Proprietor: Dr Darren R C Kelly, **ABN:** 67 677 268 579, **Post:** PO Box 1816, Bondi Junction, NSW 1355, Australia
Phone/Fax: +61 2 9386 0090, **Mobile:** +61 405 029 008, **Email:** info@webel.com.au, **Web:** https://www.webel.com.au

Proposal for a LogicalWrapper element for UML and SysMLv2

Version date: 2019-11-18
Author: Darren Kelly, Webel IT Australia
Intended audience: OMG SysML and UML Revision Task Force members and related community

Abstract

The LogicalWrapper is a new graphical grouping element proposed for UML and SysML[1]. The LogicalWrapper may not own elements that are not LogicalWrappers[2], it merely groups elements logically and graphically (visually) according to a logical context indicated by its name, with traceability of the logical members of the grouping. Any element that may be a logical member of a LogicalWrapper may be a member of more than one LogicalWrapper.

The proposed symbol for the LogicalWrapper is a simple rectangle, with a name header similar to that of a Class symbol, a compartment for logical graphical grouping, and an optional listing compartment for logical **members**. A LogicalWrapper may be used in any UML or SysML diagram that supports element symbols that can be graphically contained within the proposed rectangular symbol for a LogicalWrapper.

Graphical containment, in supporting diagrams, of wrapped symbols by the proposed rectangular symbol for a LogicalWrapper implies logical membership (grouping) of the elements represented by those symbols, but does not imply ownership or containment in any sense in the underlying model. A LogicalWrapper may itself be wrapped by another LogicalWrapper (nested) if the outer wrapper provides a relevant context for the inner wrapper.

A proposed relationship Wraps can be used to indicate members of a wrapper when graphical containment is not used. Two LogicalWrappers may be related to each via binary relationships.

Logical wrapping is presented as a *parasitic* organisational mechanism that is *orthogonal* to the "physical" ownership of elements afforded by Packages, Models, and other Namespaces.

The LogicalWrapper shares some of the membership tracking powers of the SysML ElementGroup, and – like the SysML ElementGroup – it may be extended to help support elicitation of model elements through *Graphical Parsing Analysis* (although that is not its primary application).

1 While familiarity with SysML may be helpful, it is not essential for understanding this proposal.
2 Whether a LogicalWrapper may own another LogicalWrapper is a discussed variation point.

Proprietor: Dr Darren R C Kelly, **ABN:** 67 677 268 579, **Post:** PO Box 1816, Bondi Junction, NSW 1355, Australia
Phone/Fax: +61 2 9386 0090, **Mobile:** +61 405 029 008, **Email:** info@webel.com.au, **Web:** https://www.webel.com.au

Table of Contents

1) Primary example: Telescopes of various kinds

The primary example used is various kinds of telescopes: optical, infrared, radio, x-ray, gamma, UV; ground-based, space-based, airborne. The main purpose of the proposed LogicalWrapper is shown in an adapted SysML Block Definition Diagram (BDD) in Figure 1.1, where some telescopes – represented by SysML Blocks – are logically grouped by LogicalWrapper elements (indicated by the stereotype keyword «wrapper») according to whether they are considered ground-based or space-based, where the wrapped Blocks happen to have been packaged according to whether they are considered primarily «optical» or «infrared» (indicated additionally here by stereotype keywords[3] independent of ownership by Packages/Models to help tracking under owner change).

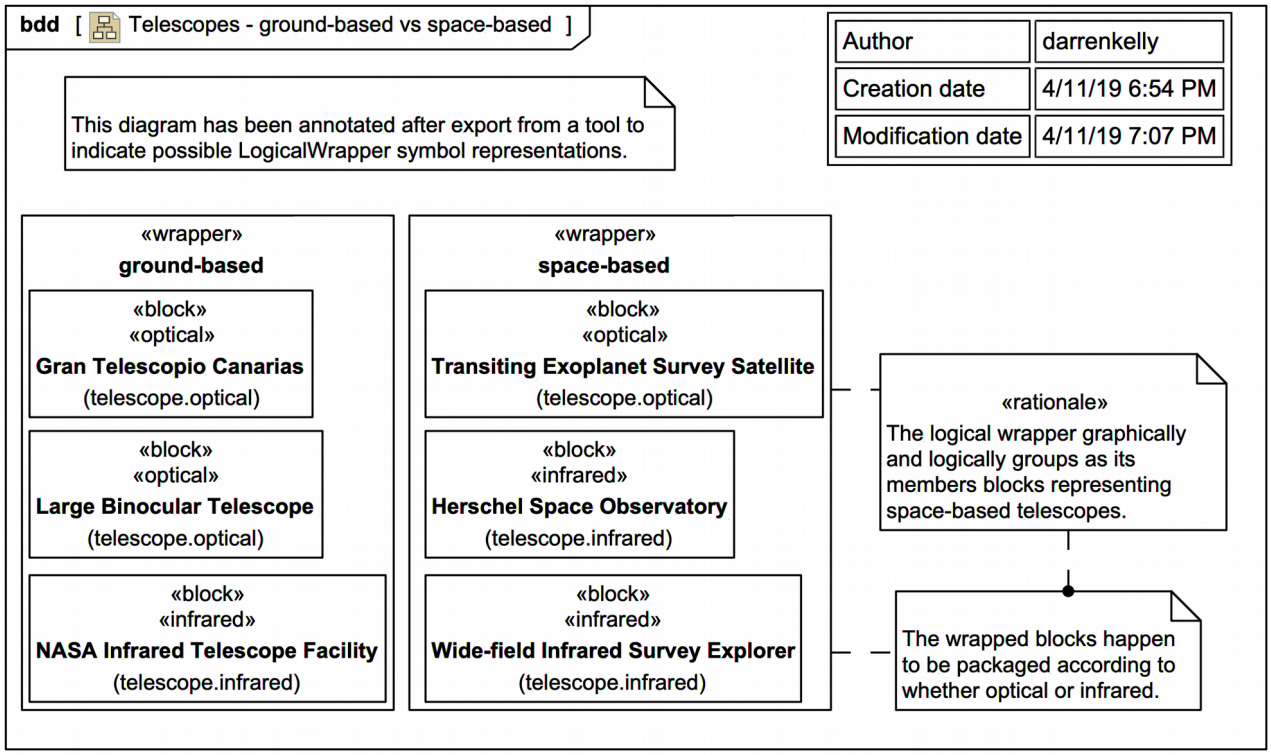

Figure 1.1: LogicalWrapper elements logically and graphically grouping SysML Blocks in a BDD

Note that the packaging can be changed by the modeller at any time without any impact on the logical grouping, which grouping is described here as *orthogonal* to the model packaging and also *parasitic*, in the sense that it does not disrupt other aspects of the underlying model.

3 The use of stereotype keywords to indicate operation wavelength ranges of the telescopes is merely a communication convenience for this proposal; one could use instead, for example, a SysML ValueType enumeration and a value property on an abstract Telescope base block, redefined in inheriting blocks.

2) DISCLAIMER concerning the use of the UML Artifact in wrapped element examples

The UML Artifact is not currently part of the UML4SysML intersection. The author has long advocated for the acceptance of the UML Artifact in SysML for representation of external source document artifacts, which may provide source text for elicitation of model elements using graphical *Parsing Analysis*. Thus, the UML Artifact is used freely in some examples here.

If a modeller is working with a tool that supports UML and SysML in parallel – and if that modeller does not have the requirement of consistent exchange of compliant SysML models between tools – the use of Artifacts in SysML models on real-world problems is in practice of little consequence.

3) DISCLAIMER concerning the use of the UML Component in placeholder wrapper examples

This proposal includes examples of graphical logical wrapping by re-appropriation of some of the logical and graphical grouping capabilities of the UML2 Component, which re-appropriation the author has used with effect previously to support a wide range of software engineering and systems engineering modelling. These examples are to be considered illustrative placeholders for the purposes of this proposal only, and the approach is not advocated long term by the author.

Note that the UML2 Component is not part of the UML4 SysML intersection of supported UML elements in SysML. The UML2 Component is also limited in its graphical grouping capabilities to a small (useful but limited) subset of UML elements, and can't be used in all diagram types.

Indeed, a main point of the proposed LogicalWrapper is to overcome the limits of the UML2 Component under this re-appropriation, and to be ultimately fully adopted also by SysMLv2, without reference to elements outside the UML4 SysML intersection.

In some tools, whether a UML Component symbol that appears to graphically contain an element symbol for a realized Classifier in fact "steals ownership" depends entirely on manipulations by the modeller in the tool, i.e. one can't infer ownership from the graphical containment. For clarity, in most diagrams in this proposal the ownership of wrapped elements is explicitly displayed.

