INFORMATION SYSTEMS IN MANAGEMENT

Information Systems in Management (2016) Vol. 5 (1) 144−158

CONTEXT-DRIVEN META-MODELER (CDMM-META-MODELER) APPLICATION CASE-STUDY

PIOTR ZABAWA [a], GRZEGORZ FITRZYK [b], KRZYSZTOF NOWAK [c]

[a] *Department of Physics, Mathematics and Computer Science, Cracow University of Technology, Cracow, Poland*
[b] *graduate of Department of Physics, Mathematics and Computer Science, Cracow University of Technology, Cracow, Poland*
[c] *Department of Civil Engineering, Cracow University of Technology, Cracow, Poland*

The main contribution of this paper is the working case study for meta-modeling process performed in open ontologies. It contrasts to close ontology based approaches well known from software engineering discipline. Moreover, in place of ontological standards like Resource Description Framework (RDF) defined by World Wide Web Consortium (W3C) or Web Ontology Language (OWL) by W3C and Object Management Group (OMG), the presented meta-modeling approach is based on notions characteristic for software engineering, like class, relationship, Unified Modeling Language (UML), UML Profile, stereotype, meta-model as well as for enterprise applications. This approach is feasible as it refers to the concept of Context-Driven Meta-Modeling (CDMM) introduced in previous papers and implemented in the form of Context-Driven Meta-Modeling Framework (CDMM-F). The case study is realized in the form of graphical UML modeling of the modeling language (meta-model) in the Context-Driven Meta-Modeling Meta-Modeler (CDMM-Meta-Modeler) Thus the presented case-study constitutes the proof-of-the-concept for graphical meta-modeling for all mentioned concepts and their implementations. It also displays the nature of the meta-modeling process in this paradigm and explains some mechanisms that play important role when process effectiveness and convenience of the meta-model designer are taken into account.

Keywords: meta-model; application context; open ontology; modeling language; meta-modeling process; visual modeling; UML; UML Profile

1. Introduction

This paper presents a case study for the modeling language design process. It is based on Context-Driven Meta-Modeling Paradigm (CDMM-P) [31] and Context-Driven Meta-Modeling Framework (CDMM-F). The CDMM-F constitutes one possible implementation of the CDMM-P. The name and the special role of the application context in CDMM-F implementation of the CDMM-P are explained in [27]. The diagramming problem for open ontology based approach to meta-modeling is the main subject of the paper.

The meta-modeling case study is performed in the Context-Driven Meta-Modeling Meta-Modeler (CDMM-Meta-Modeler) tool presented earlier in [7] but introduced on the CDMM background in [29]. The CDMM-Meta-Modeler is the Eclipse PlugIn based on UML2 Eclipse PlugIn and implemented with the aid of relatively large number of technologies named in [7, 29]. The paper [29] is focused on the implementation issues of CDMM-Meta-Modeler while the presented paper is addressed to the visual meta-modeling process issues.

Wide and careful research of scientific and commercial literature was performed and can be found mainly in [31]. The conclusion from the research was that there are no direct references to the approaches similar to the one introduced in [31]. Some literature located on the border of ontology and software engineering domain can be identified, however it explores RDF or OWL standards for ontologies [1, 3, 4, 5, 8, 9, 14, 15, 16, 23] or refers to the systems of notions (ontologies) used in software engineering discipline and addressed mainly to the notions of software engineering process [6, 10, 17, 22, 24].

In the paper some notions which are well known in software engineering discipline are used and they refer to Object Management Group (OMG) standards like Unified Modeling Language(UML) [2, 21, 12, 13]. There are also papers that are focused on the process of meta-modeling, like [20, 18, 19, 26, 25]. These two groups of papers are the closest to the system of notion used in the paper. According to [11] open ontologies are not known and are not applied to software engineering discipline. The authors are not aware of any reference in the scientific literature to date to the presented CDMM approach, so the paper constitutes the first contribution to the subject, related to visual modeling.

2. Context-Driven Meta-Modeling Meta-Modeler

The CDMM-Meta-Modeler tool was presented in the context of CDMM concept in [29]. Nevertheless, it is shortly characterized in this section.

The tool has the form of Eclipse PlugIn and was implemented in the following technologies: Eclipse RCP, Java SE, Equinox OSGi, JavaFX, SWT, JFace and UML2 SDK. The role of it is to offer the graphical UML modeling framework for meta-model designer and integrate to CDMM-F and UML2 SDK PlugIn. This way the model of the modeling language can be created in standard UML way, it can be explored from any software system via UML2 PlugIn API and, first of all, the application context file for CDMM-F can be created, as shown in [27].

The main functionalities of the CDMM-Meta-Modeler were presented in [29], so only the most critical ones are contained in the paper in the form of Table 1.

