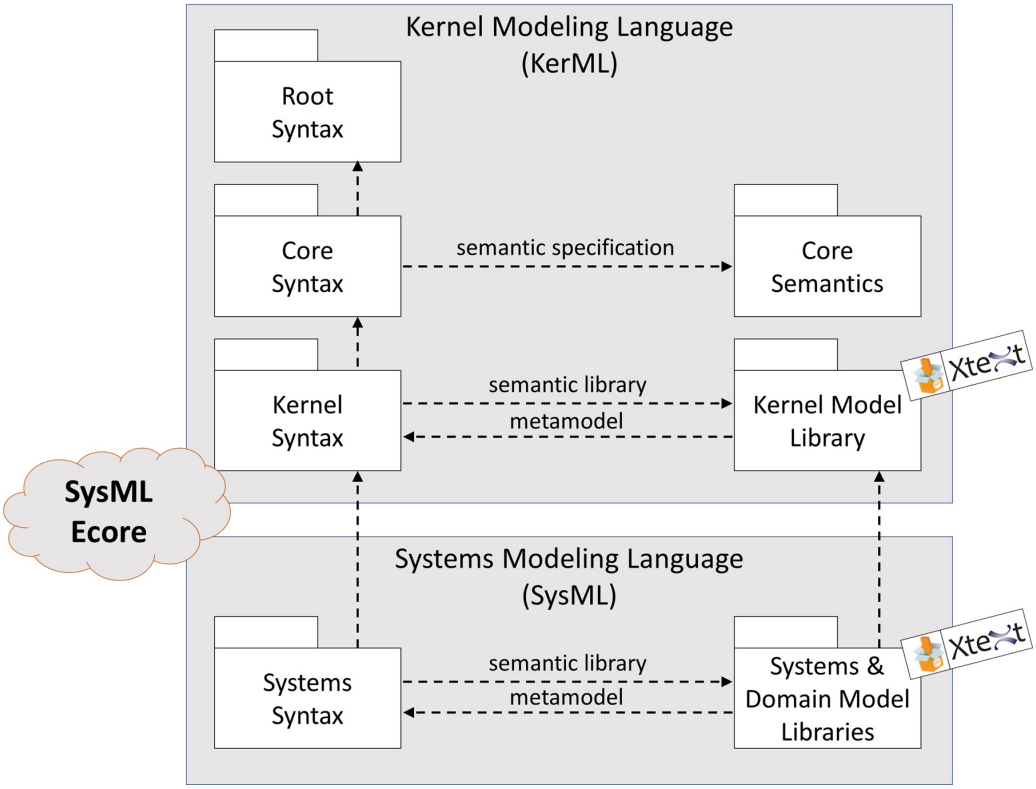


FIGURE 3: The KerML/SysMLv2 ecosystem.

As shown in Fig. 4, the SysML language is constructed as an extension of the KerML Kernel metamodel [5]. The SysML abstract syntax reuses the abstract kernel syntax, providing specialized constructs for modeling systems. Additionally, the SysML System Model Library extends the Kernel Model Library to provide the semantic specification for SysML. Finally, SysML provides an additional set of domain libraries to provide a rich set of reference models in various areas important for systems modeling (such as quantities and units, and basic geometry). (SysML p.69 [6])

Finally, it is important to note that concrete elements of KerML cannot be instantiated from SysML.

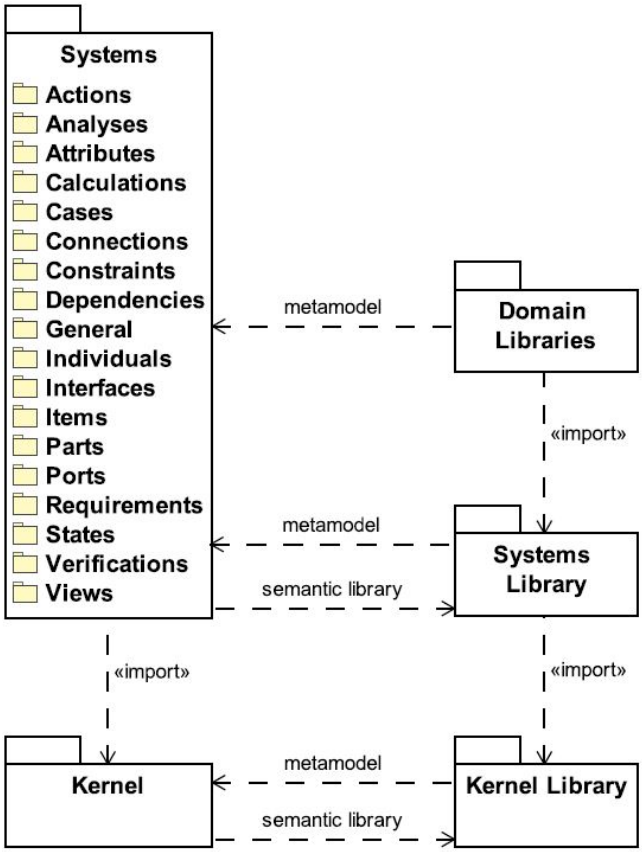


FIGURE 4: Architecture de SysML.

2.1.3 Elements, Relationships, and Annotations

KerML is organized around three fundamental concepts: Elements, Relationships and Annotations. The first two form the structure of the language and the third allows an orthogonal approach to it. Every element in SysML derives from these concepts.

More precisely, KerML makes it possible to represent complex structures in the form of graphs in which the nodes are Elements and the arcs are Relationships. Since a Relationship is itself an Element, relationship chains are also authorized and allow the nesting of features (*nested features*). Fig. 5 specifies the links that exist between Element and Relationship. The *ownership* is not detailed in this document but deserves to be mentioned here. Every Element is *owned* by another. When one element is deleted, all the elements it *owns* are also deleted. For information purposes, Figures 7 and 8 show the hierarchy derived from Elements and

