TECHNICAL TRANSACTIONS | CZASOPISMO TECHNICZNE
FUNDAMENTAL SCIENCES | NAUKI PODSTAWOWE

1-NP/2015

PIOTR ZABAWA*, GRZEGORZ FITRZYK**

ECLIPSE MODELING PLUGIN FOR CONTEXT-DRIVEN META-MODELING (CDMM-META-MODELER)

PLUGIN DO MODELOWANIA W ECLIPSE DLA METAMODELOWANIA STEROWANEGO KONTEKSTEM (CDMM-META-MODELER)

A b s t r a c t

The novel and crucial point of this paper is the illustration of the application of close ontology based meta-modeling to defining the open ontology based construction of modeling languages (meta-models). This approach is a generalization of Object Management Group (OMG) standards related to Model-Driven Architecture (MDA). The existing modeling tools supporting the traditional approach can be reused when open ontology based meta-modeling tools are implemented. This paper describes Context-Driven Meta-Modeling Meta-Modeler (CDMM-Meta-Modeler) – the Eclipse Plugin that constitutes such an open ontology based meta-modeling tool. The tool constitutes an implementation of the diagramming aspect for the Context-Driven Meta-Modeling Framework (CDMM-F), which is one of several possible implementations of the Context-Driven Meta-Modeling Paradigm (CDMM-P).

Keywords: *meta-model*, *class diagram*, *modeling paradigm*, *Eclipse Plugin*, *OSGi*, *ontology, visual modeling*

S t r e s z c z e n i e

Nowym i kluczowym elementem artykułu jest ilustracja zastosowania metamodelowania opartego o ontologie zamknięte do konstruowania języków modelowania (meta-modeli) opartego o ontologie otwarte. Podejście to jest uogólnieniem standardów Object Management Group (OMG) związanych z Model-Driven Architecture (MDA). Do implementowania narzędzi modelowania opartych o ontologie otwarte można wykorzystać istniejące narzędzia modelowania oparte na tradycyjnym podejściu. Artykuł opisuje Context-Driven Meta-Modeling Meta-Modeler (CDMM-Meta-Modeler) - PlugIn do Eclipse'a, który stanowi takie narzędzie do metamodelowania oparte na ontologiach otwartych. Narzędzie to stanowi implementację problemu konstruowania diagramów dla frameworku Contex-Driven Meta-Modeling Framework (CDMM-F), który jest jedną z możliwych implementacji paradygmatu Contex-Driven Meta-Modeling Paradigm (CDMM-P).

Słowa kluczowe: *meta-model*, *diagram klas diagram*, *paradygmat modelowania*, *Eclipse Plugin*, *OSGi*, *ontologia, modelowanie wizualne*

DOI: 10.4467/2353737XCT.15.120.4157

* Department of Physics, Mathematics and Computer Science, Cracow University of Technology; pzabawa@pk.edu.pl.

** Graduate of Department of Physics, Mathematics and Computer Science, Cracow University of Technology; gfitrz@gmail.com.

1. Introduction

Contemporary approaches to the creation of modeling languages for software engineering discipline are based on close ontologies – close systems of notions reflected by the static structure of modeling languages represented by meta-models. The characteristic feature of such systems is that each meta-model class is placed in a meta-model graph and this class depends on all other meta-model classes it is related to. As the result, close ontologies are difficult to change, in contrast to open ontologies.

This paper is related to a different approach to meta-modeling which is based on open ontologies named Context-Driven Meta-Modeling (CDMM). In contrast to the traditional approach briefly characterized above, in the CDMM approach, the whole meta-model graph consists of meta-model classes placed in meta-model graph nodes while the relationship classes characteristic for the CDMM approach are located at the graph edges. Moreover, the meta-model classes are unrelated to one another. This results in a very high level of flexibility in defining the meta-model structure as well as in defining the meta-model elements (entity classes and relation classes).

