TECHNICAL TRANSACTIONS 4/2017
CZASOPISMO TECHNICZNE 4/2017
COMPUTER SCIENCES
DOI: 10.4467/2353737XCT.17.055.6366

Piotr Zabawa piotr.zabawa@pk.edu.pl
Institute of Computer Science, Faculty of Physics, Mathematics and Computer Science of Cracow University of Technology

Simulation of the CDMM-P paradigm-driven meta-modeling process

Symulacja procesu metamodelowania sterowanego paradygmatem CDMM-P

Abstract

This article presents a simulation of the process of meta-model creation. The meta-model (modeling language) is created according to the Context-Driven Meta-Modeling Paradigm (CDMM-P) with the help of its implementation – Context-Driven Meta-Modeling Framework (CDMM-F). The simulation process may be applied to meta-model creation in an evolutionary approach to meta-modeling or may be used to check correctness of the meta-model, or to test the CDMM-F framework. This paper is focused on the verification and testing mentioned above.

Keywords: simulation, meta-modeling, open ontology, software engineering, software development process

Streszczenie

Artykuł prezentuje symulację procesu tworzenia meta-modelu. Metamodel ten (język modelowania) tworzony jest zgodnie z paradygmatem Context-Driven Meta-Modeling Paradigm (CDMM-P) z wykorzystaniem szkieletu (framework) stanowiącego jego implementację – Context-Driven Meta-Modeling Framework (CDMM-F). Proces symulacji może być zastosowany do tworzenia metamodeli zgodnie z podejściem ewolucyjnym do metamodelowania albo może być stosowany do weryfikowania poprawności metamodeli lub do testowania CDMM-F. Niniejszy artykuł koncentruje się na wspomnianym weryfikowaniu i testowaniu.

Słowa kluczowe: symulacja, metamodelowanie, ontologie otwarte, inżynieria oprogramowania, proces wytwórczy oprogramowania

1. Introduction

This paper presents a study focused on the software development process as the application domain of modeling and simulation. The modeling aspect is addressed here in order to create modeling languages, while the simulation is focused on research of the nature of this process.

The business goals of software development enterprises are constantly evolving. One of the goals is to increase reusability in order to shorten the time-to-market, minimizing business and product risks while also minimizing project and product costs. At the beginning, the reusability was addressed to subsystems and components. As a result, they were moved from one project to another in the binary form. It was a very good result as long as the customization of the system can be done by configuring it. This approach fails if the character of the configuration evolves from the linear via the tree-like to the graph-like ones, which is a frequent phenomenon. At the same time, enterprises were trying to create a universal, highly reusable source code – classes, packages, libraries. The pursuit of this goal could give good results as long as the responsibilities of software systems were taxonomised unambiguously. Unfortunately, they were not. The main reason was that owners, managers, architects of IT enterprises were starting from the premise that everything changes all the time. Nevertheless, each software system can be decomposed into the following elements: immutable elements, elements with immutable structure and mutable functionality, elements with immutable functionality and mutable structure, mutable elements.

In the late 90's, we witnessed the beginning of the standardization process of many model-driven technologies supporting software system development. It led to the standardization of modeling languages, leading to the creation of the Unified Modeling Language (UML), and at the beginning of XXI century – to the introduction of Model-Driven Architecture (MDA) standards. The application of these standards moved the subject of reusability slightly into the so-called reusable assets – models, knowledge base elements usually consisting of pieces of parameterized source codes or parameterized meta-codes. However, the same problems with correct decomposition of software systems as the ones mentioned above limited the application of new standards. Moreover, all of them, including Rational Unified Process (RUP), being the result of software development process standardization, as one of the possible instances of Unified Processes (UP), are very large. Their constantly growing size and complexity result from the assumption of incorporating each technology into the standards. That is, these standards are built as the supersets of everything available on the constantly growing IT market. In addition, this is a real barrier for people when they want to apply these standards in production process. Moreover, the process of acquisition of some technologies is unacceptably slow or it even fails.

