TECHNICAL TRANSACTIONS | **CZASOPISMO TECHNICZNE**
FUNDAMENTAL SCIENCES | NAUKI PODSTAWOWE

3-NP/2014

PIOTR ZABAWA, MAREK STANUSZEK*

CHARACTERISTICS OF THE CONTEXT-DRIVEN META-MODELING PARADIGM (CDMM-P)

CHARAKTERYSTYKA PARADYGMATU META MODELOWANIA STEROWANEGO KONTEKSTEM (CDMM-P)

A b s t r a c t

The paper introduces a novel Context-Driven Meta-Modeling Paradigm (CDMM-P) and discusses its properties. The CDMM-P changes the traditional division of responsibilities within the data layer in software systems. It facilitates the interchangeable usage of both objects representing data and objects representing relationships. The decomposition of specific responsibilities results in the weakening of internal data model dependencies. This in turn allows for run-time construction of the whole data model. The proposed paradigm facilitates exceptional flexibility in the implementation of the data layer in software systems. It may be applied to domain modeling in enterprise applications as well as to the modeling of any ontology, including the construction of modeling and meta-modeling languages. As such, CDMM-P underpins a broad domain of Context-Driven Meta-Modeling Technology (CDMM-T).

Keywords: Data layer, model layer, paradigm, model parameterization, UML, entity class, pure entity class, entity relationship class, domain model, meta-model, meta-programming, meta-language

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

W artykule wprowadzono nową koncepcję Paradygmatu Meta Modelowania Sterowanego Kontekstem (CDMM-P) oraz przedyskutowano jego własności. CDMM-P zmienia tradycyjny podział odpowiedzialności w warstwie danych systemów softwerowych. Ułatwia wymienne stosowanie zarówno obiektów reprezentujących dane, jak i obiektów reprezentujących relacje. Podział tych specyficznych odpowiedzialności skutkuje osłabieniem wewnętrznych zależności modelu danych. Pozwala to z kolei na konstruowanie całej warstwy danych w czasie wykonania. Proponowany paradygmat zapewnia wyjątkową elastyczność implementowania warstwy danych systemów softwerowych. Może być on stosowany do modelowania dziedzinowego aplikacji korporacyjnych i do modelowania dowolnego systemu pojęć (ontologii) z konstruowaniem języków modelowania i metamodelowania włącznie. CDMM-P stanowi podstawę szerokiej dziedziny Technologii Meta Modelowania Sterowanego Kontekstem (CDMM-T).

Słowa kluczowe: warstwa danych, warstwa modelu, paradygmat, parametryzacja modelem, UML, klasa encyjna, klasa czysto encyjna, klasa relacji encyjnej, model dziedzinowy, meta model, meta programowanie, meta język

* Institute of Computer Science, Faculty of Physics, Mathematics and Computer Science of Cracow University of Technology; pzabawa@pk.edu.pl, mareks@riad.pk.edu.pl.

1. Introduction

There is a disparity between the UML's (Unified Modeling Language) system of notions and its application to programming languages. This disparity has been known for many years. In particular, there is no the programming language that offers a built-in mechanism for management of life-time of objects, in the meaning of associative relationships defined in UML – the association, aggregation, containment. As the result, the management of an object's life-time is traditionally assumed as the responsibility of applications implemented in these languages, thus it is the programmer's responsibility.

One possible solution to this problem is the new concept of meta-modeling presented here. This concept offers several new and very useful opportunities for software construction. This solution, as well as its accompanying features, constitutes a new CDMM-P (Context-Driven Meta-Modeling Paradigm) for both software design and development. The considerations presented here apply to the software system's data layer and the main focus of attention was the associative relationships and wide spectrum of applications of CDMM-P.

The CDMM-P presented here plays an important role in software engineering – it allows for the breaking of limits in UML's mapping of associative relationships into programming languages. Moreover, it helps to break limits characteristic of all known approaches to the modeling system of notions from any application domain, including the modeling of software systems.

2. State of the art

The topic of this paper overlaps software engineering and artificial intelligence. This is the first reason why the analyzed scientific literature is so differentiated. The second is the fact that domain modeling can refer to vertical domains as well as to the construction of modeling languages – meta-languages. The last reason is new and shows that, as a consequence of CDMM-P implementation, the same notions can be implemented in enterprise applications as well as being applied in meta-modeling.

