Using Metamodel Composition to Unify User Story and Use Case Metamodels

Larissa Mangolim Amaral[1], **Anarosa Alves Franco Brandão**[1], **Fábio Levy Siqueira**[1]

[1]Escola Politécnica – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

{larissa.mangolim.amaral,anarosa.brandao,levy.siqueira}@usp.br

Abstract. *Requirements languages are often defined as metamodels for the standardization of a system's requirements specification. The unification of these metamodels benefits general analysis and interoperability between requirements models. Even tough some authors already discuss systematic composition approaches, this application for requirements languages' metamodels is still little explored. Therefore, we applied non-domain-specific composition frameworks to create a unified metamodel for User Story and Use Case diagram by assessing common points in related work. This application enabled reducing the subjectivity of the requirements languages composition process but still strongly depended on human curation.*

1. Introduction

One of the first activities executed in a software project is speaking to stakeholders in order to understand their requirements, i.e., needs, and associated constraints and conditions for the system [IEEE 2017]. The requirements are then documented to establish a contract as a specification of what has to be fulfilled by the system [IEEE 1990], usually expressed in requirements languages.

Requirements languages determine a set of constructs and verification protocols that are used to shape requirements documentation [IEEE 2017]. There is a wide range of different requirements languages that vary in their aspects and purposes, like expressiveness, visualization, and formality. Formal languages are composed of a syntactical and a semantic definition, the latter being responsible for defining the domain of information supported by the language and for mapping the meaning of expressions validated by the former [Harel and Rumpe 2003]. These definitions for requirements languages are often expressed by taking advantage of metamodeling techniques [Amaral et al. 2022], if not by their original authors, by later authors.

Every model is fundamentally focused on a specific purpose [Bézivin 2005], which justifies the existence of multiple requirements languages - or even versions of the same language - for different perspectives, contexts, or domains. The unification of requirements languages metamodels is approached by some authors, each of them indicating various benefits coming from the unification result and the consequent interoperability of metamodels. Saidi et al. (2013), who proposed a metamodel unifying the semantics of i*, CREWS, and PREview from different requirements approaches (goals, scenarios, and points of view, respectively), focus on the benefit of the translation of information between models written in different languages. Patricio et al. (2011), who unified i* and KAOS requirements languages, indicate that the creation of tools that interoperate with

both popular and unpopular notations disseminates the usage of the unpopular one among industry practitioners. Rauh, Golubski and Queins (2017) emphasize two other benefits: (1) the identification of inconsistencies and redundancy in system specifications; and (2) the creation of more complete specifications through the identification of missing information in models.

Although there are some studies that propose a unified metamodel for requirements languages, the majority of them do not use a generic iterative method for this composition — one such as those examined by Abouzahra et al. (2020) that could be systematically used to unify other metamodels. In general, their unification process is composed by the analysis and abstraction of each term that appears in the languages' definition and the identification of any correspondence between them [Kaindl and Wagner 2009, Patricio et al. 2011], always focused on the languages being unified or on a specific domain. Rauh, Golubski and Queins (2017) connected the main concepts from use case description and UML diagrams through their relations to system services, which is a service-specific strategy. Saidi et al. (2013) apply an extension of the Lesk algorithm that calculates the distance between concepts' synonyms using definitions from the lexical database WordNet, which characterizes one of the common heuristics used for the matching operation.

Our research question is related to the evidenced gap for a composition method in the context of requirements languages, and how their matching is usually subjectively done: Does the application of existing non-domain-specific model composition methods for the unification of multiple requirements languages metamodels reduce process subjectivity? In addressing this question, we select a method assessed by Abouzahra et al. (2020) and apply the selected heuristics to two existing requirements languages' metamodels: User Story (US) and Use Case diagram. We were able to create an unification of these metamodels and discuss the process execution and results. In the future, we hope to develop a unified metamodel that takes into account a variety of requirements languages.

This paper is organized as follows. In Section 2, we define the strategies for each step of the requirements metamodels' composition. In Section 3 we apply a combination of existing composition methods in an example, unifying two existing requirements languages' metamodels. In Section 4 we discuss how the method was carried out and its the results, and the identified limitations of our study. Finally, Section 5 presents the conclusions and future work directions.

