Meta-Modeling
Decomposition of Responsibilities

Piotr Zabawa(✉)

Cracow University of Technology, Warszawska 24, 31-155 Krakow, Poland
piotr.zabawa@pk.edu.pl

Abstract. Contemporary known and applied approaches to defining graph modeling languages are standardized. One their characteristic feature, which is fixed and static structure defined at compile time, contains generalizations and thus is difficult to change. The paper presents the results of the process of decomposing responsibilities which may be identified in meta-models. Such decomposition may be done if a meta-model is defined from the compile-time independent meta-model node and meta-model arc classes joint into meta-model graph at run-time. The Context-Driven Meta-Modeling Framework (CDMM-F) was designed to support defining such meta-model. The process of the responsibilities migration from one place they were originally concentrated to the right place is shown in the paper as well and this migration results in mapping them to the right elements of the CDMM-F.

Keywords: Modeling languages · Meta-model
Decomposition of responsibilities

1 Introduction

Meta-modeling is a process of creating rules, constraints and syntactical as well as semantical elements of the method of reality abstraction. Graph-based modeling is one possible approach to meta-modeling. Meta-modeling in the form of graph models and meta-models is a widely accepted approach in software engineering. Meta-models are used in this domain to define general-purpose modeling languages, like the Unified Modeling Language (UML) or to define Domain-Specific modeling Languages (DSLs). Their roles and motivation for leading the research are characterized below.

The role of the general-purpose modeling languages (GPML) is to support software development teams with the language focused on the specification of the software systems under development (horizontal market). The models can be also used for generating software system artifacts (UML) as well as they can be executable (BPMN2). This way of software development processes automation is very important from the economical reasons and is a common technique for improving competitiveness of the IT enterprises.

In contrast, the domain-specific modeling languages (DSML) are extensively used in many application domains (vertical markets) to define easy to use small textual or graphical languages. They are useful for solving simple domain-specific problems and are popular e.g. in enterprise systems [5].

The paper is related to a new approach to defining graph modeling languages. The approach is named Context-Driven Meta-Modeling (CDMM) [14]. The modeling languages defined in CDMM approach are named CDMM-compliant meta-models and they can be general-purpose graph modeling languages or domain-specific ones. Thus, the subject of the paper is related to the OMG concepts as well as to graph DSLs, both mentioned above. And the paper is focused on identification of modeling language responsibilities and constructing a query language for the transformed meta-model.

In contrast to the previous papers dedicated to CDMM-related problems this paper introduces and names different kinds of meta-modeling responsibilities not known from the existing approaches. However, none of the previous papers explained the specific nature of the run-time meta-modeling nor introduced meta-modeling responsibilities characteristic for this approach to meta-modeling. The responsibilities superposition characteristic for the run-time meta-modeling was never published before.

2 State of the Art

There are several well known Object Management Group's (OMG) standards, like Meta-Object Facility (MOF), Unified Modeling Language (UML), Business Process Model and Notation (BPMN2), both built on top of the MOF as well as other modeling standards. Some of them constitute Model-Driven Architecture (MDA) standard. The MDA is dedicated to automating software development processes in the model-driven approach. All MDA standards are general purpose standards and they are re-defined from time to time to be shared among modeling and software development communities to support model-driven software development processes.

The modeling languages defined by the OMG are created at compile-time and thus they are named compile-time meta-models or compile-time modeling languages further. In contrast, the modeling languages created at run-time are named run-time meta-models or run-time modeling languages. The paper is dedicated to run-time meta-models, nevertheless it refers also to compile-time modeling languages.

The paper contains a discussion of responsibilities that can be identified in contemporary modeling languages. However, the discussion is applicable for run-time modeling languages only. It may be, however perceived as a strong motivation to defining modeling languages this way due to many advantages.