The approach presented in this article is different – it offers a constructive approach to limit models and meta-models (modeling languages) to the notions that are necessary to support reusability and automation of software development process – the automation addressed at the creation of software development project artefacts from models. However, it is also based on some concepts mentioned above. One of the consequences resulting from the application of UML is the transformation of the evolutionary approach from the software development process (RUP) to the evolutionary approach to modeling

activities. In the approach presented in this paper, this evolutionary approach also affects meta-modeling. This is a natural consequence of limiting the modeling language at the beginning of the particular product creation and enriching this modeling language during this process. The process may encompass one product, many products, product-line, many product-lines. There are already some publications dedicated to the problem of meta-model evolution [14, 19, 15, 17, 21].

So, what is now the business goal of software development enterprises? It seems that the goal is to create a mechanism that supports reusability of meta-models being subject of evolution in software development processes. That is why the CDMM-P was invented and CDMM-F was implemented.

The name of CDMM underlines the significance of Context in this approach. This context plays the role of the configuration for the CDMM-F framework. It is a graph-like structure, so the disadvantages of the concept of configuration mentioned above are limited as a result. It is worth noticing here that there are two kinds of contexts: the static context, which helps to create static structures with well-defined responsibility (meta-models) [26, 23, 24, 25]; and the dynamic context, which helps to enrich functionalities of the static structure [3, 2, 4]. This article is focused on the static contexts only.

Thus, the natural application field for this approach is the creation of software system elements with immutable functionality and mutable structure, according to the classification presented above. However, this article is focused on the simulation of the process of creation meta-models, so it is not dedicated to any application domain.

2. CDMM-P Basics

The CDMM-P Paradigm was introduced in [26] and the special role of Context was illustrated in [23]. This paper presents the most important concepts of this approach.

The main idea of the CDMM-P is to assume that associative relationships (UML composition, aggregation, association) between classes (pure entity classes) are represented in the form of classes (entity relationship classes) and all these classes are not interrelated in any form in their source codes. The objects of entity relationship classes are injected into pure entity classes at run time on the basis of the graph-like configuration file. Thus, the mentioned file plays the role of the static context for the CDMM-F framework that supports the CDMM-P paradigm. This paradigm may be applied for the implementation data layer of target domain-specific software systems as well as just for the creation of graph modeling languages, like in the case of this article. In consequence, a meta-model architect can design a modeling language from scratch or can customize an existing modeling language (e.g. UML). In the last case, he/she can remove unnecessary elements from the UML and/or add extra elements to the UML meta-model. The meta-model created this way may be also the subject of evolutionary changes together with models created in the former versions of the modeling language.

The concept of the responsibility division introduced by the CDMM-P is illustrated in Figure 1 for the sample meta-model.

