Emerging Topics in Textual Modelling[*]

Achim D. Brucker[1], Gwendal Daniel[2], Martin Gogolla[3], Frédéric Jouault[4],
Christophe Ponsard[5], Valéry Ramon[5], and Edward D. Willink

[1] Department of Computer Science, University of Exeter, Exeter, UK
`a.brucker@exeter.ac.uk`
[2] Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya (UOC),
Barcelona, Spain
[3] Computer Science Department, University of Bremen, Bremen, Germany
`gogolla@informatik.uni-bremen.de`
[4] ESEO-TECH, Angers, France
[5] Center of Excellence in Information and Communication Technologies (CETIC),
Charleroi (Belgium)
`{christophe.ponsard,valery.ramon}@cetic.be`
[6] Willink Transformations Ltd, Reading, England,
`ed_at_willink.me.uk`

Abstract The 19th edition of the OCL workshop featured a lightning
talk session where authors were invited to present their recent work and
open questions related to textual modeling in general and OCL in partic-
ular. These 5 minute presentations triggered fruitful discussions within
the OCL community on the usage of textual modeling, model validation,
and specific technical points of the OCL specification. This community
paper provides an overview of the presented contributions (one per sec-
tion), as well as a summary of the questions and discussions they have
triggered during the session.

Keywords: OCL, Textual Modeling, Graphical Modeling

1 Introduction

Textual modeling in general and OCL in particular is a well-established but still
active field of research. The OCL standard, which was first released in 1997,
has been continuously enriched since then, and is now the cornerstone of sev-
eral model querying, verification, and transformation approaches. Current work
on textual modeling covers its foundation (e. g., the formal semantics of tex-
tual modeling languages) to its usage in cutting edge applications like database
querying, or AI specification.

The lightning talk session of the 19th edition of the OCL workshop was
animated by three experts of the field. They presented their ongoing work on
the usage of textual modeling with respect to graphical modeling, discussed

---

specific constructs of the OCL language, and introduced optimization strategies for model verification.

The following sections are contributed by these experts. They summarize both their recent works and the related discussions that took place during the workshop.

2 Tweaking Class Model Validation with Specialized Association Bounds
Martin Gogolla

In the context of validating and verifying UML and OCL class models, we have developed a so-called model validator [1] that automatically generates valid object models satisfying in particular the stated OCL invariants. For each class and optionally for each association, one states a lower and upper bound, respectively, for the number of objects in a class and for the number of links in an association. Tight bounds often lead to faster validation and verification processes. Apart from plain associations, UML allows the developer to apply different kinds of part-whole relationships: Aggregation and composition. In our implementation, both part-whole relationships have to be acyclic on the instantiation level, i. e., in the object diagrams. Concerning the difference between aggregation and composition, we have implemented aggregation as a part-whole relationship establishing a weak connection between the part and whole, whereas composition implies a strong connection between part and whole: In aggregations, a part may be included in many wholes, whereas in compositions a part may be included at each point of the part's life cycle in at most one whole (allowing for possibly changing, different wholes). Associations are interpreted in object diagrams with arbitrary connections in which no particular restrictions are made with respect to structure. These requirement can be formulated again as OCL constraints [2].

