On UML's Composite Structure Diagram

Ian Oliver, Vesa Luukkala

Nokia Research Center
Helsinki, Finland
{ian.oliver,vesa.luukkala}@nokia.com

Abstract. The composite structure diagram and related notions have been introduced into UML2.0 to supplement already existing artifacts such as classes. However the usage of these constructs by engineers and/or modellers is not always in the spirit of inventors of these constructs. A number of additional interpretations develop which are not always consistent with the intended usage of the structure nor with the language itself. Understanding these additional usages assists in understanding areas of ambiguity, extension, inconsistency and the future development of the language.

1 Introduction

The composite structure diagram's and related structures' uses and semantics are well described in [1–3] while the notions of composition are adequately described in [4, 5]. Its function is to extend the modelling capabilities of the UML beyond that of classes and their relationships and is primarily aimed to assist the modelling of the internal structures of classes with a more well defined notion of decomposition. Similar notions exist in methods such as ROOM [6] (capsules) and languages such as SDL [7] and SysML [8] for example.

As tools become more UML compliant and support more UML constructs, engineers and/or modellers start to use these additional constructs. The effect of this is that the semantics of these constructs is often learnt through an implicit process based around the name of the construct and what the tool appears to allow; the semantics are often based on the engineer's expectations and *perceived meaning* [9] rather than on the actual, intended semantics. This we term as *implicit profiling* of the UML; engineers or modellers tend to invent their own subtle variations on the semantics (and syntax) of the UML during their work.

In this paper we discuss the nature and experience of the use of the composite structure diagram as introduced into the Unified Modelling Language version 2.0 [10]. The usages described here are based upon primarily the engineer's expectations and usages and so do not necessarily conform to the original, intended meanings. As additional interpretations have emerged, we have at times attempted to standardise these so that the diagram is used in a clear and consistent manner.

It is important to recognise and understand how constructs such as the composite structure are being used in reality and how close (or far) these usages are from or to the inventors' aims. If these facts are understood then the development of languages such as the UML becomes more relevant to the current practise. We have collected this body of results over a number of years of working with internal consulting projects utilising the UML and its various profiles within Nokia [11–13]. The range of projects has been from enterprise database systems to embedded, real-time components.

2 Classes and Composite Structures

In this section we outline the relationship between classes and composite structures. This is made in a similar way to that seen in [14] with their description of diagram reconstruction between the class, sequence and state diagrams.

In all of these cases we assume the composite structure diagram to be complete in that it shows all the information and the class diagram to be the smallest class structure that admits the given composite structure (cf: [15]).

2.1 Basic Structures and Relationships

In figure 1 we see the trivial situation where a trivial composite structure diagram consistent of a *context A* implies the existence of class A. All diagrams of this nature in this document have the composite structure diagram on the left and the implies class structure on the right.

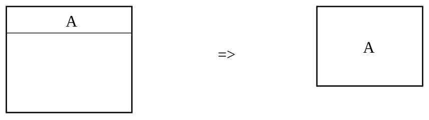

Fig. 1. Trivial Case

The case in figure 2 denotes the situation where the composite structure of A contains a number of B parts. This implies existence of classes A and B with some kind of relationship. Once a relationship is instantiated then the two objects may communicate by calling each other's operations.

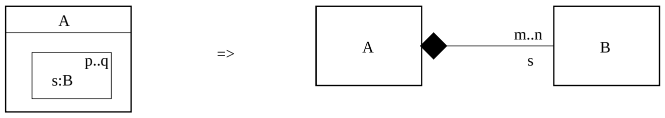

Fig. 2. Simple Composite Structure

In this case, the part with type B denoted by the solid rectangle is part of A's composition, thus the relationship between classes A and B must be a composition (black diamond in the class diagram). The choice of black diagram or a composition relationship is made because of the strong(er) semantics for this concept and that parts [16] are unsharable and lifetime owned by the parent object.

The composite structure must conform to the constraints set out in the class model of the description. The properties of the composite structure are a refinement (in the mathematical sense) [17] of the properties described in the class diagram, or in other words, the state space admitted by the composite structure is smaller than or equal to the state space admitted by the class diagram.

It is stated that the part(s) B are named s:B which states that the parts are referenced through a relationship B and that the number of parts (p..q) must be within the multiplicity bounds stated in the class diagram for this role, ie: m..n, the following invariant holds:

$$\begin{aligned}&m..n \sqsubseteq p..q \Rightarrow\\&\quad (0 \leq p \leq q \quad \wedge \quad q > 0) \wedge (0 \leq m \leq n \quad \wedge \quad n > 0) \quad \wedge\\&\quad (p \leq m) \wedge (q \leq n)\end{aligned}$$

Note that the naming of the part (in figure 2's composite structure diagram) implicitly denotes a connection while in other more complex cases (as we shall see) the connection must be explicitly drawn.