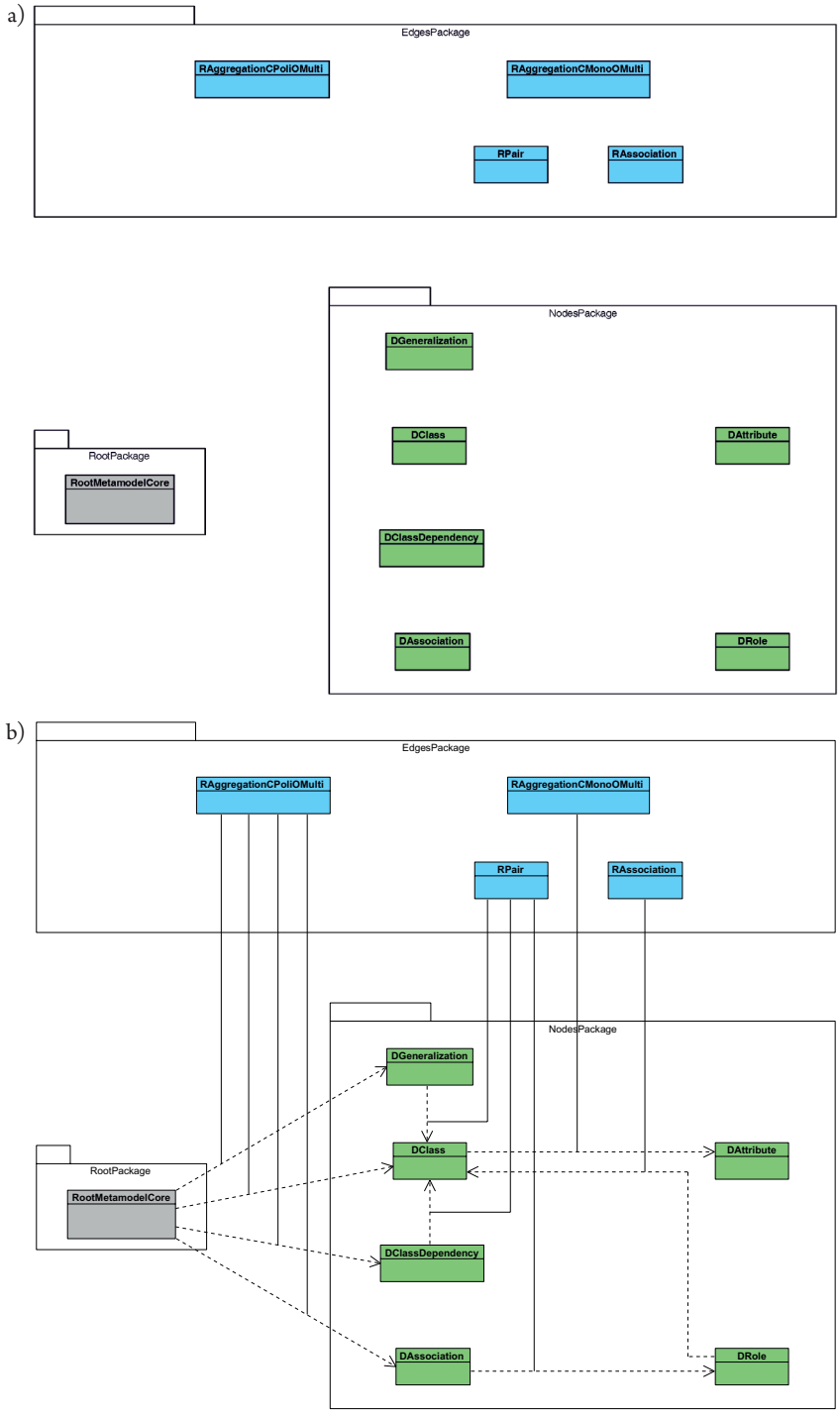

Fig. 1. Decomposition of responsibilities introduced by CDMM-P paradigm: a) unrelated meta-model classes, b) inter-related meta-model classes after relationship injections

The following color convention is applied in diagrams presented in Figure 1:

- Grey – framework elements that are immutable, as they belong to the framework,
- Green – user-defined meta-model elements representing meta-model nodes,
- Blue – user-defined meta-model elements representing meta-model edges.

It is worth noting that the concept of CDMM-P is based on open ontologies. This is new in comparison to the traditional closed ontology based modeling standards, like MOF, UML, MDA. As far as the author is aware, there are no similar solutions discussed in the literature, but some common ideas can be found in the context of ontologies. The open ontology concept not connected to software engineering is presented in [5, 10, 11]. There are some papers focused on the application of ontologies to domain model construction [8, 6, 22, 12] and its customization [20] dedicated to vertical OMG standards. Application of ontology for static class model inference [9] and for meta-language construction [18] are also known. The ontology can be used for application design, like in the case of JOINT system [16]. Ontologies can be designed from MOF-based MDA-compliant meta-model [7] or vice-versa [1].

3. Fundamentals of the CDMM-F Architecture

The CDMM-F framework was presented in [24]. Here, we present only the basics of its architecture that are relevant in the context of the simulation of the meta-modeling process.

The architecture of the CDMM-F framework was created in a form that fulfills the following criteria:

- should support CDMM-P paradigm,
- should be as convenient for the client of this framework as possible.

In order to satisfy the first criterion, the following technologies were chosen: Java, Spring and AspectJ. The graph configuration file is implemented in the form of a Spring application context file. The injection mechanism is based on injecting default interface implementations supported by both Spring and full version of AspectJ.