In Figure 3.1, a re-appropriated UML Component is used to explore logical grouping of some SysML Blocks. The ***realizations*** compartment is re-appropriated to show the logical members; the proposed LogicalWrapper would instead have a dedicated **members** property (much like the SysML ElementGroup) as shown in the mockup Figure 3.2.

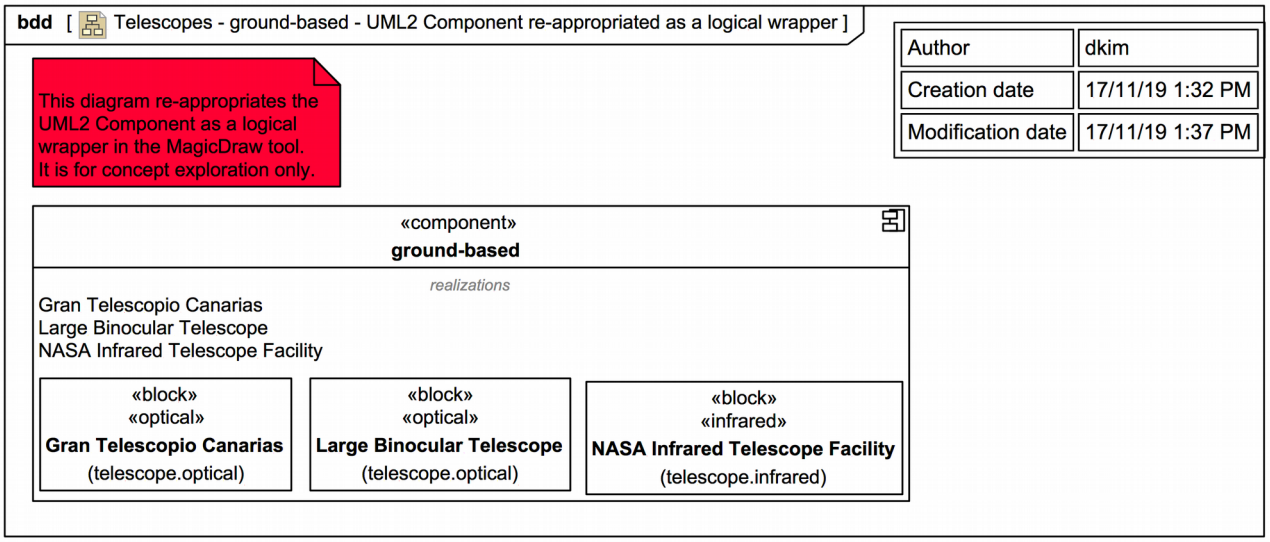

Figure 3.1 Re-appropriation of a UML2 Component to show logical grouping of SysML Blocks

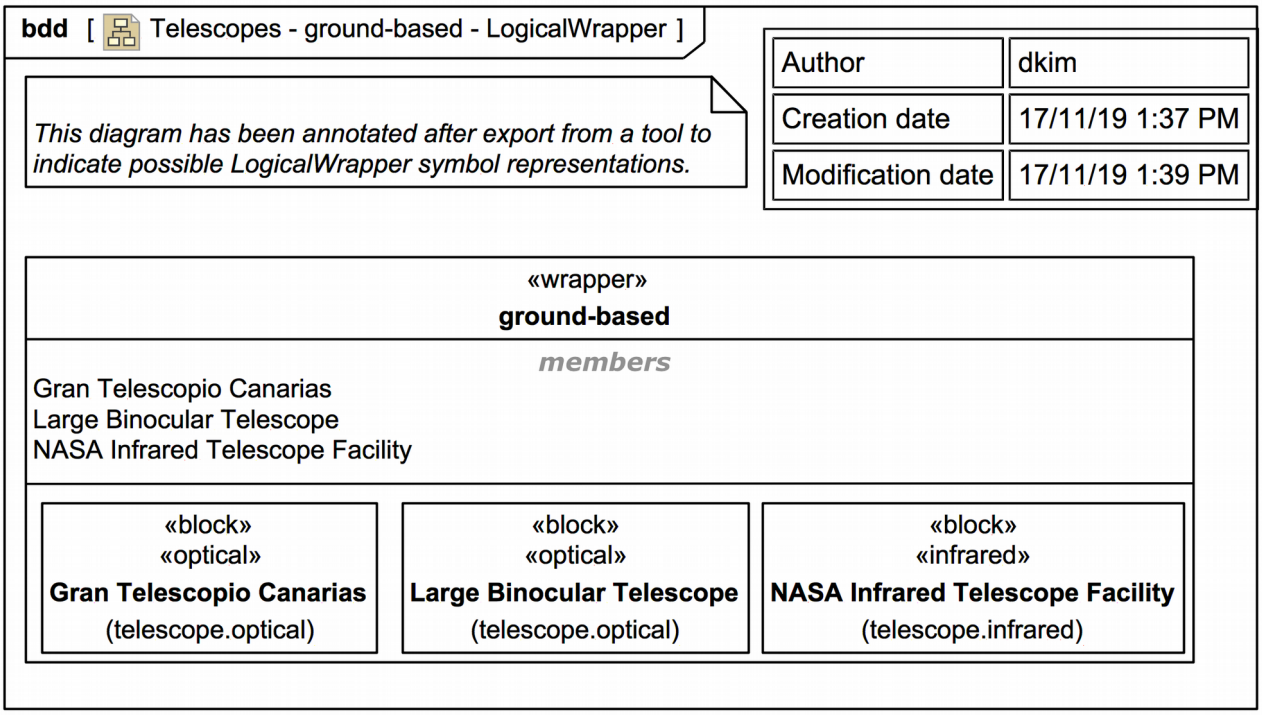

Figure 3.2 Mockup of a LogicalWrapper grouping some SysML Blocks as logical members

The mockup Figure 3.2 shows how the LogicalWrapper would have a lower compartment (optionally demarcated) for logical, graphical, grouping of member elements; it is recommended in this proposal that this logical grouping compartment not by default have a label. The diagram also shows a ***members*** listing compartment, the display of which should be optional.

4) Elements that are logically and graphically wrappable in a UML Class Diagram

Figure 4.4 illustrates elements that should be logically and graphically (visually) wrappable in a UML Class Diagram, including: Class, DataType, Enumeration, PrimitiveType, Signal, Interface, UseCase, Artifact, Signal, Comment, Component, Actor, Collaboration, Node, Package, Model, Profile, and InstanceSpecification (including instances of Node and Artifact). A LogicalWrapper may also wrap (nest) a child wrapper.

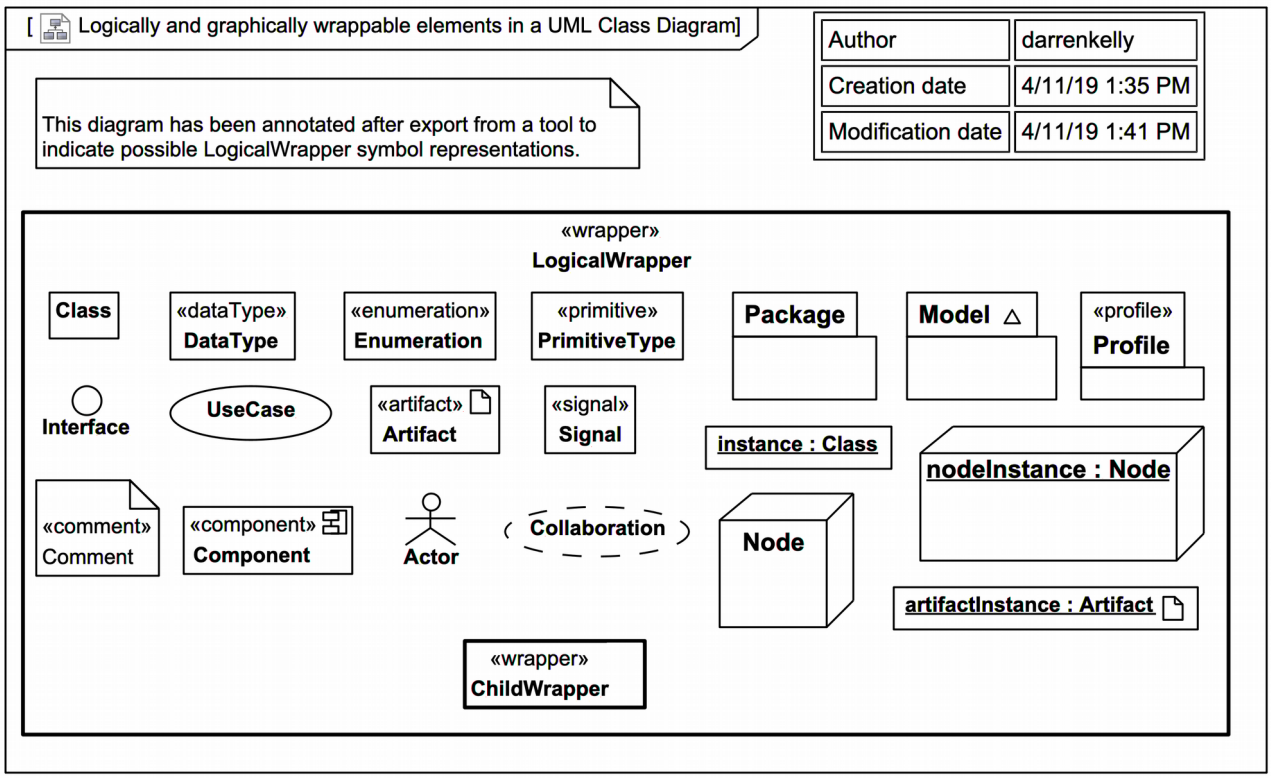

Figure 4.4 Elements that the proposed LogicalWrapper can wrap in a class diagram

The display of all valid relationships between wrapped elements should also be supported, as well as the display of any relationships between wrappers.