Composite structure diagrams can also imply 'normal' relationships using the dashed rectangle notation denoting a referenced object as can be seen in figure 3.In figure 4 we can see a more complex example of this involving a collaboration between a number of objects.

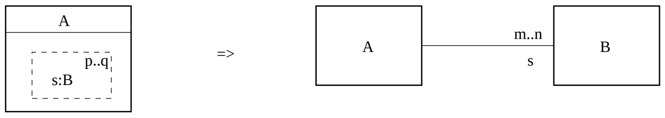

Fig. 3. Referenced Part

Similarly to the case in figure 2 the nature of the relationship in the class needs to be considered. We suggest as can be seen in figure 3 that a connection to a referenced part implies a normal association or relationship in the class model. This is the weakest possible interpretation of this kind of connection. The use of aggregation or white diagram we are against due to its weak semantics; of course the modeller is free to use this. Similarly, directionality of the relationship can not be inferred, unless one explicitly restricts the ability of parts to call the parent.

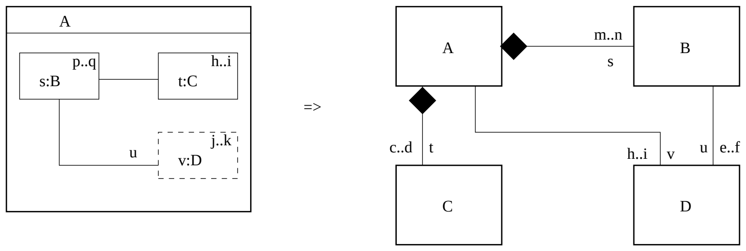

Fig. 4. Complex Structure

The multiplicities of the parts and their relationships in the composite structure is restricted that if a relationship is denoted in the composite structure then it must exist at the class level. In figure 5 we can see a simple composite structure where part s:B has a multiplicity of 1..2 and part t:C has a fixed multiplicity of 2.

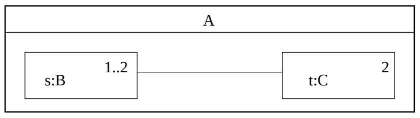

Fig. 5. Multiplicities and Relationships

When this composite structure is instantiated we obtain two possible collaborations of objects as shown in figure 6. The patterns for this can be seen in [18].

We have not discussed the effects of generalisation and specialisation here as we have just wished to show the minimum implied class structure. Of course the presence of other classes in a given generalisation/specialisation hierarchy does introduce more possibilities with the mappings. We do not discuss these here but refer the reader to [19].

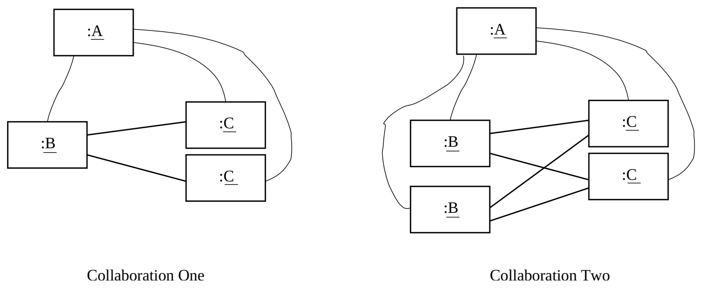

Fig. 6. Possible Instantiations

2.2 Port Based Connections

The connection between parts in a composite structure need not be restricted to being instances of relationships in the class diagram. A connection may also be made between port artifacts in which case the connection is used for send and receiving signals via these ports.

Figure 7 shows the trivial case where the composite structure diagram contains a port, while is thus also implied in the class model.

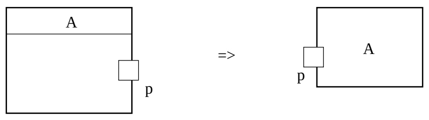

Fig. 7. Trivial Case with Port

A connection between ports, which we term 'wire', states that the ports can communicate; a signal sent from one port is directed along the wire to the receiving port and nowhere else. This is analogous to an operation call over a normal connection. Figure 8 shows this situation.