An important special feature of the CDMM concept is that it is not build on top of standards defined to support ontologies like the Resource Description Framework (RDF) by the World Wide Web Consortium (W3C) or the Web Ontology Language (OWL) by the Object Management Group (OMG). As a consequence, the CDMM approach differs significantly from [1, 11, 3, 4, 7, 8, 9, 10, 6]. In contrast to the ontology research domain, it is defined in the manner known from the meta-modeling of software intensive systems known, for example, from [2]. As a result, the CDMM approach can be compared to such extensively used OMG standards like the Meta-Object Facility (MOF) [13], the Unified Modeling Language (UML) [14] and Model-Driven Architecture (MDA) [12]. Thus, the CDMM constitutes the generalization of all the above-mentioned MDA-related standards.

This generalization was achieved through the introduction of a new modeling paradigm named the Context-Driven Meta-Modeling Paradigm (CDMM-P) presented in [15]. This paradigm makes it possible to replace traditional interrelated compact systems of notions both specified and implemented (references between classes inside classes to represent class relationships) via class diagrams by meta-models created during run-time from decoupled meta-model entity classes and meta-model relationship classes. The CDMM approach replaces fixed data structures with dynamically created data structures.

The CDMM-P has its implementation in the form of the Context-Driven Meta-Modeling Framework (CDMM-F) introduced in [16] and explored in [17] from the special perspective of the role the context takes in this framework implementation. The CDMM-F plays the role of proof of the concept for the CDMM-P idea. The CDMM-F framework is implemented in Java technologies – Java SE enriched by Spring and AspectJ. An illustrative case-study for the application of the CDMM-F framework is presented in [18], where the modeling language is defined from scratch just to illustrate the features of the CDMM-F. In this paper, the same meta-model is created from the CDMM-Meta-Modeler especially to illustrate the difference between both approaches.

This paper is related to the CDMM-F framework and the system presented here constitutes an extension for the CDMM-F. The first version of the CDMM-Meta-Modeler was presented in [5]. The software system introduced in the paper is dedicated to creating static UML models of modeling languages (meta-models) via class diagrams. A meta-model created in this way is then used to create CDMM-F instance just for this meta-model (the model). The model can be then accessed via the API of the CDMM-F. In this way, the concept of MDA-like modeling in the open ontologies is achieved and implemented. Moreover, the paper contains a constructive illustration of the way the commonly known and widely applied close ontology based approach to meta-modeling can be used to define an open ontology based approach to the construction of modeling languages. Thus, the paper constitutes the proof of concept for such a new approach to meta-modeling.

2. Context-Driven Meta-Modeling

Two elements of the Context-Driven Meta-Modeling set of tools that together constitute Context-Driven Meta-Modeling Technology (CDMM-T) are presented below in this section. Firstly, the most fundamental CDMM-F framework characteristic features are shown. Secondly, the role and the concept of visual modeling support for the CDMM-F is presented.

2.1. Framework

The CDMM-F framework is the core element of the whole CDMM-T. It constitutes an implementation of the CDMM-P paradigm. The name of all the above-mentioned CDMM elements originated just from the way that the CDMM-F framework was implemented. The data structure representing the meta-model has the form of the application context XML file, as the CDMM-F is implemented in Java, Spring and AspectJ. As a consequence, the meta-model takes the role of the context for the CDMM-F application. The running framework is created automatically from that file and the structure of the running framework reflects the meta-model structure. The XML file mentioned above has a relatively complex format and contents. It could be created from any text editor, but this approach is error-prone – this is why the visual modeling tool is desirable to design the meta-model which the application context file can be generated from by this tool.

2.2. Visual Meta-Modeling

The Visual Meta-Modeling feature was added to the CDMM-F to simplify the meta-model designers' job. This simplification was achieved by making the graphical tool available for meta-model graph creation and by minimizing the number and complexity of the solution's user tasks. The graph is built from two kinds of elements:
– classes that are associated with graph nodes – meta-model entity classes,
– classes that are associated with graph edges – meta-model entity relationship classes.