Table 1. Most critical functional features of CDMM-Meta-Modeler

Functional feature	Purpose
UML Profile diagramming	Defining stereotypes for meta-model entity and relation classes
UML class diagramming for meta-model elements	Defining meta-model entity and relation classes
	Stereotyping meta-model entity classes
UML class diagramming for meta-model graph	Placing meta-model entity classes (stereotyped or not) as meta-model graph nodes
	Stereotyping meta-model entity classes
	Introducing associative (composition, aggregation, association) and dependency relationships as meta-model graph edges connecting meta-model entity classes
	Stereotyping meta-model graph edges with the names of meta-model relation classes
Creating meta-model graph XML representation	Exporting simple graph representation or application context to be loaded by CDMM-F

The functionalities shown in Table 1 play the key role in the case study discussed in section 3.

3. CDMM-Meta-Modeler Application Case-Study

This section presents a case study for the application of the CDMM-Meta-Modeler Eclipse PlugIn. In the case study a sample modeling language (meta-model) is created from the CDMM-Meta-Modeler according to open ontology.

3.1. Sample Meta-Model

Our goal in the case-study is to create the modeling language (meta-model) presented in the Figure 1. The following coloring convention is assumed in the whole case-study: a) red color represents CDMM-F elements; b) green color represents meta-model entities, that is - elements that can be placed in meta-model graph nodes; c) blue color represents meta-model relations, that is - elements that can be placed in meta-model graph edges; d) grey color represents elements of the meta-model graph which are not expressed in CDMM-Meta-Modeler notation.

The diagrams presented in Figure 1, Figure 4, Figure 5 and Figure 7 were created in commercial modeling tool while the remaining diagrams (Figures 2-3 and Figure 6) were created in CDMM-Meta-Modeler.

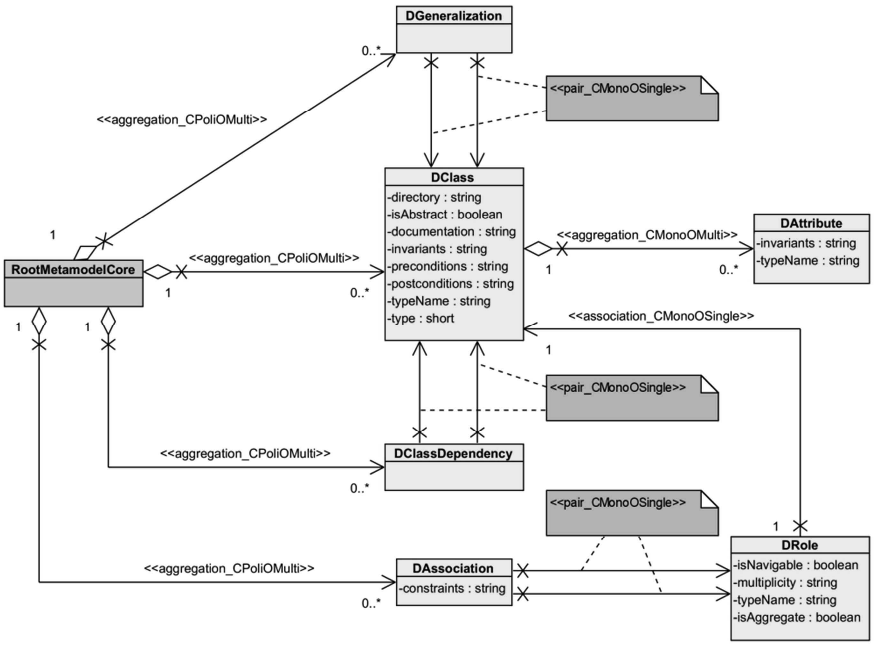

Figure 1. Sample CDMM-F meta-model defined in a UML modeling tool

The meta-model presented above is used to construct modeling language through diagramming in CDMM-Meta-Modeler. The modeling language defined in the Figure 1 can be used to model static information about a software system. Models created in this language may contain information known from UML class diagrams. However the meta-model contains pairs of relationships of the same kind which are very frequently met in meta-models. In contract to CDMM-Meta-Modeler notation shown in the Figure 5, they are expressed in UML.

It is worth noticing that in this meta-model some meta-model entity classes (DGeneralization, DClass, DClassDependency and DAssociation) are connected to the CDMM-F core meta-model root class (RootMeatmodelCore) via relationships. This way the user-defined meta-model classes are associated to CDMM-F directly (via mentioned relations) or indirectly (via user-defined relationships already existing in the meta-model). In the case of the CDMM-Meta-Modeler the association of meta-model to the CDMM-F root class is achieved via stereotyping of classes according to the subsection 3.4.