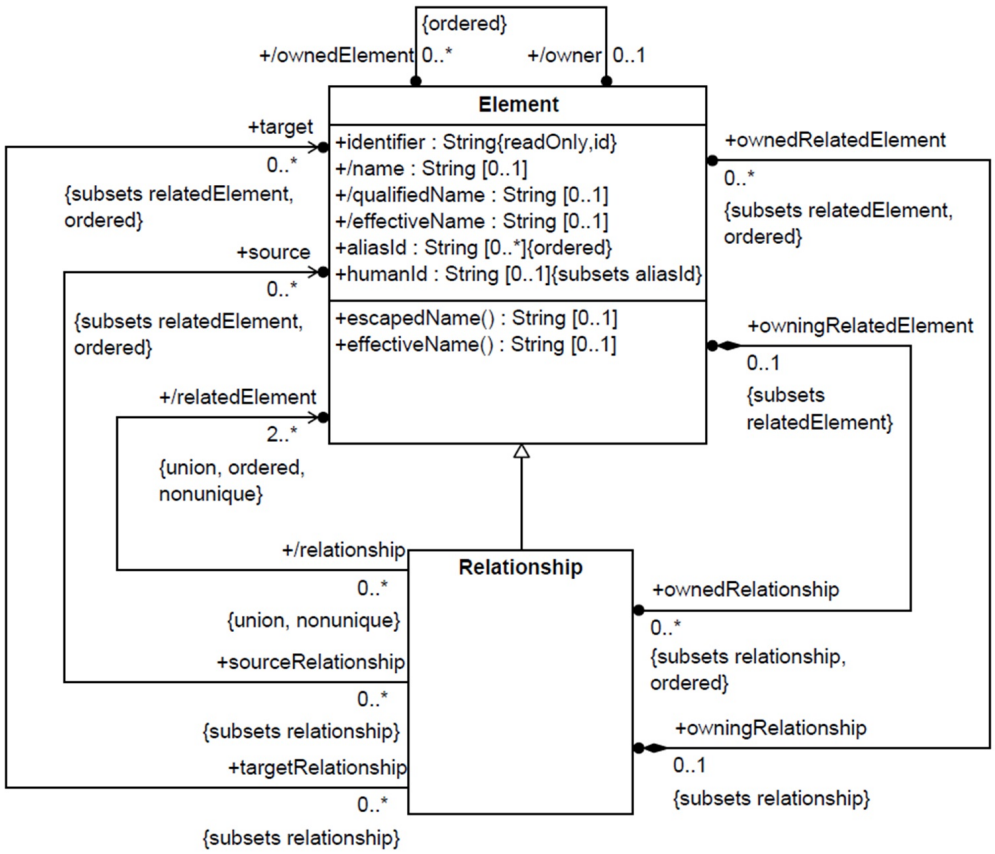


FIGURE 5: Elements et Relationships dans KerML.

Relationships in KerML.

On the other hand, "an Annotation is a relationship between an element and an AnnotatingElement that provides additional information about the annotated element. Each annotation falls between a single AnnotatingElement and a single element being annotated, but an AnnotatingElement can have multiple annotation relationships with different annotatedElements, and any element can have multiple annotations." (KerML Specification, p. 37 [5]).

In the case of traceability, annotations can be used to identify types of links and tag *relevant* links (*i.e.,* according to the specific use we want to make of them). We will speak of an *orthogonal* approach by annotations.

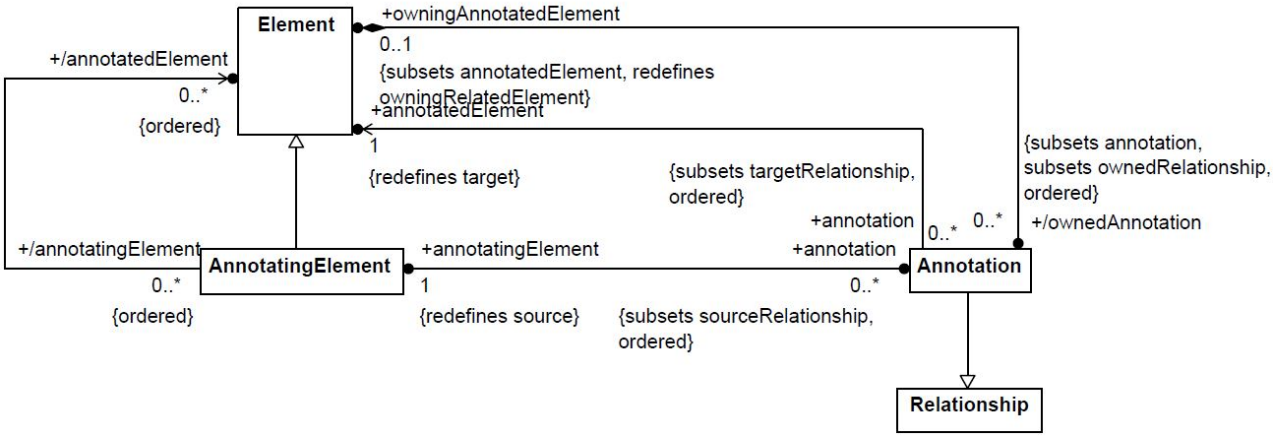


FIGURE 6: Annotations in KerML.

2.2 Goals

The needs in terms of traceability have been established in Deliverable 1 cite deliverable1. We summarize here the high level needs to satisfy a quality traceability, *i.e.,* allowing to qualify quantitatively and qualitatively the identified links useful for the tracing.

2.2.1 Mastering complexity

The tracing must allow to measure or approximate the complexity of the system. Since the concept of "Complexity" suffers from a high degree of volatility, its measurement (or its *proxy*) requires a consequent adaptability.

2.2.2 Increase the ability to scale the system

To facilitate the monitoring of the evolution of the system over time, tracing makes it possible to concentrate the measurement of the impact (*eg,* of a change) on (types of) specifically chosen links.

2.2.3 Maintain a degree of independence from the target system

Tracing must not influence the system. It must remain independent, or *orthogonal*, so that *i)* it does not alter the evolution of the system, *ii)* it does not depend on decisions relating to the development of the system, and *iii)* it remains possible to plot systems of systems (*ie,* written in different languages, on different platforms).