The main idea of the proposed paradigm is that information about the interconnections of entity classes is moved out from these classes and put into separate *entity relationship classes*. This way, the new responsibility distribution to the data layer is applied. Furthermore, the literature is considered in the context of the data layer only from several perspectives.

2.1. Architectural perspective

Generally speaking, there is no literature on the CDMM-P approach as long as the presented problem is analyzed within the perspective of traditional software engineering. This observation is confirmed by the contents of [29] which discusses architectural approaches to the data layer – there is no parameterization of the data layer by model. Moreover, in all available publications, such models are constructed statically through explicit embedding of relationships in interrelated classes in the form of pointers or references. More general software engineering subject reviews like [5] or [23] also confirm this observation. The same conclusion results from the literature dedicated to more detailed problems. In [34], the ADOM-UML (Application-based DOmain Modeling) is used to enrich modeling in order to fill the gap between domain and application models, while in [35], the SDM (Semi-Automated Domain Modeling) concept is introduced to infer domain models from the former application models stored in a repository. In [28], the system for gathering information about modeling is introduced, but its meta-model has a fixed structure. In paper [7], the domain model is composed from many interrelated heterogenic project artifacts and the emphasis is put on the important role of the relationships performed in that approach. In [2], the challenge for the research is to make use of very large and very complex UML and Ecore meta-models [11]. In paper [13], the evolutionary introduction of changes to running software system without it stopping is shown. However, this goal is achieved through carrying out a priori off-line analysis. Paper [14] presents information regarding how to mix interrelated ontologies in a direct and static way. In [25], the meta-model is inferred from several models.

2.2. Ontological perspective

The notion of ontology in its technical meaning originated from artificial intelligence research. In software engineering, it can be applied to end-user domain enterprise applications (the product), to the problems of modeling and meta-modeling or to the software development process. The analysis presented below addresses all of the above perspectives with the exception of processes.

Monograph [4] contains traces of references to the concepts of open ontology, but without references to software engineering. More references to ontology can be found in [15] and [16].

Paper [10] shows how to define the domain model with the aid of ontology. Then, with the application of technologies well suited for DSL (Domain-Specific Language) construction and Magic Potion [8] like Clojure/Lisp [19, 21, 22], the compilable and executable source code in the domain-specific language for making operations on this model is created. They refer to the Semantic Web which is a kind of open ontology and does not have a meta-models hierarchy above the model layer – this is typical of OMG (Object Management Group) approaches.

Another very important publication is [24] which presents the system for ontology-based application design. The access to object's domain takes place through object-oriented paradigm, which is not new compare to [37, 30]. The cited work contains the whole IDE (Integrated Development Environment) named JOINT (Java Ontology Integrated Toolkit). It offers the ability of generating Java source code directly from the ontology.

Paper [33] is focused on the method of the application of ontology to the customization of the domain model for the needs of a particular software system. In this way, the ontology as an expression medium for OMG vertical standards is applied. In paper [12], ontology is used to infer a static class model, while in [26], a special ontology must be chosen to build OGML (Ontology Grounded MetaLanguage) meta-language on top of it.

The analysis of the "whole-part" relationship is presented in [32]. However, this analysis, based on [20], assumes that the WP (Whole-Part) relationship is defined as a class of property. This is why the analysis is so complex and needs dedicated frameworks.

In [9], the MOF-based (Meta Object Facility) MDA-compliant (Model Driven Architecture) meta-model is introduced for modeling ontologies, thus, software engineering and artificial intelligence are combined. While in paper [3], the proposed framework is integrated into several ontologies through their injection into the more general meta-ontology.

2.3. Paradigmatical perspective

Further we may analyze the literature from the paradigmatical perspective. In paper [8], the meta-modeling role, as a way of application of many paradigms at the same time, is presented, referring directly to [10]. A very interesting solution named "MagicPotion" which has already been discussed in section 2.2, is introduced there.

According to [36], meta-programming is the old concept of "using programs to manipulate other programs" and in case of Java, the meta-programming concept is supported by annotations. This paper defines the requirements of meta-programming.