Both groups of classes must be predefined with the same tool. For the association of classes to the meta-model graph, the UML Profiles, and specifically UML stereotypes are applied. The association is via the stereotype name which is generated from the name of the class that is intended to be associated to the node or to the edge of the graph. The process of stereotype generating is one of the functions of the visual meta-modeling tool.

The key concept of the tool is the application of the functionalities of close ontology--based modeling tools for constructing the simple tool that makes it possible to define open ontology-based modeling languages.

3. CDMM-Meta-Modeler Features

According to the concept of visual meta-modeling presented in section 2, and according to the characteristics of CDMM-F presented in [16, 17], the most important features of the CDMM-Meta-Modeler can be specified. They are shown below in this section.

The most desired features of the CDMM-Meta-Modeler at the current stage of research and implementation work are as follows:

- **reuse of existing UML modeling (not diagramming) components**,
- **reuse of the mechanism of UML Profile modeling (not diagramming) offered by available components**,
- **reuse of the mechanism of extending UML modeling components via UML Profiles (the two above-mentioned components must be compatible)**,
- **reuse of existing diagramming framework**,
- **potential reuse of existing modeling components for additional tasks** like generating source code from the model, serialization of the meta-model via the standard UML XMI format for interoperability purposes,
- **potential integration of the CDMM-Meta-Modeler with existing commercial modeling products (all features listed above support this feature)**,
- **the ability to graphically model (diagramming) limited UML models containing:**
  - **stereotypes in UML Profiles**,
  - **classes with stereotypes for classes introduced in UML Profiles**,
  - **associative relationships (composition, aggregation, association) with stereotypes for relationships introduced in UML Profiles**,
  - **dependency relationships with stereotypes for dependency introduced in UML Profiles**,
  - **packages with their hierarchies**,
- the ability to define meta-model entity classes in separate package,
- the ability to define meta-model relation classes in separate package,
- **the ability to customize predefined CDMM-F framework meta-model classes contained in the separate package**,
- the mechanism of the reuse of the predefined CDMM-F framework meta-model relation class in the user-defined meta-models and vice versa,
- the ability to round-trip engineer the UML Profile for stereotyping entity and relationship meta-model classes,
- **the mechanism of drag-and-drop meta-model entity classes to the meta-model diagram**,
- **the mechanism of introducing the UML Profile stereotypes for entity meta-model classes to these classes on the meta-model diagram**,

- **the mechanism of joining meta-model entity classes on the meta-model diagram via associative and/or dependency relationships (creation of the meta-model graph composed of the user-defined entity and relationship meta-model classes)**,
- **the mechanism of introducing UML Profile stereotypes for meta-model relations to the relationships put on the meta-model diagram**,
- the ability to load/save the meta-model through the XMI file,
- **the ability to export the meta-model to the XML file for the CDMM-F**,
- **the ability to generate the application context XML for the CDMM-F** and to send this application context to the CDMM-F Plugin in the form of the string.

The most important functionalities from the list shown above were already implemented in the presented CDMM-Meta-Modeler (in bold) while the remaining functionalities are under development. A large number of both functional and non-functional requirements was formally specified according to the RUP's FURPS+ classification and known good practices; however, these are not contained in this paper due to size limitations. However, both functional and non-functional tests discussed shortly in section 5 were implemented according to the requirements correctly specified.

4. CDMM-Meta-Modeler Tool Implementation

The visual modeling concept was materialized in the form of the CDMM-Meta-Modeler tool. The existing modeling tools can be used for the implementation of the CDMM-Meta--Modeler as the concept of the visual modeling assumes that close ontology based modeling can be applied to construct the open ontology based meta-modeling tool. The right source of such close ontology based modeling tools is the Eclipse framework with its Plugins – this is why the CDMM-Meta-Modeler tool was also implemented in the form of the Eclipse Plugin.

The implementation of the Eclipse Plugin makes it possible to achieve functionalities sufficient for the effective creation of open ontology based modeling languages with low expense and independently from commercial modeling software providers. At the same time, the approach typical for meta-modeling can be used in place of the approach characteristic for design and specification of ontologies which is based on RDF or OWL standards.