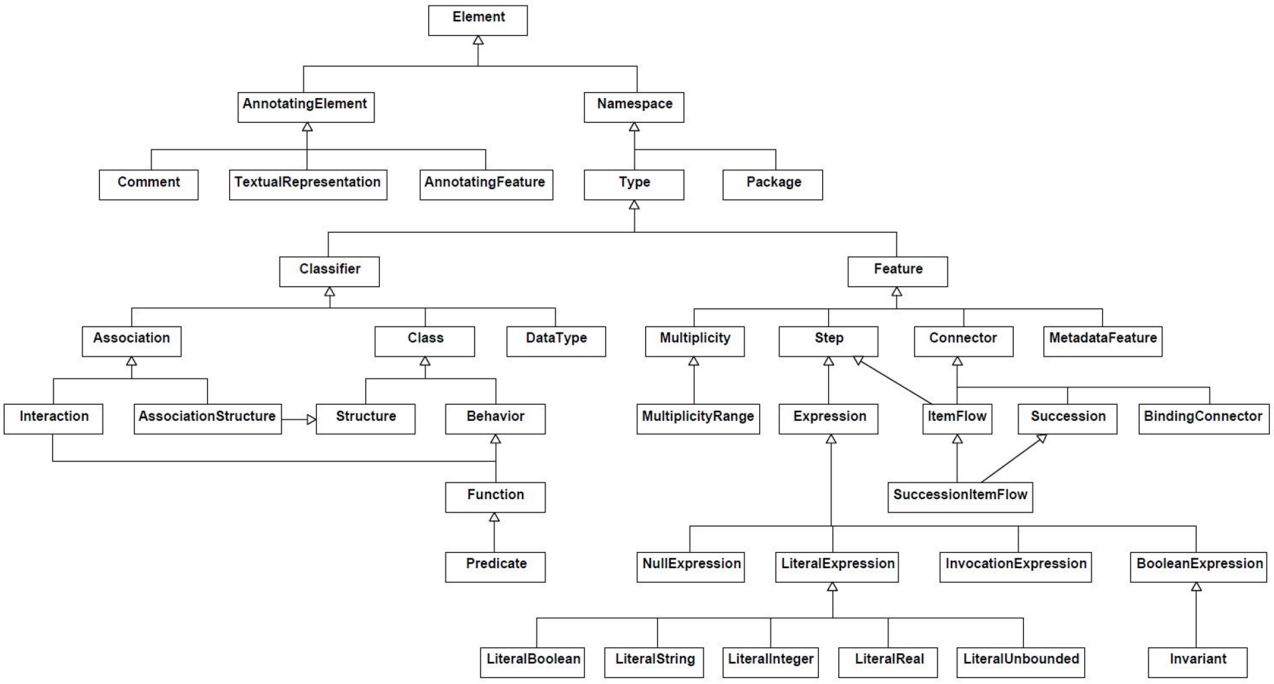


FIGURE 7: Hierarchy of KerML Elements.

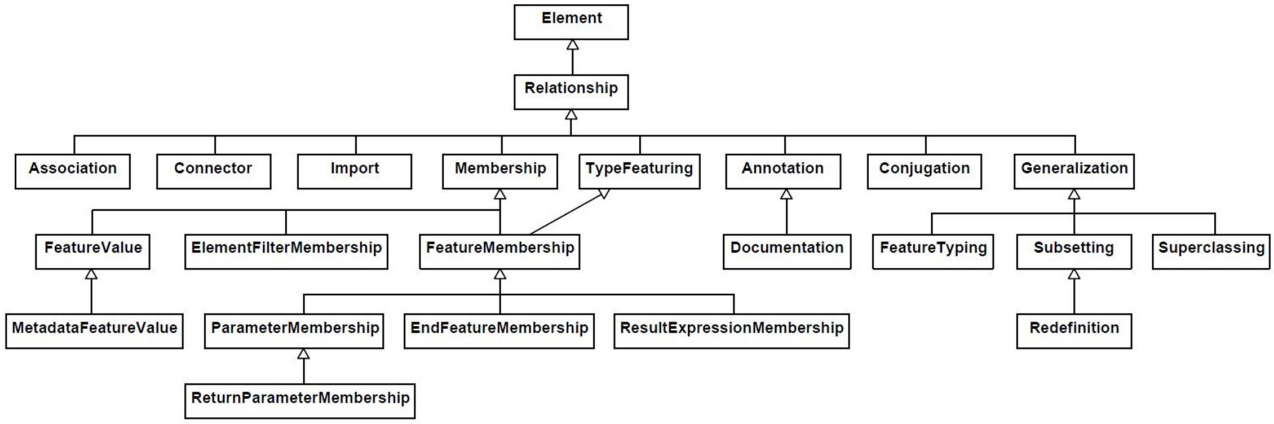


FIGURE 8: Hierarchy of KerML Relationships.

3 Strategies for traceability of SysML elements

During this fourth deliverable we explored different strategies offered by SysML to allow trustable traceability.

This section presents the 4 strategies considered to integrate the functionalities of Trace*a* into SysMLv2.

In this section, we present the four strategies considered, their strengths and weaknesses, as well as the one promoted by those in charge of the SST. This section details the integration of the concepts of trace and trace link within KerML to be able to harness the notions of confidence and justifications (Evidence) of these latter.

3.1 Root-level adaptations

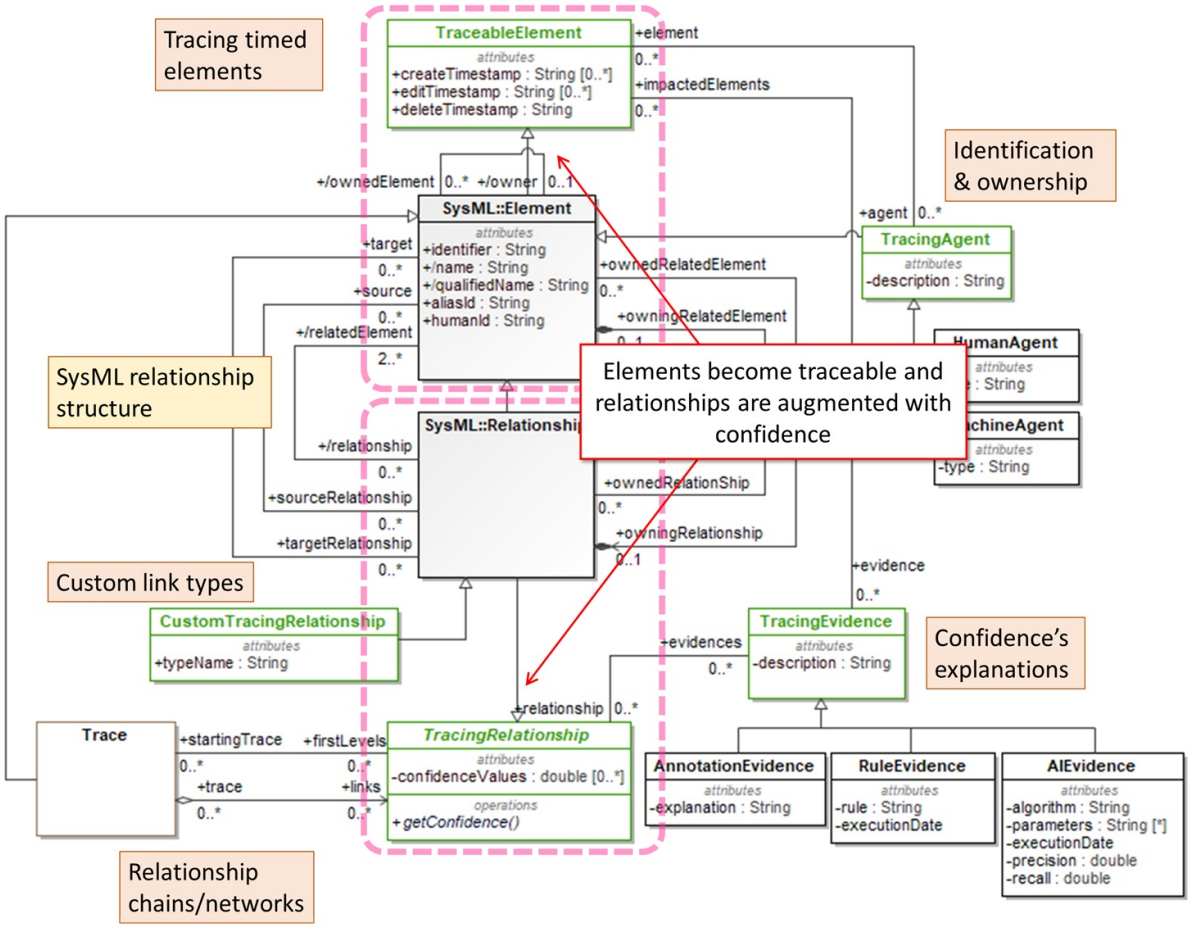


FIGURE 9: Adaptation of root-level elements.

