The Journal of Systems and Software 122 (2016) 327–341

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Model-based M2M transformations based on drag-and-drop actions: Approach and implementation

Tomas Skersys[a,b,*], Paulius Danenas[a], Rimantas Butleris[a,b]

[a] Center of Information Systems Design Technologies, Kaunas University of Technology, K. Barsausko str. 59, Kaunas, Lithuania
[b] Department of Information Systems, Kaunas University of Technology, Studentu str. 50, Kaunas, Lithuania

A R T I C L E I N F O

Article history:
Received 20 April 2016
Revised 25 September 2016
Accepted 26 September 2016
Available online 28 September 2016

Keywords:
Model-to-model transformation
Model-driven development of transformations
Drag-and-drop actions
UML profile
MDA
CASE tool

A B S T R A C T

In the context of model-driven development, model-to-model (M2M) transformations are often positioned as one of the key selling features of this ever-growing paradigm. Indeed, M2M transformations can speed up the development process by automating certain modeling tasks, enable reusability of existing models within a single or even multiple projects, and bring other actual benefits to a systems developer. Nevertheless, CASE tool-supported M2M transformations are quite often represented as hard-coded "black-box" solutions lacking flexibility and customization features. This paper presents main conceptual and implementation aspects of a practical approach for both the development and application of model-based, customizable M2M transformations. The transformation is triggered by so called drag-and-drop action, which is enacted after a certain element is dragged from a model browser and dropped into a diagram or onto some other element representation in the diagram. Another distinctive feature of the presented approach is the introduction of "partial M2M transformation". The term assumes a transformation of a user-defined fragment of the source model into a fragment of the target model, instead of taking whole models as in case of full M2M transformation. The presented solution improves overall usability of such M2M transformations in an actual CASE tool environment.

1. Introduction

After the introduction of Model Driven Architecture (MDA) in 2003 (OMG, 2003), model-to-model (M2M) transformations (Biehl, 2010) have become an essential part of the modern model-driven development paradigm. Additional boost was given by the introduction of UML 2.0 featuring powerful model extension mechanism. Indeed, after the adoption of UML 2.0, M2M transformations became a common means for the generation of platform-specific models (PSM) from platform-independent models (PIM) (OMG, 2003), which in its turn made M2M transformations a highly desired feature in any advanced MDA CASE tool.

Our personal modeling experience, empirical observations and constant feedback from our professional IT partners also show that a certain subtype of M2M transformations, i.e. *user-interacted partial* M2M transformations, are in particular demand, especially when it comes to agile, highly iterative model-driven development. By the term *partial M2M transformation* we assume a transformation of a user-defined fragment of the model (i.e. a set of selected source model elements) into a fragment of another model (i.e. a set of target model elements), instead of the whole model as in full M2M transformation. Such demand may be argued by the fact that a CASE tool user prefers to have certain degree of "freedom of choice" when it comes to selecting a process of work, possibilities to create few models at a time while reusing each other's concepts, modify these models on demand, or even develop and execute your own M2M transformations dedicated for specific uses. These are just few common situations where the advantage of *partial model-based* M2M transformations over traditional *full* M2M transformations is the most obvious.

Also, traditional M2M transformation approaches are not effective when it comes to the development of semantically rich business and platform-independent models – these transformations simply cannot provide sufficient levels of flexibility, customization, and user interaction. In many cases, this could be explained by the fact that M2M transformations in CASE tools are implemented either as hard-coded solutions or using dedicated transformation languages (e.g., ATL (Eclipse Foundation, 2016a), QVT (OMG, 2011)), thus the possibility for a CASE tool user to somehow inspect or modify the predefined transformations, or even develop the new ones, is highly limited. Overall quantity of such M2M transformations in a CASE tool is also limited and may not satisfy the needs of a tool user.

* Corresponding author.
E-mail addresses: tomas.skersys@ktu.lt (T. Skersys), paulius.danenas@ktu.lt (P. Danenas), rimantas.butleris@ktu.lt (R. Butleris).

http://dx.doi.org/10.1016/j.jss.2016.09.046
0164-1212/© 2016 Elsevier Inc. All rights reserved.

Another common feature of traditional full M2M transformations is working with source and target models in XMI format. Such approach makes real time user-interacted transformations (e.g., based on drag-and-drop (D&D) actions or popup menu-based selection of model concepts) practically unusable, especially if a transformation is executed by an external transformation engine. This is due to the fact that an "XMI-to-XMI" transformation requires reloading a project from the generated XMI file each time the transformation is performed in a CASE tool, unless the tool supports in-memory transformations.

In this article, a practical model-driven approach of partial M2M transformations is presented.[1] This approach provides one the ability to *use, customize* and *develop* new M2M transformations within a single CASE tool platform overcoming most of the above mentioned issues inherent to traditional full M2M transformations. In the proposed approach, transformations are invoked by so called drag-and-drop (D&D) actions and then executed by the internal transformation engine without reloading any working models. D&D action is triggered after a certain element is dragged from a model browser and dropped into a diagram or onto some other element representation in the diagram.

The main work was carried out within the IDAPI project.[2] The paper presents an extended and updated version of the research presented in Skersys et al. (2015a) and Skersys et al. (2014). Compared to previous publishing, this paper presents a stable M2M transformations metamodel and extensive specification of the engineering solution itself; in addition to that, certain experimental findings on model-driven development and usability of partial M2M transformations are also reported in this paper.

Further, the paper is structured as follows: in Section 2, conceptual aspects of the approach is presented and discussed in more details; Section 3 describes the implementation architecture of the approach, while illustrative example of M2M transformation specification, which is also referenced from other parts of the paper, is presented in Section 4; the results of the experiment are drawn in Section 5; Section 6 explores basic threats to validity of the experiment results; in Section 7, overview of related work is presented; conclusions and future work insights conclude the paper.

2. Approach of model-based M2M transformations based on drag-and-drop actions

2.1. Introductory illustrative example

Before we go into conceptual and technical details of the approach itself, let us start with two simple examples illustrating the usability of the approach in actual modeling activities, as well as its capabilities of developing new and configuring the existing M2M transformations in a model-driven way.

For the first example, let us assume a business analyst has a BPMN business process model (BPM) designed for some business domain. Now, it is up to a system analyst to develop a user functional requirements model based on the business knowledge provided by the previously designed BPM (and some other sources of knowledge, which we will omit for simplicity reasons). The system analyst starts doing it by designing a UML use case diagram (UCD). This is not nearly a straightforward process; therefore, the analyst would normally start designing this diagram from the scratch just by looking at some process diagrams of the BPM. But instead of creating new UML concepts and putting them into UCD, the analyst starts selecting certain concepts from the BPM itself and then dragging and dropping them right into the worksheet of UCD; at the same time, CASE tool reacts to each analyst's drag-and-drop (D&D) action by creating sets of new interrelated UML use case model (UCM) concepts and automatically deploying them into the diagram. An illustrative example of one of such D&D action-invoked M2M transformation is presented in Fig. 1: a selected task from BPMN business process model (Tag 1) is being transformed into a set of interrelated concepts (*Actor-Association-UseCase*) of UML UCM, which are then automatically deployed into a use case diagram (Tag 2). Moreover, the analyst may also drag and drop certain participant (e.g. "Clerk") from his process model into the use case diagram, and the system will generate not only the corresponding actor, but also a set of related use cases corresponding to the activities performed by that participant in the process model. The *M2M transformation model* (specification) of this illustrative transformation example is presented in Fig. 8 (Section 4).