The second criterion is satisfied by an exchangeable API implementation and by introduction of proxy layer. The last element was introduced to eliminate conflicts between the names of methods injected from more than one interfaces.

4. Sample Meta-Model

In this section, a sample meta-model is shown in order to illustrate the concept of CDMM approach for defining modeling languages. It is then explained on the basis of the CDMM concept that relationship classes tend to be more complex than node classes of the meta-model. The classification of relationship classes is also introduced in this section. This classification is general, but it refers to the sample meta-model just to point the relationship classes under consideration.

The simple, but powerful, from the application perspective, sample modeling language is shown in Figure 1b. It consists of the three following packages, like in the case of each CDMM compliant modeling language:

- RootPackage – package containing CDMM-F framework elements (grey classes),
- NodesPackage – package containing user-defined meta-model node classes (green classes),
- EdgesPackage – package containing user-defined meta-model edge classes (blue classes)

All user-defined classes are implemented as completely independent of one another.

In order to define a new required modeling language, the CDMM-F user (meta-model architect) should:

- customize existing meta-modeling standard,
- customize his/her former meta-models,
- define new meta-model from scratch,
- reuse his/her former meta-models.

The last item is not interesting from the perspective of this article. In all remaining cases, the meta-model architect probably should introduce some nodes of a meta-model graph (meta-model classes) and/or introduce some new kinds of meta-model graph edges (meta-model relationships).

It seems from the analysis of available meta-models and from the IT market experience that new nodes are introduced more frequently than new edges. That is why the answer to the question if the edges should be injected into nodes or vice-versa is that nodes should be injected into the relationship. In consequence, the introduction of new node classes is simpler than the introduction of new edge classes. It increases the risks of the meta-modeling process connected to the implementation of relationship classes. That is why the simulation process is focused on this aspect of meta-modeling.

There are the following kinds of relationship classes when nodes are injected into edges:

- originated in separate parent class after injection (like RAggregationCMonoOMulti, RAssociation, RPair),
- originated in shared parent class after injection (like RAggregationCPoliOMulti),
- terminated in single object (like RAssociation, RPair),
- terminated in many objects (like RAggregationCMonoOMulti, RAggregationCPoliOMulti).

The main implementation risk is connected to correct handling of the dual nature (class sharing, number of objects) of each relationship class. That is why these classes should drive both testing and simulation processes.

5. Simulation of Meta-Modeling Process

The problem of testing model-driven tools is very complex and typically is performed in a way common for other kinds of software. However, the approach for testing CDMM-F is special and could also be applied for other software systems based on models.