The first strategy consists in augmenting the root elements with the attributes necessary for traceability. Fig. 9 shows the modifications required. In gray are the SysML classes, in green the hinge/border classes. The purple dotted lines underline the areas of interest. A TraceableElement class increases the KerML Elements by aggregating evolution markers (timestamps) and references to the agents responsible for their identification. The latter are particularly interesting when using non-deterministic algorithms for identifying the tracing links (see the detail of the metamodel of Trace*a* for more details [4]). Another class comes to increase the Relationships by attributing to it a value of confidence.

This strategy must have been left because it turns out to be too intrusive in the language. Changing the root elements potentially impacts the entire language structure and therefore could not be incorporated into the KerML/SysMLv2 specification process. Moreover, this strategy implies the definition of a concrete syntax which fits into the KerML language schema – which can be very expensive in terms of consistency evaluation and verification.

3.2 Annotation type dedicated to tracing

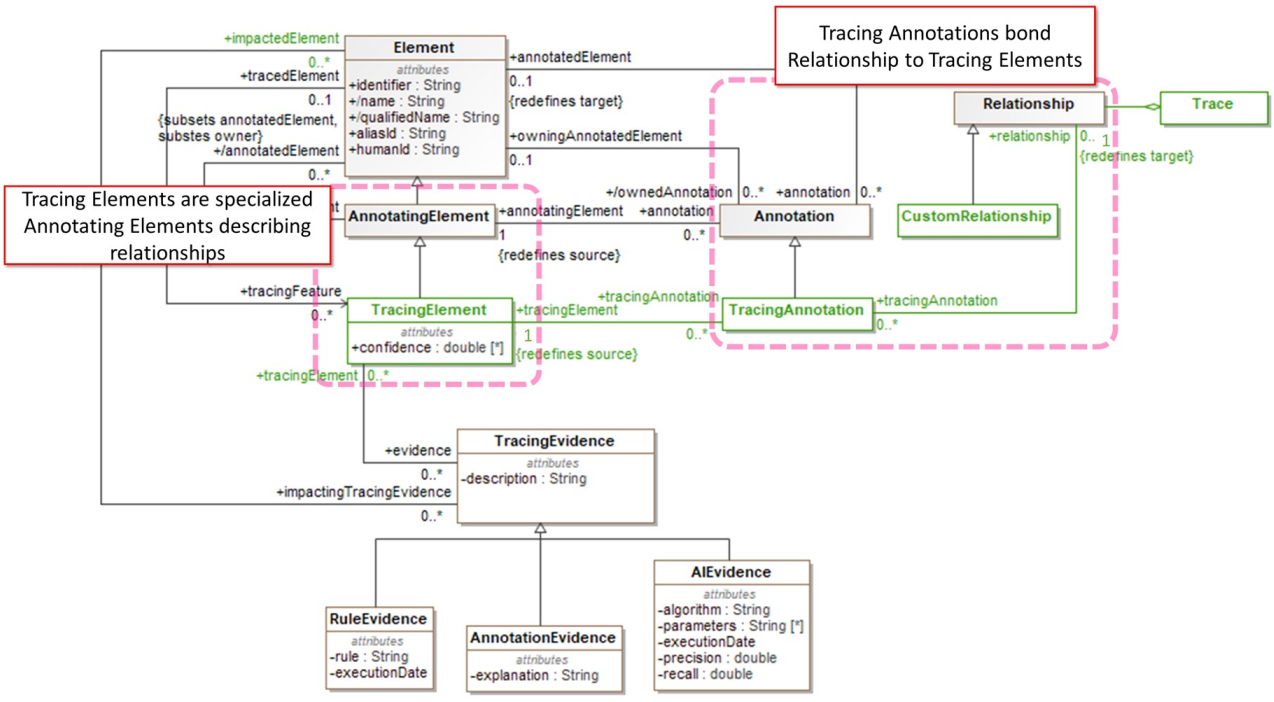


FIGURE 10: A new type of annotation for KerML.

To avoid modifying the fundamental elements of KerML, our second idea was to redefine an annotation type dedicated to tracing. To do this, a TracingAnnotation links a Relationship

to a TracingElement just as an annotation links an Element to an AnnotatingElement. The idea is to allow the relevant relationships to be annotated with confidence values (and associated with a set of information justifying the calculation of these values). Fig. 10 illustrates the modifications necessary to apply this extension to KerML.

This strategy reuses the existing concrete syntax (and to add the keywords necessary to enter the confidence value and the justifications). An illustrative example is provided in Listing 1. However, this strategy poses a problem in terms of tooling. Indeed, by modifying the abstract syntax (the metamodel) to add a new type of annotation, all the tools allowing the manipulation of this syntax must be updated. This is an intrusive solution again. In a language still in the process of stabilization, the chances of making these changes happen are slim.

```
1  trace Req2Source {
2     from elt1 to elt2 /** Elements */
3     type CustomLinkForClasses
4     confidence 0.83
5     impacts D, E, F
6     description "Something"
7  }
```

Listing 1: Sample of a concrete syntax for tracing annotations

Nonetheless, the concept of annotation is to be retained here – it suits perfectly the idea of traceability *orthogonal* to the software. With tracing annotation, a change in the traced elements does not necessarily impact the tracing elements, and vice versa.

3.3 A new type of (*annotating feature)*

The KerML specification states that "*an AnnotatingFeature is a kind of AnnotatingElement that allows the definition of structured metadata with attributes specified by the modeler.*" (KerML specification, p. 178 [5]). Fig. 11 presents a snippet of the KerML metamodel where the AnnotatingFeature class is defined. AnnotatingFeatures use MetadataFeatures which redefine new features for associated elements (respecting the typing defined in DataTypes). These features will be evaluated at the *execution of expression at model level* and a value will be associated with each feature.

Fig. 12 shows the modifications to be made to KerML to allow the redefinition of a new type of AnnotatingFeature dedicated to traceability. A TracingAnnotation associates a TracingFeature with a Relationship. This Feature includes a level of confidence and possibly evidence (references to other elements).