The extension to wrapping of SysML elements in a Block Definition Diagram (BDD) is considered self-explanatory; Class becomes Block, DataType becomes ValueType, etc.

Please note that – while the LogicalWrapper lends itself well to use in the UML Class Diagram and SysML Block Definition Diagram (BDD) – the proposed LogicalWrapper is for use in any UML or SysML diagram that supports element symbols that can be graphically contained in a rectangle.

Proprietor: Dr Darren R C Kelly, **ABN:** 67 677 268 579, **Post:** PO Box 1816, Bondi Junction, NSW 1355, Australia
Phone/Fax: +61 2 9386 0090, **Mobile:** +61 405 029 008, **Email:** info@webel.com.au, **Web:** https://www.webel.com.au

5) On graphical containment by a LogicalWrapper

The ability to graphically contain elements using a LogicalWrapper can be extremely useful to a modeller for compact layout of grouped elements in diagrams (beyond that provided by layout features in modelling tools), and for promoting workflow. The visual organisation of logically grouped elements via graphical containment can also help make diagrams easier to read.

When used in combination with nesting of LogicalWrappers, very powerful and efficient modelling and diagramming idioms and progressive modelling workflows can be achieved.

(Hierarchical nesting of LogicalWrappers used as graphical *Parsing Analysis* containers – where the context for logical wrapping is provided by text extracts/snippets sourced from authoritative domain documents – can be used to aid interpretation of semantically related text extracts, but the details of this application are beyond the scope of this proposal here.)

5.1) LogicalWrappers can be nested

A LogicalWrapper can nest another LogicalWrapper graphically, which implies directly that the nested inner wrapper is a logical member of the outer wrapper. A simple example with one level of nesting is shown in Figure 5.1.1.

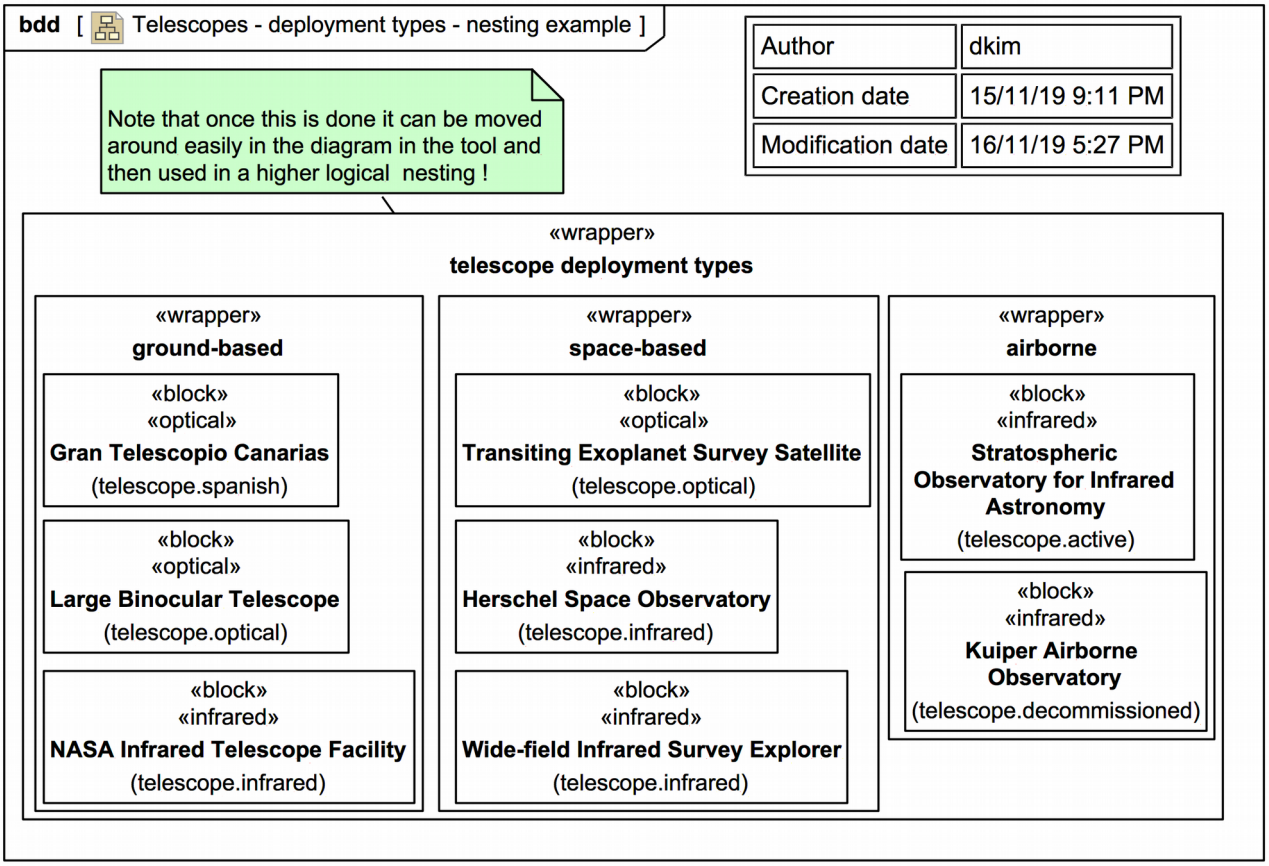

Figure 5.1.1 Example of nesting of LogicalWrappers

The ability to nest wrappers graphically is a powerful, time-saving diagramming device in tools:

1. It promotes a workflow for creating structured diagrams with nested logical contexts.
2. It affords layouts that are far more powerful than automatic layouts for many cases.
3. As a bonus, it can be used to create robust grouping grids behind featured elements.

When combined with un-wrapping (see section 8.1) and relationships between wrappers (see section 8.2) it can help support a form of semantic modelling, and aids the break-down of text extracts when combined with *Parsing Analysis*.

6) Logical wrapping of a subset of Properties within an owning context

The need for a modeller to logically and visually group subsets of Properties within an owning context in Composite Structure Diagrams can arise. The desired logical grouping may be independent of the AggregationKind of the properties or of the Type of the Properties.

For example, as shown in the left diagram of Figure 6.1, in a SysML model of a Newtownian reflecting telescope one may wish to group part property and value property symbols in an Internal Block Diagram (IBD) according to whether they are involved with optical systems or electrical systems – indicated additionally here for clarity with the keywords «optical» and «electrical» for stereotypes applied to the underlying Types (so-called *secondary stereotypes*).