For the second example, let us assume our system analyst is not satisfied with the limited set of available predefined partial M2M transformations. While working with UML use case models, he wants to have the same D&D-invoked transformation not only for BPMN *Task* concepts (as shown in Fig. 1) but for *SubProcess* concepts as well. To make this happen, he simply opens up the BPMN_BPM-to-UML_UCM transformation model, finds the transformation specification (diagram) dedicated to BPMN *Task*, and uses it as a pattern to develop a new transformation dedicated to BPMN *SubProcess*. With a bit of practical experience, it all took him less than 5 min to have a new partial M2M transformation ready for use.

The two above mentioned examples illustrate some of the main "selling" features of the proposed approach, which will be described in more details further below.

2.2. Conceptual architecture of the solution

Fig. 2 describes basic interacting structural elements in a CASE tool, which are enacted during the execution of partial M2M transformations based on D&D actions.

After a user of a CASE tool performs a predefined D&D action (*D&D Action*) on a particular concept from a source model (*Model*), it acts as a trigger to invoke a DSL engine (*DSL Engine*), which uses certain M2M transformation specification (*M2M Transformation Specification*) to perform M2M transformation. The result of the transformation is a generated fragment (a set of concepts) of a target model. Each D&D action specification uses model concept types from either UML metamodel (*UML Metamodel*) or/and one or more UML profiles (*UML Profile*), which are used in that particular M2M transformation. M2M transformation specification may also be augmented with a transformation pattern (*M2M Transformation Pattern*) to specify more complex M2M transformations (an example of such transformation specification is presented in Fig. 8).

Technical aspects of the implementation are discussed in Section 3.

2.3. Basic requirements for a CASE tool to support the approach

In its essence, the approach of model-based partial M2M transformations is platform-independent; however, its implementation via D&D actions puts certain requirements for a target CASE tool. In this section, we will define basic set of requirements for a CASE tool, which must be met in order to use this approach:

– *Support of UML extension mechanism via UML profiling*. Today, the majority of state of the art CASE tools (e.g. Visual Paradigm,

[1] The latest version of the implementation of partial M2M transformations approach, also explanatory demos and other useful information can be found at: https://bitbucket.org/pauliusdan/magicdraw-d-d-transformations/.

[2] Project VP1-3.1-ŠMM-08-K-01-018 „Research and Development of Internet Technologies and their Infrastructure for Smart Environments of Things and Services" (acronym: IDAPI, duration: 2012-2015), partially funded by the European Social Fund (ESF).

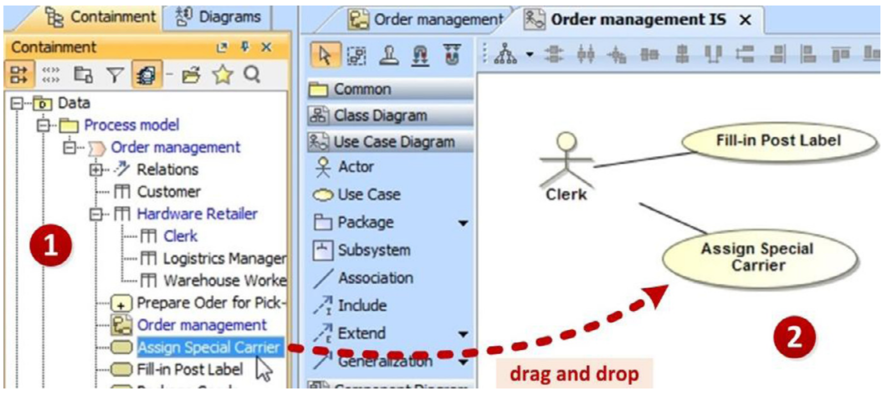

Fig. 1. Illustrative example of a partial M2M transformation using D&D action.

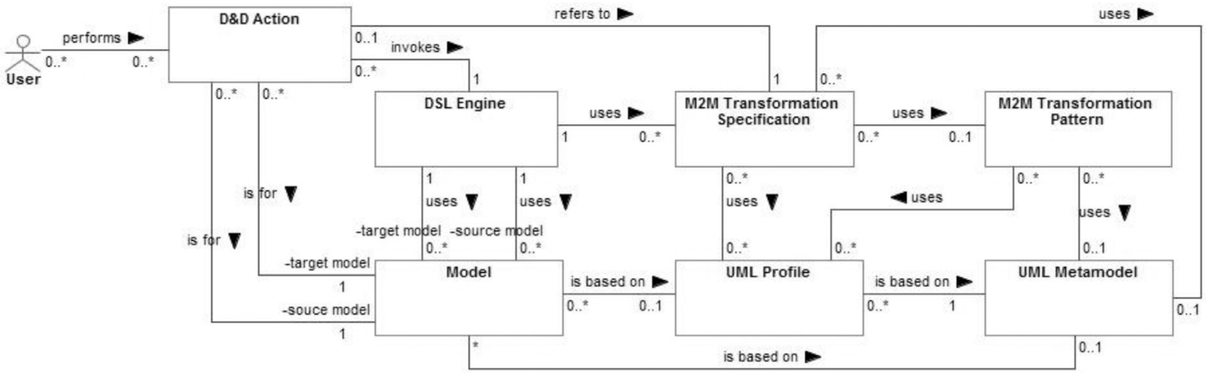

Fig. 2. Conceptual architecture of M2M transformations based on D&D actions in a CASE tool.

ArgoUML, MagicDraw, Enterprise Architect, Modelio) support this kind of extensibility. This capability can also be achieved by providing the ability to develop custom metamodels from scratch (e.g. Rational Software Architect); however, such approach is more time and effort consuming. Our approach is based on UML profiling.

– *Extensibility of CASE tool functionality*. In this regard, UML CASE tools can be divided into functionally extensible and non-extensible ones. This capability of functional extensibility is inherent to open source solutions or/and the ones supporting plug-in-based implementation architectures with stable public API. Our approach is based on (but the conceptual part is not bound to) the latter technology.
– *DSL engine and its extensibility*. If a CASE tool has an implemented UML extension mechanism, then it certainly has some DSL engine as well. The critical question is whether the engine itself is extensible and how this is achieved. In our approach, DSL engine extension is needed in order to allow the customization of so called D&D handlers. Anyhow, if the previous two requirements are satisfied by quite many CASE tools, the third one limits this number to only few choices. Today, only several well-known advanced CASE tools support model-based DSL customization. Here, the strongest are MagicDraw by No Magic Inc. (2016) and Enterprise Architect by Sparx Systems (2016).

Provided all the above mentioned capability requirements are met, each CASE tool still has its own ways of implementing things. Our current implementation is based on a CASE tool MagicDraw; all of the diagramming and explanatory examples in this paper are also developed using this CASE tool.

2.4. Application scenarios of M2M transformations based on drag-and-drop actions

Two main scenarios of *application* of the M2M transformation approach are presented Figs. 3 and 4.

Fig. 3 describes the 1st scenario of a user-system interaction when a user initiates a M2M transformation by dragging and dropping one concept *onto* another concept within the boundaries of the *same* diagram. The scenario has two cases resulting in two different outcomes:

– The 1st case is when dragging and dropping concept $a(i)$ onto concept $a(j)$ assigns the concept $a(i)$ as a new *value* to a predefined *property* of the concept $a(j)$;
– The 2nd case is when dragging and dropping concept $a(i)$ onto concept $a(j)$ creates a new *relationship* of a predefined type between these two concepts.

An example of the 1st case of the 1st scenario could be assigning resource “Clerk” to a lane set in BPMN process diagram. This is done by simply dragging and dropping the resource onto the chosen lane set; the executed M2M transformation would assign the resource “Clerk” to the lane set’s system property *represents*, which was defined in the specification of that particular M2M transformation (please, refer to Section 2.5 for more comprehensive presentation of M2M transformation specification). For the illustration of the 2nd case of the 1st scenario, let us consider that actor “Clerk” is not associated with use case “Assign Special Carrier” in the use case diagram of our introductory example (Fig. 1). In such case, dragging and dropping the actor onto the use case would create an association relationship between these two use case model elements.