This approach is the most interesting. KerML / SysMLv2 maintainers informed us that this "AnnotatingFeature / Datatype" structure is in their sights because it allows the addition

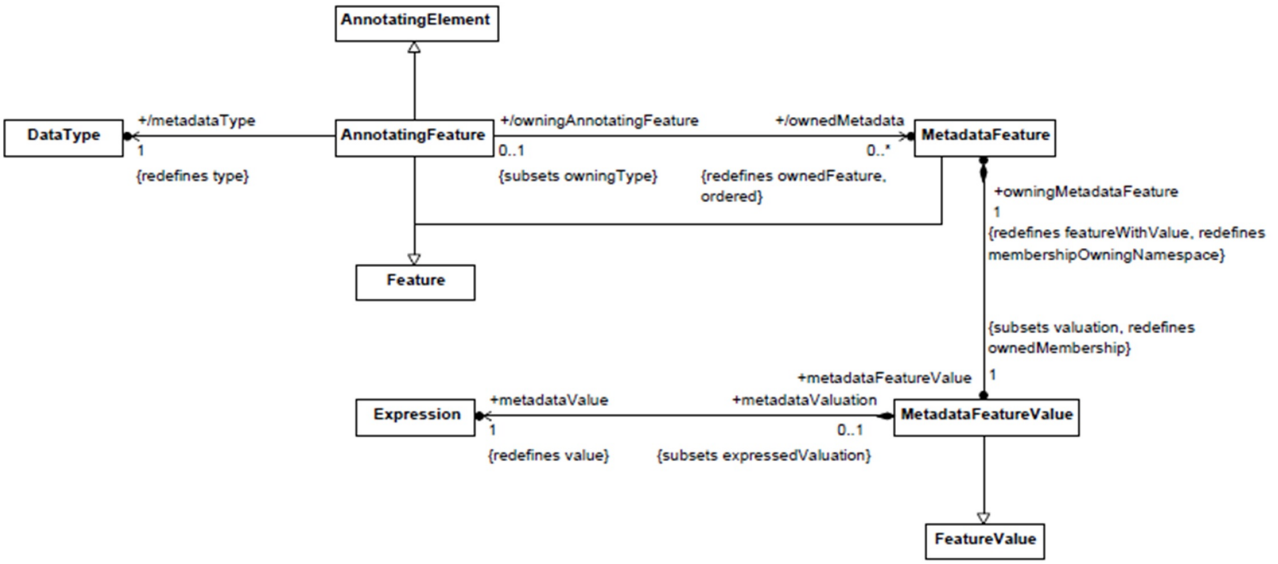


FIGURE 11: Snippet of the KerML metamodel: AnnotatingFeature and valued Expressions for adaptable metadatatypes.

of functionality in the form of metadata libraries without directly modifying the language elements (what we call ”orthogonality”). More precisely, they plan to use only this type of annotation and to ban its refinement in favor of this structure to allow the simplified writing of specialized function libraries (*feature libraries*). From September 2021, the different types of AnnotatingElement (Comment, TextualRepresentation and AnnotatingFeature) will converge to a representation based on AnnotatingFeature. This migration will simplify, first of all, the elaboration of comments specific to a project / domain thanks to the use of dedicated Datatypes.

3.4 A feature library for traceability

The last strategy considered will be the one implemented. Instead of modifying the abstract syntax structure of KerML (*i.e.,* metamodel level), the idea is to create a library of features dedicated to traceability (model level). It is about redefining datatypes specific to the tracing features in order to join them to the structures concerned at the model level. It is an orthogonal method comparable to the use of *stereotypes* in UML. The comparison is based on the ability offered by the AnnotatingFeature to define new (libraries of) features without altering the base of the language itself. The process is particularly suited to building a tracing strategy, often implemented *after* the software has been put into production.

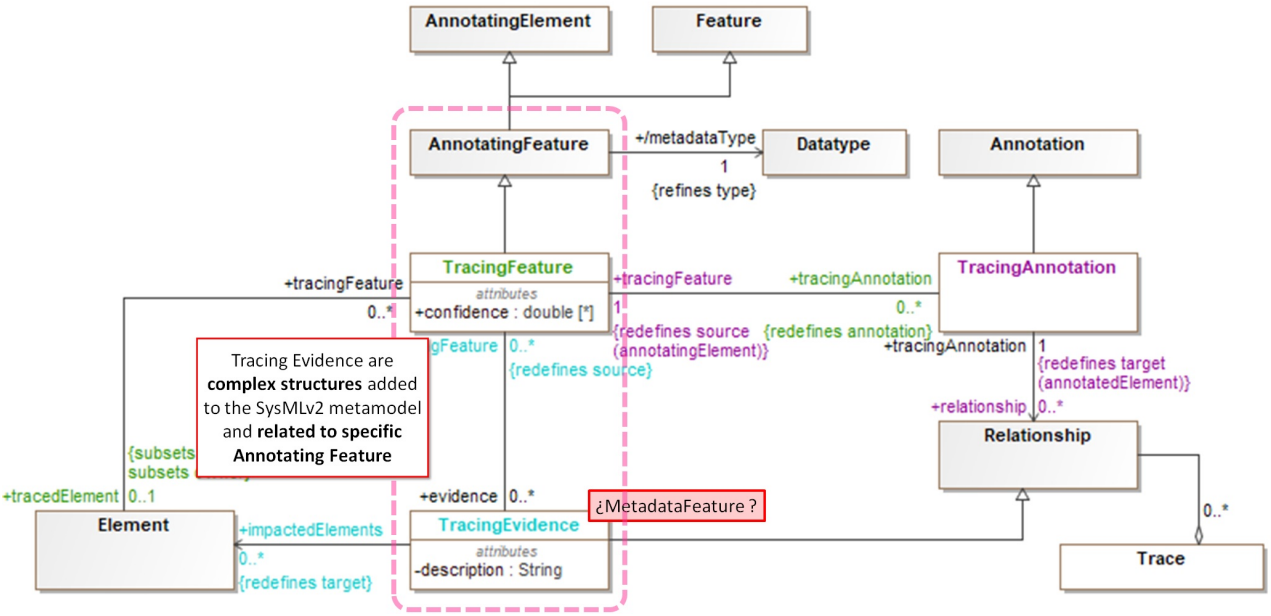


FIGURE 12: A new type of annotating feature dedicated to traceability .

The Listings 2 and 3 present our feature library in SysML. Listing 2 contains the library declaration; Listing 3 contains an example of application of such datatype defined on a concrete example. It illustrates the allocation of a confidence value of 0.7 to a connection between a requirement (req) and a package. The second case (line 16 to 20) attributes a description and points to an element impacted (by the evaluation of the confidence),

```
1  package TracingAnnotations {
2   attribute def ConfidenceTracing {
3     attribute confidence : Real;
4     attribute impact : Anything[*]
5     assert constraint
6       { confidence >= 0.0 && confidence <= 1.0 }
7   }
8
9   attribute def ExplainableTracing {
10    attribute description : String;
11    attribute evidence : Evidence;
12    attribute agent : Agent;
13  }
14 }
```

Listing 2: Definition of a datatype dedicated to traceability (partial listing).

```
import package TracingAnnotations::*;

/*Definition of the target system. */
part vehiculetest {}
req RE01_MLV {}
package UMLCD_CORE {}

/*Assignment of a confidence of 0.7 to the Req2Design link.
  */
connection Req2Design connect RE01_MLV to UMLCD_CORE {
  @ConfidenceTracing {
    confidence = 0.7;
  }
}

/*Assigning a description and an impact to the Req2Design
  link.*/
connection Req2Design connect RE01_MLV to UMLCD_CORE {
  @ExplainableTracing {
    description = "Something";
    impact = (vehiculetest);
  }
}
```

Listing 3: Use of metadata features for traceability

4 Metadata feature library for quality traceability

This section presents the limitations imposed by the state of the art of KerML / SysMLv2 in terms of meta-annotation. It then presents the design that we have implemented to overcome it.