Two ports can only be (sensibly) connected if their interfaces match. While we have not drawn the required and provided interface graphical icons we can state that ports q and r have compatible interfaces; however this only provides the weakest of possibilities as we can not reason effectively about the actual interface specifications themselves nor the relationships between the interfaces in terms of potential generalisations/specialisation. If ports q and r were **not** connected then we would not be able to infer any details about their interfaces and possible or otherwise compatibility.

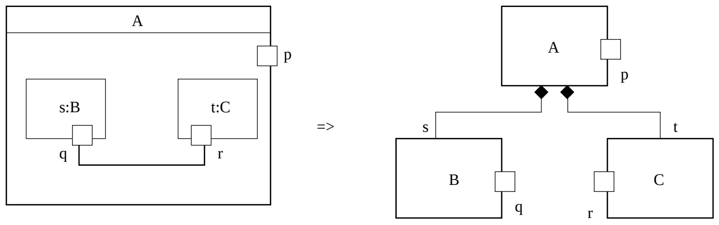

Fig. 8. Two Wired Ports

If in figure 8 port p has been connected to, say, port r, then we can guess that p's visibility is internal or private to class A and thus only visible to the ports of the composed parts. But, this fact is not guaranteed so we can not reasonably infer and properties regarding visibility: in this case port p on A is by default public and visible to both internal parts and external objects.

It is worth detailing the meaning of compatible interfaces further. Let us assume that in the example in figure 8 the interface situation is described as in the class diagram in figure 9.

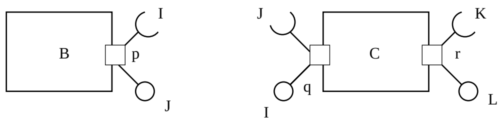

Fig. 9. Various Interface Combinations

Two ports can sensibly communicate signals defined in the required interface to the provided interface, if these interfaces are compatible. In figure 9 this is possible in both directions for ports p and q as each either requires or provides interfaces I and J.

A connection between, say, ports p and r for example would not be sensible - no messages would be understood and thus no meaningful communication may take place as neither the provided nor required interfaces match. In this case, drawing a connection between these ports on the composite structure diagram would be a violation of the typing constraints defined in this class diagram. If either K or L we related to I and J via generalisation or specialisation then some from of communication *is* possible. The introduction of the possibilities of generalisation and specialisation further complicate the matters with regards to the possible interpretations and permutations of interface combinations.

3 Usage of Composite Structures

In our work we have identified a number of common ways of utilising the composite structure diagrams in conjunction with the class diagram for describing various systems. While the UML2.0 Superstructure document describes the syntax and provides some informal semantics we have found it necessary to either enhance these descriptions and in some cases extend the meaning in order so that this particular UML artifact is used consistently and effectively. In this section we describe the uses of composite structure diagrams and their interpretations:

- Architecture Specification
- Scenario Descriptions
- Modes
- Operation Effects

3.1 Architecture Specification

In a number of tools, for example, Borland's Together Architect, the composite structure is used to describe the configuration of a class and associated parts. For example, figure 10 shows a composite structure diagram viewing a model. This model contains two classes A and B, where class A has three parts (named Part1, Part2 and Part3). The composite structure diagram shows class A's parts in completeness.

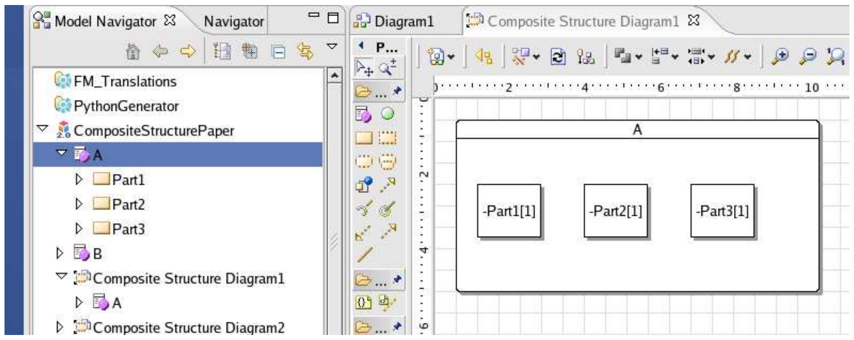

Fig. 10. Example Tool Implementation of a Composite Structure Diagram

This is per the UML2.0 semantics and each composite structure diagram acts as a *view* [20] onto the internal structure - the parts - of a given class; this view *may omit* entities that are inside the given class. The issue here is that the composite structure diagrams do not stand alone as being artifacts of the model and that the diagram does not necessarily show a complete view of the internal structure - this is something that is often misunderstood and not just with these particular diagrams but also with the class diagram.