The requirements on the instantiations, i. e., the objects diagrams, can be summarized as follows: Plain associations are interpreted by plain, directed graphs having no particular restrictions, aggregations yield acyclic graphs (so-called dags, i. e., **d**irected, **a**cyclic **g**raphs), and compositions end up in graphs with tree-like structures. Fig. 1 states an overview and an example. With respect to the number of upper bound of links in associations, aggregations, and compositions the requirements imply the following upper bounds. We assume the simple case that we consider one class with one association, one aggregation and one composition defined on the class in a reflexive manner. Furthermore we assume that n objects are present in the considered class. The specialized association bounds can be stated as follows: At most $n * n$ links are allowed in the case of association, at most $(n - 1) * (n - 2)/2$ links in the case of aggregation, and at most $n - 1$ links in the case of composition.
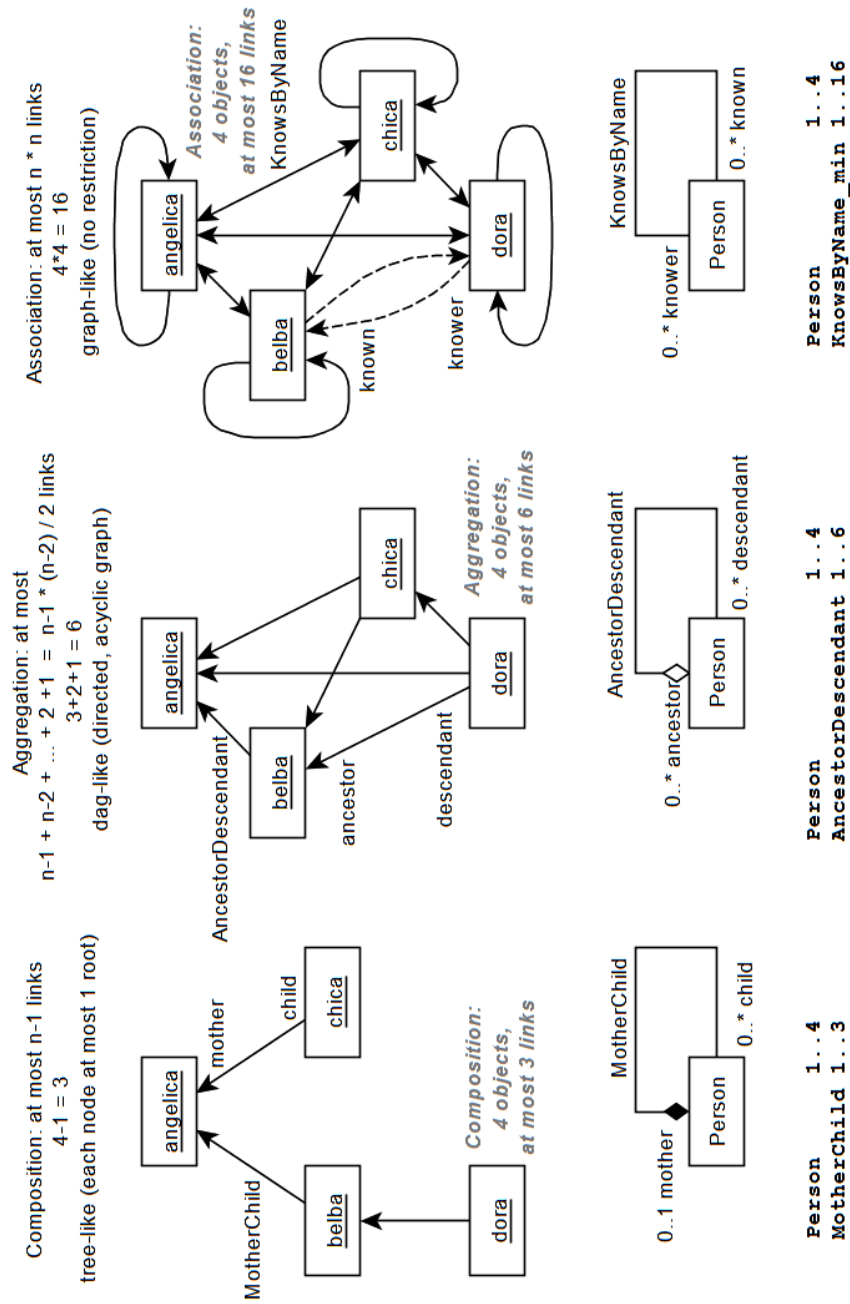
Figure 1. Composition, Aggregation, and Association: Population Differences

3 An OCL Map Type
Edward D. Willink

OCL's family of `Collection` types is well known, but a `Map(K,V)` type is missing. Some distinguished authors have suggested that the deficiency can be remedied by a `Set(Tuple(K, V))`, but this is clearly misguided since a `Set(Tuple)` can have many different-valued entries for the same key, whereas a map can only have one value per key.

The Java Map type is very familiar and might perhaps inspire an equally familiar library type for OCL, but this too is misguided since a Java Map is mutable while an OCL `Map` should be immutable.