3.2. Sample Meta-Model Entity Classes

In order to define the entities of the meta-model presented in the Figure 1, which is created in CDMM-Meta-Modeler, the appropriate classes should be defined in the form of Java source code or in the form of CDMM-Meta-Modeler classes. The round-trip engineering technique can be applied for their definition. The CDMM-Meta-Modeler class diagram that contains definitions of the meta-model entities is presented in the Figure 2a.

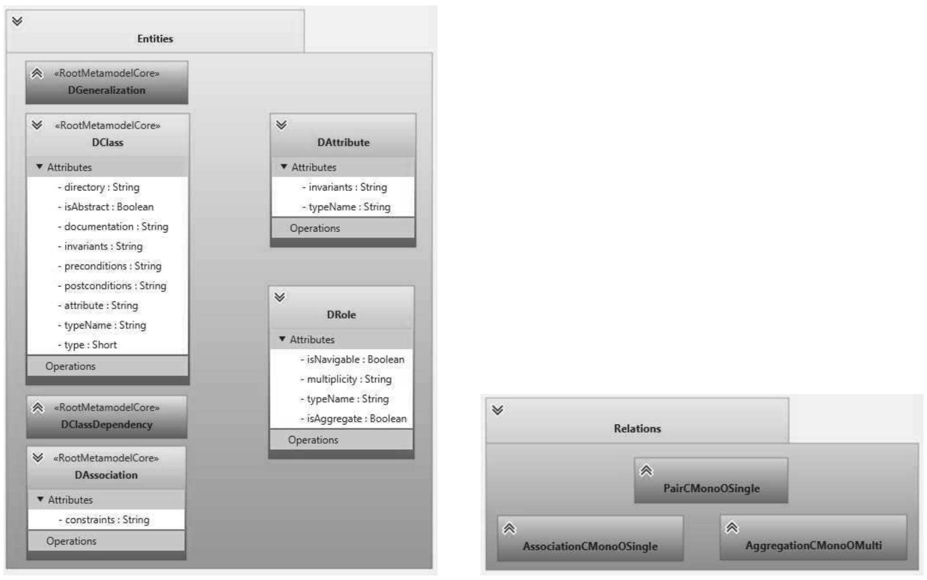

Figure 2. Sample CDMM-F meta-model a) entities and b) relations defined in CDMM-Meta-Modeler

It is worth noticing that these classes: are not interconnected in any form; are not connected to any user-defined meta-model relationship classes; the classes that are intended to be connected to the root CDMM-F class are stereotyped by the name of the root class (RootMetamodelCore).

All these classes will be placed in the nodes of meta-model graph, what is presented in subsection 3.6.

3.3. Sample Meta-Model Relations Classes

The meta-model user-defined relationship classes are also defined in CDMM-Meta-Modeler in the way similar to definition of meta-model entity classes. Their definition for the sample meta-model from the Figure 1 is presented in the Figure 2b. The user-defined meta-model relationship classes are not interconnected in any form and are not connected to any user-defined meta-model entity classes.

One of these classes can be reused in order to associate «RootMeatmodelCore» stereotyped meta-model entity classes to CDMM-F root class. Nevertheless, one such default relationship class is predefined in CDMM-F and distributed with the framework. This class can be reused when meta-model graph is designed. The way these meta-model relationship classes are used while meta-model graph definition is explained in subsection 3.6.

3.4. Connection to CDMM-F

The characteristic feature of open ontology based meta-modeling is that there are no compile-time relationships between any classes of the meta-model. The only relationships allowed are with the CDMM-F classes, however they are not presented in the diagrams above (except the one from the Figure 1) for simplification (except the «RootMetamodelCore» stereotype). However, the nature of these meta-model associations to the CDMM-F classes is shown in the Figure 3.

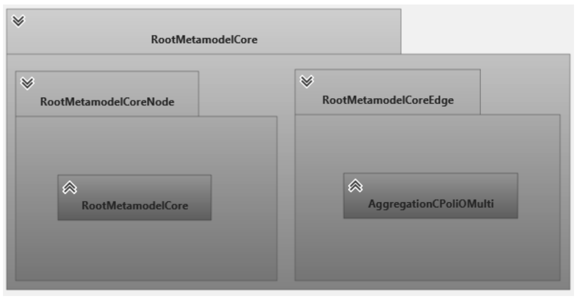

Figure 3. CDMM-F and thus CDMM-Meta-Modeler root meta-model classes

The Figure 3 displays the CDMM-F package structure and two following classes:

- RootMetamodelCore that belongs to the CDMM-F - this class cannot be redefined by the CDMM-F and, in consequence CDMM-Meta-Modeler user
- AggregationCPoliOMulti which is predefined in CDMM-F - this class can be redefined by the user of both CDMM-F and CDMM-Meta-Modeler systems or the CDMM-Meta-Modeler may be instructed to reuse one of user-defined relation classes to play the role of the CDMM-F framework class.