For the practitioner, a common interpretation of a composite structure diagram, as currently implemented in the UML2.0, is that it describes the system as a whole or the 'architecture of the system.' A note must be made about the definition of 'architecture': while definitions do exist [21], it is often unclear to the practitioner (and sometimes even to the persons developing the methods) what architecture really is. Suffice, we often find that the term is misapplied to mean a top-down, functional decomposition of objects which omits the architectural definition or development step that requires human imagination and inventiveness to adhere to design requirements in favour of a mechanical process of simple decompositions.

This usage is often in conjunction with the concept of a 'system' class much in the same was as a top-capsule is used in ROOM or as the block interaction diagram is used in SDL. Almost invariably in these cases the class diagram (if ever expressed) appears as a top-down decomposition structure with an entry or start point in that system class.

While we have no overall problems with the interpretation, it is often the case that the composition mechanisms are misused such that they are confused with the development method. In this case, the system class is decomposed many times with the behaviour being moved down to the leaf nodes of the decomposition hierarchy. This is identical to the SDL usage where structural blocks are repeatedly decomposed to form a static hierarchy; SDL only allows behaviour at the leaf blocks, while the UML allows behaviour at all levels of decomposition.

An example of this is that a top-down, functional decomposition based method starts with a block with some behaviour and during the process of development, this behaviour is decomposed down into smaller, functional units. UML's composite structure mechanism should be used to describe the composition of classes and their parts while allowing each entity its own internal behaviour regardless of whether it contains parts which themselves have behaviour. In the UML case, the action of decomposition in the software process should not be confused with decomposition of a class into its parts.

3.2 Composite Structures for Describing Scenarios

A second usage of the composite structure is more in line with the UML view. Here the diagram or diagrams are used to describe the various configurations of the system. In a process which then is similar to *describing scenarios with object* *diagrams* as a way of requirements elicitations and then *reconstructing the class diagram*.

Various composite structures are combined and generalised to produce the class diagram which admits all the given composite structures as shown in figure 11. In this figure we can see three scenarios each modelled as a complete composite structure diagram; from this, one can imply the weakest inferable class structure, but often one sees an unconstrained class diagram produced instead.

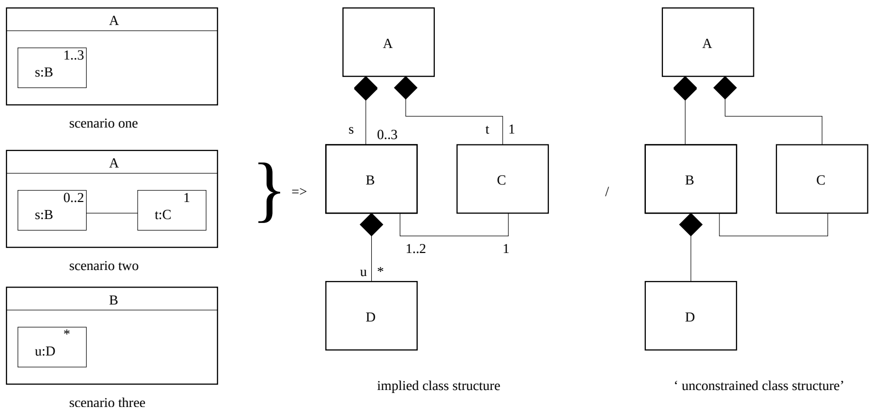

Fig. 11. Scenario Modelling and Class Reconstruction

The interaction between class and composites structure has a subtle effect for the modeller: it is recommended that multiplicity constraints, the type (aggregation, relation, composition) and directionality are always denoted on associations between classes. However this sometimes is difficult to envisage when working from a class perspective and it requires scenarios to be modelled using object diagrams and such multiplicity constraints and other properties to be inferred from these scenarios [22, 23]. This particular way of working can be difficult if not foreign, especially as the drawing object diagrams are not supported by many tools (we have used USE for this [24]) and that most OO practitioners are not trained to utilise object diagrams.

This style of using composite structures is not covered by the UML2.0 but can be ‘admitted’ by some tools[1].

It can now be argued that the UML collaboration artifact fulfils the same purpose and an example of this can be seen in figure 12 which shows a collaboration depicting a situation to that seen in the second composite structure diagram from figure 13.