The ordered `Collection`s are therefore a better source of inspiration. The following have obvious functionality:

`=`, `<>`, `isEmpty()`, `notEmpty()`, `size()`.

New `keys()` and `values()` operations can access the two halves of the map.

Similarly obvious functionality with respect to the keys can be provided by:

`excludes(k)`, `excludesAll(c)`, `excluding(k)`,

`excludingAll(c)`, `includes(k)`, `includesAll(c)`

Further emulation of ordered Collections suggests that `at(k)` accesses the map at index k. `including(k,v)` creates a new map with an additional or replacement `k<-v` binding. It returns `null` for a null value and `invalid` for a missing value.

Richer support can be provided by:

`excludesMap(m)`, `excludesValue(v)`, `excludingMap(m)`,

`includesMap(m)`, `includesValue(v)`, `includingMap(m)`

Construction of a `Map` literal can re-use the `Tuple` literal syntax in conjunction with a new binding operator `<-`. Thus a map literal with two entries for two value to key bindings my be created by:

`Map{k1<-v1, k2<-v2}`

The above facilities were prototyped in Eclipse OCL 2015-06. They provide an adequate ability to use a `Map` but prove very inefficient for Map construction since creating a Map with `N` entries requires progressive construction of `N-1` intermediate maps; the execution performance is therefore at best quadratic. The Eclipse OCL 2019-03 release therefore extends the create/operation `Map` functionality with iteration support.

A `Map` is treated as a set of keys each with a bound co-value. All the standard `Collection` iterations apply to `Map`s using the set of keys as the iteration domain. Additionally a co-iterator may be bound to the iterator to avoid the need to invoke `at(k)` to obtain the value of each key. Thus a map can be checked to ensure that each `v` bound to its `k` iterator is equal to the squared value of the iterator.

`let c : Map(Integer, Real) = ... in c->forAll(k<-v | v = k*k)`

A new `collectBy` iterator, that may be used on `Collection`s or `Map`s, supports creation of a `Map` by collecting an expression value for each iterator key. A map from ten integer values to their squares may be built by:

`Sequence{1..10}->collectBy(k | k*k)`

Future work might generalize `k<-v` from special purpose punctuation to an expression operator. The downside of this generalization is the cost/complexity of a new `Entry(K,V)` type for the new expression result and of course many new operations to allow an `Entry` type to be used sensibly. The upside is that the `collectBy` body may use let variables and may compute both key and value. A map that binds an integer value to its string value could be built by:

```
Sequence{1..10}->collectBy(k | k.toString()<-k)
```

A consequence of permitting both key and value to be computed, is that, in general, uniqueness of the key values cannot be guaranteed. To avoid non-deterministic loss of colliding values, the `Map` parameterization would need to be a multimap: `Map(K,Bag(V))`.

4 Some Guidelines About When to Use Textual And/Or Graphical Modelling in MBSE
Christophe Ponsard and Valéry Ramon

Model-based systems engineering (MBSE) is the application of modelling techniques throughout the whole system development lifecycle to support system requirements, specification, design, verification and validation activities [3]. No matter they are general-purpose or domain-specific, modelling languages can take different forms and rely on different textual and/or graphical notations:

- textual modelling languages cover a large spectrum, from natural/structured languages (e.g. for requirements) to more formal ones (e.g. B, Event-B, Alloy)
- graphical modelling languages are mostly semi-formal (e.g. UML, SysML) but some have formalised notations (e.g. Petri nets, Finite State Machines).

Textual and graphical modelling languages can also be combined and complement each other. Indeed, textual model elements may appear in graphical-based languages for stating precise properties (e.g. OCL constraints in UML class diagrams, formal layer of KAOS [4] using temporal logic). Besides, tools exist that enable the combination of both modalities: either they provide textual edition inside graphical editors or they enable to edit the same model both graphically and textually (with both views continuously synchronised with each other). An example of this latter case is the integration of Sirius graphical editor and Xtext textual editor to edit EMF models [6].