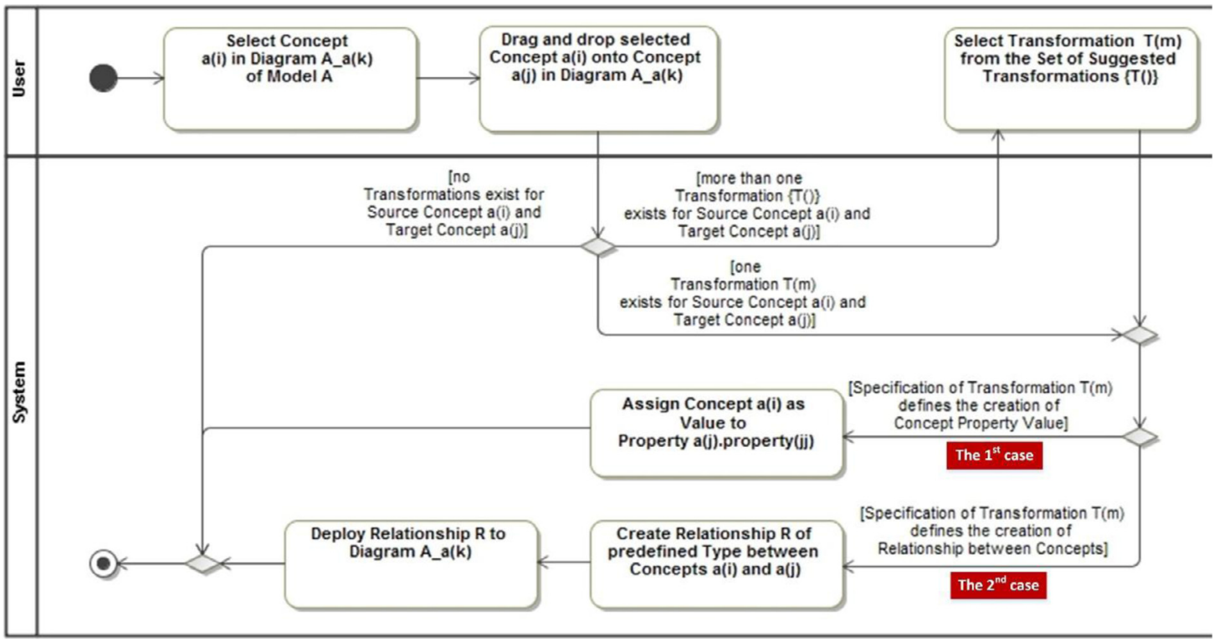
User
System
Select Concept a(i) in Diagram A_a(k) of Model A
Drag and drop selected Concept a(i) onto Concept a(j) in Diagram A_a(k)
Select Transformation T(m) from the Set of Suggested Transformations {T()}
[no Transformations exist for Source Concept a(i) and Target Concept a(j)]
[more than one Transformation {T()} exists for Source Concept a(i) and Target Concept a(j)]
[one Transformation T(m) exists for Source Concept a(i) and Target Concept a(j)]
Assign Concept a(i) as Value to Property a(j).property(jj)
[Specification of Transformation T(m) defines the creation of Concept Property Value]
The 1st case
Deploy Relationship R to Diagram A_a(k)
Create Relationship R of predefined Type between Concepts a(i) and a(j)
[Specification of Transformation T(m) defines the creation of Relationship between Concepts]
The 2nd case

Fig. 3. The 1st scenario of *using* partial M2M transformations.

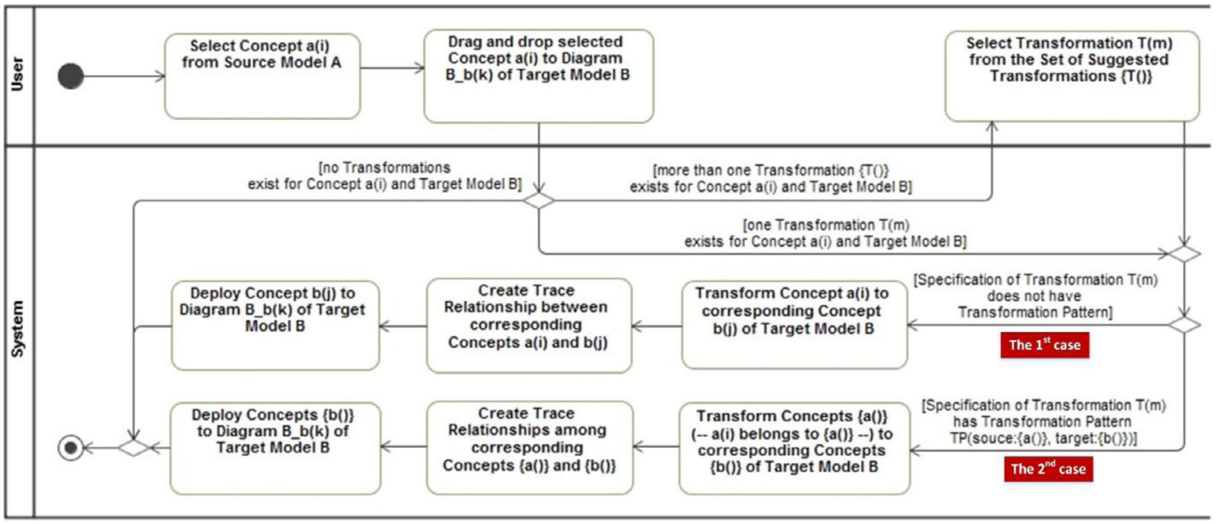
User
System
Select Concept a(i) from Source Model A
Drag and drop selected Concept a(i) to Diagram B_b(k) of Target Model B
Select Transformation T(m) from the Set of Suggested Transformations {T()}
[no Transformations exist for Concept a(i) and Target Model B]
[more than one Transformation {T()} exists for Concept a(i) and Target Model B]
[one Transformation T(m) exists for Concept a(i) and Target Model B]
Deploy Concept b(j) to Diagram B_b(k) of Target Model B
Create Trace Relationship between corresponding Concepts a(i) and b(j)
Transform Concept a(i) to corresponding Concept b(j) of Target Model B
[Specification of Transformation T(m) does not have Transformation Pattern]
The 1st case
Deploy Concepts {b()} to Diagram B_b(k) of Target Model B
Create Trace Relationships among corresponding Concepts {a()} and {b()}
Transform Concepts {a()} (-- a(i) belongs to {a()} --) to corresponding Concepts {b()} of Target Model B
[Specification of Transformation T(m) has Transformation Pattern TP(souce:{a()}, target:{b()})]
The 2nd case

Fig. 4. The 2nd scenario of *using* partial M2M transformations.

In both presented cases, transformations do not use supplementary transformation patterns and, therefore, could be considered as the least complex ones. Also, it should be mentioned that prior to our implementation, DSL engine of MagicDraw already had a limited support of both aforementioned cases of the first scenario; however, this capability was used very inexplicitly, no predefined transformation libraries existed, and some transformations were entirely hardcoded.

Fig. 4 describes the 2nd scenario of a user-system interaction when a user initiates a M2M transformation by dragging and dropping a concept from a source model *into* a diagram of a target model. The scenario has two cases:

- The 1st case is when a transformation specification does not have a supplementary transformation pattern;
- The 2nd case is when a transformation specification is supplemented with a transformation pattern.