[1] It is unfortunate that Visio and Powerpoint are considered modelling tools

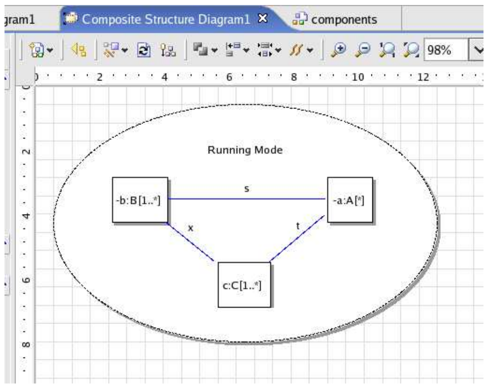

Fig. 12. UML Collaboration

But the collaboration, at least to the modeller, does not appear as a diagram in its own right but as an artifact that requires a diagram - this might explain why modellers are reluctant to use it.

The other main reason is that the collaboration artifact or element is not read in the same way as a composite structure diagram, the usage is more akin to that described in [25]. By 'read' here we mean in the semiotic sense [26] such that persons used to engineering the structure of systems interprets and writes the descriptions of those systems with the tools (in our case the composite structure diagram) that they are used to thinking with. Anecdotal evidence suggests that engineers think of modes as the system comprising of objects in a certain *structure* and not as a collaboration of objects, where as the collaboration is used to describe particular scenarios and not modes of operation. One can also argue that an additional or yet another UML construct to learn is also complicating matters.

3.3 Modes of Operation

Similar to describing individual scenarios, the use of composite structure diagrams for describing modes of operation of the system is often seen. Consider the situation as in figure 13 which shows three individual and complete composite structure diagrams each depicting a mode of operation of some system. This kind of division of major functionality or behaviour of the system is common in many real-time systems [27], particularly in our case: digital signal processors.

Here we see three distinct, complete composite structure diagrams which by way of a tagged value describe modes of the system. Each composite structure diagram here is used as an artifact that completely describes the the configuration

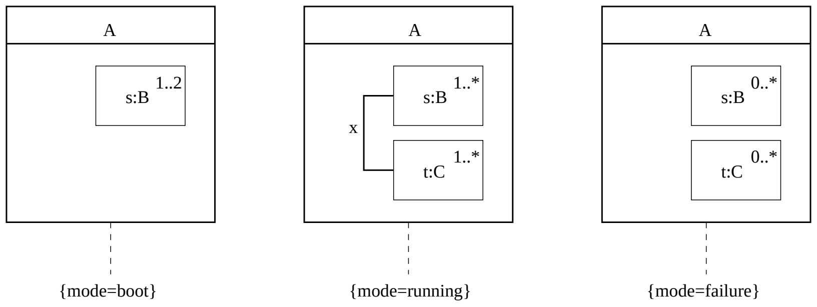

Fig. 13. Three Modes

of parts of various objects. Each composite structure diagram then describes a state-space which characterises the structures seen while the system is operating in that given mode.

State spaces of the modes may overlap, we do not provide any semantics or rules for this in this paper, although it has been suggested that these overlaps can be more easily seen in a visual sense from the use of these diagrams. These overlaps can be used to suggest suitable pre-conditions for actions which are capable of changing the mode of the system

3.4 Operation Effects

When using the composite structure, one can supply information about the effect of an operation through the use of a composite structure instantiation and stereotyped dependency relationship as the example in figure 14 shows.

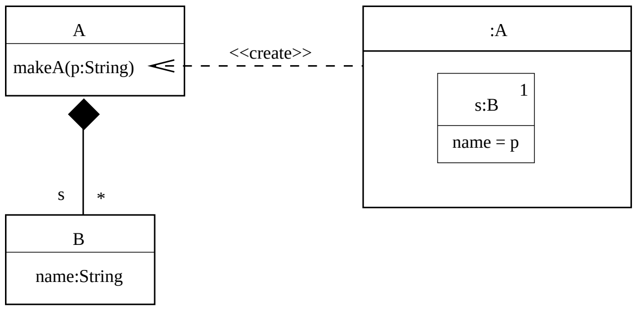

Fig. 14. Operation Output

The UML states that this kind of construct denotes the **constructor** for a given class and the use of the stereotype << *create* >> semiotically reinforces this notion. There is a very subtle change between the 'composite structure'-like artifact in figure 14 and the 'class'-like structure normally seen in composite structure diagrams (see fig.1) in that the output of the operation is an instantiation of some class; this has caused some confusion when trying to teach these constructs.