There are known some frameworks for defining meta-models, like for example [7]. The CDMM approach presented in the article is also supported by the appropriate CDMM-F framework [13]. The paper refers both the CDMM-F and the meta-models to determine which meta-model responsibility should be mapped to the framework or to the meta-model. As the result of the analysis described in Sect. 4.1 the paper the Context-Driven Meta-Modeling Framework (CDMM-F) [11] was designed and implemented. The design was based on the concept of decomposition of responsibilities and their correct mapping to framework and meta-model elements. In consequence, the CDMM-F constitutes the feasibility case-study for the decomposition of responsibilities discussed in the paper. The CDMM-F is introduced first in the article as it forms the base for identifying responsibilities, their decomposition and mapping mentioned above. The special role of application context mentioned further in the paper is presented in [10].

3 Context-Driven Meta-Modeling Fundamentals

The Context-Driven Meta-Modeling (CDMM) approach to defining graph modeling languages (meta-models) is different from the OMG concept. The key assumption in CDMM is that meta-models are treated as data models [8]. In the consequence the generalization relationships are excluded from CDMM-compliant modeling languages. It contrasts to other approaches where generalization constitutes an important modeling element, as it was described for AODB in [4,6]. Moreover, in the CDMM a meta-model designer may introduce classes not only for meta-model graph nodes but also for graph edges. All these classes are independent one of the other at compile-time. Such a meta-model graph can be semantically enriched by introducing entity classes both to meta-model graph nodes and graph edges. The entity classes may form aggregation hierarchies. In the consequence of such assumptions, the whole such a graph structure can be defined at run-time. The formal representation of the CDMM approach is illustrated in the Fig. 1 in the form of the CDMM meta-meta-model class diagram [13].

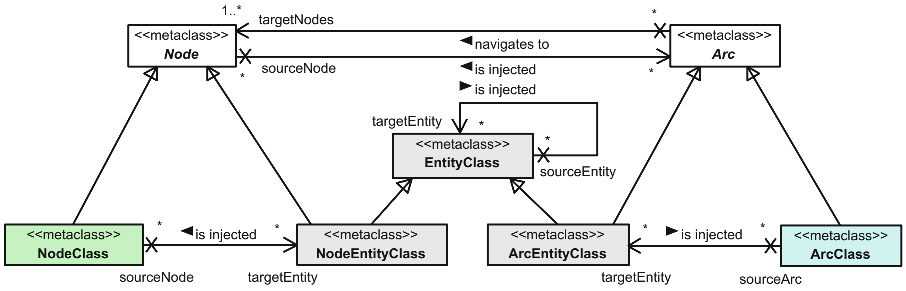

Fig. 1. CDMM meta-meta-model - from [13].

The CDMM approach to defining modeling languages offers significant ease of change introduction to modeling languages as they are defined at run-time.

Meta-models specified this way may easily become graph domain languages, that is - graph DSLs. On the other hand, the CDMM-compliant modeling languages may be created in the form of OMG modeling or meta-modeling languages customization.

The CDMM concept presented in the Fig. 1 is implemented in the form of the CDMM-F, the framework which supports defining CDMM-compliant modeling languages. The framework is implemented in the combination of Java-related technologies, namely Java, Spring, AspectJ, Guava and Apache commons.

The universality of CDMM approach manifests in the fact that the modeling language can be defined in any form not necessarily addressed to class-object paradigm. In the consequence the application of CDMM approach makes accommodating new technologies and new paradigms easier than in the case of MDA approach.

The next sections present some observations made during several stages of the evolutionary approach applied to the CDMM-F design and implementation process and resulting in the CDMM meta-meta-model presented in the Fig. 1. The special focus of this kind of research was put just on the identification and decomposition of meta-model and CDMM-F responsibilities. Then, the identified responsibilities were mapped to the right element of the CDMM-F or the meta-model.

4 Migration of Responsibilities in the CDMM-F

The analysis presented in this section leads from the identification of existing but not evident meta-model responsibilities to their mapping to the right place in the run-time meta-model or the framework. This mapping is required for the correct CDMM-F design.