We have previously mentioned the advantages that the use of feature libraries (orthogonal) offers compared to more intrusive strategies (modifying the structure of the language).

However, the development of datatypes dedicated to traceability encounters a certain number of limitations due to the state of progress of the implementation of KerML / SysMLv2. In this section, we present the limitations encountered and then our design for quality traceability for KerML *and* SysMLv2.

4.1 Orthogonality - priority and limitations

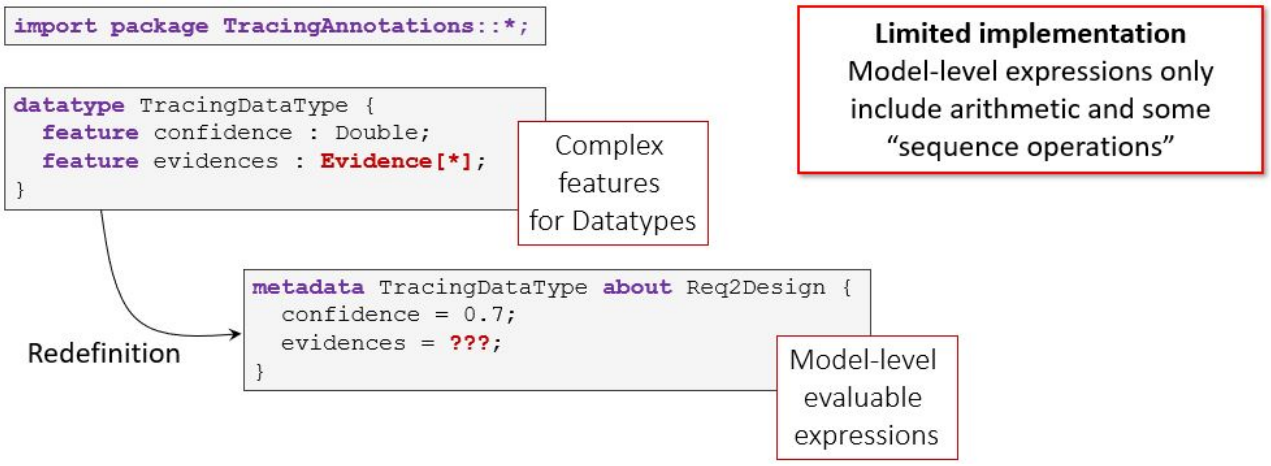


FIGURE 13: Defining complex Metadata structures and evaluating expressions at model level: implementation limits.

To use the words of Ed Seidewitz, one of the main promoters of the new version of SysML and one of the main people responsible for its delivery, the idea of using the AnnotatingFeatures to define *a priori* external needs to the language specifications is "extremely relevant". However, the next revision will not contain any specific article on the matter. A number of decisions remain to be clarified on the subject. In particular, the ability to define and reuse complex structures within feature datatypes is under consideration. For now, the definition of complex datatype is not guaranteed to work, as illustrated in Fig. 13.

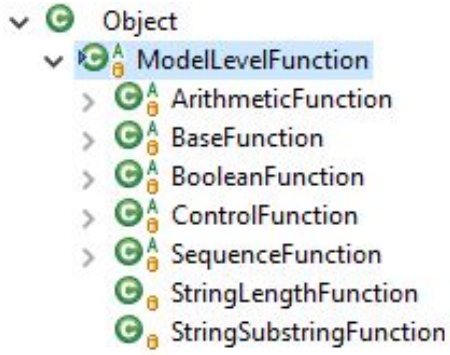


FIGURE 14: Types of functions available for evaluating expressions at the model level.

Until the delivery of the second revision (RFP2) scheduled for late August 2021, the SST remains busy with other priorities. Ed Seidewitz foresees the deployment of the features related to modelv level expression in October or November 2021. At the time of writing, annotating features are ”only” templates in which some points of variability are allowed (the authorized functions are derived from ModelLevelFunction, see Fig. 14).

Consequently, the definition of complex structures within datatypes is limited as stated above. Only Ed Seidewitz is aware of the code related to the question and he himself struggles to remember the state of the module dedicated to the assignment of values to complex datatypes. As an example, the use of enumeration (*enum*) in annotations does not allow their use for evaluation (see Section 4.4). Finalizing this implementation will only be possible when a subset of the modeling language SysML will be defined. In view of the SST's plans to converge the different types of annotations towards AnnotatingFeatures using metadatatypes, these points will be addressed with high priority – but only after the submission of the new revision.

4.2 Iterative design: confidence and explainability of trace links

To favor the use of specialized libraries (such as the SST encourages), we have designed SysML and KerML datatypes dedicated to tracing. Fig. 15 shows the design of these structures. At the top, we distinguish between a design allowing the assessment of confidence and a design allowing to go further by entering the supporting information for this assessment. We distinguish between the two for the sake of iterativity of the development and implementation processes (as we have just seen, the mechanisms allowing the manipulation and evaluation of datatypes are not fully completed in KerML / SysMLv2).

The `ConfidenceTracing` datatype has been implemented to overcome this disappointment and initially allows the attribution of a real value of confidence to the annotated links. We add a description field (free String) and a field to reference to potentially impacted elements