It is unclear in the UML specification whether this construct can be read in a more general way that the structures created are just merely the effects in terms of structures created of any operation rather than explicit constructor functions. The example (figs.14 and 15) can be seen describing the SDL notion of instantiating a process at system initialisation. In similar vein the latter example (figs.16 and 17) can be seen as describing runtime creation of a process.

Two uses have been found here, the first is that the << *create* >> stereotype means that the operation in question is a constructor function and that the composite structure instantiation given must have the owning class of that constructor as its context and that it represents the instantiation of that class and related parts. The filmstrip (or sequence of object diagrams) in figure 15 demonstrates the use of `A::makeA(...)` as a constructor function.

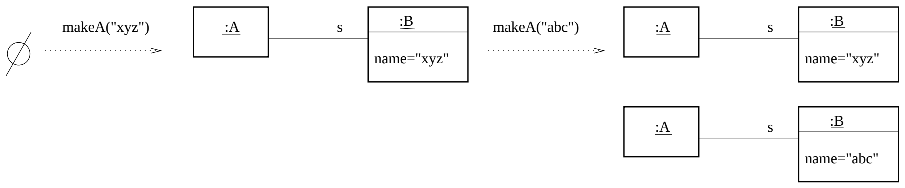

Fig. 15. Filmstrip of Constructor Function Applications

The more general case is where the construct seen in figure 14 is generalised to show the effect with respect to instantiations of any operation not just constructors. This would extend the idea expressed on §9.3.1 of [10]. Figure 16 shows a possible syntax for this while figure 17 shows a filmstrip.

Further to this, although we have not explored this in detail yet, it has been suggested in our work that these kinds of effects could be translated to object constraint language [28] expressions - the postcondition of actions. This provides an enabling of design-by-contract ideas [29] within the graphical language alone.

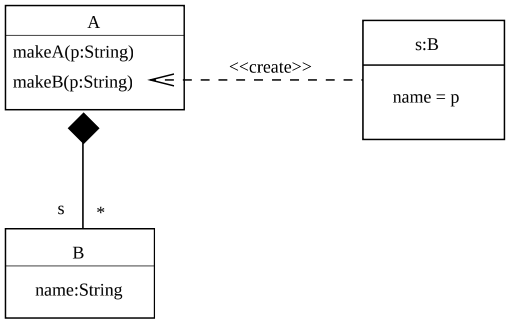

Fig. 16. General Operation Output

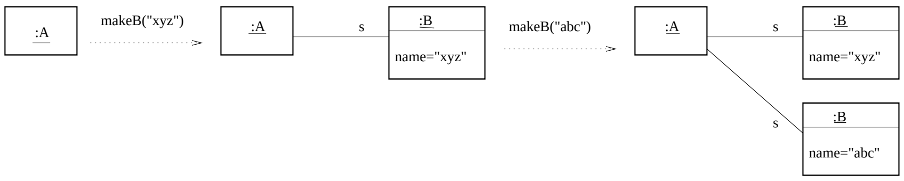

Fig. 17. Filmstrip of Operation Applications

4 Potential Solutions

We offer a few potential solutions:

– Best Practise
– Standardisation
– Formal Semantics
– Removal

The cases described in this paper have all been seen in use in the field. It is obvious in many cases that individual practitioners are 'inventing' usages as needed. While this can be argued that it is in the spirit of UML to do this, in reality it makes for the situation where a single diagram is read in many, potentially conflicting ways. The results of this ambiguity are well known - one only needs to read existing literature on this subject to see the problems. One way this has manifested itself is through the amount of time lost in engineers trying to understand and explain each others' semantics rather than discussing the design of the system.

Common, coarse grained usages do emerge and these can be standardised through best practise **at project level**, not company wide unless the work environment is small and focused across a number of projects. In the case of earlier versions of UML and lately UML2.0 we have produced a company-wide standard usage [30]. However this has concentrated on class, state and sequence diagram elements rather than the lesser used composite structure. One reason for this is that engineers often feel compelled to use as much notion as possible without regard for its purpose and restricting this abundance of choice results is a more focused usage of allowed notation.

For example, the most useful usage of composite structures we have found is as restrictions to the class diagram for describing the configuration of objects during various modes of the system (section 3.3).

Best practise works if the level of description of the practise/usage is precise, relevant to the job at hand and usually not formal in the mathematical sense. The problem is that the practise varies between projects and engineers working on more than one project often have problems adapting to two or more subtly varying syntaxes and semantics. A cross-company best practise tends to become too loose in its ideas in order to admit all possible variations.