The following stages were applied to identify the right place of implementation for particular responsibilities:

– STAGE ONE - the meta-model responsibilities were identified as it is described in Sect. 4.1;
– STAGE TWO - the maximal number of responsibilities was associated to the `Arc` classes for the meta-model while removing them from meta-model `Node` classes;
– STAGE THREE - the graph query language was designed in such a way that each meta-model element instance (CDMM-compliant model element) may be obtained from `Arc` class instances;
– STAGE FOUR - the responsibilities focused in the query language elements were moved to different elements of CDMM-compliant meta-models or to the elements of the CDMM-F.

The stages are characterized in the succeeding subsections.

4.1 STAGE ONE: Responsibilities in CDMM-Compliant Meta-Models

In this section the analysis of the CDMM meta-meta-model presented in the Fig. 1 is performed in order to identify the meta-modeling responsibilities being the subject of migration. Thus some responsibilities are inferred from the mentioned class diagram. The CDMM meta-meta-model is conceptual. In the consequence, the CDMM-compliant meta-models are not instances of the meta-meta-model. They should be rather perceived as meta-meta-model concretizations or realizations. The CDMM meta-meta-model as a conceptual one is not created in the CDMM-F in any form. The advantage of it is being just a model of roles the meta-model classes play regarding the CDMM meta-model graph.

The responsibilities inferred in this section are further mapped to the different elements of the meta-model and CDMM-F. This mapping drives the CDMM-F design decisions as well as meta-model design decisions.

The characteristic feature of MDA standards is the fixed structure (hierarchy) of their meta-models. Moreover, the meta-models of the MDA sub-standards are monolithic - their responsibilities are decomposed to packages and to generalizations from abstract classes or implementation of marker interfaces, sometimes abstract classes are used as markers. Also, the relationships between MDA sub-standard meta-model nodes are represented in the form of references (or inheritance or implementation relationships) and not in the form of classes [1–3,9]. In consequence, the set of available relationships is limited. In this section the results of the analysis of possible responsibilities that can be found in meta-models (also in the MDA meta-models) are presented from the perspective of CDMM approach. These results are inferred from the CDMM-F implementation efforts.

As it results from the CDMM meta-meta-model presented in the Fig. 1, each CDMM-compliant meta-model class can be a `Node` class or an `Arc` class or an `Entity` class. According to the CDMM assumptions these classes are not related at compile-time and the meta-model is created from these classes at run-time. The `Node`, `Arc` and `Entity` classes are defined by a meta-model designer for the purpose of a particular modeling language. They are also highly reusable both in the source code level (Java source code files) and in the byte code level (compiled class files or jar files). They can be easily exchanged between different meta-model projects and between end user applications and meta-model projects. The character of `Node`, `Arc` and `Entity` meta-model classes is determined by their role. For the `Node` classes it is enough just to exist and be empty. `Entity` classes should store data in the form of the attributes. `Arc` classes must be defined in the form resulting from the Fig. 1. More specifically, each `Arc` class must define its `targetNodes` element. This element plays role of the container for information about type of the `targetNodes` and for the object being instances of the `targetNodes` class. The `targetNodes` classes are determined when a CDMM-compliant meta-model is constructed from the meta-model definition (application context) while the `targetNodes` objects are put into the `targetNodes` container while instantiating the meta-model (model creating). Both `Node` and `Arc` classes may also contain data in attributes. In such a case they share their role with the role of `Entity` classes. Each `Arc` class is associated to a `Node` class at run time on the stage of meta-model creation. The concept of injections is applied here. The same injection mechanism is also applied for run-time `Entity` classes associating to `Arc` or `Node` classes as well as to `Entity` classes. As a special case an `Entity` class may be shared between several `Node` and several `Arc` classes by injecting the `Entity` class to each of the mentioned `sourceNode` and `sourceArc` classes. It is shown in the Fig. 1 that the cardinalities of relationships between `Node` and `Arc` classes are many-to-many. It means, that each `Node` class may have many `Arc` class injected and each `Arc` class may be injected to many `Node` classes. However, each `Arc` class must have at least one `targetNodes` end (otherwise it does not relate anything). So, the responsibility of `Arc` classes is to have a `targetNodes` end. The reflexive (that is self) association in `EntityClass` form the Fig. 1 means that the `Entity` classes may form association (aggregation) hierarchies.