2. Metamodel Composition

Abouzahra et al. (2020) did a systematic review of studies that propose frameworks and tools for model composition. They vary mainly in their strategies defined by their purposes, but they mostly converge on a structure based on four steps, which are presented in Figure 1. The first step is the *matching operation*, that produces a *matching model* by searching for equivalences between elements from two input models. The matching model contains the links between the input models and can be used for automatic processing depending on how formally it is defined [Abouzahra et al. 2020] - for example, Del Fabro and Valduriez (2007) and Di Ruscio et al. (2012) use weaving models for that. The second step is a *composition operation* that uses the *matching model* together with the input models to produce the resulting composed model. Boronat et al. (2007), Kolovos

et al. (2006) and Anwar et al. (2013) also have a step for *conflict resolution* between elements that considered equivalent but with structural differences that conflict, and there must be a defined strategy for determining which structure will be retained in the resulting metamodel - for example, Boronat et al. (2007) choose one input model whose structure will be prioritized. Another step that is considered is a *revision and manual update* of inconsistent links by a domain specialist. Considering the partial or complete automation of the previous steps, this revision can help reduce the loss of information from the adopted heuristics. In Figure 1, *revision* and *conflict resolution* were considered to be executed together as the designer would analyse the resulted matching model.

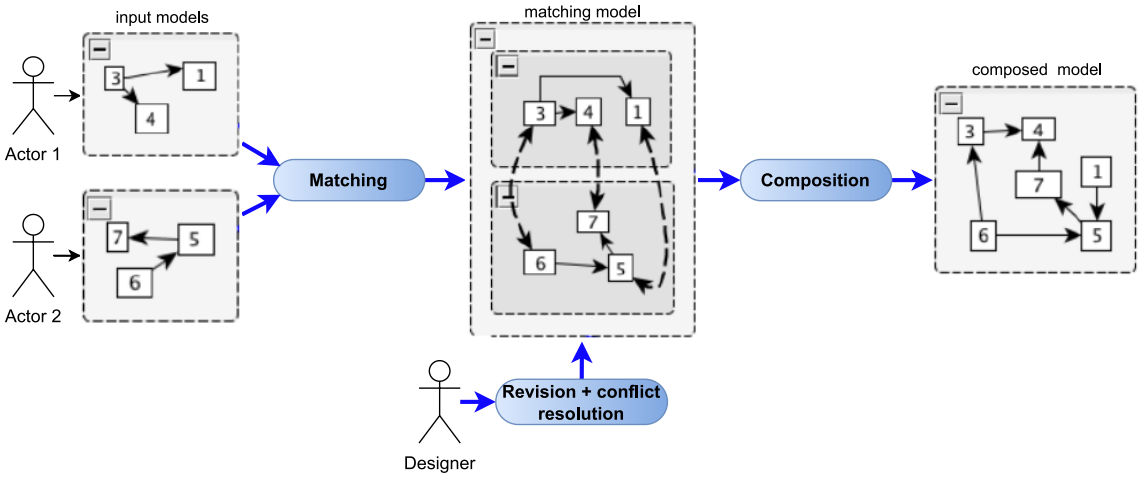

Figure 1. Model composition flow - adapted from Anwar et al. (2013). The different models' elements are represented by boxes with identification numbers, their relationships being the arrows between them. The steps inputs and outputs are indicated by the blue arrows.

Anwar et al. (2013) and Del Fabro and Valduriez (2007) are the model-related works best ranked by Abouzahra et al. (2020). Del Fabro and Valduriez (2007) propose a very complete set of heuristics for calculating matching elements between two input metamodels and building a weaving model as the matching model. Nevertheless, their composition operation bypasses the unified metamodel and uses the matching model to directly generate a transformation for merging two models that each conform to their respective input metamodel - meaning their process actually aims at a composed model instead of a composed metamodel like we intend to have.