Standardisation of the usage of composite structure through the OMG, ETSI, ITU or other standards body has been seen to be difficult with a number of major approaches being taken: UML-RT, SDL and then a number of minor variants and additions being proposed. While this would probably be the best route, it is obvious after nearly six years of UML2.0 standardisation work that a common agreed, consistently used semantics will probably not emerge.

One argument for the apparent weakness in the semantics of the composite structure comes from UML's open and extensible semantics approach. While we agree that it is in UML's interests to utilise this idea, it would be better to insist on a common,***core*** semantics and reading for the language; especially with lesser understood concepts such as composition. One can compare the semantics of composite structures with that of classes and note the difference in formality, see [31] for the latter and [32] for precise UML semantics. Again as with best practise ideas, the will for a formal semantics for a core of UML while existing is not being translated into the standard.

Even if a formal semantics is never produced as a part of a final standards document it can be argued that the effort of doing so will force the standardisation bodies to concentrate more on what is being written and described. There is already a body of results regarding the development of OCL with confirms this [33, 34]

The final solution, which we have advocated unless the situation demands, is to use a small subset of the UML as possible: this normally means just class and sequence diagram in most cases. This approach is favoured by profiles such as xtUML [35] which takes an extreme position by not using the composite structure diagram at all and leaves creation, deletion and management of objects to the relevant operations in the action language. As composite structures are semantically complex with many different interpretations the best solution is often just to forbid its usage.

5 Conclusions

It has been the aim of this paper to document some common, in the field, usages of the composite structure diagram. Overall, we surmise that the composite structure diagram with its additional capabilities over what has been presented before (in UML 1.x) and it greater degree of expressiveness over constructs such as capsules (ROOM) is an important and valued addition to the range of UML artifacts.

However, its position at this moment in time as a view-like structure rather than an artifact as the class or collaboration means that it is invariably misunderstood. We have noticed that engineers with a ROOM or ObjecTime background tend to use the composite structure as a description of the constructor function of a class or the starting point for the system as a whole. While those with more of an SDL or hardware background tend to use it instead of the class diagram for describing a static collection of components, subsystems or objects.

Many training courses do not cover the composite structures but rather concentrate on how to model with use cases and classes or more commonly how to develop object oriented systems/architecture from just use cases. If composition is mentioned at all then it is usually with an informal and often misleading discussion of the black vs white diamond symbols without reference to how compositions are actually described.

We would suggest that the composite structure diagram is elevated from its model viewing role to that of a first class UML modelling artifact with a clear and precise relationship with other (object and classes) UML elements. This would allow more accurate modelling of notions such as modes, constructor functions and so on. To facilitate this we would propose a detailed semiotic analysis of the notation which would combine what the notation suggests with the background of the user and thus their ‘preferred reading’ of the notation.

6 Acknowledgements

This work has been made in coöperation with ETSI and is partially funded by the EU project Rodin (IST-511599).

References

1. Haugen, Ø., Møller-Pedersen, B., Weigert, T.: Structural modeling with uml 2.0, classes, interactions and state machines. In Lavagno, L., Martin, G., Selic, B., eds.: UML for Real, Design of Embedded Real-Time Systems. Kluwer Academic Publishers (2004) 53–76
2. Bock, C.: Uml 2 composition model. Journal of Object Technology **3**(10) (2004) 47–74
3. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley (1999) 0201325713.
4. Bock, C., Odell, J.: A foundation for composition. Journal of Object Oriented Technology **7**(6) (1994) 10–14
5. Odell, J.: Advanced Object-Oriented Analysis and Design Using UML. SIGS Reference Library. Cambridge (1998) 0-521-64819-X.
6. Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modelling. Wiley (1994)
7. Ellsberger, J., Hogrefe, D., Sarma, A.: SDL, Formal Object-oriented Language for Communicating Systems. Prentice Hall (1997) 0-13-632886-5.
8. partners, S.: Systems Modeling Language (SysML) Specification. www.sysml.org (2005) version 1.0 alpha, SysMLv1.0a-051114R1.
9. Gonzalo Genova, M.C.V., Nubiola, J.: A semiotic approach to uml models. In: 1st International Workshop on Philosophical Foundations of Information Systems Engineering (PHISE'05) 13 June 2005, Porto, Portugal. (2005)
10. Object Management Group: UML Superstructure Specification v2.0). Omg document number ad/02-09-02 edn. (2004) OMG Document Number formal/05-07-04.
11. Oliver, I.: Experiences of model driven architecture in real-time embedded systems. In: Proceedings of FDL02, Marseille, France, Sept 2002. (2002)
12. Oliver, I.: Model driven embedded systems. In Lilius, J., Balarin, F., Machado, R.J., eds.: Proceedings of Third International Conference on Application of Concurrency to System Design ACSD2003, Guimarães, Portugal, IEEE Computer Society (2002)
13. Oliver, I.: Some issues in rigorous system design in a model driven development context. In: ECSI UML-SystemC System Design Flow Workshop, May 4, 2004, Lille, France. (2004)
14. Selonen, P., Koskimies, K., Sakkinen, M.: Transformations between uml diagrams. Journal of Database Management **14**(3) (2003) 37–55
15. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artificial Intelligence **13** (1980) 27–39
16. Henderson-Sellers, B., Barbier, F.: Black and white diamonds. In France, R., Rumpe, B., eds.: UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings. Volume 1723 of LNCS., Springer (1999) 550–565
17. Back, R.: Refinement Calculus: a Systematic Introduction. Springer-Verlag (1998) 0387984178.
18. Selic, B.: Architectural patterns for real-time systems, using uml as an architectural description language. In Lavagno, L., Martin, G., Selic, B., eds.: UML for Real, Design of Embedded Real-Time Systems. Kluwer Academic Publishers (2004) 171–188
19. Simons, A.J.H.: The theory of classification. part 4: Object types and subtyping. Journal of Object Technology **1**(5) (2002) 27–35