All responsibilities mentioned in this section are connected to meta-model elements (user defined classes), to the meta-model itself (its representation in memory) or to the definition of the meta-model (the file which defines an application context for the CDMM-F).

4.2 STAGES TWO and THREE: CDMM-F Query Language

In this section the results of STAGE TWO and THREE are discussed together as the goal of the analysis presented in the paper is just the STAGE FOUR, which in turn is illustrated in Sect. 4.3.

The decision about constructing meta-model query language around `Arc` classes was driven by the fact that in the available modeling languages the number of different types of relationships (`Arc` classes in CDMM) is significantly smaller than the number of different types of nodes (`Node` classes in CDMM). Moreover, the `Node` classes are unique in a meta-model while the `Arc` classes may appear many times in a modeling different (say, between two pairs of `Node` classes in the case of bnary relationships) of `Node` classes and inside a set of `Node` classes (say, between two `Node` classes);

The graph query language was designed in such a form that being in a particular meta-model `Node` all `Arc` classes injected to this `Node` class the `Node` class object is casted to the `Arc` class and the right `Arc` class `targetNodes` attribute is found by the query.

The construction of a sample query `Arc` class that fulfils the concept from the last paragraph is presented below. First the CDMM-F elements referenced further are presented in the Figs. 2 and 3.

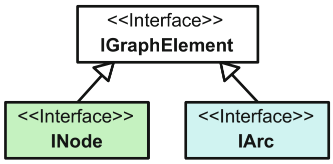

Fig. 2. CDMM-F top hierarchies for the roles in the graph.

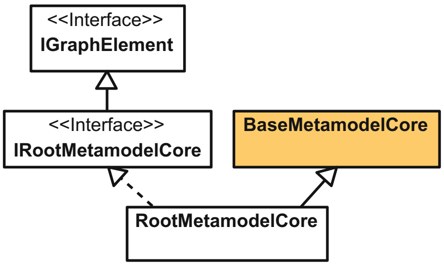

Fig. 3. CDMM-F top hierarchies for the roles in the meta-model core elements.

The CDMM-F query language was designed according to the following assumptions:

- the subject of each query is a `Node` class; this `Node` class plays the role of a `sourceNode` class of a relationship injection as was depicted in the Fig. 1
- the query contains the `sourceNode` object, the `Arc` class, the `targetNode` class as its arguments
- the query returns the list of `targetNode` objects.

The meta-model query language design rules presented above fulfill the requirement of making it possible to identify all relationship instances and return the list of all objects on the `targetNode` end of each relationship instance. It is sufficient to traverse the whole meta-model graph instance, that is the graph which is composed of meta-model objects (CDMM-compliant model) interrelated according to the injections (CDMM-compliant meta-model).

Below, the design of classes implementing this language is analysed.

The most general representation of the type of a class of the most complex relationship applied so far has the following form in the query language mentioned above:

```
Multimap<List<Pair<IGraphElement,Class<?>>>, BaseMetamodelCore>
```

This form of type reflects the superposition of the responsibilities identified in Sect. 4.1. The superposition is expressed in the Expr. 1. The appropriate elements of the type and responsibilities the elements of this type reflects are displayed in the same colors.

$$\textbf{\textit{inter-object-relations}} \circ \textbf{\textit{arity}} \circ \textbf{\textit{binary-relation-id}} \circ \textbf{\textit{targetNode-objects}} \tag{1}$$

where:

- *inter-object-relations* stands for the inter-object relation instances for a particular `Arc` class injected to `Node` classes
- *arity* stands for the number of `Node` classes interrelated by an `Arc` class (N-ary nature of a relationship), thus the number of injections of an `Arc` class to the `Node` classes represented by the `sourceNode` multiplicity in the Fig. 1

- *binary-relation-id* stands for the key used to uniquely identify a binary relationship
- *targetNode-objects* stands for the number of objects located at the `targetNode` association roleName presented in the Fig. 1.