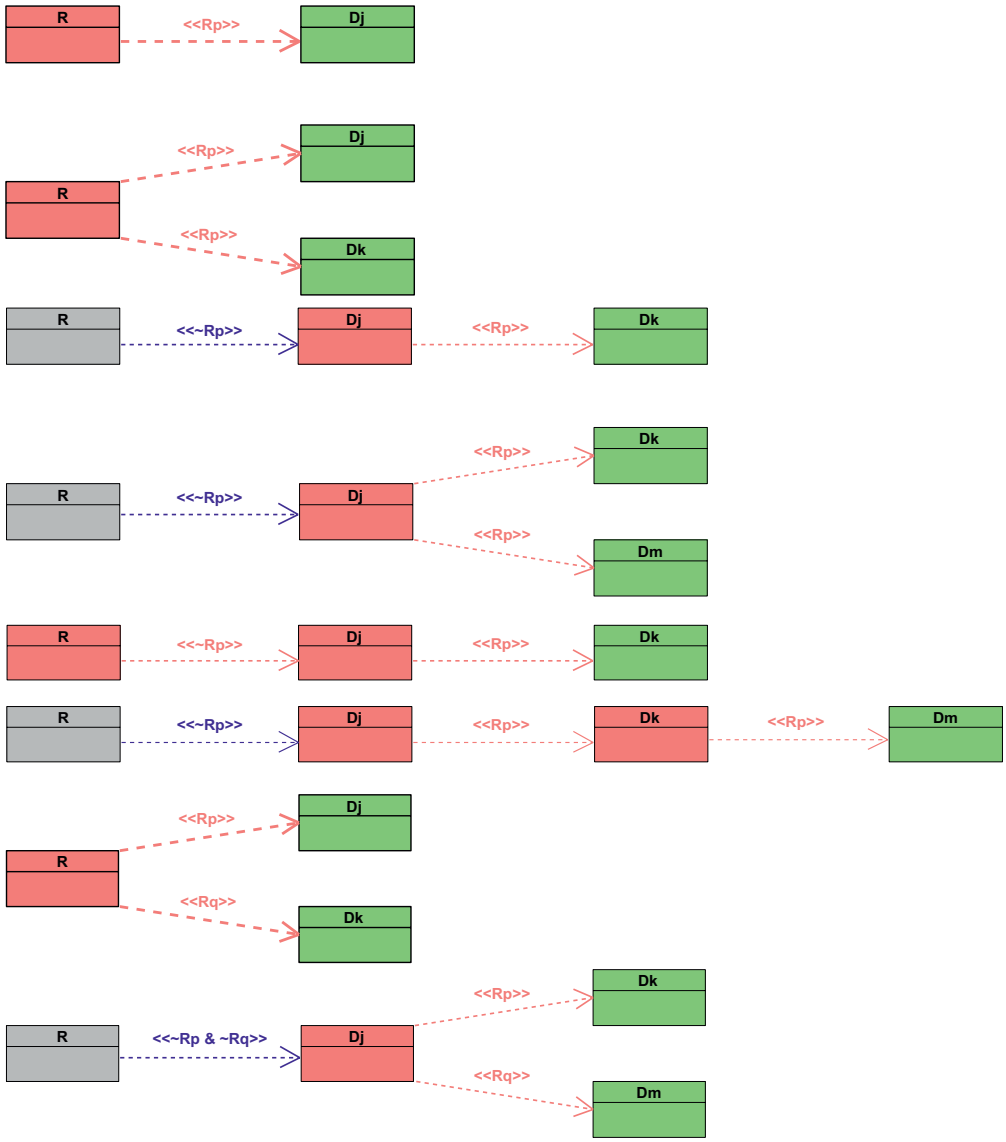

Fig. 2. Set of meta-model graphs identified as test kernels

Two testing stages were applied for the CDMM-F quality verification purposes:

- manual creation of specially designed meta-models,
- automated simulation of the usage of the CDMM-F framework in production.

The first stage was described in detail in [23]. Theoretical analysis of risks was performed first to create meta-models, which are useful for testing. Then, meta-models were created to address these risks. The meta-models taken into account are presented in Figure 2.

The following color convention is applied in diagram presented in Figure 2:

- Grey – framework elements that are immutable, as they belong to the framework,
- Green – user-defined meta-model elements representing meta-model nodes,
- Blue – user-defined meta-model elements representing meta-model edges,
- Red – meta-model elements that are subject of test.

From the perspective of the title of this paper, the second stage that is focused on simulation is more interesting. The simulation was designed according to the risk-driven rule, which is good for simulation purposes. The goal of simulation was verification of the quality of the CDMM-F correctness for many application context files. The risks are focused on manually created models, which were called test kernels. The simulation process idea was to surround each test kernel by generated pseudorandom graphs. They were called test contexts. Therefore, the simulation addressed the risk of a wrong cooperation of test kernels with the rest of almost any meta-model. This approach limited simulation effort costs and maximized the chance to identify potential problems connected to the implementation and usage of the CDMM-F framework.

The simulation process was performed in several stages:

- the set of meta-model graph nodes and the set of meta-model edges (two sets of classes) was prepared,
- all possible graphs built of these classes were generated automatically (the size of graph was limited),
- pattern matching algorithm was executed to recognize graphs that contained test kernels,
- manual creation of client code scanning meta-model via API for each recognized graph took place.