3. Composing Requirements Metamodels

A challenge for the systematic model composition we intend to apply is that requirements metamodels usually contemplate abstract concepts whose semantics are not always completely clear. In this paper we merge two requirements metamodels: a metamodel of the unification of UML diagrams (class, use case, activity, and state machine diagrams) [Rauh et al. 2017] (M_1) and a user story (US) metamodel [Wautelet et al. 2014] (M_2). The choice of which requirements languages to unify considered the most popular languages among Software Engineering (SWE) researchers [Amaral et al. 2022]. The metamodels' elements are shown in Figure 2. The choice of what metamodel to use considered only that the metamodel should not be domain-specific, since our intention for

future work is to apply this method for the unification of other non-domain-specific requirements languages. M_1 was a strong candidate because it addresses many of the most common requirements languages and is already a work of language unification, therefore it could bring more insight to the analysis of possible problems during the composition process. Similarly, M_2 is an extended version of an US metamodel. M_1 was restricted to restricted to the metaclasses that compose the Use Case diagram, which contemplates the metamodel's static structure partition and the state-related concepts.

3.1. Composition Method Used

To compose the metamodels, we used steps from different metamodel composition methods. The **matching operation** was based on Del Fabro and Valduriez (2007), which consists of: (1) applying a cartesian product between every element from each input metamodel; (2) calculating elements' similarity using predefined heuristics; and (3) filtering links below the determined threshold. For that, we used the AMW tool[1] itself, SimMetrics API[2] and the JAWJAW API[3] - which is a wrapper for WS4J (WordNet Similarity for Java). We did **manual revision and conflict resolution** altogether, choosing the links and structures that made more sense in requirements context. This stage contemplates Del Fabro and Valduriez (2007)'s link rewriting step, when it is possible to transform simple links of equivalence into non-obvious links like inheritance and nesting that are more directly related to the application scenario. The authors do not define an explicit strategy for either of these steps, leading to a more customizable analysis. Anwar et al. (2013) suggest that conflict resolution and the definition of more complex relationships may be done manually by a designer in some cases. For the **composition operation**, we used Anwar et al. (2013)'s out-target merging strategy, which keeps input models unchanged, computes elements' links from the matching model, and translates every element that has no correspondence links into the resulting new metamodel. Merging is a composition strategy that considers that all the information from the input models will be present in the composed metamodel, ideally with no duplication [Abouzahra et al. 2020]. Table 1 summarizes strategies for each step of the execution.

Step	Strategy
Matching operation	Cartesian product; matching heuristics (Levenshtein distance for name similarity, cardinality, type and conformance, Lesk (WordNet)); and threshold filtering [Del Fabro and Valduriez 2007]
Revision	Manual filtering and link rewriting [Del Fabro and Valduriez 2007]
Conflict Resolution	Case by case analysis [Del Fabro and Valduriez 2007]
Composition Operation	Out-target merging strategy and translation [Anwar et al. 2013]

Table 1. Strategies for each step of the composition method.

3.2. Matching Operation

First, both metamodels were represented on Ecore[4] so they could be used as input for the AMW tool. For an initial run of the matching operation, we executed the following

[1] Available at https://www.inf.ufpr.br/didonet/amw/
[2] Available at https://sourceforge.net/projects/simmetrics/
[3] Available at https://code.google.com/archive/p/jawjaw/
[4] Available at https://wiki.eclipse.org/Ecore_Tools

methods with configured weights with the values suggested in the Match Configuration Menu section of the AMW's User Guide[5]: 0.7 for name similarity, 0.2 for cardinality, 0.1 for type and conformance, and 0.4 as threshold. This execution resulted in an empty-link weaving model, shown in Figure 2. That makes sense since the elements had very little name, type, and cardinality similarity. We also tried running the separate methods, and neither one had a higher similarity than the determined threshold.