An example of the 1st case of the 2nd scenario could be dragging and dropping task “Assign Special Carrier” from BPMN process model into UML use case diagram, and automatically creating use case “Assign Special Carrier” as a result. The 2nd case of the 2nd scenario enacts the most complex transformations in our approach, because the specifications of these transformations are augmented with so called transformation patterns allowing one to perform many-to-many concept transformations. The scenario of using partial M2M transformations presented in the introductory example (Fig. 1) is a good illustration of the 2nd case of the 2nd scenario. The specification of that particular transformation is presented and explained in Section 4. Note that both cases of the 2nd scenario also include the creation of traceability relationships among the corresponding source and target elements; the semantics of traceability relationships is briefly discussed further in Section 2.5.

2.5. Metamodel of model-based M2M transformations based on drag-and-drop actions

Fig. 5 presents an implementation-independent metamodel of model-based partial M2M transformations, which are based on a

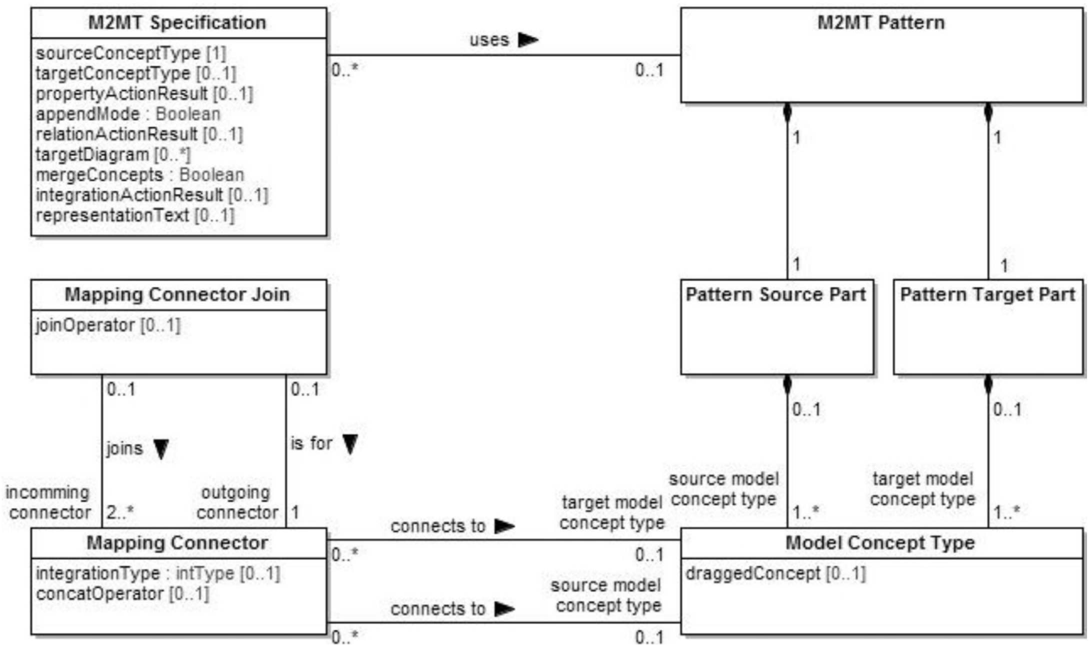

Fig. 5. Metamodel of model-based M2M transformations.

certain type of user interaction – D&D actions. Metaclasses in the metamodel could be divided into two sets: the first set holds a single metaclass *M2MT Specification*, which specifies all the main characteristics of an M2M transformation; the second set holds the rest of the metaclasses, which specify supplementary M2M transformation patterns for more complex transformations.

Let us describe the metamodel in more details:

– *M2MT Specification* is one of the core metaclasses of the M2M transformation metamodel. It holds a set of properties specifying all main properties of an M2M transformation. These properties are described in Table 1 – we suggest getting familiar with them before proceeding further. A single *M2MT Specification* class is enough to specify an atomic transformation containing *one* source model concept type (SMCT) and *one* target model concept type (TMCT). To specify more complex transformations involving multiple concepts from target and source models, a transformation specification should be augmented with a transformation pattern (*M2MT Pattern*). When used together, those two parts of specification compose a complete M2M transformation model, which provides the following possibilities to the transformation:
– Any concept type or its property can be mapped to any other concept type or its property.
– One can specify mappings within a single model or between different models expressed in the same or even different modeling languages (e.g. UML and BPMN, BPMN and SoaML) implemented as UML profiles.
– Many-to-many elements mappings can be specified.
– The M2M transformation pattern is specified by a set of metaclasses in the metamodel:
– The basic structure of *M2MT Pattern* is composed of two parts, namely, *Pattern Source Part* and *Pattern Target Part*. When a pattern is completed, it must contain both of these parts. A source part contains source model concept types, and a target part – target model concept types. A complete pattern must have at least one concept type (CT) in each of the two mandatory parts.
– SMCT from the source part may be mapped to a TMCT in the target part using *Mapping Connector*. The specified mapping between SMCT and TMCT means that the instance of TMCT will be created in the actual target model, and this concept will be assigned with the name of the particular instance of SMCT from the actual source model (e.g. name "*Assign Special Carrier*" of a task from BPMN process model is assigned to a newly created use case in UML use case model). For specific cases, string processing expression may be assigned to *concatOperator* – it is used when only a certain specified part of the name of SMCT instance needs to be assigned to the name of the corresponding instance of TMCT (e.g. name "*Special Carrier*" for the newly created class in a target model could be acquired by subtracting the first word from "*Assign Special Carrier*", which is the name of an existing use case in a source model). Integration type property *integrationType* may also be set for mapping connector; when this property is assigned with certain value (taken from the set of predefined values specifying types of model integration), it overrides the value defined in property *integrationActionResult* of *M2MT Specification* (see Table 1 for more information about the latter property of *M2MT Specification*).
– *In* each part of a transformation pattern, CTs are also interconnected with connectors. However, in this case connectors are used to preserve structural rules of the underlying metamodel, from which CTs are taken.
– Two or more SMCTs from the source part may be mapped to a single TMCT in the target part using *Mapping Connector Join*. Property *joinOperator* is used to define a certain sequence of how to join the names of incoming instances of SMCTs in order to form the name of the corresponding instance of TMCT (e.g. use case "*Assign Special Carrier*" in a target model could be created from the combination of two interrelated source model elements put in the predefined order: 1) association "*Assign*", 2) class "*Special Carrier*").

Please, refer to Section 4 for more comprehensive example and explanation about some of the above mentioned metamodel elements. In Section 4, Fig. 8 provides an illustrative example of the particular M2M transformation specification model.

Description of the properties of *M2MT Specification* metaclass is presented in Table 1.

Table 1
Description of the properties of *M2MT Specification* metaclass.