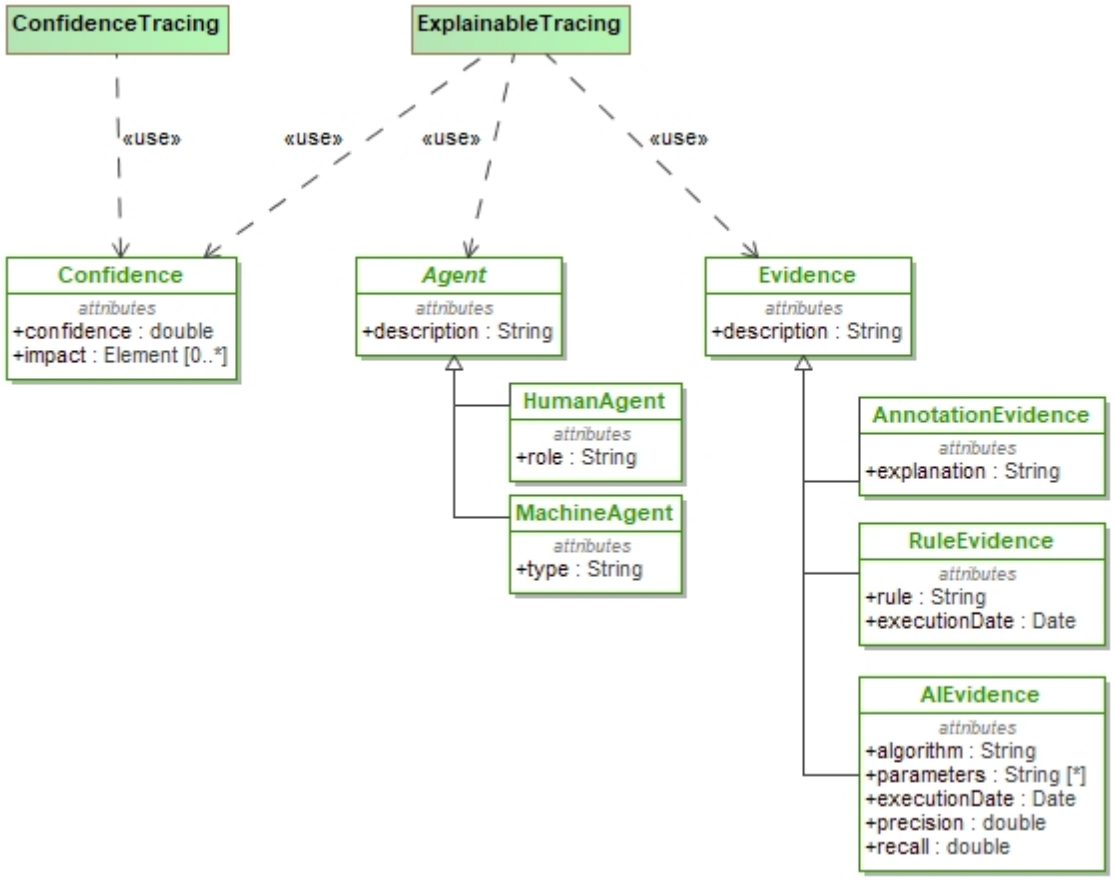


FIGURE 15: Metadatatypes for quality traceability including an assessment of confidence and monitoring of the identification processes used.

– *e.g.,* used in the calculation of the confidence value. The `ExplainableTracing` datatype has the same attributes as ConfidenceTracing (description, confidence and impact) with in addition *i)* the notion of Evidence to store information about the identification process used for tracing, and *ii)* the possibility to inform which (type of) agent is responsible for identification.

4.3 Consequence of the separation of KerML and SysML languages

SysML is based on KerML. However, if the language dedicated to systems reuses the concepts of the kernel language, these are not accessible from the former. Thus, the definition of a Datatype (and its associated Features) does not work in SysML but only in KerML. To allow the use of Trace*a* in both languages, we have transformed the Datatypes and their features into *attribute definitions and uses* (resp. "Attribute def" and "Attribute"). Indeed, an attribute definition is a particular type of structure dedicated to the construction of Datatypes for the design of systems (see Fig. 16).

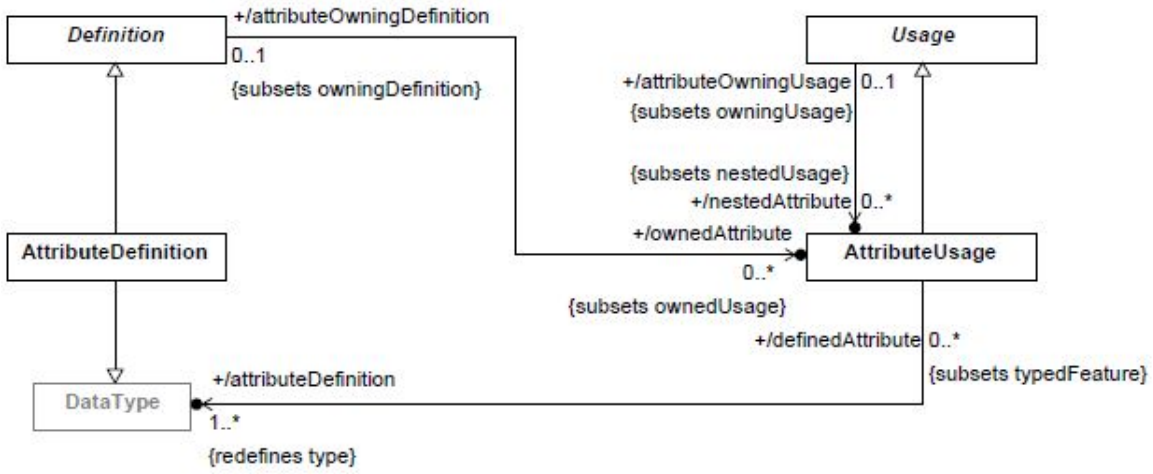


FIGURE 16: The definition and use of attributes in SysML is a specialization of datatypes and their features in KerML.

4.4 Usage example: filtering

This section presents an example: the filtering of links satisfying a confidence greater than a given threshold. The first 30 lines of listing 4 (with graphical representation in Fig. 17), show the structure of the system and links being traced. Lines 33 to 39 show the code needed for filtering. This is a reuse of the existing machinery to support SysMLv2 metadata annotations. Fig. 18 shows the result of the query: the five links with a confidence greater than 0.5 are filtered.

```
1  package Tracing_FilterExample {
2    import TracingAnnotations::*;
3
4    /* System under tracing. */
5    part end1 {}
6    part end2 {}
7    part end3 {}
8    part end4 {}
9    part end5 {}
10   part end6 {}
11   part end7 {}
12   part end8 {}
13
14   /* Trace links, confidence valued. */
15   connection testLink95 connect end1 to end2
16     {@ConfidenceTracing { confidence = 0.95; }}
17   connection testLink85 connect end1 to end3
18     {@ConfidenceTracing { confidence = 0.85; }}
19   connection testLink75 connect end1 to end4
```

```
20      {@ConfidenceTracing { confidence = 0.75; }}
21    connection testLink65 connect end5 to end6
22      {@ConfidenceTracing { confidence = 0.65; }}
23    connection testLink55 connect end6 to end7
24      {@ConfidenceTracing { confidence = 0.55; }}
25    connection testLink45 connect end7 to end8
26      {@ConfidenceTracing { confidence = 0.45; }}
27    connection testLink35 connect end8 to end7
28      {@ConfidenceTracing { confidence = 0.35; }}
29    connection testLink25 connect end8 to end6
30      {@ConfidenceTracing { confidence = 0.25; }}
31  }
32
33  import TracingAnnotations::*;
34  /* Filter example - full notation. */
35  package ConfidenceLevel {
36    /*Connections that satisfy a threshold confidence level (0.5).*/
37      import Tracing_FilterExample::**;
38      filter @ConfidenceTracing && ConfidenceTracing::confidence >=
        0.5;
39  }
```

Listing 4: Definition and filtering of tracing annotations to collect links satisfying a certain level of confidence.