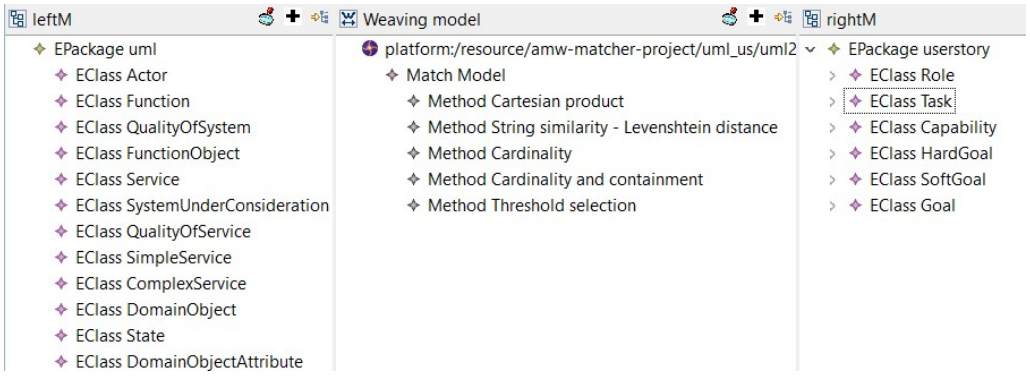

Figure 2. Weaving model after the automatic run of the model matching methods from the AMW tool.

We then applied the Lesk method, which is another heuristic considered by Del Fabro and Valduriez (2007), but not present in the AMW tool version we had access. For that, we implemented a Java matcher using the JAWJAW API[6]. Since the authors of M_2 detailed the history of elements that were unified to each metaclass, we also included this history for the Lesk execution. The similarity score results were filtered with a threshold value of 30, which seemed like a low enough value for us not to lose any meaningful links but also high enough to filter irrelevant ones. The links shown in Figure 3 were inserted manually into the weaving model in the AMW tool.

3.3. Revision and Conflict Resolution

We followed the matching operation with a revision of every link it produced. We kept the equality link for `Actor` and `Role` because it had a very high similarity score since `Role` had previously been unified with `Actor` in M_2 so they would represent the same concept.

`Function`, `Service`, `System` and `State` matched with a lot of elements, probably due to the wideness of their meanings (even inside the requirements context). As a result, equivalences to `System` were all ignored as none of the elements in M_2 had the same level of abstraction as `System`. Equivalences with `State` were discarded as well, since M_2 does not contemplate elements with a requirements' dynamic component. `Function` and `Role` link was rejected, because, even though their general meanings might be similar, they relate to different objects (function refers to the service, and role refers to the actor). `Function` and `Task` link was considered to make more sense and was maintained, as both concepts are about a process that is executed by the system. We considered the comparison between `Service` and `Task`, which had the highest similarity scores. Looking at the semantics given by the authors of the metamodels for these

[5]Available at https://www.inf.ufpr.br/didonet/amw/userguide/index.html
[6]Available at https://github.com/uneluneravie/matching-metamodels

elements, both concepts are focused on achieving a goal: "a 'Service' can provide the 'Actor' a value by supporting to reach a goal of the 'Actor"' [Rauh et al. 2017, p. 2], and "a task specifies a particular way of attaining a goal" [Yu et al. 2011, p. 57]. The main difference, which led us *not* to unify these concepts, was that the `Service` is a more complex entity that *supports* the achievement of a goal through functionality definition, whereas `Task` is a more direct means to this achievement. To make this relationship explicit, we changed their equality link to a relationship between these ununified metaclasses.