Property of M2MT Specification	Description
sourceConceptType	The property defines SMCT, an instance of which will be selected by a user to perform a D&D action (let us denote that instance "I_{SMCT}"). For a valid transformation specification this property cannot be empty.
targetConceptType	The property defines TMCT. It also acts as a D&D action trigger to launch the M2M transformation itself. When an I_{SMCT} is dragged and dropped onto an instance of TMCT defined in *targetConceptType* (let us denote that instance "I_{TMCT}"), the M2M transformation is invoked. When the property is not set, it is assumed that the I_{SMCT} is dragged and dropped into the diagram itself (the type of the diagram is defined in *targetDiagram* – see below) – this is quite common case for transformations augmented with transformation patterns.
propertyActionResult	It defines a certain property of TMCT defined in *targetConceptType*. When the property *propertyActionResult* is not empty, dragging I_{SMCT} and dropping it onto I_{TMCT} will assign I_{SMCT} as a new *value* to the defined property of I_{TMCT}. The property may also have an empty value.
appendMode	The property is of Boolean type and complements *propertyActionResult*. If the value is set to "*true*", then each time the specified D&D action is performed onto I_{TMCT}, new value is stacked to the pre-existing values of the specified property of the I_{TMCT}. If the value is set to "*false*", then the new value will overwrite all the existing values.
relationActionResult	The property is non-mandatory. When the property is not empty, the result of the D&D action is a new *relationship* between I_{SMCT} and I_{TMCT}. The property *relationActionResult* defines a *type* of a relationship that will be created. The property may also have an empty value.
targetDiagram	Defines the types of target diagrams, onto which a D&D action can be performed in order to invoke the transformation. The property also works as a guard to invoke only particular transformations. This is due to the fact that I_{TMCT} (specified by *targetConceptType*) may be deployed in a number of diagrams of different types within a single model, but not all of those diagrams may be valid candidates to perform that particular transformation.
mergeConcepts	The property is of Boolean type. Using D&D-based partial M2M transformations it is often the case, when newly created model concepts coincide with the pre-existing concepts of the model. Setting the property *mergeConcepts* to "*true*" will assure that those coinciding concepts will be merged. If the property is set to "*false*" then the names of newly created coinciding concepts will be altered automatically to avoid model validation errors. Concepts' similarity is decided by comparing their names.
integrationActionResult	The property is used to specify types of traceability relationships, which may be created among the source model concepts and corresponding target model concepts for model integration management purposes. Model integration management system was developed as a separate solution (plug-in), which falls out of the scope of this paper and, therefore, will not be discussed any further.
representationText	The property presents a short description of the particular M2M transformation.

For more explicit understanding of the main properties of M2M transformation specification, let us present few examples based on the transformation usage scenarios presented in Section 2.4.

An example of using the 1st case of the 1st scenario was the assignment of resource "Clerk" to a lane set in BPMN process diagram. In the specification of this particular transformation, we set certain values to the following properties:

- *propertyActionResult* = 'represents' – in BPMN model, property *represents* of concept type *LaneSet* binds a lane set with a resource;
- *sourceConceptType* = 'Resource' – here we specify concept type of a draggable model element for the transformation;
- *targetConceptType = "UseCase"* – here we specify concept type of a model element, onto which the draggable model element is dragged;
- *targetDiagram* = 'UseCase Diagram' – again, values set for this property define types of diagrams, in which this particular transformation can be triggered.

For the illustration of the 2nd case of the 1st scenario, we considered that actor "Clerk" was not associated with use case "Assign Special Carrier" in the use case diagram of our introductory example (Fig. 1). Then we dragged and dropped the actor onto the use case; as a result, an association between these two use case model elements was created by the system. In this particular transformation, the creation of an association between two specific concept types (i.e. *Actor* and *UseCase*) is defined by setting the following properties in the specification:

- relationActionResult = 'Association';
- sourceConceptType = 'Actor';
- targetConceptType = 'UseCase';
- targetDiagram = 'UseCase Diagram'

In Section 2.4, for the illustration of the 1st case of the 2nd scenario we presented an example, where a user drags and drops task "Assign Special Carrier" from BPMN process model into UML use case diagram; this D&D action triggers the execution of certain transformation, which results in automatic creation of use case "Assign Special Carrier", which is then represented in the use case diagram. The specification of such transformation is also pretty straight forward – we define the type of draggable element (*sourceConceptType* = "Task"), and the type of the element we drag onto (*targetConceptType* = "UseCase"); additionally we specify use case diagram as the type of target diagram (*targetDiagram* = 'UseCase Diagram').

As it was already mentioned, the 2nd case of the 2nd scenario enacts the most complex transformations in our approach, because the specifications of these transformations are augmented with transformation patterns. The specification of an exemplary transformation is presented and explained separately in Section 4.

Note that a comprehensive formal specification of the developed M2M transformation language is out of scope in this paper, therefore, we will not elaborate on all the details and specific cases of this language any further. In Section 4, an illustrative example of the particular M2M transformation specification model is presented and explained in an informal manner – please, refer to that part of the paper for more explanatory details about the transformations metamodel presented in this section.

3. Implementation aspects of the approach

3.1. Implementation of M2M transformations metamodel in a CASE tool

The developed metamodel of model-based M2M transformations (Fig. 5) was implemented in a CASE tool MagicDraw. In this respect, the main contribution to the approach was the development of *M2M Transformations* profile.

The overall architecture of UML profiles supporting the solution is presented in Fig. 6 (here, the *M2M Transformations* pro-

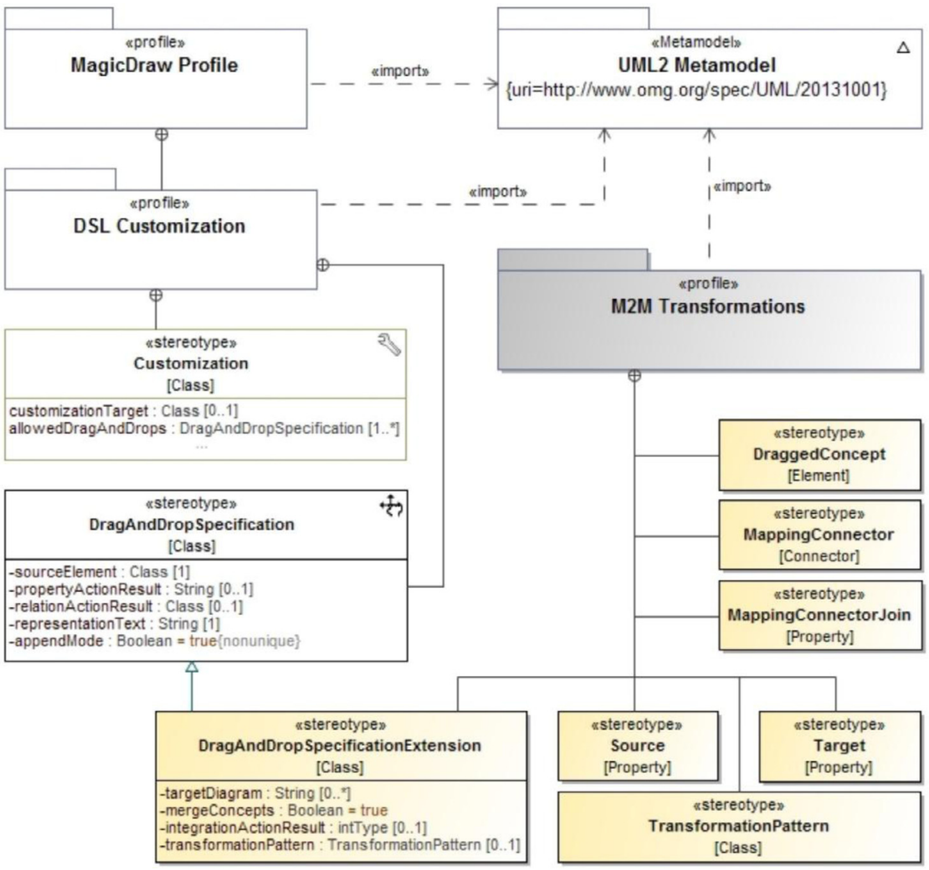
«profile»
MagicDraw Profile
«import»
«Metamodel»
UML2 Metamodel
{uri=http://www.omg.org/spec/UML/20131001}
«profile»
DSL Customization
«import»
«import»
«profile»
M2M Transformations
«stereotype»
Customization
[Class]
customizationTarget : Class [0..1]
allowedDragAndDrops : DragAndDropSpecification [1..*]
«stereotype»
DragAndDropSpecification
[Class]
-sourceElement : Class [1]
-propertyActionResult : String [0..1]
-relationActionResult : Class [0..1]
-representationText : String [1]
-appendMode : Boolean = true{nonunique}
«stereotype»
DraggedConcept
[Element]
«stereotype»
MappingConnector
[Connector]
«stereotype»
MappingConnectorJoin
[Property]
«stereotype»
DragAndDropSpecificationExtension
[Class]
-targetDiagram : String [0..*]
-mergeConcepts : Boolean = true
-integrationActionResult : intType [0..1]
-transformationPattern : TransformationPattern [0..1]
«stereotype»
Source
[Property]
«stereotype»
Target
[Property]
«stereotype»
TransformationPattern
[Class]