The important assumption here is that there must be defined exactly one class in each of the RootMetamodelCoreNode and RootMetamodelCoreEdge (see the Figure 3) packages. These classes are dedicated for the purpose of associating meta-model elements with the CDMM-F when the meta-model graph is defined in CDMM-Meta-Modeler. The way the graph is created from the classes presented so far is explained in subsection 3.6.

3.5. Profiles for Meta-Model Graph Elements Stereotyping

The UML stereotyping mechanism is used in CDMM-Meta-Modeler for associating meta-model entity classes to CDMM-F root class (see the Figure 2a) and for associating meta-model graph relationships to meta-model relationship classes (see the Figure 5).

In order to introduce these stereotypes into CDMM-Meta-Modeler the UML extension mechanism is applied – the required stereotypes are defined through UML Profiles in CDMM-Meta-Modeler.

So, the next step is to define two profiles:

- RootMetaModelProfile
- GeneratedProfile

The first profile RootMetaModelProfile is defined for associating meta-model entity classes to CDMM-F root class. The second one is dedicated to associating meta-model graph relationships to meta-model relationship classes.

In fact, the need for user activities is very limited here, as the RootMetamodelProfile is predefined in the framework. However, if the user wants to exchange the predefined AggregationCPoliOMulti to his/her own implementation, then he/she must change the name of it manually. The remaining, GeneratedProfile does not need any user activities as this profile is generated on the fly by the CDMM-Meta-Modeler on the basis of meta-model relation classes shown in the Figure 2b. The contents of both profiles for the meta-model are shown in the Figure 4a for RootMetaModelProfile and in the Figure 4b for GeneratedProfile.

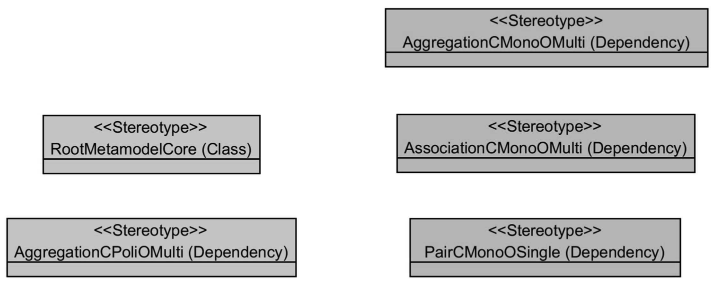
<<Stereotype>>
AggregationCMonoOMulti (Dependency)
<<Stereotype>>
RootMetamodelCore (Class)
<<Stereotype>>
AssociationCMonoOMulti (Dependency)
<<Stereotype>>
AggregationCPoliOMulti (Dependency)
<<Stereotype>>
PairCMonoOSingle (Dependency)

Figure 4. CDMM-Meta-Modeler Profiles: **a)** RootMetaModelProfile, **b)** GeneratedProfile

Both profiles are used to manually stereotype entity classes when they are created according to subsection 3.2 and relationship classes when meta-model graph is created according to the subsection 3.6.

Manual stereotyping is time consuming and error prone. That is why automation is planned to be used in future implementation of the CDMM-Meta-Modeler.

3.6. Definition of Sample Meta-Model Graph

Now, all meta-model elements that are required for meta-model creation from CDMMMeta-Modeler are available. So, the only thing is to associate all of them into the graph structure that represents the modeling language. In order to do that the UML diagram (to define the meta-model graph structure) must be created and some meta-model elements must be stereotyped.

The required and sufficient meta-model graph diagram created in CDMM-Meta-Modeler for the CDMM-like representation of the sample meta-model from the Figure 1 is presented in the Figure 5.

It is worth noticing in the Figure 5 that:

- meta-model entity classes are defined to be interconnected to the root Root-MetamodelCore class of the CDMM-F (that is why they are displayed in red font on the green background according to the coloring convention introduced in subsection 3.1); the only stereotype «RootMetamodelCore» is predefined in CDMM-MetaModeler and can be applied to meta-model entity classes only;
- meta-model entity classes that can be reached from the stereotyped classes are not stereotyped (they are displayed in green);
- for directed meta-model graph the dependency relationships can be used;
- the meta-model graph relationships (displayed in blue according to the coloring convention) are associated to the meta-model relationship classes via stereotypes the names of which are equivalent to the meta-model relationship class names; the stereotype names for the relationships are taken from the GreneratedProfile UML Profile defined in CDMM-Meta-Modeler; more relationships can be used in the meta-model graph diagram - all kinds of associative relationships (composition, aggregation, association) especially to represent two-directional meta-model relationships;
- some relationships cannot be represented in UML as they do not exist in this standard's meta-model; for example the RPair relationship does not exist while it is useful in meta-modeling; in consequence the lacking RPair relationship is modeled in the Figure 1 by two association relationships with the appropriate note (in grey) connected to each pair (OCL expression can be used to define this constraint as well); however the same RPair relationships are modeled in CDMM-Meta-Modeler meta-model graph diagram as one meta-model notion - a relationship, what is visible in the Figure 5; dual nature of the relationship is represented by the appropriate implementation of the relationship class that is associated to the dependency relationship via stereotype; (see for example DAssociation - DRole relation in both Figure 1 and Figure 5); this is one of advantages of open ontology based meta-modeling. Another good example is N-ary association from UML which does not have its representation in UML meta-model and can be easily introduced to the CDMM.