2.4. DSL perspective

DSL languages are always designed in the context of a particular application domain. This domain is usually limited to the operations executed on the data layer. This is why the approach to the construction of this layer results from the way in which the DSL is perceived. This conclusion also relate to presented work.

The work reviewed in [27] shows that, independently of the approach to DSL design, the closed character of the ontology is assumed at the very beginning. This publication describes the abstract syntax of DSL as a meta-meta-model or as an Abstract Syntax Graph (ASG)/ Abstract Syntax Tree (AST), but each of these approaches closes the ontology by MOF or by grammar respectively. In publication [31], localized in the area of Computer Supported Cooperative Work (CSCW), the appropriate DSL is constructed on the basis of close ontology (Ecore). Moreover, article [6] is focused on the embedding of abstractions into domain-driven approaches (DSML – Domain Specific Modeling Languages) to DSL construction. These abstractions can be reused in many application domains. The mapping between them, and particularly between modeling languages, is implemented in the form of annotations.

2.4. Mixed perspective

There are some papers dealing simultaneously with several of the perspectives pointed out above. In [38], DSL, ontology notions and UML modeling are joined and the idea of DSL construction based on fixed static entity classes structure is presented. Paper [18] presents an approach to the change of meta-model via the application of a universal ontology introduced in [17]. Combination of the ontological approaches with meta-modeling is shown in [1]. It is dedicated to providing a better transformation from modeling of general concepts in domain languages (ontology) to models/meta-models that are MDA-compliant. However, fixed entity models are used again.

3. Decomposition of data model responsibilities

The main goal of the CDMM-P is to take the responsibility for lifetime management of the objects, inter-related through associative relationships characteristic of the data layer. This goal can be achieved through the change of responsibilities, that is the typical solution implemented for design. It transpires that the entity classes encompass too many responsibilities. On the one hand, they map application objects to their data source and on the other hand, they store relationships that in turn define how the lifetime of related objects must be managed. At the same time, entity classes or application codes are responsible for lifetime management. However, this should be the responsibility of relations because they know their own nature best. This is why the split of both responsibilities through the introduction of the notions of pure entity (in contrast to the established notion of entity) and entity relationship is. Such a decomposition was not considered in the literature till now. Pure entity classes are responsible for the representing of data while the entity relationships are responsible for the lifetime management of pure entity class instances involved in associative relationships. The sample data models of a class named Company for both traditional and new approaches are presented in Fig. 1.