Fig. 6. Overall architecture of UML profiles supporting the developed solution.

file is painted dark; native MagicDraw profiles are left unpainted). Here, a subset of properties of the *M2MT Specification* metaclass (Fig. 5) is realized by the pre-existing «*DragAndDropSpecification*» stereotype from the native MagicDraw's *DSL Customization* profile; *TargetConceptType* property is realized by *customizationTarget* property of pre-existing «*Customization*» stereotype; other properties of the *M2MT Specification* metaclass are implemented in «*DragAndDropSpecificationExtension*» stereotype, which is a specialization of «*DragAndDropSpecification*».

A transformation pattern is realized by a custom stereotype «*TransformationPattern*». The pattern itself is realized as a structured class that has its internal structure, called *structure compartment*. The structure compartment of a transformation pattern holds two parts: source part (represented using *«Source»* property) and target part (represented using *«Target»* property). The source part holds one or more concept types and/or properties of concept types of a source model. The target part holds one or more concept types and/or properties of concept types of a target model. In each part, concept types are interconnected using *mapping connectors* (UML *Connectors* with «*MappingConnector*» stereotype); when designing a model of particular M2M transformation, these bindings are defined based on the underlying metamodel of the particular type of model (e.g. concept types representing a fragment of UML Class model are interconnected preserving structural rules of the UML metamodel). In its turn, mappings *between* source and target models are also defined using connectors – those bindings represent actual transformation mappings among source and target concept types. Such visual mapping structure allows one to develop various structures of transformation patterns; in a sense, such approach may be viewed as a visual pattern-based M2M transformation modeling language.

«*Customization*» is a specialized stereotype used to denote customization classes, which invoke the execution on D&D action specifications (once again, «*DragAndDropSpecification*» realizes most of the properties of *M2MT Specification* metaclass). Here, *allowedDragAndDrops* property specifies a set of transformation specifications, which are invoked for execution after a CASE tool user performs a D&D action on the specified target element (*customizationTarget*). With each activation of D&D action, only one transformation specification may be executed. This means that the user will be asked to choose one particular transformation at a time, if *allowedDragAndDrops* property specifies more than one transformation specification (item selection menu is used for this purpose).

Note that in general the syntactical validity of a transformation model is ensured by a CASE tool's model validation engine. In other words, if a CASE tool supports a certain level of enforcement of model validation, the syntactical invalid model cannot be created or at least syntactical errors are highlighted for correction. On the other hand, at this stage of development, our solution does not restrict a user from developing syntactically invalid *transformation patterns*. However, during the execution of such invalid transformation, a MagicDraw user will be provided with a system warning message stating that the action could not be executed as it would lead to the creation of an invalid model (such message is generated if the creation of the model fails). Also, some basic structural checks are performed by our solution itself while loading M2M transformation libraries to the project.

3.2. Implementation architecture

The developed solution for model-based partial M2M transformations is composed of six core components:

- CASE tool MagicDraw as the implementation platform for our approach. The CASE tool supports recent versions of various OMG modeling standards, such as UML 2.5, BPMN 2.0. It also supports UML profiles, as well as provides facilities for development of customizations and domain specific languages (DSLs). Custom plug-ins can be developed in Java language using internal Open API.
- *DSL engine* of the MagicDraw. The engine provides functionality to use domain specific profiles as customized extensions of UML for development of customized diagrams with user-defined toolbars, stereotyped elements, symbol styles, customization of element specifications and representations, application of custom real-time semantic rules (No Magic, Inc., 2015). All DSL customizations in the MagicDraw are stored as UML models.
- *Drag and Drop Transformations* plug-in for MagicDraw, which provides the development environment for model-driven development of M2M transformations. It also extends the processing capabilities of the previously described extended D&D action specifications. More details on this plug-in are provided further below in this section.
- *MagicDraw Model Profiles* package holding native MagicDraw profiles, which are necessary for the development and execution of D&D M2M transformations.
- *Custom transformation Libraries* component, which stores custom built M2M transformation specifications (models) for specific modeling languages, e.g. UML, BPMN, SBVR, UML-BPMN, BPMN-SBVR, UML-SBVR. More details on this plug-in are provided further below in this section.
- *VEPSEM model integration* plug-in for MagicDraw UML (optional), which provides model integration capability. This plugin developed in R&D project VEPSEM,[3] in which the authors of this paper also took part.

The *D&D M2M Transformations* plug-in is developed using our *M2M Transformations Core* API, which implements all the necessary abstract M2M transformation processing and could be used to develop other similar M2M transformation solution for other compatible CASE tools. Such development solution enables its reuse and integration of the developed transformation engine, while at the same time providing flexibility and extendibility essential for custom implementations.

Two relevant UML profiles are created and distributed with the *D&D M2M Transformations* plugin:

- *M2M Transformations* profile, which contains the transformation metamodel and other related elements, such as *Customization* elements used by MagicDraw DSL engine to customize the representations of model elements.
- *Model Integration* profile, which contains UML elements and custom stereotypes used to provide the required model integration capability.

One of the core features of our approach is custom transformation libraries (*Custom Transformation Libraries* package in Fig. 7). These transformation libraries are developed for different modeling languages or their sets, and contain model-based partial M2M transformation specifications (models). The libraries can be developed and exchanged among MagicDraw users, provided they have installed the D&D M2M Transformations plugin in their MagicDraw tools. A MagicDraw project may use more than one transformation library at the same time, depending on particular needs of a user.

[3] Project VP1-3.1-ŠMM-10-V-02-008 „Integration of Business Processes and Business Rules on the Base of Business Semantics" (acronym: VEPSEM, duration: 2013-2015), partially funded by the European Social Fund (ESF).

Two additional native MagicDraw profiles (from *MagicDraw Model Profiles* package) are required to run our solution:

- *DSL Customization* profile (distributed together with MagicDraw), which is used by the transformation engine and provides a set of necessary stereotypes.
- *UML Metamodel with attributes* (can be installed as a resource using MagicDraw Plugin Manager), which is used to represent UML metaclasses together with their properties. This is mandatory for the development of transformation patterns.

4. Illustrative example of M2M transformation specification

Further, let us shortly discuss an illustrative model of M2M transformation specification (Fig. 8) describing two D&D action-based transformations from the running example presented in Section 2.1.

The model is composed of one customization class (tag 1), two D&D action specifications (tags 2 and 3), and transformation pattern (tag 4). Customization class lists two D&D action specifications (tags 2 and 3). The listed D&D action specification is to be executed by the DSL engine upon dragging certain source element *sourceElement* specified in that specification and dropping it onto target element *customizationTarget* specified in the customization class. In this particular example, customization class does not have any defined customization target element because the target is the target diagram itself (the target diagram is defined in *targetDiagram*). So, the customization class and the *first* listed D&D action specification (tag 2) defines that this particular transformation will be invoked upon dragging an instance of *Task* from the source model and dropping it into the Use Case Diagram. In its turn, the customization class and the *second* listed D&D action specification (tag 3) defines that this particular transformation will be invoked upon dragging an instance of *LaneSet* from the source model and dropping it into the Use Case Diagram.