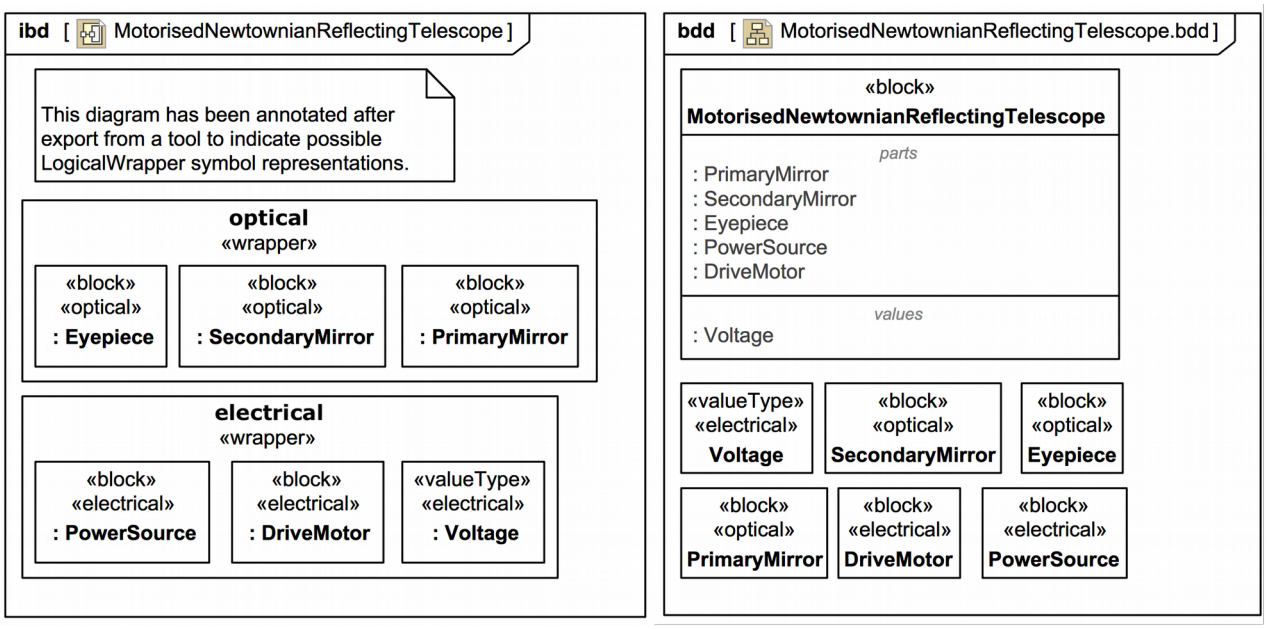

Figure 6.1 Logical and graphical grouping of Properties by LogicalWrappers in an IBD vs BDD

Note that the LogicalWrappers in the IBD of Figure 6.1 are completely "parasitic"; they have no effect on the ownership of the Properties (as seen in the BDD), they merely serve to offer a logical grouping along with a graphical, visual containment that reinforces this logical grouping.

7) On logical grouping with the SysML ElementGroup vs the LogicalWrapper

The SysML1.6 ElementGroup affords powerful "parasitic" logical grouping of a wide range of model elements, and tracking of logically grouped elements via its **members** attribute. It has a **/criterion** attribute (derived from the Comment **body**), which can be interpreted as a logical

grouping context if desired; in many cases, displaying the bound Comment **body** of an ElementGroup suffices for communication in diagrams.

This mechanism can, for example, be leveraged for traceable elicitation of model elements from sourced text extracts using a *Parsing Analysis*[4] approach, as illustrated in Figure 7.1.

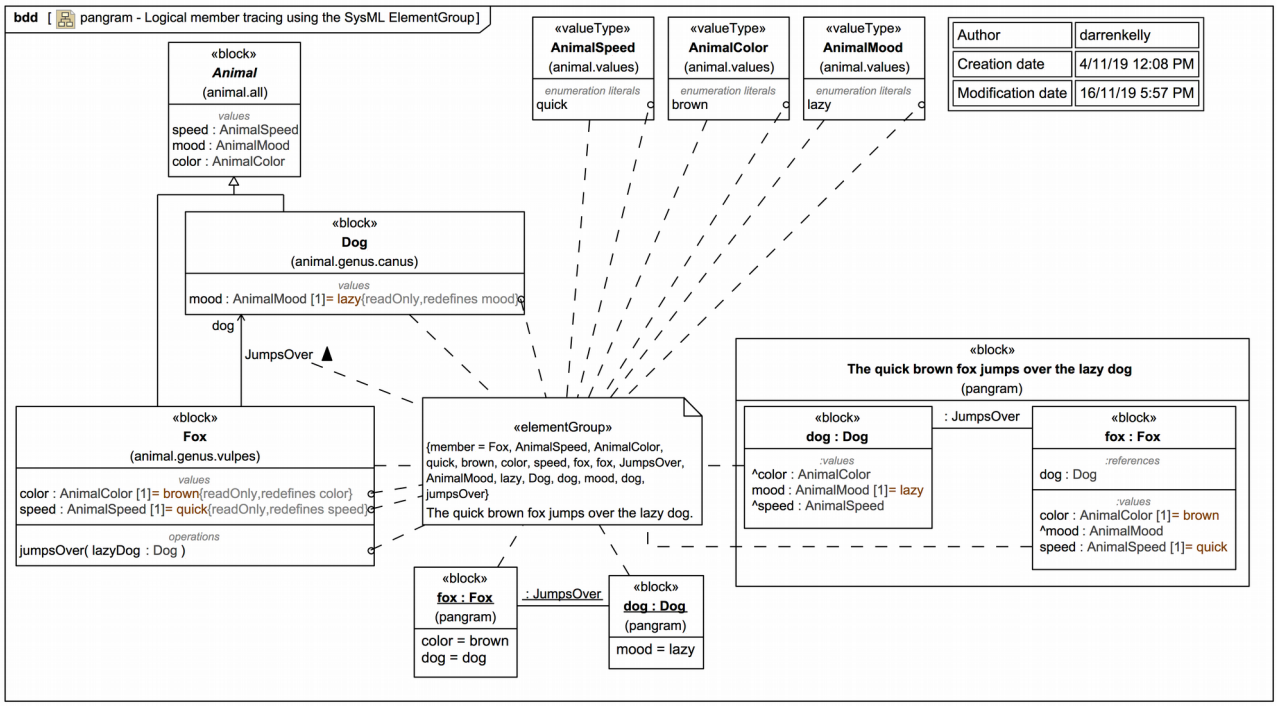

Figure 7.1 Logical tracing of a wide range of types of SysML elements using an ElementGroup

4 In the author's Webel *Graphical Parsing Analysis* recipe for SysML, the SysML ElementGroup is in fact extended with a custom stereotype that additionally carries a reference to a stereotyped Artifact representing an authoritative domain document from which extracts (called "snippets") are sourced, and from which traceable model elements are elicited. An example profile is given in Section 11 (Appendix A).

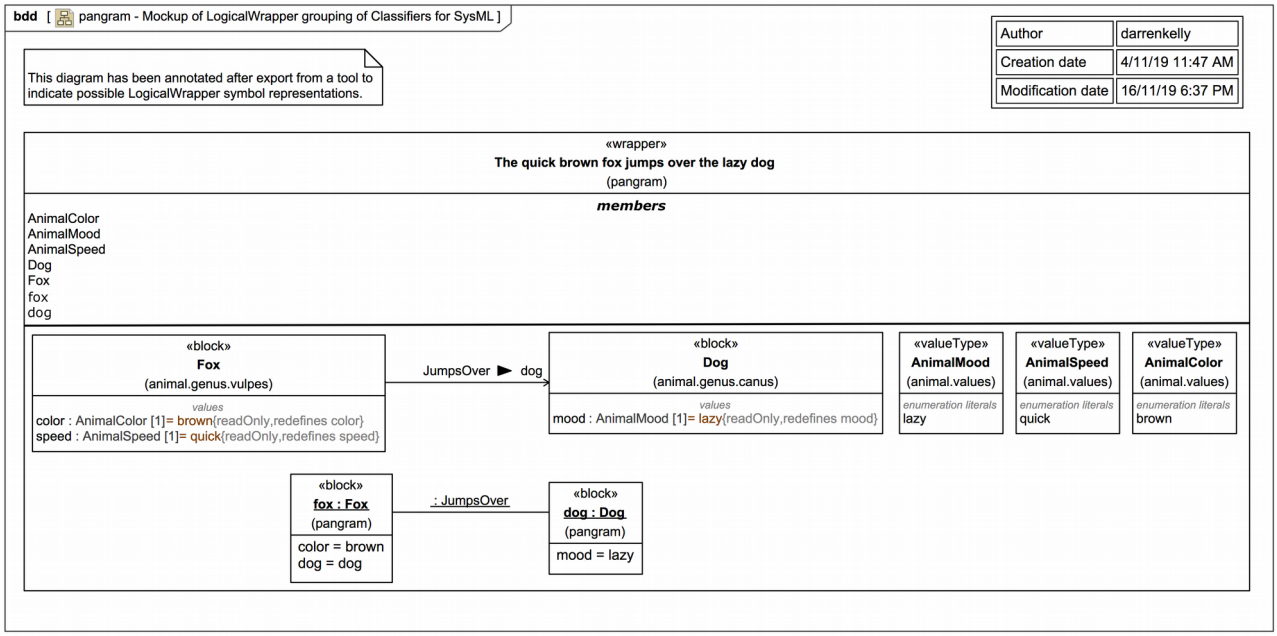

Figure 7.2 Attempt at elicitation of model elements using Parsing Analysis with a LogicalWrapper

Figure 7.2 shows an attempt to likewise elicit model elements using graphical *Parsing Analysis* using a LogicalWrapper. It does not have clear tracing to the Properties of the Classifiers (SysML Blocks in this case), nor to the enumeration literals, nor the 'JumpsOver' Association. Compare this with the SysML ElementGroup, for which traceable *handles* can be drawn to also easily collect Properties – as well as many other element kinds – as members, as shown in Figure 7.1.

The SysML ElementGroup is better suited to creating logical groupings for *Parsing Analysis*.