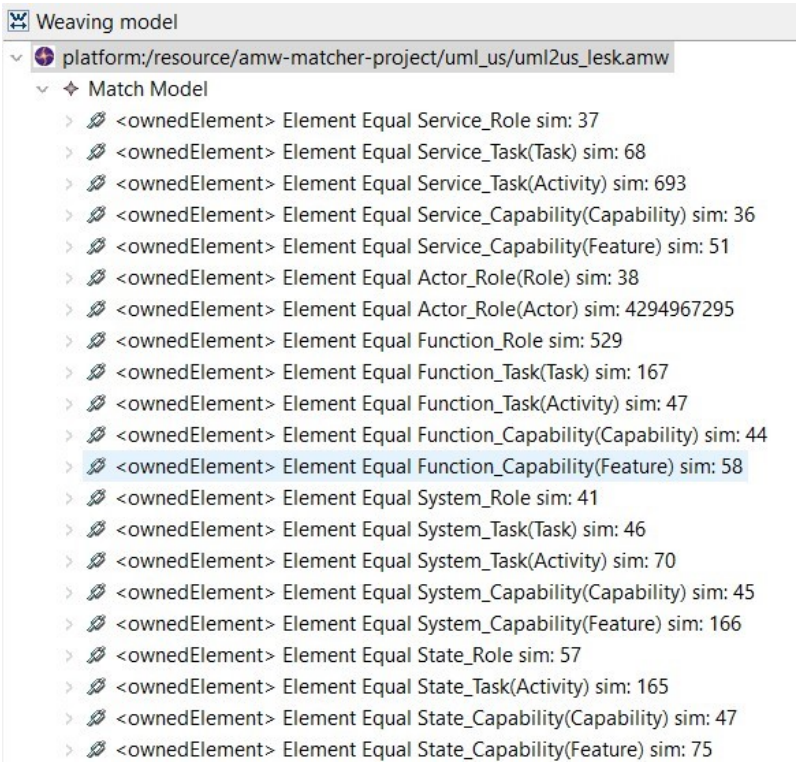

Figure 3. Weaving model after manual insertion of links with similarity generated by the Lesk (WordNet) heuristic.

Afterwards, in conflict resolution, we chose to keep M_2's properties for both unified metaclasses because M_1 does not declare attributes for its elements. The only other conflict happened when unifying `Function` and `Task`, since `Function` already defined a relationship with `Service` and we had declared a special link between the latter and `Task`. We therefore opted for keeping the original relationship and discarding the new link. We also checked for consistency as there would be defined a relationship between `Function` and `Actor`, since it already existed with `Task`. The cardinality did make sense, as the `Actor` can use many `Services` and each `Service` is defined by `Task`, so one or more `Actors` could want many `Tasks`.

For the merging operation, we added `NotFound` links to track the elements in both input models that had no match and would be translated to the composed metamodel. We then resolved every link manually, resulting in the metamodel presented in Figure 4.

4. Discussion

Regarding the composition execution process, it was easier to identify conflicts while composing the final metamodel, raising suspicion that the automation of these steps would

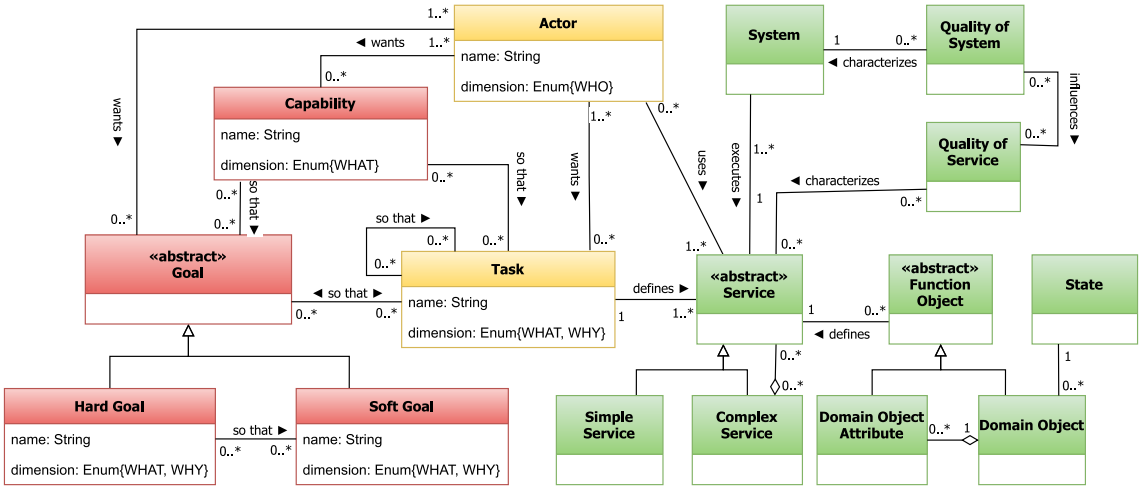

Figure 4. Resulting composed metamodel. Metaclasses from M_1 are in green, from M_2 are in red, and the ones that were a unification of concepts from both models are in yellow.

possibly lead to information loss or inconsistency. Another noteworthy aspect is that the history tracked from M_2 assisted in the recovery of concepts' meanings and links that could have been lost. It would be interesting that in future iterations on the integration of new models, the previous matching models were considered as concepts' history.