The consequences of a special approach to representing N-ary relationships, which was applied in CDMM meta-model graph query language is toched on further in the paper and will be published in separate paper.

It is worth noting that the superposition from the Expr. 1 is similar to the Decorator design pattern applied to generic types in place of the object types. Thus, the superposition of responsibilities may be seen as the Decorator design pattern moved from class-object paradigm to generic paradigm. Objects are decorated by relations, relations are decorated by N-arity, N-arity is decorated by the objects cardinality. Moreover some elements of this superposition are defined at meta-modeling level while some of them are defined at modeling level.

Another observation is that the order of superposed element must remain unchanged while responsibilities migration or while simplification of the relationship.

The Expr. 1 constitutes the result of the STAGES 2 and 3 discussed at the beginning of the current section.

4.3 STAGE FOUR: Responsibilities Mapping

The results of the STAGE 4 are presented in the Tables 1 and 2. The same coloring convention as the one used in Sect. 4.2 was applied in both tables to show the result of mapping all colored responsibilities to meta-model and CDMM-F locations.

All identified meta-model responsibilities were mapped to the right place in run-time CDMM-compliant meta-model elements according to the Table 1. They were also implemented according to the Table 2 in the CDMM-F framework both as the meta-model graph definition specified in the form of the application context and in the form of the CDMM-F software itself. The framework was then extensively tested both by manually defined meta-models oriented to testing the correctness of each responsibility mapping and in the form of semi-automatic simulation-like approach presented in [12]. In the last approach to testing the manually defined meta-models constituted the test kernels surrounded by automatically generated test context - the supergraphs for test kernels. The tests confirmed the correctness of the responsibilities decomposition.

As it was shown in the Tables 1 and 2, some responsibilities are mapped to more than one locations, like for example "being a subject of entities injections" which is distributed between `Node` class, `Arc` class and `Entity` class. Also, each location is mapped to many responsibilities. It means, that the mapping is many-to-many.

It is worthy of notice that the N-ary relationship responsibility is associated to the application context in the Table 2. In contrast to compile-time meta-models, arity is not the responsibility of the `Node` class. There are many consequences

Table 1. Mapping from the CDMM responsibilities to the meta-model elements.

Responsibility	Location
Being a meta-model node or arc	Element class
Being a meta-model graph node	`Node` class
Being a subject of arcs injections	
Being a subject of entities injections	
Being a meta-model graph arc	`Arc` class
Being able to be injected to the `sourceNodes`	
Being a container for `targetNodes` both for `Node` classes and `Node` objects (relationship end)	
Being directed from the `sourceNode` to `targetNodes`	
Being a subject of entities injections	
Being possibly bi-directed	
Storing and exposing meta-model element attributes	`Entity` class
Belonging to an aggregation hierarchy of entities	
Being able to be injected to the nodes, arcs or entities	
Being a subject of entities injections	

Table 2. Mapping from the CDMM responsibilities to the CDMM-F framework elements.

Responsibility	Location
Defining a scope for Elements	Application context
Defining a unique name for Elements	
Defining an arc as being a binary or N-ary relationship (arity)	
Defining an arc as a reflexive relationship	
Defining the whole meta-model graph composed of node, arc, entity classes and their injections	
Having APIs for meta-model graph traversal	CDMM-F
Having API for meta-model graph elements instantiation (creating model in memory as a graph of objects)	
Having a meta-model graph query language	
Having scope management factories	
Having configuration for a meta-model project	
Being able to instantiate the meta-model classes (creating meta-model in memory as a graph of meta-model element classes)	

of moving arity responsibility from a `Node` class to the application context. The detailed discussion of the N-ary relationships in run-time meta-models is out of scope of the paper.