However, while the SysML ElementGroup may be superior to the proposed LogicalWrapper in flexibility of logical grouping, and serves its intended purpose well, it does not offer graphical or visual containment, and is not relatable, which makes it unsuitable for some modelling tasks.

8) A LogicalWrapper is relatable

One of the most important powers of the LogicalWrapper is that (unlike the SysML ElementGroup) it is relatable, and relationships between wrappers – and between wrappers and other element types – can be drawn in UML and SysML diagrams, and traced in the underlying model. (The author has used this device with compelling effect previously with stereotyped, re-appropriated, UML Components.)

8.1) Graphical un-wrapping and the Wraps relationship

The Wraps relationship may be used instead of graphical containment in diagrams to indicate logical membership of elements in a LogicalWrapper. This process is called here *un-wrapping*.

The Wraps relationship is a directional binary relationship from a LogicalWrapper to a wrapped Element, so the UML Dependency is a candidate metaclass for extension.

In Figure 8.1.1 a wrapper with the name 'space-based' has Wraps relationships to each of its three logical member Blocks, and a wrapper named 'telescope deployment types' has a Wraps relationship to the wrapper 'space-based' by virtue of a nesting of the wrappers (see Figure 5.1.1).

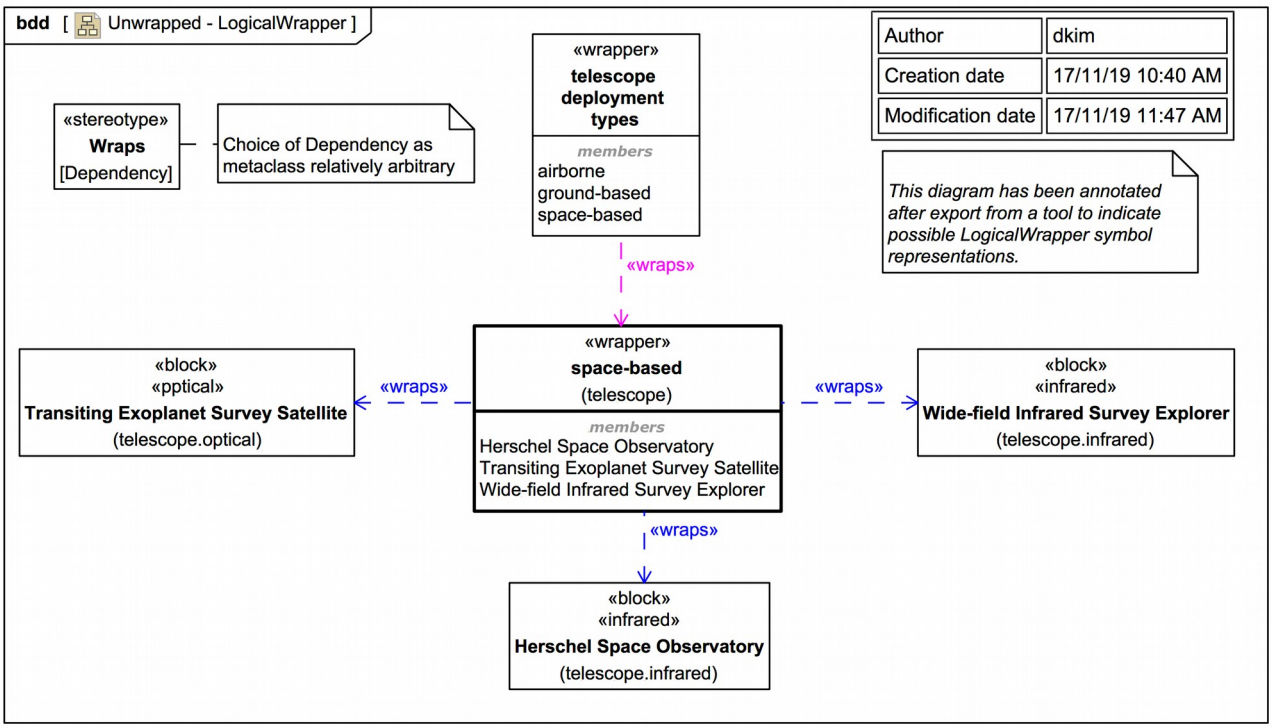

Figure 8.1.1 The un-wrapped 'space-based' wrapper with Wraps relationships

As is done for many SysML relationships, a corresponding derived attribute **/wrappedBy** could be be defined on NamedElement, as illustrated in Figure 8.1.2 using both a compartment and a SysML-style callout.

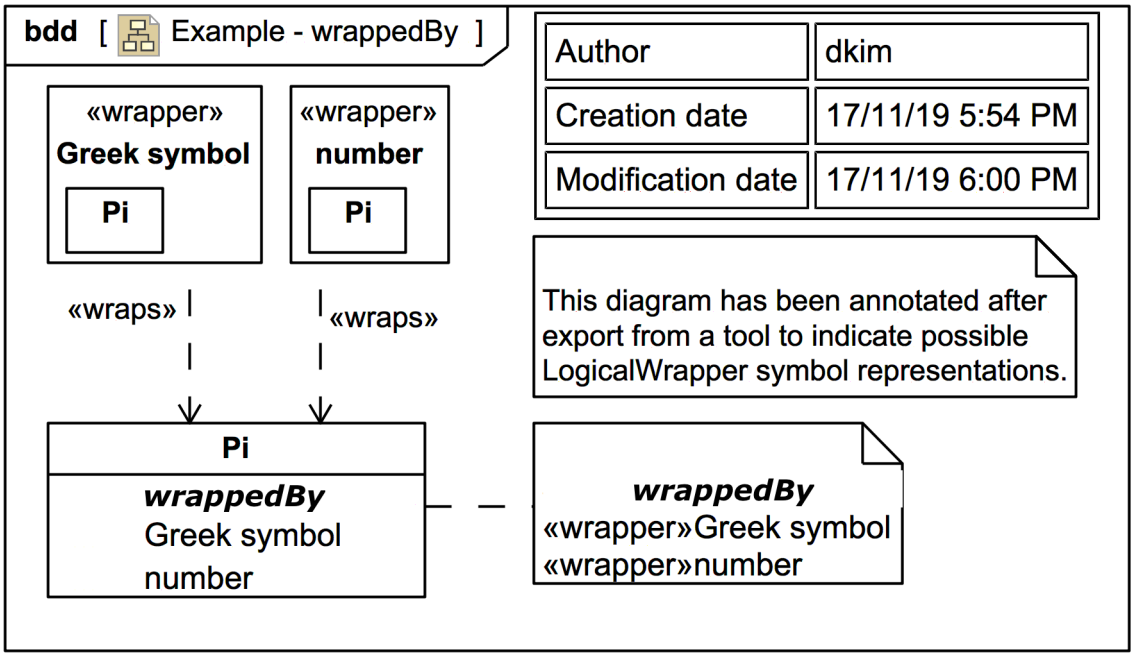

Figure 8.1.2 Compartment and callout for wrappedBy attribute vs Wraps relationship

8.2) Relationships between LogicalWrappers

The example in Figure 8.2.1 shows how relationships between LogicalWrapper elements can be used to analyse text extracts[5], in this case to illustrate a claimed «contradiction», while also offering illustration of some elicited model elements. While this example is relevant to graphical *Parsing Analysis*, the basic principle can be used for quite general UML and SysML modelling.

5 In the author's full *Webel Parsing Analysis* recipe, when text extracts are quoted a tagged value of a custom stereotype extending the LogicalWrapper is used to trace such quoted text to a stereotyped source Artifact.