Although textual and graphical notations may capture the same information, one modality might be more adapted to some context modelling related activities involving both humans and tool chains. We give here a synthetic comparison of the benefits of both kinds of notations according to relevant factors such as analysability, learning curve or scalability. Our work is based on both a review of results from some empirical studies [5, 7] (including themselves wider results reported in the literature) and on our own return of experience as a research and industry technology transfer centre. Table 1 summarises those results: + and (+) respectively indicate that the notation is better or is better with some restrictions explained below; deb means "debatable".

Table 1. Comparison of textual and graphical notations

Factor	Textual	Graphical
Analysability and Modifiability - Accuracy $^{(*)}$	$(+)^{(a)}$	
Effectiveness$^{(**)}$	$(+)^{(b)}$	
Intuitiveness and Learning Curve	$(+)^{(c)}$	
Representation of high-level/more abstract information		$(+)^{(d)}$
Complex/large model management	$\text{deb}^{(e)}$	$\text{deb}^{(e)}$
Versioning - Comparison (diff)	$+^{(f)}$	
Preference - Satisfaction		$(+)^{(g)}$

Most criteria are self-explanatory but we give some precision about:

(*) Accuracy is the degree to which a model enables deeper reasoning as well as error/deficiencies/inaccuracies detection and correction (based on definitions of analysability and modifiability from SQuaRE (ISO/IEC 25010)).

(**) Effectiveness is the effort/time to understand and maintain a model.

The following comments and lessons learned can be summarised from Table 1.

a *Accuracy*: textual is better according to the study of [5] whereas no statistically significant difference is noticed in [7]. From our experience, textual notation is potentially more accurate (assuming well-defined semantics) notably for stating precise properties and in the context of formal modelling languages.

b *Effectiveness*: textual notation is more effective according to the study results of [5, 7]. Effectiveness with graphical notation significantly improves with training [7], which matches our experience.

c *Learning curve*: except for simple diagrams (e. g. UML use cases), it is usually shorter with textual notation [5, 7]. *Intuitiveness* depends on the language expressiveness and intricacy but also on the user profile (technical vs non technical). Graphical notation is usually more intuitive for non-technical profiles. Using a Domain Specific Language (DSL) improves this dimension.

d *Representation of high-level information*: from our experience and the literature, graphical models better enable an overall view of the modelled system, especially in the first engineering phases of complex industrial systems.

e *Complex/large models* lack studies relying on industrial models. Combining both modalities is probably recommended to manage different levels of details (hierarchical views, easy navigation) and the variety of profiles.

f *Versioning - Comparison*: it is easier and largely supported for textual notations thanks to Version Control Systems. Support for model versioning is progressing. However, clear visual representation of differences in graphical modelling tools remains tricky.

g *Preference - Satisfaction*: usually goes to graphical modelling [5, 7]. However, when balanced with other qualities like accuracy or effectiveness, more experienced subjects end up preferring textual representations.

In conclusion, the above evidence gives some hint about the respective benefits of textual and graphical modalities in MBSE. However, they are not yet strong enough, given the small number of the empirical studies as well as their simplified models and engineering tasks not necessarily realistic from a real-world perspective. Our plan is to enrich this comparison based on MBSE on-going and future deployments, in connection with a more systematic literature survey.

Bibliography

[1] Gogolla, M., Hilken, F., Doan, K.H.: Achieving Model Quality through Model Validation, Verification and Exploration. Journal on Computer Languages, Systems and Structures, Elsevier, NL (2017). Online 2017-12-02

[2] Gogolla, M., Richters, M.: Expressing UML Class Diagrams Properties with OCL. In: Clark, T., Warmer, J. (eds.) Advances in Object Modelling with the OCL, pp. 86–115. Springer, Berlin, LNCS 2263 (2001)

[3] INCOSE: Systems Engineering Vision 2020. INCOSE-TP-2004-004-02 (2007)

[4] van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley (2009)

[5] Melia, S., Cachero, C., Hermida, J., Aparicio, E.: Comparison of a textual vs a graphical notation for the maintainability of mde domain models: an empirical study. Software Quality Journal **24**(3), 709–735 (2016)

[6] Obeo: Xtext/Sirius Integration Use-Cases. `http://bit.do/obeo-sirius-xtext` (2017)

[7] Sharafi, Z., et al.: An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension. In: IEEE 21st Int. Conf. on Program Comprehension (ICPC) (2013)