In order to further minimize the costs, the deterministic algorithm for generating graphs was not implemented. However, in place of it the non-determinism contained in Java hash tables implementation and included in Jung (Java Universal Network/Graph Framework) framework was used. It helped to surround risk kernels presented in Figure 2 by risk contexts – graphs generated by non-deterministic algorithm.

6. Simulation-Based Testing

The typical approach to the verification of the quality of model-driven systems, such as the CDMM-F one, is via testing. In the case of the subject, system tests were also implemented in iterative software development process. However, they were limited to unit tests. In fact, the CDMM-F framework should also be verified via integration tests. These tests are typically performed in a form that is not enough in this case – testers tend to manually define and implement only some non-trivial data structures that drive integration testing. This observation led to the significant improvement of the traditional approach, as the typical approach was exchanged by an automatic generation of a large set of data structures and thus integration testing became a simulation process.

Simulation technique and testing have similar goals – they are applied in order to maximize the probability of error identification by IT company customers. That is why software development company staff are interested in finding as many errors as possible – typically via testing. Nevertheless, both approaches do not guarantee that the software system will be finally free of errors as well as the verification process may not detect errors. Therefore, the IT companies go further – they try to maximize the probability that the product quality verification process (an investment) will find some errors. The best technique is to perform product and process risk analysis and to elaborate the testing strategy (also known from RUP as test plan) that maximizes the chance for finding errors.

The same is true both for the testing and the simulation technique. During implementation of CDMM-F, the simulation did not detect any errors, in contrast to unit-testing. This fact resulted from the asymmetry that was intentionally introduced to the development process. The main stress was put on careful design, iterative software development process and intensive iterative testing. The simulation was performed at the end of the process – thus not iteratively. This approach resulted from the very high risk level of new technology implementation. In contrast to typical production process, the success of the system creation was not guaranteed. In addition, the feasibility study approach cannot be performed, as the system itself is its feasibility study. That is why the investment into simulation was postponed in this special case. Thanks to the fact of simulation delay, the cost of the development was low and the risk of overinvestment was controlled. However, the duration of implementation was extended and the project risk was moved to the end of the simulation process. In the IT industry, there is usually no time for such careful design and testing due to time pressure. In addition, in this situation, the importance of the presented simulation technique grows significantly. The simulation allows to move the responsibility from design just to the simulation while keeping development cost low, allows to reduce the development process time and to address risk early if the simulation process is performed iteratively.

Manually created unit tests for CDMM-F were focused just on test kernels. The simulation process was executed for these test kernels surrounded by test contexts. As it was already mentioned above, the simulation process did not identified any defects. Some assumptions were made regarding the meta-model graphs generated during the simulation process. The graphs were simple directed and connected graphs and they did not contain transitive relationships or N-ary relationships. The number of nodes generated by the simulation was limited to the doubled number of the nodes in the largest test kernel, that is to 8 nodes. The test kernels were identified among the graphs generated by non-deterministic algorithm in loops limited to 20,000 iterations. It was observed during the experiments that exceeding the number of loops above this limit does not produce a noticeable number of new graphs, thus it is ineffective. The number of all different graphs generated this way was equal to 3100, while the theoretical number of such graphs is 882033440. The test kernels were found in 2800 graphs. Thus, the effectiveness of generating graphs, interesting from the testing perspective, was 90%. The number of graphs generated in this approach does not seem to be large enough in comparison to the theoretical one, but it is very attractive from the testing point of view, especially when compared to manual graph/test defining process.

7. Conclusions

The CDMM-F framework usage was the subject of the simulation process. The goal of this process was to check how the system might behave in a real production environment. A new approach used for the model-driven software system confirmed that this approach is useful for such a system. Moreover, the verification results obtained from the simulation process confirmed that this is a good approach to such verification, as the costs are naturally limited by this approach. At the same time, the risk-driven approach to simulation helps to maximize the probability of the success of this process, that is – to identify problems. The subject is interesting from the quantitative point of view and there are plans to investigate it in the future.

References

[1] Aßmann U., Zschaler S., Wagner G., *Ontologies, meta-models, and the modeldriven paradigm*, Calero C., Ruiz F. & Piattini M., eds, Ontologies for Software Engineering and Software Technology, Springer, 2006, 249–273.