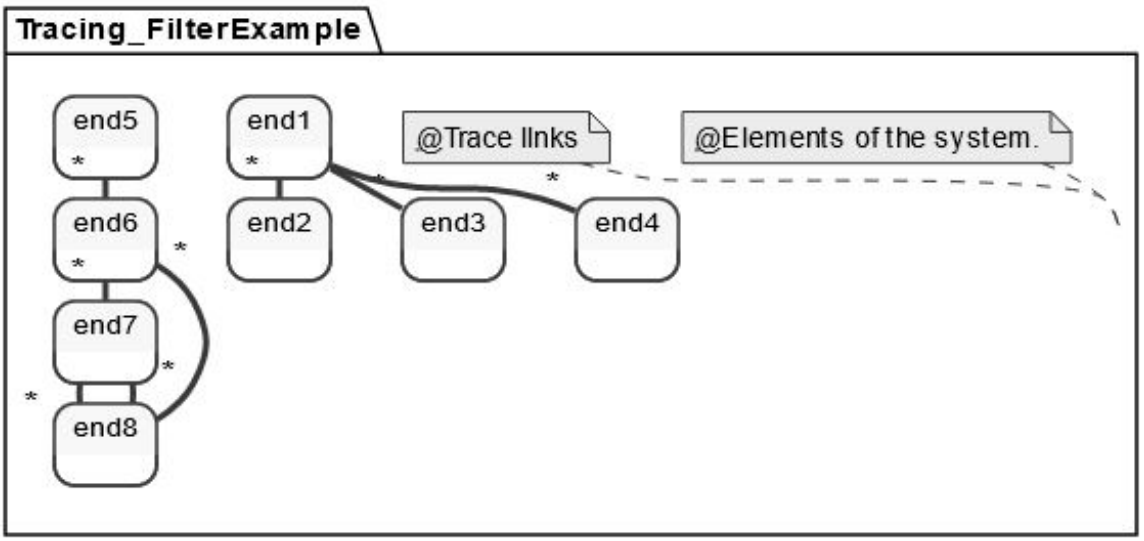


FIGURE 17: Example using the SysMLv2 filtering tool for trace annotations.

The same process can (could) be used to ”tag” links with specific types – or any other

```
1  %list Confidencelevel::*
```

```
Comment (cd2e95dd-44ff-409e-92dc-0e331134b280)
AnnotatingFeature (38a370fc-bf07-4c0a-a3c9-811e248ad491)
AnnotatingFeature (027e4618-290f-4eeb-bcf5-5ae75bb16b4c)
AnnotatingFeature (0a000fbb-4457-4ca1-adba-cdfc86ca76fa)
AnnotatingFeature (04654814-0344-4a5d-a570-d3fb16d19163)
AnnotatingFeature (934304f6-a447-4d40-a770-b0b0bdabd945)
ConnectionUsage testLink55 (a710d03a-87f9-4563-9288-f35c26256fd0)
ConnectionUsage testLink65 (a1d41392-18ea-4167-b817-b1c47862629b)
ConnectionUsage testLink75 (3626785f-4544-4e9c-a1d4-a9f3e38618af)
ConnectionUsage testLink85 (7e501da2-e5ba-4f43-ba0c-079ed1d8b6c6)
ConnectionUsage testLink95 (7e50580c-2a17-42a1-88aa-dbad7c886bac)
```

FIGURE 18: Filtering result: filtered annotations (i.e., satisfying a specific confidence).

pre-defined attribute. The implementation of the evaluation of expressions is unfortunately limited. As to this day, the use of strings of characters (*String*) or of enumerations (*enum*) in the definition of datatypes generates a compilation error. Fig. 19 shows this limitation (the figure has been generated with JupyterLab, see Section 6).

```
1  import TracingAnnotations::*;
2  ⟶/* Filter example - full notation */
3  ⟶package ConfidenceLevelnType {
4  ⟶⟶/* Connections that satisfy a threshold confidence level (0.5). */
5        import Tracing_FilterExample::**;
6        filter @TraceType && @TraceType::type == TraceTypes.test1;
7    }
```

```
ERROR:Couldn't resolve reference to Element 'test1'. (112.sysml line : 6 column : 61)
ERROR:Must be model-level evaluable (112.sysml line : 6 column : 9)
ERROR:Must be a valid feature (112.sysml line : 6 column : 50)
ERROR:Must be a valid feature (112.sysml line : 6 column : 61)
```

FIGURE 19: Filtering is not (yet) implemented for enumerations.

5 Conclusion

This deliverable presents means to integrate quality traceability into SysMLv2 using Trace*a*. With Trace*a* datatypes, SysMLv2 relationships can be annotated with valuable information related to their quality – *i.e.,* their *trustability*. The degree of confidence as well as the information necessary to justify it can be associated to SysML links (connections) through metadata definition.

This integration is *orthogonal*. It does not impact the structure of the language itself – changes happen at the model level with new feature libraries, not at the metamodel level. This allows the (re)use of the machinery supporting (meta)annotations and eases the (re)definition of tracing structures specific to a certain project or company. We showcase these benefits in one small example that also reveals the current limitations of SysMLv2's implementation.

The SST confirms that annotating features are valuable and salient artefacts in the development of SysML. They shall take more and more importance in the future releases. Work on the evaluation of expressions at the model level is highly required and will be part of the agenda of the fourth quarter of 2021. Finally, as a sign of encouragement, the SST invites us to further investigate in the direction we took.

6 Software artefacts

The software artifacts we have implemented have been uploaded to the Modelia Git repository and can be accessed at: `https://github.com/modelia/tracea/3-sysml-integration`.

The file contains the two definitions presented above along with a simple example and a filtering usage example. Subfolders are `/kerml` and `/sysml` respectively to KerML and SysML versions.

SysML datatypes can be viewed in the JupyterLab of the SST. To do so, open the following link (`https://www.sysmlv2lab.com/`) in a web browser and upload the code contained in `/sysml/jupyterlab/TraceaTracingAnnotations.ipynb` from the Tracea Git repository. Its Markdown counterpart (*.md*) shows a *Git readable* version of the Jupyter document.

REFERENCES

[1] Edouard R. Batot. First deliverable: Survey of traceability techniques with a focus on their applications in ai-based software techniques, august 2020.

[2] Edouard R. Batot. Second deliverable: Traceability language – definition and editor, december 2020.

[3] Edouard R. Batot. Third deliverable: Traceability solutions – evaluation and extension, april 2021.

[4] Edouard R. Batot, Sebastien Gerard, and Jordi Cabot. (Not) yet another metamodel for software traceability. In Proceedings of the 12th System Analysis and Modelling Conference, SAM '20, page 1–10. Association for Computing Machinery, 2021.

[5] Collegial industrial partners and Ed Seidewitz. Kernel Modeling Language (KerML), version 1.0, release 2021-04, submitted in partial response to systems modeling language (SysML®) v2 RFP, may 2021.

[6] Collegial industrial partners and Ed Seidewitz. OMG Systems Modeling Language tm (SysML®), version 2.0, release 2021-04, submitted in response to systems modeling language (SysML®) v2 RFP, may 2021.