Next, both D&D action specifications define the same transformation pattern *transformationPattern*, which means that upon the execution both transformations will refer to the same transformation pattern. In our example, the pattern defines: a source part describing BPMN Process Model pattern composed of a *Task* that is deployed in a *LaneSet*; a target part describing UML Use Case Model pattern composed of an *Actor* associated with a *UseCase*; mappings between certain inner parts of the source and target parts – the mapping define actual element-to-element transformations to be executed by the DSL; the connectors among inner elements of both source and target parts – these connectors preserve certain structural rules from the underlying BPMN and UML metamodels respectively.

So, upon dragging the task “Assign Special Carrier” (Fig. 1) from the process model and dropping it into the use case diagram, all instances of *LaneSet* associated with that task will be also selected for transformation; in our case, the lane set “Clerk” will be selected. This set of the source elements will be transformed into a set of interrelated target elements, i.e. a use case “Assign Special Carrier”, an actor “Clerk”, and an association relating those two concepts; text processing rule *{left(,0,)}*, defined in *concatOperator* property of the mapping connector, specifies that the name of an instance of *LaneSet* should not be assigned to an instance of *Association* in this transformation (i.e. name of the lane set is subtracted to a zero-length string, which is then assigned to the newly created association in the target use case model).

Following our example, the second D&D action specification will be invoked, when the lane set “Clerk” is dragged and dropped into the use case diagram. In this case, a set of *Task* instances, associated with the “*Clerk*”, will be selected (i.e. “Assign Special

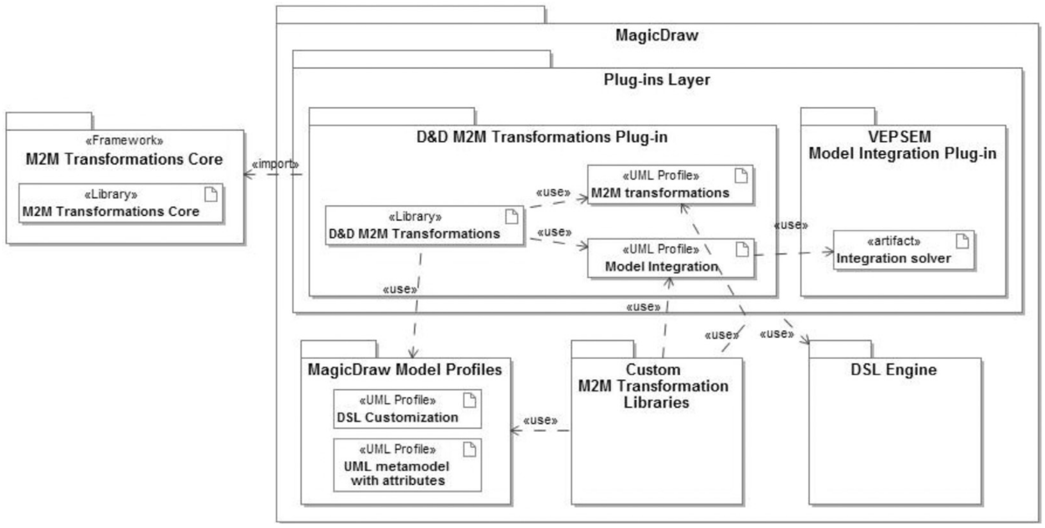

Fig. 7. Implementation architecture of the solution.

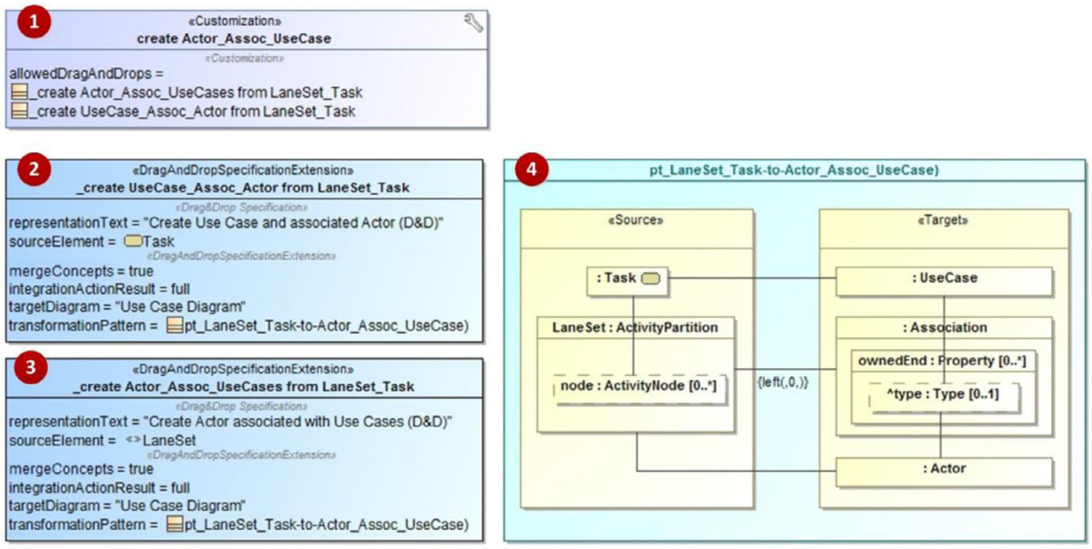
1
«Customization»
create Actor_Assoc_UseCase
«Customization»
allowedDragAndDrops =
_create Actor_Assoc_UseCases from LaneSet_Task
_create UseCase_Assoc_Actor from LaneSet_Task
2
«DragAndDropSpecificationExtension»
_create UseCase_Assoc_Actor from LaneSet_Task
«Drag&Drop Specification»
representationText = "Create Use Case and associated Actor (D&D)"
sourceElement = Task
«DragAndDropSpecificationExtension»
mergeConcepts = true
integrationActionResult = full
targetDiagram = "Use Case Diagram"
transformationPattern = pt_LaneSet_Task-to-Actor_Assoc_UseCase)
3
«DragAndDropSpecificationExtension»
_create Actor_Assoc_UseCases from LaneSet_Task
«Drag&Drop Specification»
representationText = "Create Actor associated with Use Cases (D&D)"
sourceElement = LaneSet
«DragAndDropSpecificationExtension»
mergeConcepts = true
integrationActionResult = full
targetDiagram = "Use Case Diagram"
transformationPattern = pt_LaneSet_Task-to-Actor_Assoc_UseCase)
4
pt_LaneSet_Task-to-Actor_Assoc_UseCase)
«Source»
«Target»
: Task
: UseCase
LaneSet : ActivityPartition
node : ActivityNode [0..*]
{left(,0,)}
: Association
ownedEnd : Property [0..*]
^type : Type [0..1]
: Actor

Fig. 8. Example of two D&D action specifications using the same transformation pattern.

Carrier" and "Fill-in Post Label"). The set of selected source elements will be transformed into a set of interrelated target elements following the same transformation pattern, i.e. the created actor "Clerk" will be associated with the created use cases "Assign Special Carrier" and "Fill-in Post Label". In both D&D action specifications, the property *mergeConcepts* is set to "*true*", which means that if any newly created concept coincides with any pre-existing concept, those concepts will be merged. In our example, if both transformations were executed subsequently, the target use case model would be augmented with actor "Clerk", use case "Assign Special Carrier" and association between them after the execution of the first transformation, and the second transformation would augment the model (and the diagram) with only one new use case "Fill-in Post Label" and association relating that use case with the pre-existing actor "Clerk".