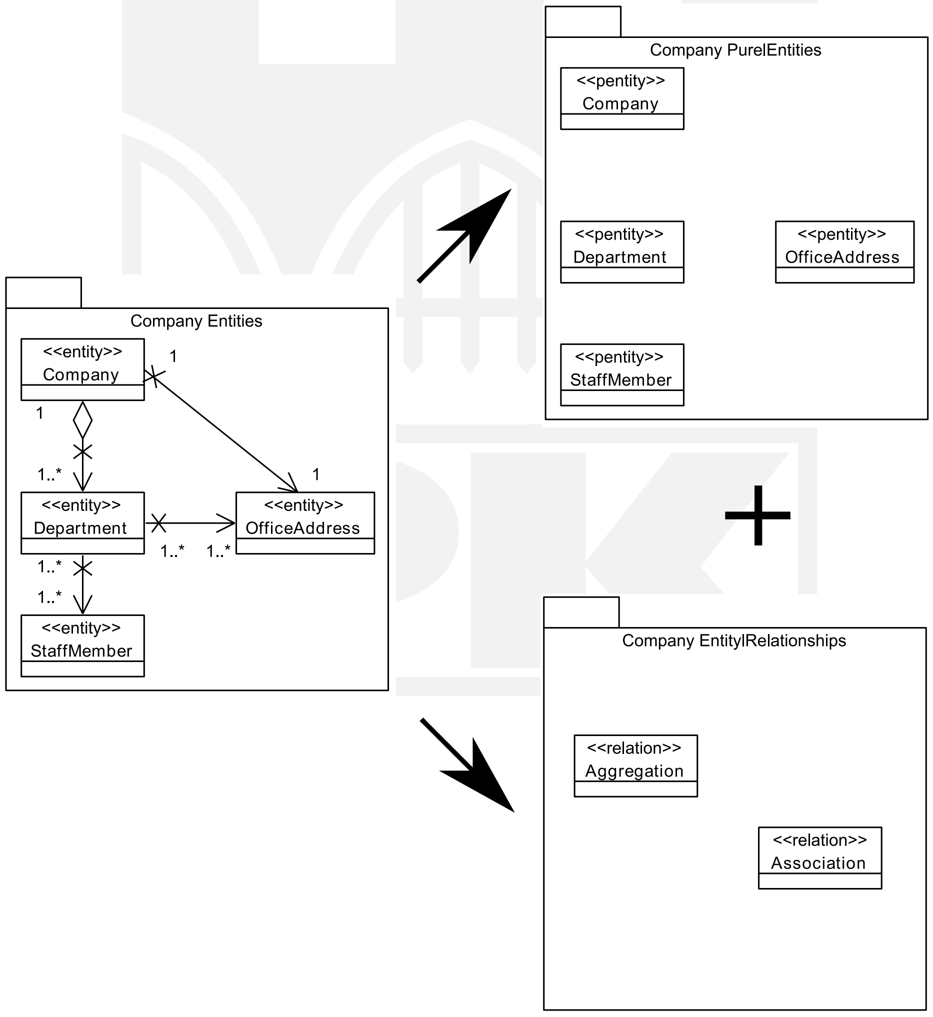

Fig. 1. Conceptual transformation from fixed entity model to context-driven entity model for sample Company model

With the introduction of this crucial modification, the application's source code and (optionally) entity classes are released from the responsibility of representing relations. In effect, it leads to the weakening of dependencies on the following borders:

- pure entity class – pure entity class,
- entity relationship – entity relationship,
- pure entity class – entity relationship.

Maximal weakening of the above dependencies can be described by not measurable but intuitive quality criterion 'weak coupling & strong-cohesion' which is commonly accepted as good design practice in software engineering. This weakening plays the key role in the CDMM-P concept. Moreover, it constitutes the starting point for reaching and achieving the goals described below.

4. Influence of responsibilities shifting on the ease of change management

The elimination of circular dependencies as well as the limitation of dependencies were introduced as good practice in software engineering. In the context of the CDMM-P, significant weakening allows for the achieving of two succeeding goals:

- maximal limitation of the scope of introduced change,
- possibility of run-time data layer change.

The first goal is quite obvious – the introduction of change into the pure entity class does not impact the entity relationship. Similarly, introduction of change into the entity relationship does not influence pure entity classes involved in this. The second goal is however, not as clear. All classes introduced by the CDMM-P can be interconnected at run-time due to the fact that they are independent of each other. Specifically, all pure entity class objects may first be loaded, and then all entity relationships between them can be created. The point here is not, however, just to interconnect existing objects, but also to store information about their classes. These allows to:

- obtain information from each *pure entity class* about the *entity relationship classes* that the *pure entity class* is involved in,
- obtain information from each *entity relationship class* about the *pure entity classes* involved in this *entity relationship*.

An XML file describing both classes and relationships can be used as the source of information about both *pure entity classes* and their relationships. Such a file can be interpreted by the implementation of the CDMM-P, which can take the form of the framework named Context-Driven Meta-Modeling Framework (CDMM-F). Such a framework will be presented in further publications. As a result of this approach, all classes involved in the model can be run-time interconnected, as depicted in Fig. 2. The class objects can be created from the CDMM-F's client level through the API (Application Programming Interface) after loading the whole data layer model from the XML file mentioned above.

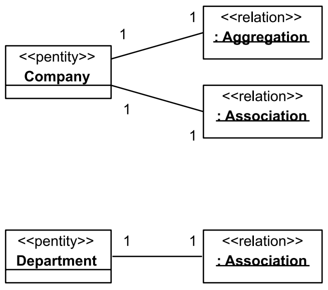

Fig. 2. Implementation-dependent context-driven entity model of a sample Company model from Fig. 1 with injected relationships