The entity class stereotypes can be introduced:

- manually in the Entity package while entity classes are defined
- manually in the meta-model graph diagram (better practice)
- automatically for the meta-model graph elements (the best practice)

The relation class stereotypes are generated automatically on the basis of the meta-model relationship class names and placed in the Generated Profile. The list of stereotypes available for a particular relation is offered by the CDMM-Meta-Modeler when meta-model graph diagram is created by meta-model designer.

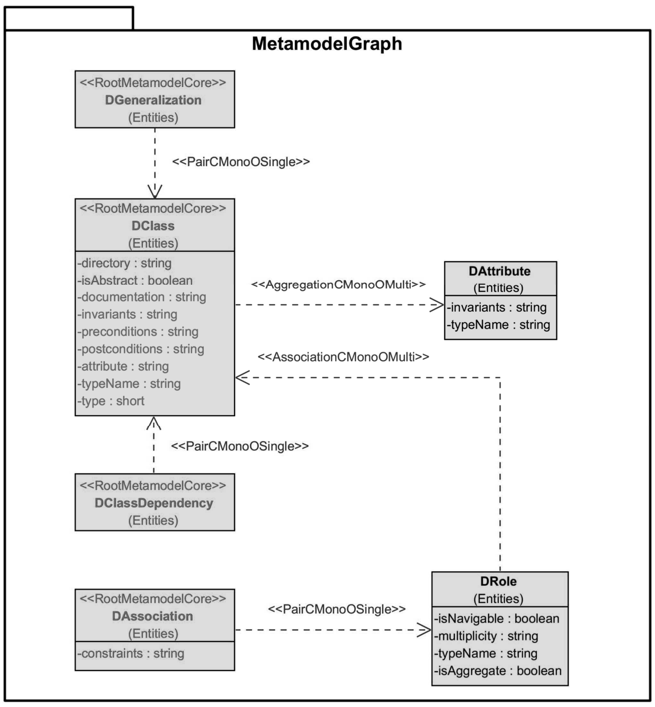

Figure 5. The meta-model graph structure arranged from meta-model elements

In order to illustrate the diagramming functionality of the presented software tool the same meta-model graph created in CDMM-Meta-Modeler is shown in the Figure 6. This diagram does not follow the coloring convention introduced for the purpose of the paper.

The whole meta-modeling process, thus the process of the design of a modeling language as a whole is presented in subsection 3.7. The process is general but is placed in the subsection 3.7 in the section 3 which is focused on the case study, as the discussion of the process refers to the notions connected to the case study.

3.7. Meta-Modeling Process

One of the goals of CDMM-Meta-Modeler was to simplify the work that must be performed by the person who defines a modeling language. It was achieved by minimization of the required user activities - both number and scope. More specifically, the "convention over configuration" and automation approaches were used for profile definition tasks as it was already mentioned above in subsection 3.5. In the case of diagramming the number of elements that must be managed manually is also limited and the manual tasks are simplified.