4.1. Technologies

As previously mentioned, the Eclipse framework was chosen for CDMM-Meta-Modeler tool implementation. The tool itself consists of several Eclipse Plugins. The motivation for such a technological choice is as follows:

- reduction of specialized integrated development environments (IDEs) required by the CDMM-T user to the one based on Eclipse,
- extensibility of the CDMM-T implementation into newly required functionalities via the installation of new Plugins,
- operating system independence of the CDMM-Meta-Modeler tool resulting from Eclipse IDE operation system independence,

- reuse of the UML2 SDK Eclipse Plugin for the UML model of the modeling language storage and for its conformity to the OMG UML standard,
- natural integration with CDMM-F which was already implemented in Java; in fact, it was implemented in Java because all contemporary modeling tools are implemented just in Java, including the UML2 SDK.

The Eclipse framework itself is implemented in Java and is based on Equinox – the Eclipse implementation of Open Services Gateway initiative (OSGi).

Graphical libraries the Eclipse is based on are SWT and JFace. They offer application rendering which is native for Eclipse platform. This is why, in order to unify the presentation style of all platforms, the JavaFX library, the successor of Swing, was applied.

All these technologies are open source which helped to limit cost of the CDMM-Meta--Modeler tool significantly.

The short characteristics of the technologies mentioned above and remaining ones is presented below. This characteristics constitutes motivation for choosing just these technologies.

- Eclipse RCP – This Eclipse Rich Client Platform facilitates the implementation of Rich Client applications based on the dynamic Plugin model via the introduction of a set of dedicated Plugins to the Eclipse Platform.
- Equinox – This is an OSGi implementation dedicated to Eclipse. It helps to manage Plugins and offers extensions that can extend the functionality of other Plugins.
- UML2 SDK – This is the Eclipse PlugIn that includes the implementation of OMG UML version 2.x. The main goals of this component are the provisions of:
  - useful implementation of UML for the support of modeling tool development,
  - common XMI schema for the facilitation of the exchange of semantic models,
  - test cases for the verification of the specification correctness,
  - validation rules as the medium for defining and executing conformance levels.
- SWT – graphical library for Java as an alternative for AWT/Swing libraries. This library has been developed by the Eclipse Foundation. The main idea of this library is to provide native system aspects such as key shortcuts, reactions to mouse behavior etc. as well as the creation of a native application view – this means that the view is unique for each platform.
- JFace – This is one of the Eclipse Foundation's projects which provides graphical components based on SWT. The main goal of the library is to provide such components that are frequently used and, at the same time, are difficult to implement in SWT. As the JFace components constitute an extension of SWT components, they can be combined with SWT.
- JavaFX – This is graphical library which is a successor to Swing. It offers a wide spectrum of components together with the mechanism of their simple customization via CSS files. Moreover, it allows the nesting of its components into other graphical libraries like Swing or SWT. The possibility of nesting JavaFX in SWT allows the use of JavaFX in Eclipse. As a consequence, this allows creating advanced components through the combination of JavaFX, JFace and SWT.
- SWTBot – This is the Java library that allows the implementation of a user interface and/or functional tests for SWT and Eclipse. SWTBot integrates with Tycho very well.

As a result, these tests can be executed by Maven. SWTBot works on all platforms available for SWT. Both Tycho and Maven are used in the development process for the Plugin.
- JUnit – This is the most popular Java-oriented tool for unit testing support. These tests allow for the continuous testing of the product.
- TestFX – In contrast to SWTBot, this library is designed to support the manual development of automated user interface tests and/or functional tests for Swing and JavaFX.

The Eclipse RPC distribution was chosen as it is dedicated to support Eclipse Plugins development and constitutes the execution environment for the CDMM-Meta-Modeler. It allows the integration of all the above-mentioned technologies. The choice of technology was simple as there is no competing open source environment on the market.