5. Parameterization of data layer by its model

The authors noticed that in the data layers of software systems, like in UML, there is a strong asymmetry in the number of available entity classes (potentially infinite) and the number of available relationships among them (three associative relationships are usually sufficient) in most domain models. It thus allows for the introduction of predefined *entity relationship classes* and for further parameterization of the data layer through *pure entity classes* defined by the user of CDMM-P. Predefined entity relationship classes are independent of user-defined *pure entity classes* thanks to the maximal weakening of dependencies in the CDMM-P. At the same time, user-defined *pure entity classes* are independent of predefined entity relationship classes. As a result, the set of entity relationship classes can be modified independently of the set of *pure entity classes*. This means that in such a situation, the relationships between any set of classes can be shaped as needed without any limits – there is nothing that closes or limits these sets. Thus, the data model can be created according to the CDMM-P on the basis of its UML model. More specifically, this model can be loaded at run-time for the purpose of the dynamic creation of the application data layer as long as the CDMM-P is used. The application of the UML class model, as a widely accepted standard, is quite obvious, but the whole structure of the data layer can be loaded from any resource describing the whole structure both sufficiently and unambiguously.

The presented method of the dynamic loading of the data model into the application, based on CDMM-P, is possible because of the dependency weakening presented above. Consequently, there is no information in the entity classes about other entity classes. According to the CDMM-P, there are no property fields in *pure entity classes* that represent entity relationships just as in entity relationship classes, there are no fields representing *pure entity classes*. *Pure entity classes* interconnecting via entity relationships in the CDMM-P do not have compile-time character but have run-time only. As a result, the data layer is parameterized at run-time by the data model.

6. Reusability consequences

The data layer creation based on CDMM-P allows for any customization of previous data models being constructed in an IT enterprise. It also significantly improves the possibilities for the reuse of former *pure entity classes* (they may be used in any relationships context) as well as the reuse of specific entity relationship classes (they may be used in any context of *pure entity classes*). Thus, the approach presented here allows for unlimited changes of the context through introducing *pure entity classes* and entity relationship classes in the data model. Moreover, this context can be established at run-time. In the result, the *pure entity classes* can be run-time changed or replaced in the application containing them. Achieving any of the goals presented here with the application of traditional approaches to software engineering is not possible.

7. Class-based perspective for non-class-based paradigm

The CDMM-P cannot be implemented in class-based versions of object-oriented technologies. Nevertheless, it should be taken into account that its potential users (developers) implement this approach only in the most popular class-based version of the object-oriented paradigm. In order to make the CDMM-P useful for the developers community, it should be matched to the way they make use of entity classes in the class-based variant of the object-oriented manner. Thus, the next goal of the CDMM-P is the ability to offer its users class-based access to CDMM-P's specific notions – *pure entity class* objects and entity relationship objects. This goal can be achieved via introduction of the additional layer with both access to entity relationship objects through the *pure entity objects* perspective and access to *pure entity objects* through entity relationship objects being possible. Based on the introduced layer, the CDMM-P's client application could not only see all objects available in the CDMM-P from the traditional class-based version of the object-oriented approach, but could also use any of the following graph data model scanning strategies:

- *pure entity classes* based strategy,
- *entity relationship classes* based strategy,
- mixed strategy adjusted to current needs.

Two first strategies are mutually symmetrical and can be used interchangeably.

8. Reflectiveness vs. static nature of data model

The construction of an application based on CDMM-P may have one of the following characters:

- static – it is assumed that the developer knows a priori the data model which is run-time loaded,
- dynamic – it is assumed that the developer does not know the data layer model.

In the first case, the application source code is simpler for implementation, but it does not allow for any change of static structure of the data model. In the second case, the developer can dynamically query the *pure entity classes* for their entity relationships. The developer can also query the entity relationship classes regarding their *pure entity classes*. The developer can then execute appropriate operations on objects of these classes. It is important from the CDMM-P perspective that there are no limitations for the choice between static and/or dynamic model of implementation of the application based on CDMM-P. Moreover, these two approaches can be mixed.