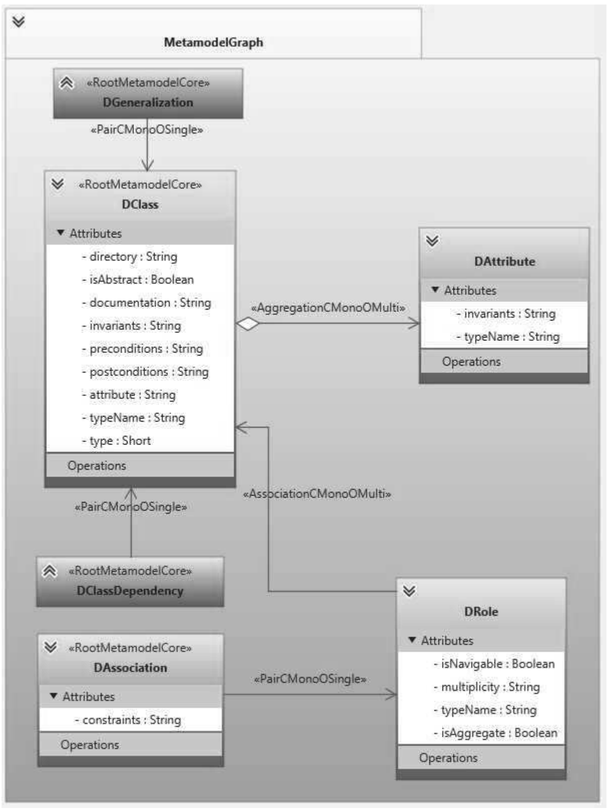

Figure 6. Sample meta-model graph diagram created in CDMM-Meta-Modeler

The manual tasks in the final version of the CDMM-Meta-Modeler can be limited to:

- defining entity classes
- defining relationship classes
- associating some meta-model entity classes via meta-model relationship classes on the meta-model graph diagram

- stereotyping all relationships on the meta-model graph diagram by the stereotypes equivalent to the names of already defined (not necessarily already implemented)meta-model relation classes

The meta-model classes mentioned above must be, of course, implemented. But they are subject of extensive reuse in meta-modeling, that is why defining and implementing them is required for the first version of the meta-model and could be required on the very limited extent for future versions of the meta-model. The main subject of introduction/change are meta-model entity classes that are very easy for implementation (the whole source code of these classes may be generated from their CDMM-Meta-Modeler definition - from the model of the meta-model stored in CDMMMeta-Modeler in the form of UML2 Eclipse PlugIn model representation). This model of meta-model can be thus accessed from external software through the UML2 Eclipse PlugIn API. This way the CDMM-Meta-Modeler has the open architecture.

The steps of the process of constructing the meta-model graph shown in the Figure 5, as well as any other meta-model, are as follows:

- the entity classes are created in the CDMM-Meta-Modeler
- the relationship classes are created in the CDMM-Meta-Modeler
- the green classes are dragged and dropped from the Entities package to the MetamodelGraph diagram; they represent nodes of the meta-model graph
- the RootMetaModelProfile is optionally updated in case the default relation class is exchanged to the user-defined one
- the GeneratedProfile is generated by the CDMM-Meta-Modeler
- the dependency relationships are introduced into MetamodelGraph diagram
- the dependency relationships on the CDMM-Meta-Modeler diagram are stereotyped through the CDMM-Meta-Modeler Eclipse PlugIn GUI which offers the list of available stereotypes taken from RootMetaModelProfile and from GeneratedProfile
- some nodes of the meta-model are stereotyped by the only stereotype available for meta-model node (entity) classes, that is by «RootMetamodelCore» stereotype defined in RootMetaModelProfile

The last step is very important and crucial for CDMM-F. It introduces one root for the whole meta-model graph structure. This is important from the perspective of model (an instance of meta-model) exploration from the CDMM-F client code through the API of CDMM-F as the entry point to the API is just via the root class. This root is implemented in CDMM-F as the RootMetamodelCore class depicted in the Figure 3. This root of the meta-model is associated by CDMM-F to the classes stereotyped by « RootMetamodelCore ». These classes are presented in the Figure 5. They are associated by CDMM-F to the root class in one way through the only class defined in RootMetamodelCoreEdge package, in our case this is the AggregationCPoliOMulti class, which can be exchanged by the user to a user-defined class. The assumption of the uniqueness of this class does not introduce any limits to the approach. Associating the meta-model entity classes to the root element is a meta-model design decision that should be made by the user of CDMM-Meta-Modeler. The existence of the root element eliminates the problem of creation of multi root graph and non compact graph. Otherwise not only the implementation of the client will be complicated but also application of graph pattern recognition methods will be limited, especially if graph syntactical methods are applied. The whole process of meta-modeling in CDMM-Meta-Modeler described above is presented in the Figure 7.

The dependency relationships are unidirectional. They were introduced to the example as they are sufficient for the presented meta-model (all relationships are unidirectional in it). This simplification was introduced intentionally into the cas-estudy just to limit the complexity. In fact the CDMM-Meta-Modeler makes it possible to associate stereotypes with all kinds of associative relations. In order to introduce a two-directional relationship the GeneratedProfile must be extended. Also more complex UML relationships may be introduced to the meta-model graph. For example the relationship that may join more than one (including also the transitive case of the binary relationship) or two node classes (N-ary association) may be introduced.

Cooperation between CDMM-Meta-Modeler and CDMM-F can be achieved in the following two ways: off-line cooperation via files sharing or on-line cooperation through CDMM-F API.