4.2. Design

The most challenging part of the design discipline was to take into account the design context resulting from the technologies applied to the CDMM-Meta-Modeler implementation.

The following technologies impacted on the design discipline of the CDMM-Meta-Modeler project: Eclipse, JavaFX, SWT and UML2 Eclipse Plugin. Some parts of the CDMM-Meta-Modeler design elements and the nature of the impact of reused technologies on JavaFX is visible in Figure 1.

Another part of the CDMM-Meta-Modeler design is shown in the diagram presented in Figure 2. It shows the dependency of the design elements presented there on JavaFX and SWT.

The whole CDMM-Meta-Modeler system was carefully designed during the implementation. It is worth noticing here that both Figure 1 and Figure 2 were prepared after implementation of the presented CDMM-Meta-Modeler in this tool for the presentation purposes only.

5. CDMM-Meta-Modeler Tool testing

The CDMM-Meta-Modeler was carefully tested. The test strategy was quite simple as most risks of the product are concentrated in the diagramming functionality and in the integration of the CDMM-Meta-Modeler with modeling tools and with the third-party software presented in section 4.1.

The correctness of the implementation of the CDMM-Meta-Modeler classes was verified by extensive unit testing. The JUnit tool was used to support the process of implementation and the execution of these tests. The testing process was included in Maven's lifecycle.

Another group of tests addressed the verification of the CDMM-Meta-Modeler functionalities constitute functional tests. Some of the tests from this group were implemented with SWTBot and TestFX tools – they were executed automatically in the CDMM-Meta-Modeler development process. However, some of the tests from this group remained manual. The criterion that helped to decide which test should have which form was based on the importance of the tests. The most important tests that covered more functionality had the form of automated tests and some less important and smaller functionalities (sub functionalities) were executed manually. Moreover, due to the available testing tools, some components implemented in the CDMM-Meta-Modeler that were implemented in technologies not covered by SWT and/or JavaFX technologies had to be executed manually. A sample automated SWTBot and JavaFX dependent test case in the form of the appropriate script is presented in Fig. 3. This test verifies the correctness of the implementation of the functional requirement of creating a new meta-model project in the CDMM-Meta-Modeler.

GUI tests were performed with an approach similar to that applied to functional testing.

The remaining non-functional tests were also executed to check if all non-functional requirements were fulfilled by the CDMM-Meta-Modeler.