9. Standardization of CRUD operations on data layer

For the sake of the application of CDMM-P to the data layer, it is possible to limit the set of operations on the *pure entity* and *entity relationship classes* mentioned above to CRUD (Create, Read, Update and Delete) operations. Besides typical operations executed on entity classes in a traditional approach, here we may execute them not only on entities (operations on *pure entity classes*) but also on entity relationships. The proposition of a pre-defined set of such operations is one of the goals of CDMM-P. For the purpose of the introduction of the set of predefined CRUD operations, the CDMM-P should itself give access to entity relationships visible to the client from the perspective set to the *pure entity class*. This mechanism is crucial for CDMM-P and distinguishes it from other potential solutions. It is possible to achieve this goal through the introduction of the ac*cessor notion. The accessor is responsible for putting the appropriate perspectives which were presented above. In the case of the standardization of CRUD operations described here, such an accessor should guarantee the ability of getting access to the entity relationship from the appropriate *pure entity class*. The accessor itself can be obtained through the following polymorphic operation:

```
BaseClass getAccessor(pure-entity-class, entity-relationship-
class)
```

where

`BaseClass` – base class for all data model elements
`pure-entity-class` – *pure entity class*
`entity-relationship-class` – *entity relationship class*

Thus, the set of standardized operations, which can be executed on the object accessor to the *pure entity class* visible from the perspective put on *pure entity class*, may have the following form in pseudo-code:

```
void add(relationship-container, relationship-element)
T get(relationship-container)
int count(relationship-container)
List<T> getAll(relationship-container)
T get(relationship-container, index)
int count(relationship-container, relationship-element-class)
T get(relationship-container, relationship-element-class, index)
List<T> getAll(relationship-container, relationship-element-class)
```

where

`relationship-container` – *pure entity object* its *entity relationship* with other *pure entity objects* are visible through;
`relationship-element` – *pure entity object* bind to the *entity relationship* with *pure entity object* which gives access to object via the relationship obtained from entity;
`relationship-element-class` – information about the class of *pure entity object*
`index` – index of the *pure entity object* in a collection

It is worth noting that each *pure entity class* may have many *entity relationships*, which in turn may cause difficulties in CDMM-P's implementation in contemporary programming languages. Nevertheless, reaching this goal is possible.

10. Testing specificity

The two most important modifications of traditional approaches to testing are presented below. The first is related to unit testing of isolated classes and the second, to unit-like scenario tests responsible for testing class interconnections.

10.1. Unit testing

One of the consequences of weakening of the dependency and making classes independent is making unit tests of these classes also independent. As a result, each *pure entity class* and *entity relationship class* can be tested in isolation. Looking from the perspective of reusability, it must be underlined that both *pure entity* and *entity relationship classes* can migrate between different projects together with their unit tests, as long as the projects are based on CDMM-P. These kinds of tests constitute the additional layer of tests responsible for methods testing all data layer classes. This test layer is not known in a traditional object-oriented approach due to compilation dependencies between entity classes.

10.2. Semi-automatically generated scenario tests

For the sake of the dynamic run-time interconnecting of *pure entity classes* by *entity relationship classes,* the necessity for testing these classes in the context of their data model arises. It is however, worth pointing out here that scenario tests skeletons can be generated automatically from the same model as the CDMM-P classes are organized with the layer dynamically loaded. Thus, the proposed model plays the role of the parameter for both implementation and tests. The skeletons mentioned above could be implemented against unit testing frameworks. However, in order to generate these tests, the required objects of these classes must also be created on the basis of the association of the classes with their descriptions. Thus, such tests take the form of semi-automatic tests. The scenario test's source code is generated automatically from the objects' information that must be introduced by a developer. Based on this approach, the developer need not write the test code manually, but still keeps control over the test via the possibility of object initialization. Obviously, the CDMM-P does not eliminate the possibility of the implementation of manual scenario tests in any form, as well as introduction of modifications into already generated tests.

11. Conclusions

A new paradigm for software data layer design and implementation was introduced in this paper. The concept of the proposed approach was described at a general level to open new research fields leading to the implementation of the paradigm in different technologies. Such implementations will be presented in further research publications expanding the CDMM-P approach to the Java EE platform.

This paradigm can be used both for enterprise systems' data layer design and implementation and for the construction of modeling languages.

Finally, it is worth underlining that the concept of the paradigm CDMM-P proposed here is based upon the open ontology approach.

All diagrams were prepared with the help of Visual Paradigm UML modeling tool according to the Academic Partner Program agreement signed with Cracow University of Technology.

References

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

[2] Bendera A., Poschlada A., Bozica S., Kondova I.. *A service-oriented framework for integration of domain-specific data models in scientific workflows*, [In:] J. Smith (ed.), Procedia Computer Science 18, International Conference on Computational Science, 2013, 1087-1096.