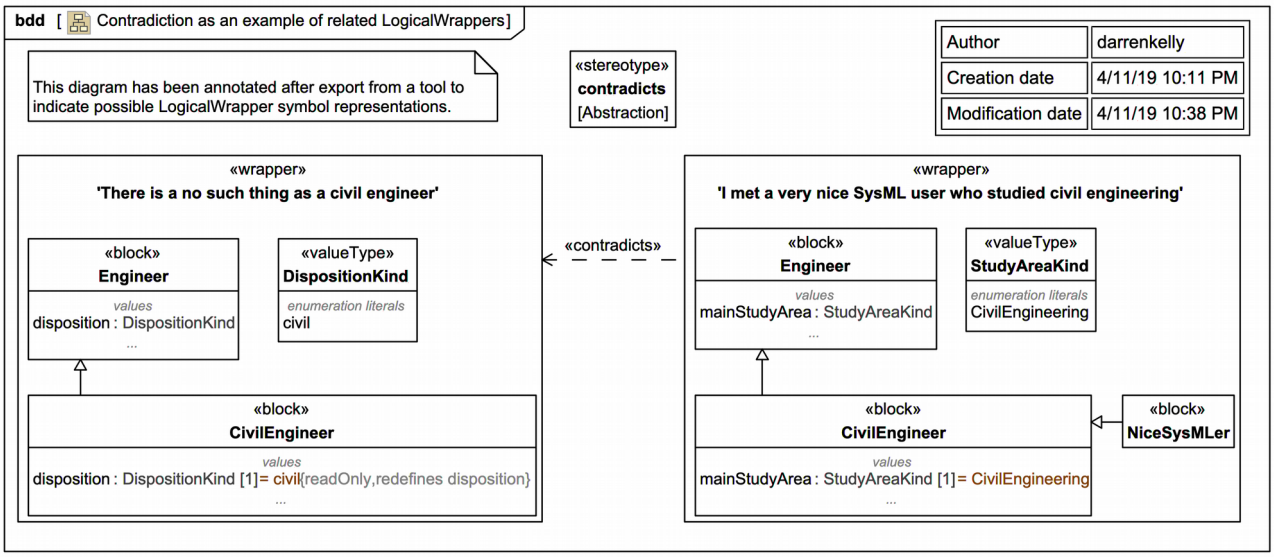

Figure 8.2.1: A stereotyped Relationship between LogicalWrapper elements for analysing text

8.3) LogicalWrappers are not Types and should not participate in Associations

Situations can arise where one might wish to establish a reciprocal relationship between wrappers. Managing this with matching pairs of directional binary relationships introduces a Single Source of Truth maintenance problem.

In Figure 8.3.1, a custom Complements stereotype is introduced and applied to reciprocal Dependency pairs to indicate that 'space-based', 'ground-based', and 'airborne' telescopes all complement each other, but changing this assertion on just one Dependency in each pair would break the logic. One might therefore be tempted to address such situations using a stereotyped symmetric Association between LogicalWrappers, but this would require the LogicalWrapper to be a Type, which is not the intention of this proposal, and would greatly complicate its specification.

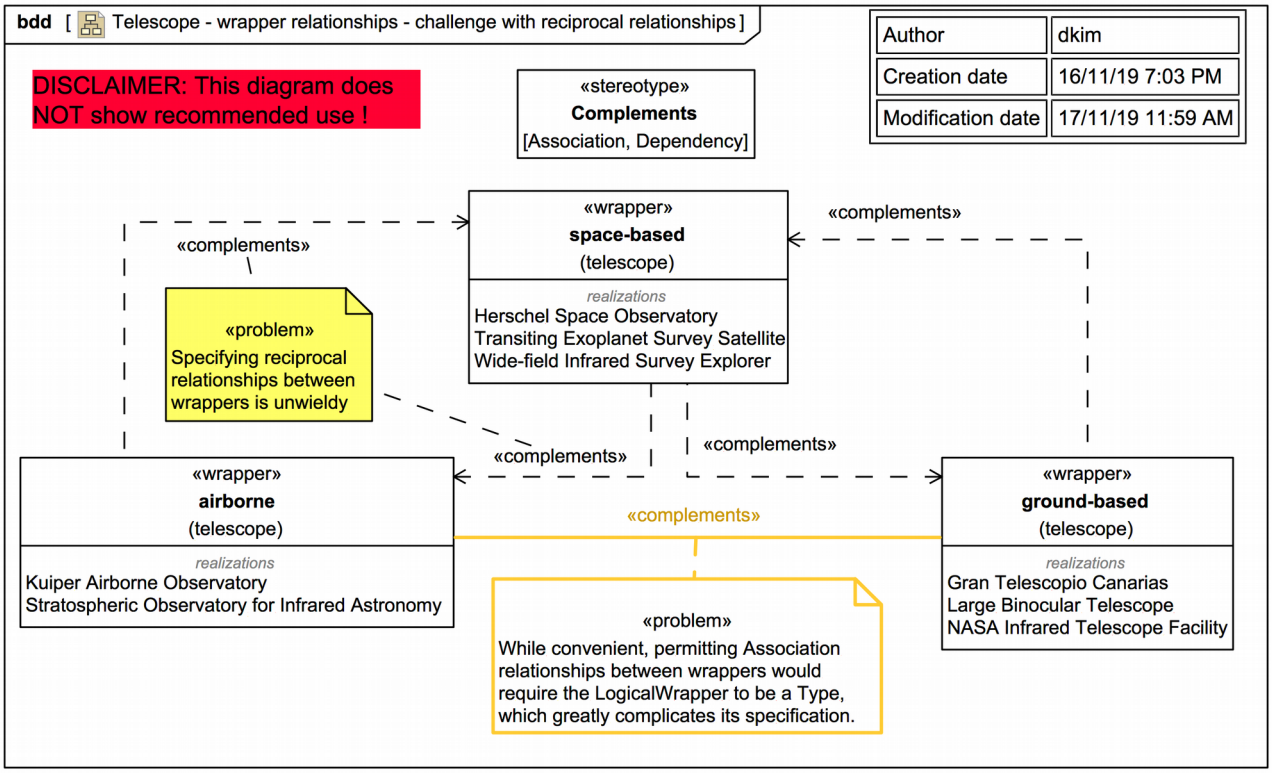

Figure 8.3.1 Challenges when modelling reciprocal relationships between wrappers

9) Variation point: whether a LogicalWrapper can own another LogicalWrapper

It has been asserted that a LogicalWrapper may not own any element that is not itself a LogicalWrapper. While it could be useful to permit a LogicalWrapper to at least own another LogicalWrapper, this would greatly complicate its specification.

If a LogicalWrapper is not permitted to own any element of any kind it suffices to make it an extension of NamedElement.

If a LogicalWrapper is permitted to own child LogicalWrappers it would have to at least be an extension of Namespace. The question also then arises whether graphical containment of the symbol of one LogicalWrapper by the rectangular symbol of another LogicalWrapper in diagrams implies ownership (which would be counter-intuitive given that it does not imply ownership for any other element kind).

This matter is indeed already a concern for the UML2 Component in the MagicDraw tool, in which the ownership of a UML2 Component by another that graphically contains it depends on the history of manipulations in the tool[6]:

- If you move one Component's symbol inside a second Component's symbol in a diagram, the ownership of the first Component in the underlying model does not change, and a ComponentRealization is created. This is equivalent to purely logical membership.
- If you move one Component in the model browser (containment tree) under another Component, the second component indeed steals ownership and becomes the owning parent of the first; no ComponentRealization is created.

In any case, one can't in the MagicDraw tool draw any conclusions in diagrams about the ownership of a Component that is graphically contained in the symbol of another Component from that graphical containment alone, as the ownership depends in subtle and complex ways on the manipulation history in the tool, as demonstrated in Figure 9.1.

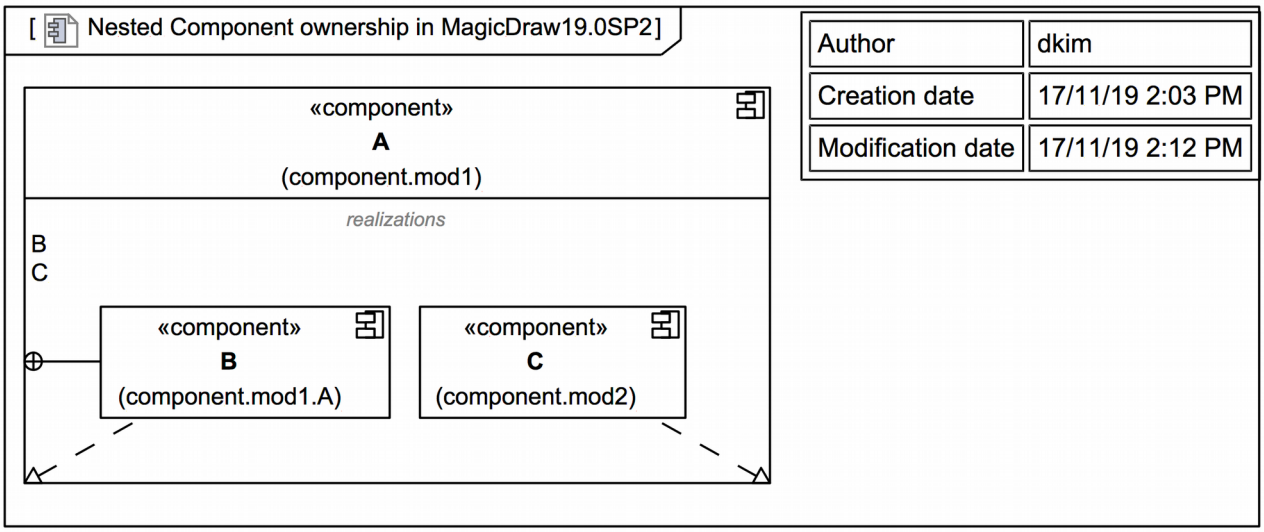

Figure 9.1 In MagicDraw ownership of Components can't be inferred from graphical containment

Specification documents such as for UML and SysML do not usually concern themselves with tool manipulations. However, a specification of the LogicalWrapper would have to clearly state whether one can infer ownership of graphically contained LogicalWrappers. The simplest approach is to not permit any form of ownership of any elements by a LogicalWrapper (in keeping with its intended purpose as a parasitic logical wrapper), even if this is at the slight cost of less flexibility in organising ownership of the LogicalWrappers themselves.