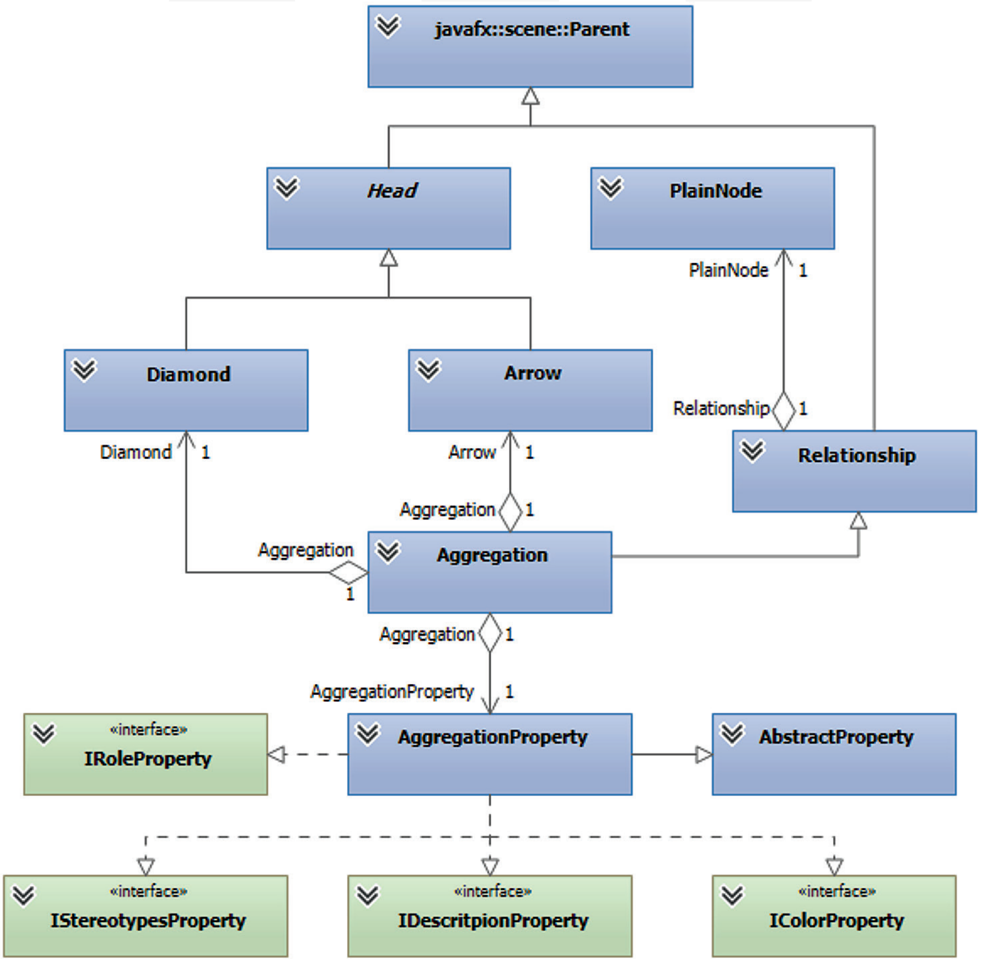

Fig. 1. Class diagram for the Aggregation class

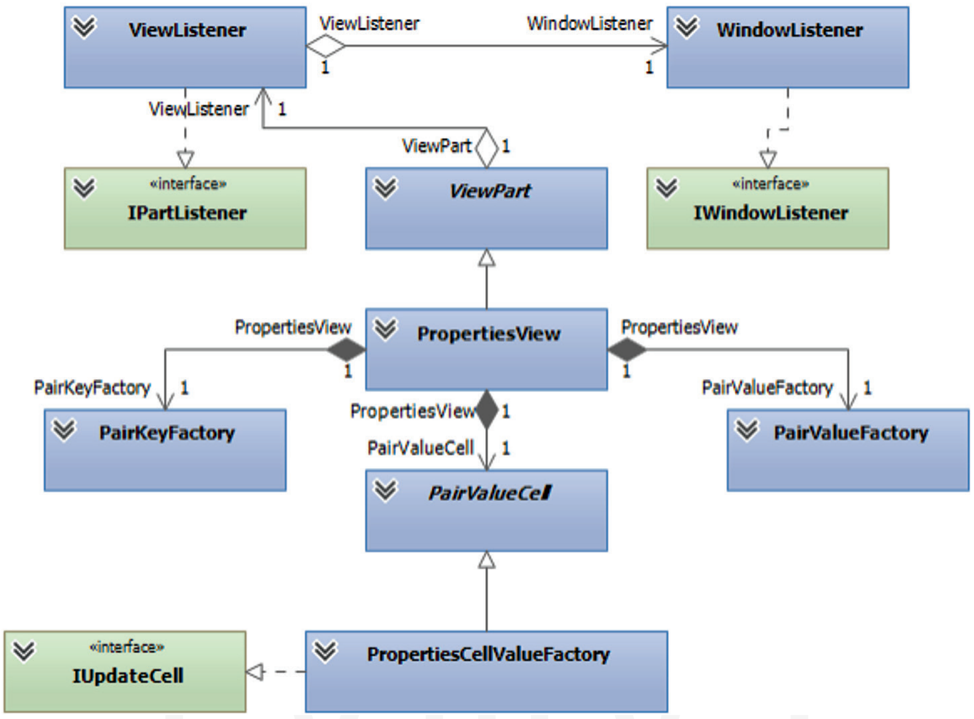

Fig. 2. Class diagram for the ‘Properties View’ class

```
package com.componentcreator.modeler.metamodel;

import static org.junit.Assert.*;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class NewProjectTest {
        private static SWTWorkbenchBot bot;
        @BeforeClass
        public static void beforeClass() throws Exception {
                bot = new SWTWorkbenchBot();
                bot.viewByTitle("Welcome").close();
        }
        @Test
        public void canCreateANewJavaProject() throws Exception {
                bot.menu("File").menu("New").menu("Project...").click();
                SWTBotShell shell = bot.shell("New Project");
                shell.activate();
                bot.tree().expandNode("Dynamic Meta-Modeling")
                        .select("Dynamic Meta-Modeling Modeler Project");
                bot.button("Next >").click();
                bot.textWithLabel("Project name:").setText("TestProject");
                bot.button("Finish").click();
                bot.waitUntil(Conditions.shellCloses(shell));
                assertTrue( ResourcesPlugIn.getWorkspace().getRoot()
                        .getProject("TestProject").exists() );
        }
        @AfterClass
        public static void sleep() {
                bot.sleep(2000);
        }
}
```

Fig. 3. Sample CDMM-F functional SWTBot and JavaFX dependent functional test case

6. Conclusions

The main goal of the research presented in this paper was to show that it is possible to implement a software system which may be used to define open ontology based meta-models in CDMM-P for CDMM-F. This software system is in the form of the Eclipse Plugin named the CDMM-Meta-Modeler which constitutes the physical form of the research result. Moreover, it was shown that such a system can be built using available tools originally designed to support the close ontology based approach to meta-modeling. Specifically, the application of the UML2 modeling Plugin and the UML Profile concept were reused in the CDMM-Meta-Modeler to express open ontology meta-models.

The CDMM-Meta-Modeler tool was carefully designed and tested. In future research, a detailed case study illustrating how the CDMM-Meta-Modeler can be used will be presented. This presentation will be done for a meta-model which is quite simple but expressive enough and applicable to commercial product software development processes.