[3] Biletskiy Y., Ranganathan G., *A semantic approach to a framework for business domain software systems*, Computers in Industry, **61**, 2010, 750-759.

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

[5] Chung L., Noguera M., Subramanian N., Garrido J., *Novel approaches in the design and implementation of system/software architectures*, The Journal of Systems and Software, **85**, 2012, 459-462.

[6] Guerra J.L. de E., Cuadrado J., *Reusable abstractions for modeling languages. Information Systems*, **38**, 2013, 1128-1149.

[7] Diaz I., Llorens J., Genova G., Fuentes J., *Generating domain representations using a relationship model. Information Systems*, **30**, 2005, 1-19.

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

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

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

[11] *Ecore meta model in*: Eclipse Modeling Framework: www.eclipse.org/modeling/emf.

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

[13] Fung K., Low G., *Methodology evaluation framework for dynamic evolution in composition-based distributed applications*, The Journal of Systems and Software, **82**, 2009, 1950-1965.

[14] 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, 1099-1118.

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

[16] Gašević D., Kaviani K., Milanovic M., *Ontologies, software engineering*, In Handbook on Ontologies, Springer-Verlag, 2009.

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

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

[19] Halloway S., *Programming Clojure*, Pragmatic Bookshelf, 2009.

[20] Henderson-Sellers B., Barbier F., *What is this thing called aggregation?*, R. Mitchell, A. Wills, J.B.B. Meyer (eds.), Proceedings of TOOLS29 EUROPE99, IEEE Computer Society Press, Silver Spring, 1999, 216-230.

[21] Hickey R., *The clojure programming language*, In Proceedings of the 2008 symposium on Dynamic languages, 2008.

[22] Hickey R., *Clojure homepage*, http://www.clojure.org, 2009.

[23] Hofmeister C., Kruchten P., Nord R., Obbink H., Ran A., America P., *A general model of software architecture design derived from five industrial approaches*, The Journal of Systems and Software, **80** (1), Jan 2007, 45-50.

[24] Holanda O., Isotani S., Bittencourt I., Elias E., Tenório T., Joint: *Java ontology integrated toolkit. Expert Systems with Applications*, 4**0**, 2013, 6469-6477.

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

[26] Laarman A., Kurtev I., *Ontological meta-modelling with explicit instantiation*, [In] M. van den Brand, D. Gašević, J. Gray (eds.), SLE 2009, LNCS 5969, Springer-Verlag, Berlin 2010, 174-183.

[27] Langlois B., Jitia C., Jouenne E., *DSL classification*, In Proc. OOPSLA 7th Workshop on Domain Specific Modeling, University Park, PA, Citeseer 2007.

[28] Malhotra R., *Meta-modeling framework: A new approach to manage meta-model base and modeling knowledge*, Knowledge-Based Systems, **21**, 2008, 6-37.

[29] Merson P.. *Data model as an architectural view*, Technical Note CMU/SEI-2009-TN-024, Software Engineering Institute, Carnegie Mellon University, 2009.

[30] Mika P., *Social networks and the semantic web. Semantic web and beyond*, Springer, 2007.

[31] Molina A., Gallardo J., Redondo M., Ortega M., Giraldo W., *Metamodel-driven definition of a visual modeling language for specifying interactive groupware applications: An empirical study*, The Journal of Systems and Software, **86**, 2013, 1772-1789.

[32] Opdahl A., Henderson-Sellers B., Barbier F.. *Ontological analysis of wholepart relationships in oo models*. Information and Software Technology, **43**, 2001, 387-399.

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

[34] Reinhartz-Berger I., *Utilizing domain models for application design and validation*. Information and Software Technology, **51**, 2009, 1275-1289.

[35] Reinhartz-Berger I., *Towards automatization of domain modeling*, Data and Knowledge Engineering, **69**, 2010, 491-515.

[36] Spinellis D., *Rational meta-programming*, IEEE Softw., **25** (1), 2008, 78-79.

[37] Szekely B., Betz J., Jastor: Typesafe. *Ontology driven rdf access from java*, http://jastor.sourceforge.net, 2009.

[38] Tairas R., Mernik M., Gray J., *Using ontologies in the domain analysis of domain-specific languages, In Models in Software Engineering*, Springer-Verlag, New York 2009, 332-342.