6 These tool manipulations are explored live in an Appendix of the screencast video referenced in Section 10.

Proprietor: Dr Darren R C Kelly, **ABN:** 67 677 268 579, **Post:** PO Box 1816, Bondi Junction, NSW 1355, Australia
Phone/Fax: +61 2 9386 0090, **Mobile:** +61 405 029 008, **Email:** info@webel.com.au, **Web:** https://www.webel.com.au

10) Accompanying screencast video resource

This proposal document is supplemented by a detailed annotated, narrated, screencast video resource available for online viewing at: https://vimeo.com/373707827

The video consists of a main tool-independent presentation on the proposed features of the LogicalWrapper, and some Appendix items with live explorations in the MagicDraw tool (including illustrating the re-appropriated UML2 Component strategy, and the power of wrapper nesting).

11) Appendix A: Webel profile for Parsing Analysis with the SysML ElementGroup

Figure 11.1 shows a typical profile in the MagicDraw tool as used in the Webel *Parsing Analysis* recipe for SysML with the Snippet extension of the SysML ElementGroup and the ParsingSource extension of the UML Document standard profile extension of the UML Artifact metaclass - in full awareness that the UML Artifact is not officially supported by SysML1.6, but is nevertheless extremely useful and can be used largely without issue parallel to SysML in the MagicDraw SysML and Cameo Systems Modeler tools. A simplified usage example is given in Figure 11.2.

Compare with the OBSOLETE profile for the re-appropriated UML2 Component in Appendix B.

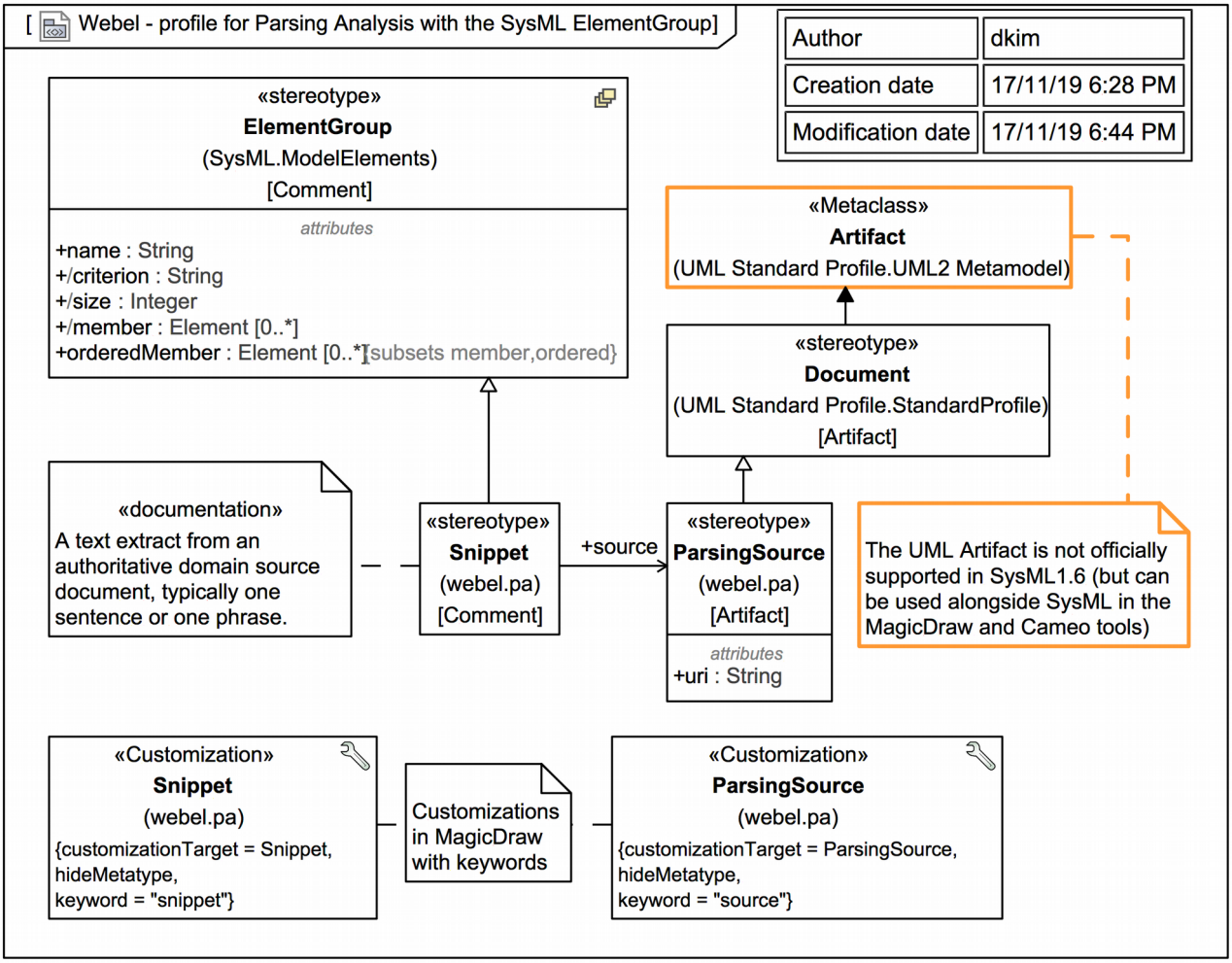

Figure 11.1 Typical profile for the Webel Parsing Analysis recipe using the SysML ElementGroup.

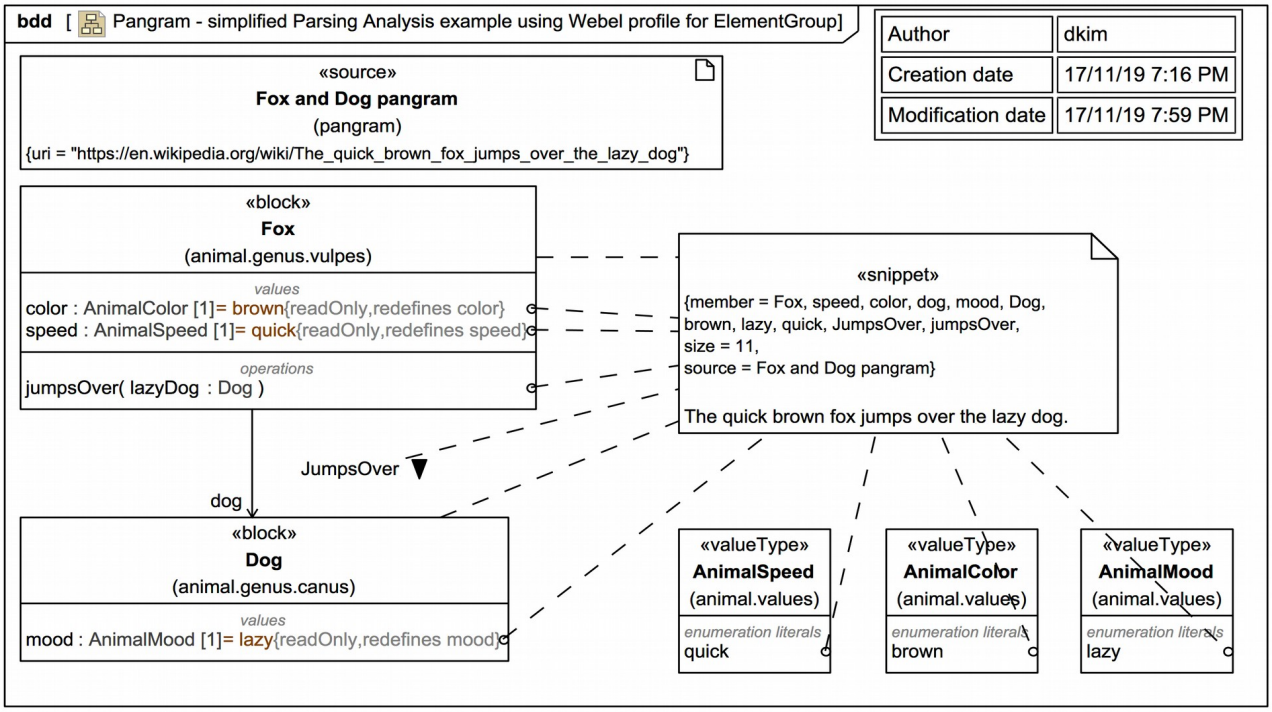

Figure 11.2 Example of Parsing Analysis using the Webel recipe for the SysML ElementGroup

12) Appendix B: OBSOLETE Webel profile for Parsing Analysis with the re-appropriated UML2 Component

The adoption of the proposed LogicalWrapper in UML and SysML would render this re-appropriation of the UML2 Component completely redundant. The approach shown here has now been largely replaced for the purposes of the Webel Parsing Analysis recipe by use of the SysML ElementGroup.