Some useful functionalities marked in section 3 are still under development – these are planned to be implemented soon in order to make the whole vision of the CDMM-Meta-Modeler presented in the paper workable.

References

[1] Aßmann U., Zschaler S., Wagner G., *Ontologies, meta-models, and the modeldriven paradigm*, [in:] C. Calero, F. Ruiz, M. Piattini (Eds.), *Ontologies for Software Engineering and Software Technology*, Springer, 2006, p. 249-273.

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

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

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

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

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

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

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

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

[10] Guizzardi G., *On ontology, ontologies, conceptualizations, modeling languages and (meta) models*, [in:] *Frontiers in Artificial Intelligence and Applications*, Vol. 155, Conference on Databases and Information Systems IV, IOS Press, Amsterdam, selected Papers from the Seventh International Baltic Conference DB and IS 2006, pp. 18-39.

[11] Holanda O., Isotani S., Bittencourt I., Elias E., Tenório T., *Joint: Java Ontology Integrated Toolkit*, Expert Systems with Applications 40, 2013, p. 6469-6477.

[12] Kleppe A. G., Warmer J., Bast W., *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA 2003.
[13] Object Management Group, *Meta Object Facility (MOF) core specification version 2.0*, 2006, http://www.omg.org/spec/MOF/2.0.
[14] Object Management Group, *Unified Modeling Language (UML) superstructure version 2.2*, 2009, http://www.omg.org/spec/UML/2.2.
[15] Zabawa P., Stanuszek M., *Characteristics of Context-Driven Meta-Modeling Paradigm (CDMM-P)*, Technical Transactions, 3-NP/2014, p. 123-134.
[16] Zabawa P., *Context-Driven Meta-Modeling Framework (CDMM-F) – Internal Structure*, 2015 (submitted for publication).
[17] Zabawa P., *Context-Driven Meta-Modeling Framework (CDMM-F) – Context Role*, 2015 (accepted for publication).
[18] Zabawa P., *Context-Driven Meta-Modeling (CDMM) – Simple Horizontal Case-Study*, 2015 (submitted for publication).