20. Institute of Electrical and Electronics Engineers, Inc.: IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems -Description. (2000)
21. Len Bass, P.C., Kazman, R.: Software Architecture in Practice. 2nd edn. Addison Wesley Professional (2003) 0-321-15495-9.
22. D'Souza, D.F., Wills, A.C.: Object, Components and Frameworks with UML - The Catalysis Approach. Object Technology Series. Addison-Wesley, Reading, Massachusetts (1999) 0-201-310.
23. Ramazani, D., v.Bochmann, G.: Approaches to the specification of object associations. In Bowman, H., Derrick, J., eds.: Formal Methods for Open Object-based Distributed Systems Vol 2. FMOODS'97, Canterbury, UK, 21-23 July, Chapman and Hall (1997) 231–246
24. Gogolla, M., Richters, M.: Development of uml descriptions with use. In Shafazand, H., Tjoa, A.M., eds.: EurAsia-ICT. Volume 2510 of Lecture Notes in Computer Science., Springer (2002) 228–238
25. Ressler, F., Geppert, B., Gotzheim, R.: Collaboration-based design of sdl systems. In Reed, R., Reed, J., eds.: SDL 2001: Meeting UML, 10th International SDL Forum, Copenhagen, Denmark. Lecture Notes in Computer Science 2078, Springer (2001) 72–89
26. Hall, S.: Encoding/decoding. In: Culture, Media, Language: Working Papers in Cultural Studies 1972-79. (1980) 128–38 London, Hutchinson.
27. Pedro, P., Burns, A.: Schedulability analysis for mode changes in flexible real-time systems. In: Proceedings, 10th Euromicro Workshop on Real-Time Systems, Berlin, Germany, IEEE Computer Society (1998) 17–19
28. Warmer, J., Kleppe, A.: The Object Constraint Language - Precise Modeling with UML. Object Technology Series. Addison Wesley Longman (1999) 0-201-37940-6.
29. Meyer, B.: Applying design by contract. Computer **25**(10) (1992) 40–51
30. Oliver, I., Ziegler, J.: Nokia UML Kernel Reference. 2.2 edn. Nokia Corporation (2005)
31. Cardelli, L., Abadi, M.: A Theory of Objects. Springer-Verlag (1998) 0-387-94775-2.
32. Evans, A., Clark, A.: Foundations of the unified modelling language. In: 2nd Northern Formal Methods Workshop, Ilkley, electronic Workshops in Computing., Springer-Verlag (1998)
33. Richters, M., Gogolla, M.: A metamodel for ocl. In France, R., Rumpe, B., eds.: LNCS 1723, UML'99 - The Unified Modeling Language, Beyond the Standard, Springer-Verlag (1999) ISSN 0302-9743.
34. Brucker, A.D., Wolff, B.: HOL-OCL: Experiences, consequences and design choices. In Jézéquel, J.M., Hussmann, H., Cook, S., eds.: UML 2002: Model Engineering, Concepts and Tools. Lecture Notes in Computer Science LNCS 2460. Springer-Verlag (2002) 196–211
35. Balcer, M., Mellor, S.J.: Executable UML: A Foundation for Model Driven Architecture. Addison Wesley (2002) 0201748045.