Addressing our research question, application of heuristics (especially the Lesk algorithm) removed subjectivity for the matching operation. The matching model also facilitates composition automation and tracking. Even tough, the results we obtained in every step of the process, considering the final metamodel, showed that the requirements domain contains multiple words with wide meaning that are present in a diversity of domains, like actor and role, or even worse, service and system. The fact that composition heuristics do not consider the semantics of these concepts applied to the specific domain endangers the precision of the results and demands more manual revision. Link types are also not able to capture the domain complexity and they too need human intervention to be able to express the proper meaning of relationships other than equality.

5. Conclusion

In this paper, we present an application of general-purpose composition methods for metamodels in the requirements domain and assess its results. We were able to build a composed metamodel for US and UML (from a requirements engineering viewpoint) with cautious manual filtering of the automatically-generated matching links.

The most evident threat to our study is that the application is still inherently dependent on the characteristics of the metamodels being composed; thus, the conclusion validity of our evaluation example is biased on these metamodel selection choices. For example, M_2 lacked any class attributes, making matching and conflict resolution easier. Also, the input models did not have many concepts in common that were matched - that was probably due to their being from different viewpoints (goal and service-oriented). In future work, we intend to systematize these choices to delineate the characteristics of the metamodels that are a better fit for the composition. Another threat is that WordNet does not work well with composed words, which we had to some extent. The matching

heuristics could be improved to better consider composed words and domain semantics. The iterative application of the same methods to aggregate more metamodels, each with its peculiarities, can bring new challenges to the composition process, which we expect to do in a future study that aims at the unification of multiple requirements languages.

References

Abouzahra, A., Sabraoui, A., and Afdel, K. (2020). Model composition in Model Driven Engineering: A systematic literature review. *Information and Software Technology*, 125:106316.

Amaral, L., Siqueira, F., and Brandão, A. (2022). A survey on requirements notations in software engineering research. *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22)*.

Anwar, A., Benelallam, A., Nassar, M., and Coulette, B. (2013). A Graphical Specification of Model Composition with Triple Graph Grammars. In *Model-Based Methodologies for Pervasive and Embedded Software*, volume 7706, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bézivin, J. (2005). On the unification power of models. *Software and System Modeling*, 4:171–188.

Del Fabro, M. D. and Valduriez, P. (2007). Semi-automatic model integration using matching transformations and weaving models. In *Proceedings of the 2007 ACM symposium on Applied computing - SAC '07*, page 963, Seoul, Korea. ACM Press.

Harel, D. and Rumpe, B. (2003). Modeling Languages: Syntax, Semantics and all that stuff. Technical report.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. pages 1–84. Conference Name: IEEE Std 610.12-1990.

IEEE (2017). ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. Technical report, IEEE.

Kaindl, H. and Wagner, P. (2009). A Unification of the Essence of Goal-Oriented Requirements Engineering. In *2009 Fourth International Conference on Software Engineering Advances*, pages 45–50, Porto, Portugal. IEEE.

Patricio, P., Amaral, V., Araujo, J., and Monteiro, R. (2011). Towards a Unified Goal-Oriented Language. In *2011 IEEE 35th Annual Computer Software and Applications Conference*, pages 596–601, Munich, Germany. IEEE.

Rauh, A., Golubski, W., and Queins, S. (2017). A Requirements Meta-Model to Integrate Information for the Definition of System Services. In *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 96–101, San Francisco, CA, USA. IEEE.

Wautelet, Y., Heng, S., Kolp, M., and Mirbel, I. (2014). Unifying and Extending User Story Models. In *Advanced Information Systems Engineering*, volume 8484, pages 211–225. Springer International Publishing, Cham.

Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J., and Fickas, S. (2011). Social modeling for requirements engineering.