The first way is possible but not convenient enough as the CDMM-Meta-Modeler must find the right directories that contain source code files for CDMM-F specific classes as well as user-defined entity and relation classes. The right locations of the classes are pointed by CDMM-Meta-Modeler configuration file. In such the case the CDMM-Meta-Modeler is able to generate the following files on the output:

- GRP file in the XML format with the minimal graph structure that represents meta-model; this file can be interpreted by CDMM-F and transformed at the start to the Spring context-file which is then loaded by CDMM-F at run-time to initiate the framework correctly
- CTX file in the XML format which is the application context file for CDMM-F; the CDMM-F loads this file at the start

Another disadvantages of this off-line cooperation is that the meta-modeling process is not dynamic enough and changes introduced into meta-model through CDMMMeta-Modeler are not reflected automatically in the CDMM-F.

That is why the on-line cooperation is promoted. In this case not only CDMMMeta-Modeler but also CDMM-F have the form of Eclipse PlugIns and can exchange data through their APIs. In this case the application context file is passed from CDMM-Meta-Modeler to CDMM-F in the form of the string. Moreover, this way of cooperation is better from the perspective of automatic testing of the implementation of entity and relation classes of the meta-model (unit testing of the meta-model) and the correctness of their association to the CDMM-F (integration testing of the meta-model).

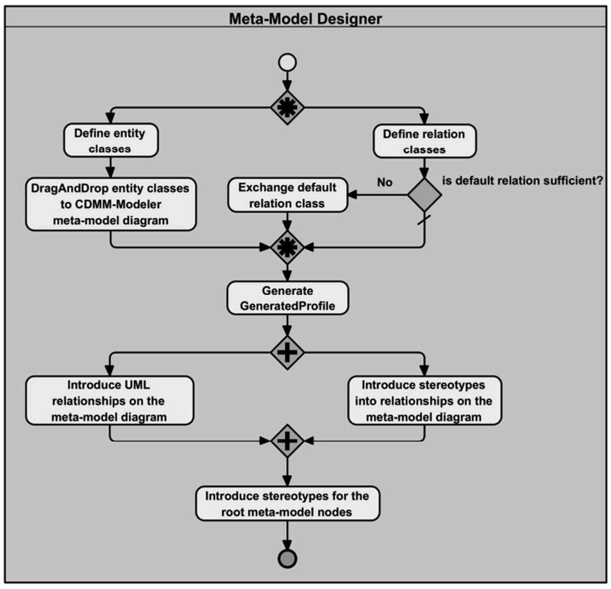
Meta-Model Designer
Define entity classes
Define relation classes
DragAndDrop entity classes to CDMM-Modeler meta-model diagram
Exchange default relation class
No
is default relation sufficient?
Generate GeneratedProfile
Introduce UML relationships on the meta-model diagram
Introduce stereotypes into relationships on the meta-model diagram
Introduce stereotypes for the root meta-model nodes

Figure 7. Business process of meta-modeling in CDMM-Meta-Modeler focused on the meta-model designer tasks supported continuously by the CDMM-Meta-Modeler tool

4. Conclusion

This paper focuses on the case study that illustrates how to construct a modeling language (a meta-model) in CDMM-Meta-Modeler software tool. It also shows how lightweight and convenient such a process is. This case study can be also a good reference for studying the nature of CDMM-F framework. The framework is an implementation of CDMM-P paradigm. So, the case study shows how this paradigm can be applied for meta-modeling. The construction of modeling languages is one of application fields of the CDMM-P. Thus, the clear explanation of meta-modeling process and its specifics can help to understand better the nature of open ontology based meta-modeling and then to apply the paradigm to construction of applications data layer, which is even more common problem, but located out of the current scope of the research. The case-study plays the role of the proof-of-the-concept for the idea of CDMM materialized in the form of CDMM-P and CDMM-F. All features of the CDMM-Meta-Modeler mentioned in the paper as planned for future work are intended to be implemented. Some of them require theoretical research before implementation.

REFERENCES

[1] Aßmann U., Zschaler S., Wagner G. (2006) *Ontologies, meta-models, and the model-driven paradigm.* In C Calero, F Ruiz, and M Piattini, editors, Ontologies for Software Engineering and Software Technology, 249–273, Springer.

[2] Booch G., Rumbaugh J., Jacobson I (2005) *The Unified Modeling Language User Guide*. Addison-Wesley.

[3] Calero C., Ruiz F., Piattini M. (2006) Ontologies for Software Engineering and Software Technology. Springer.

[4] Djurić D., Devedžić V. (2010) *Magic Potion: Incorporating New Development Paradigms through Meta-Programming*. IEEE Softw., 27 (5): 38–44.

[5] Djurić D., Jovanović J., Devedžić V., Šendelj R. (2010) *Modeling Ontologies as Executable Domain Specific Languages*. presented at the 3rd Indian Software Eng. Conf.