Figure 12.1 shows a typical profile in the MagicDraw tool as used in the (now obsolete) Webel *Parsing Analysis* recipe for UML with the ParsingWrapper extension of the re-appropriated UML2 Component and the ParsingSource extension of the UML Document standard profile extension of the UML Artifact metaclass - in full awareness that the UML Artifact is not officially supported by SysML1.6, but is nevertheless extremely useful and can be used largely without issue parallel to SysML in the MagicDraw SysML and Cameo Systems Modeler tools. A simplified usage example is given in .

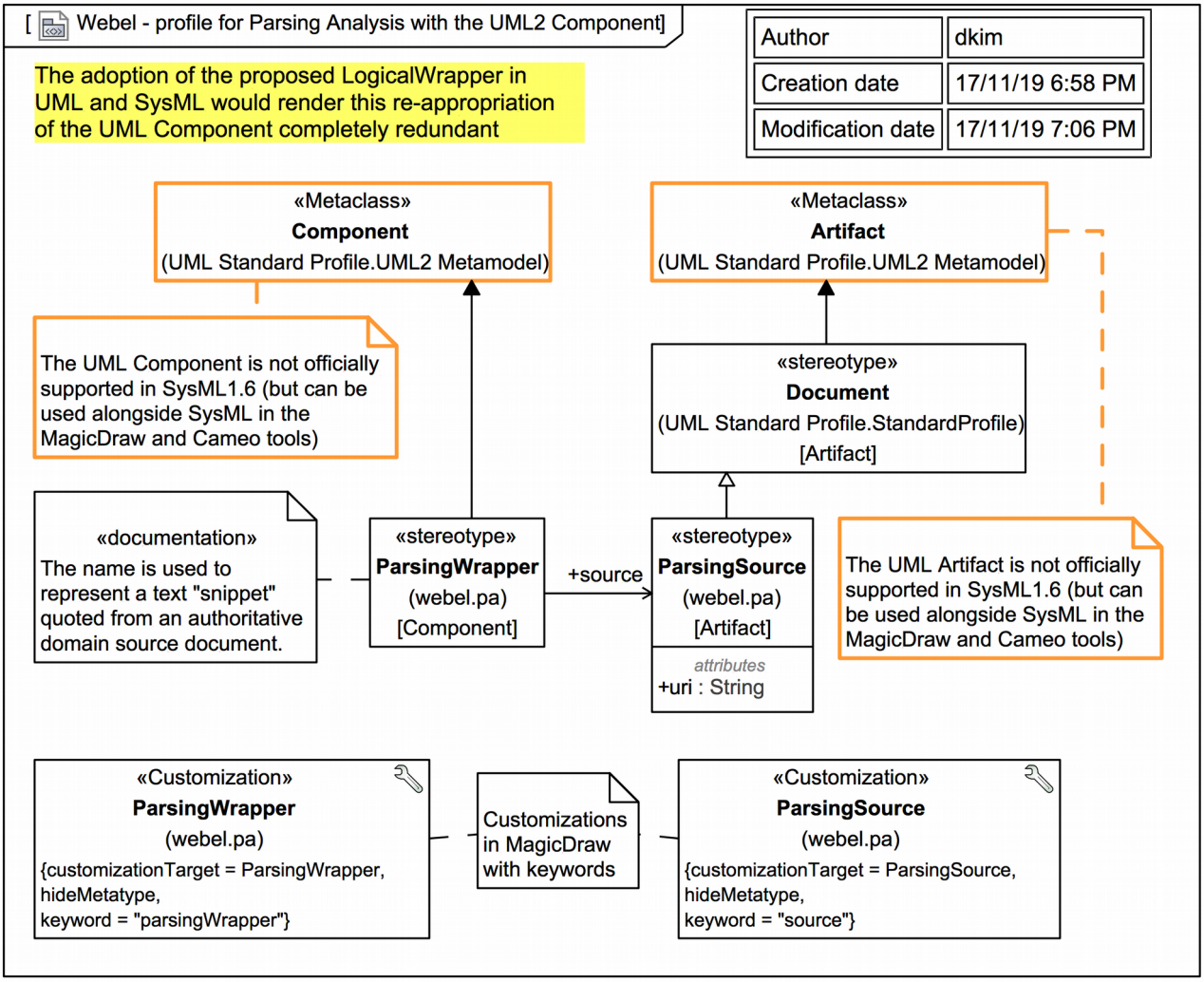

Figure 12.1 OBSOLETE profile for the Webel Parsing Analysis recipe using the UML Component.

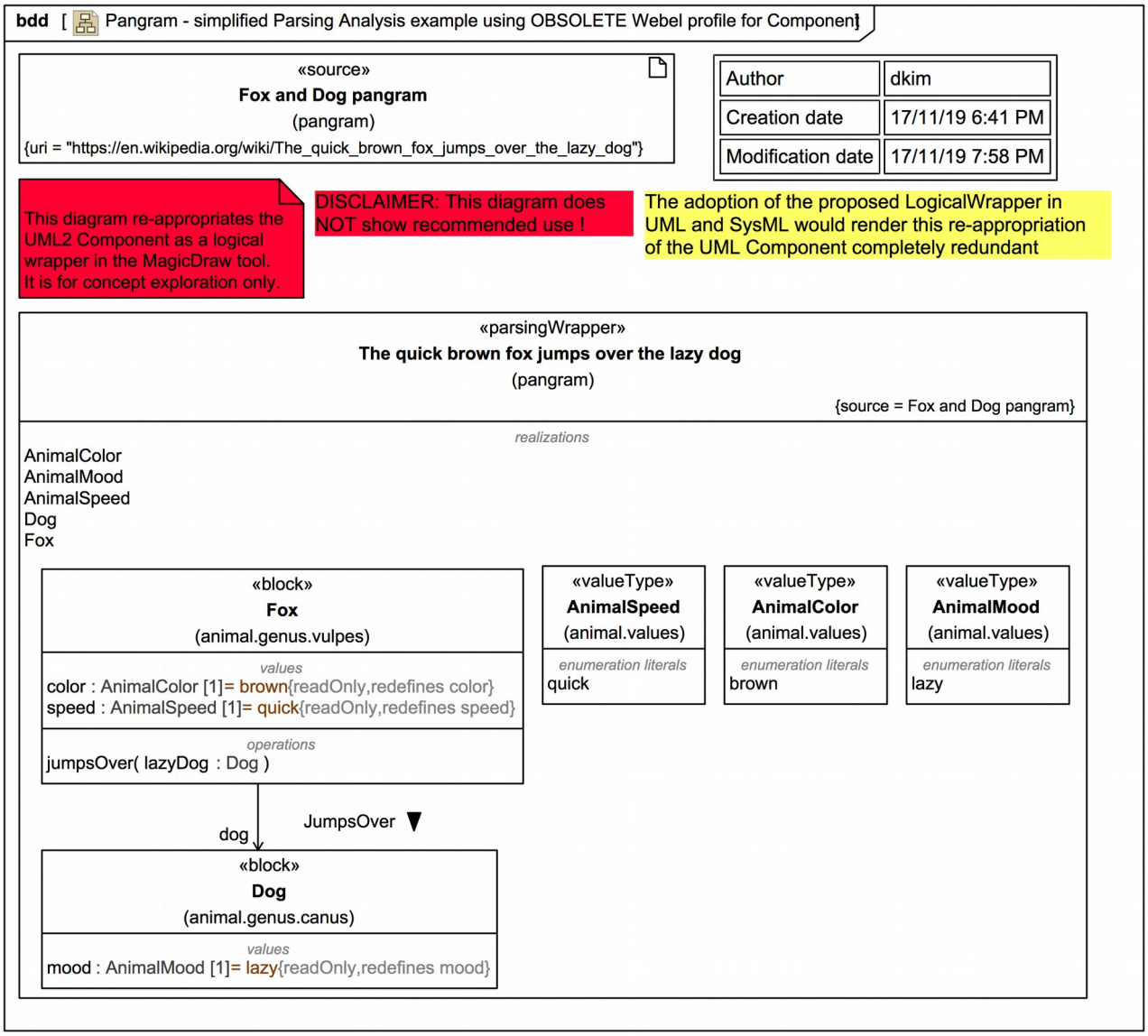

Figure 12.2 Example of Parsing Analysis using the OBSOLETE Webel recipe for the UML Component