[2] Bettini L., Bono V., *Type safe dynamic object delegation in class-based languages*, Proc. of PPPJ, ACM Press, 2008, 171–180.

[3] Bettini L., Capecchi S., Damiani F., *A mechanism for flexible dynamic trait replacement*, FTfJP '09, Genova 2009.

[4] Bono V., Damiani F., Giachino E., *On traits and types in a java-like setting*, TCS 2008 (Track B), Vol. 273, Springer, 2008, 367–382.

[5] Calero C., Ruiz F., Piattini M., *Ontologies for Software Engineering and Software Technology*, Springer, 2006.

[6] Djurić D., Devedžić V., *Magic Potion: Incorporating new development paradigms through meta-programming*, IEEE Softw. 27(5), 2010, 38–44.

[7] Djurić D., Gašević D., Devedžić V., *Ontology modeling and MDA*, Journal on Object Technology 4(1), 2005, 109–128.

[8] Djurić D., Jovanović J., Devedžić V., Šendelj R., *Modeling ontologies as executable domain specific languages*. presented at the 3rd Indian Software Eng. Conf., 2010

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

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

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

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

[13] Guizzardi G., *On ontology, ontologies, conceptualizations, modeling languages, and (meta) models*, Frontiers in Artificial Intelligence and Applications, Vol. 155, Conference on

Databases and Information Systems IV, IOS Press, Amsterdam, 18–39; Selected Papers from the Seventh International Baltic Conference DB and IS 2006, 2007.
[14] Herrmannsdörfer M., *Evolutionary Metamodeling*, PhD thesis, Technical University in Munich, 2011.
[15] Herrmannsdörfer M., Ratiu D., *Limitations of automating model migration in response to metamodel adaptation*, Proc. of the Joint ModSE-MCCM Workshop on Models and Evolution, 2009.
[16] Holanda O., Isotani S., Bittencourt I., Elias E., Tenório T., *Joint: Java ontology integrated toolkit*, Expert Systems with Applications 40, 2013, 6469–6477.
[17] Iovino L., Pierantonio A., Malavolta I., *On the Impact Significance of Metamodel Evolution in MDE*. Journal of Object Technology 11(3), 2012, 1–33.
[18] Laarman A., Kurtev I., *Ontological meta-modelling with explicit instantiation*, [in:] Brand M. van den, Gašević D., Gray J. (eds.), SLE 2009, number 5969 in LNCS, Springer-Verlag, Berlin, 2010, 174–183.
[19] Langer P., Wimmer M., Brosch P., Herrmannsdörfer M., Seidl M., Wieland K., Kappel G., *A posteriori operation detection in evolving software models*, The Journal of Systems and Software 86, 2013, 551–566.
[20] Peng X., Zhao W., Xue Y., Wu Y., *Ontology-based feature modeling and application-oriented tailoring*, Reuse of Off-the-Shelf Components, Springer-Verlag, New York 2006, 87–100.
[21] Ruscio D. di, Iovino L., Pierantonio A., *Coupled Evolution in Model-Driven Engineering*. IEEE Software 29(6), 2012, 78–84.
[22] Tairas R., Mernik M., Gray J., *Using ontologies in the domain analysis of domain-specific languages*, Models in Software Engineering, Springer-Verlag, New York 2009, 332–342.
[23] Zabawa P., *Contex-Driven Meta-Modeling Framework (CDMM-F) – Context Role*. Technical Transactions, 1-NP, 2015, p. 105–114, DOI: 10.4467/2353737XCT.15.119.4156.
[24] Zabawa P., *Contex-Driven Meta-Modeling Framework (CDMM-F) – Internal Structure*. submitted for publication, 2015.
[25] Zabawa P., Nowak K., *Context-Driven Meta-Modeling – Simple Horizontal Case-Study*, submitted for publication, 2015.
[26] Zabawa P., Stanuszek M, *Characteristics of the Context-Driven Meta-Modeling Paradigm (CDMM-P)*, Technical Transactions, Vol. 3-NP/2014, 123–134.