[6] Falbo R., Guizzardi G., Duarte K. (2002) *An ontological approach to domain engineering*. In Procs. 14th Int. Conf. on Software Eng. and Knowledge Eng. (SEKE).

[7] Fitrzyk G. (2014) *D-MMF Modeling Tool Based on Eclipse RCP*. MSc. thesis, Cracow University of Technology.

[8] Gašević D., Djurić D., Devedžić V. (2009) *Model Driven Engineering and Ontology Development*. Springer-Verlag.

[9] Gašević D., Kaviani K., Milanović M. (2009) *Ontologies, software engineering*. In Handbook on Ontologies. Springer-Verlag.

[10] Gallardo J., Molina A., Bravo C., Redondo M., Collazos C. (2011) *An ontological conceptualization approach for awareness in domain-independent collaborative modeling systems*: Application to a model-driven development method. Expert Systems with Applications, 38: 1099–1118.

[11] Goczyła K. (2011) *Ontologies in Information Systems* (in Polish). Akademicka Oficyna Wydawnicza EXIT.

[12] Object Management Group (2015) *Meta object facility (mof) core specification version 2.0*. URL: http://www.omg.org/spec/MOF/2.0

[13] Object Management Group (2015) *Unified modeling language (uml) superstructure version 2.2*. URL: http://www.omg.org/spec/UML/2.2

[14] Guizzardi G. (2005) *Ontological foundations for structural conceptual models*. Telematica Instituut Fundamental Research Series, 15.

[15] Guizzardi G. (2007) *On ontology, ontologies, conceptualizations, modeling languages, and (meta)models*. In Frontiers in Artificial Intelligence and Applications Volume 155, pages 18–39, Amsterdam. Conference on Databases and Information Systems IV, IOS Press. Selected Papers from the Seventh International Baltic Conference DB and IS 2006.

[16] Holanda O., Isotani S., Bittencourt I., Elias E, Tenório T. (2013) *Joint: Java ontology integrated toolkit*. Expert Systems with Applications, 40: 6469–6477.

[17] Javed F., Mernik M., Gray J., Bryant B. (2008) *Mars: A meta-model recovery system using grammar inference*. Information and Software Technology, 50: 948–968.

[18] Kern H., Kühne S. (2007) *Model interchange between aris and eclipse emf.* In 7th OOPSLA Workshop on Domain-Specific Modeling, Montreal.

[19] Krahn H., Rumpe B., Völkel S. (2007) *Efficient editor generation for compositional dsls in eclipse*. In Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling DSM' 07, Jyväskylä University, Jyväskylä, 2007. Technical Report TR-38.

[20] Kalnins A., Vilitis O., Celms E., Kalnina E., Sostaks A., Barzdins J. (2007) *Building tools by model transformations in eclipse*. In Proceedings of DSM 2007 workshop of OOPSLA 2007, pages 194–207, Montreal, University Printing House.

[21] Kleppe A.G., Warmer J., Bast W. (2003) MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston.

[22] Malhotra R. (2008) Meta-modeling framework: A new approach to manage metamodel base and modeling knowledge. Knowledge-Based Systems, 21:6–37.

[23] Peng X., Zhao W., Xue Y., Wu Y. (2006) *Ontology-based feature modeling and application-oriented tailoring*. In Reuse of Off-the-Shelf Components, pages 87–100. Springer-Verlag, New York.

[24] Reinhartz-Berger I. (2010) *Towards automatization of domain modeling*. Data and Knowledge Engineering, 69: 491–515.

[25] Sprinkle J., Mernik M., Tolvanen J.-P., Spinellis D. (2009) What kinds of nails need a domain-specific hammer? IEEE Software, 26: 15–18. Guest Editors' Introduction: Domain Specific Modelling.

[26] Silingas D., Vitiutinas R., Armonas A., Nemuraite L. (2009) *Domain-specific modeling environment based on uml profiles*. In Information Technologies 2009: Proceedings of the 15th Conference on Information and Software Technologies, IT 2009, pages 167–177, Kaunas. Kaunas University of Technology.

[27] Zabawa P. (2015) *Context-Driven Meta-Modeling Framework (CDMM-F) - Context Role*. Technical Transactions of Cracow University of Technology, 112 (1-NP): 105-114.

[28] Zabawa P., Fitrzyk G. (2015) Eclipse Modeling Plugin for Context-Driven Meta-Modeling (CDMM-Meta-Modeler). Technical Transactions of Cracow University of Technology, 112 (1-NP): 115-125.

[29] Zabawa P., Stanuszek M. (2014) Characteristics of the Context-Driven Meta-Modeling Paradigm (CDMM-P). Technical Transactions of Cracow University of Technology, 111 (3-NP): 123–134.