5 Conclusions

One consequence of introducing the CDMM concept of defining modeling languages at run-time is the ability to identify different responsibilities, which are not clear as long as the compile-time metamodel definitions are taken into account. It was shown in the paper that many responsibilities can be uncovered, identified, taken away from meta-model graph hierarchy and, finally placed in the right places when moving from compile-time meta-models to the run-time ones. The new responsibilities were also named and mapped to the meta-model elements according to the Table 1 and meta-modeling framework elements according to the Table 2.

The right paradigm for implementing meta-modeling query language according to the concept assumed in the paper is a generic paradigm. All meta-model Elements, that is `Node` classes, `Arc` classes and `Entity` classes may be implemented in class-object paradigm as classes unrelated at compile-time. Meta-model graphs are created in CDMM-F with the aid of aspect-oriented paradigm. The combination of mentioned object-oriented paradigms is sufficient and well suited to make all CDMM concepts including the decomposition of responsibilities feasible.

References

1. Bildhauer, D.: On the relationship between subsetting, redefinition and association specialization. In: Proceedings of the 9th Baltic Conference on Databases and Information Systems, Riga (2010)
2. Diaz, I., Llorens, J., Genova, G., Fuentes, J.: Generating domain representations using a relationship model. Inf. Syst. **30**, 1–19 (2005)
3. Génova, G., del Castillo, C.R., Llorens, J.: Mapping UML associations into Java code. J. Object Technol. **2**(5), 135–162 (2003)
4. Krótkiewicz, M.: A Novel inheritance mechanism for modeling knowledge representation systems. Computer Science and Information Systems (2017)
5. Krótkiewicz, M., Jodłowiec, M., Wojtkiewicz, K., Szwedziak, K.: Unified process management for service and manufacture system—material resources. In: Burduk, R., Jackowski, K., Kurzyński, M., Woźniak, M., Żołnierek, A. (eds.) Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015. AISC, vol. 403, pp. 681–690. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-26227-7_64
6. Krótkiewicz, M.: Association-oriented database model - n-ary associations. Int. J. Softw. Eng. Knowl. Eng. **27**(02), 281–320 (2017)
7. Malhotra, R.: Meta-modeling framework: a new approach to manage meta-model base and modeling knowledge. Knowl.-Based Syst. **21**, 6–37 (2008)
8. Merson, P.: Data model as an architectural view. Technical note CMU/SEI-2009-TN-024, Software Engineering Institute, Carnegie Mellon University (2009)

9. Tan, H.B.K., Yang, Y., Bian, L.: Improving the use of multiplicity in UML association. J. Object Technol. **5**(6), 127–132 (2006)
10. Zabawa, P.: Context-driven meta-modeling framework (CDMM-F) - context role. Tech. Trans. **112**(1–NP), 105–114 (2015)
11. Zabawa, P.: Context-Driven Meta-Modeling Framework (CDMM-F) - Internal Structure (2017, accepted for publication)
12. Zabawa, P.: Simulation of the CDMM-P paradigm-driven meta-modeling process. Tech. Trans. **4**, 143–154 (2017)
13. Zabawa, P., Hnatkowska, B.: CDMM-F – domain languages framework. In: Świątek, J., Borzemski, L., Wilimowska, Z. (eds.) ISAT 2017. AISC, vol. 656, pp. 263–273. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-67229-8_24
14. Zabawa, P., Stanuszek, M.: Characteristics of the context-driven meta-modeling paradigm (CDMM-P). Tech. Trans. **111**(3), 123–134 (2014)