5. Experiment and evaluation

In order to verify the level of acceptance of the approach by both designers of partial M2M transformations and actual users of these transformations (i.e. modelers) and identify its potential in real-world applications, we conducted an experiment with a team of researchers working on R&D project VEPSEM (see the reference to this project in previous section). One of the deliverables of VEPSEM project was the development of *automatic* QVT-based full M2M transformations for the modeling standards UML, BPMN and SBVR; therefore, one can state that the researchers, who took part in our experiment, had certain level of experience not only in modeling, but also in developing and using M2M transformations. From the total of 24 project researchers, 6 senior and 9 junior researchers were invited to participate in the experiment (note that they did not participate or were involved in the development of this approach in any way). The group of selected researchers was invited to a spacious computer class and went through a short introductory course on how to use the developed solution of partial M2M transformations. After that, the group was divided into three teams, comprising of 2 senior and 3 junior researchers each, according to their specialization in VEPSEM activities:

– UML – BPMN transformations. Each member in the first team was given the same personal task – to develop a set of given transformations between UML and BPMN models (UML $\leftrightarrow$ BPMN), and also, transformations among the elements of the

same model (UML ↔ UML, and BPMN ↔ BPMN). Transformations that needed to be developed were given as set of formal transformation rules each of them specifying a set of input and output elements, and transformation mappings. UML models comprised of UML Use Case, State Machine, and Class diagrams; BPMN model comprised of BPMN Process diagrams.
- UML – SBVR transformations. The task for the second team was similar to the first team's, with only difference in selected models. This team worked with UML ↔ SBVR, UML ↔ UML, and SBVR ↔ SBVR transformations. UML models comprised of UML Use Case, State Machine, and Class diagrams. As for the SBVR model, a specialized UML profile for SBVR was developed during the VEPSEM project phase; this profile enabled one to develop visual Fact diagrams representing SBVR Business Vocabulary (Skersys et al., 2015b).
- BPMN – SBVR transformations. Accordingly, members of the third team worked with SBVR ↔ SBVR, BPMN ↔ BPMN, and BPMN ↔ SBVR transformations.

Each member in all teams was asked to develop 24 D&D M2M transformations in total: 3 transformations for each pair of models (both directions), which did not include transformation patterns, and 3 transformations for each pair of models (both directions), which did include transformation patterns. For instance, each member of the first team was asked to develop 3 transformations without patterns for each of the following model pairs: UML→UML, BPMN→BPMN, UML→BPMN, BPMN→UML, and then 3 transformations with patterns for each of the mention model pairs. No specific instructions were given about the order, in which these transformations were to be developed, i.e. each member could choose his own preferred course of actions. A comfortable time frame of 4 h was given to perform the task.

Note that in the experiment we considered the roles of both the designer of transformations and actual users of these transformations (i.e. modelers), because while developing his transformations, each participant could instantly try them out on his own experimental models to see if the developed transformations are correct.

After the experiment, each *developed* transformation was rated as:

- *Successful* – a transformation specification was developed successfully and fully corresponded to the given description of the transformation;
- *Partially successful* – a transformation specification was developed, although it did not correspond completely with the description (e.g. some important properties where not specified), or the specification was semantically valid, yet could not be successfully processed by the developed transformation engine.
- *Failed* – an invalid transformation or no transformation at all was developed.

Results of the transformations development experiment are shown in Table 2.

The results indicate that the model-based M2M transformation language and the transformation development tool were accepted quite well. Overall, senior researchers performed their tasks slightly better than junior researchers – this could be explained by their objectively higher experience levels in modeling and especially, metamodeling. As expected, the development of transformations without patterns was not a difficult task for the most – the number of successfully developed transformations was at least two thirds of the number that each member was supposed to create. The number of failed transformation specifications was relatively low; both, senior and junior performed at approximately the same level. Please, keep in mind that the experiment took place within the pre-set time frame of 4 h; considering this, one could safely assume that the results could have been even better provided more time was given to accomplish this mentally strenuous task.

We also tried to identify possible correlations between the pairs of modeling languages and the attained results, assuming that transformations with BPMN models could be more challenging compared to UML models. However, correlation analysis did not show any significant relations across the board.

Right after the transformations development experiment was completed, each researcher was asked to fill in a survey form to evaluate their personal experiences with both, the developed transformation modeling language and the development tool. Each characteristic in the presented survey was evaluated in the discrete scale from 1 to 5. The list of characteristics was adopted from Rose et al. (2012), where a survey of similar tools was presented; yet, we added two more characteristics: *relevance* indicating the actual need for the developed solution, and *expressiveness* of the transformation language. Thus, the following list of characteristics was presented in the survey:

- *Understandability* – indicates a user's opinion about how well he managed to understand the language.
- *Correctness* – perceived semantical correctness of the M2M transformations language.
- *Conciseness* – user's opinion whether the language is not overloaded with redundant modeling concepts.
- *Expressiveness* (or *completeness*) – defines whether the language has enough modeling concepts to specify partial M2M transformations of various levels of complexity (within a given range of defined transformation modeling capabilities).
- *Usability of D&D Actions* – assumption on how well D&D actions-based transformations should perform during actual modeling activities.
- *Usability of Development Tool* – the level of usability of the model-driven design of transformation specification using the provided development tool (environment). Note that is criteria is different from the usability of D&D actions, as it is evaluated from the perspective of transformations designer, rather than modeler, who would actually use those designed transformations in his modeling routine.
- *Maturity* – the level of maturity of the presented version of the developed M2M transformations solution as a whole.
- *Relevance* – user's opinion about how relevant and important the proposed approach and its implementation is to the field of model-driven development.

Results of the survey are presented in Fig. 9 as the distribution of evaluation scores and mean values of all surveyed characteristics. These results show that the proposed M2M transformation language and the implemented solution were accepted positively. Most of the respondents felt the whole transformation development approach could be grasped quite easily. Nevertheless, the level of usability of the implemented solution was rated quite unevenly averaging to a relatively low evaluation score. Using the developed M2M transformations was easy and intuitive; however, using the development tool was somewhat complicated – arguably, the latter fact could have influenced lower solution maturity scores as well. Either way, the latter scores were not surprising, considering the fact that the implemented solution is still in its prototyping phase and is being continuously developed. At this stage of development, we did not consider the actual usability testing of the solution as the main objective of the experiment.

The understandability of the transformations language itself was rated somewhat lower than expected. However, this could be explained by the lack of documentation, tutorials and too little time spent to actually learn the language. Note, that the users were not experienced using similar tools. Therefore, this evaluation was not intended to compare this tool with other existing tools at this

Table 2
The results of the M2M transformations *development* experiment.

Team	Researcher	Qualification	Without transformation pattern			With transformation pattern		
			Successful	Partially successful	Failed	Successful	Partially successful	Failed
1st team	R6	Senior	11	0	1	9	1	2
	R7	Senior	9	1	2	9	1	2
	R8	Junior	8	1	3	7	4	1
	R9	Junior	7	3	2	8	4	0
	R10	Junior	11	1	0	6	5	1
2nd team	R11	Senior	10	2	0	8	1	3
	R12	Senior	12	0	0	8	0	4
	R13	Junior	8	1	3	7	1	4
	R14	Junior	9	2	1	8	3	1
	R15	Junior	9	2	1	6	4	2
3rd team	R1	Senior	10	1	1	9	2	1
	R2	Senior	10	2	0	8	4	0
	R3	Junior	10	0	2	8	1	3
	R4	Junior	11	1	0	6	0	6
	R5	Junior	11	0	1	7	0	5

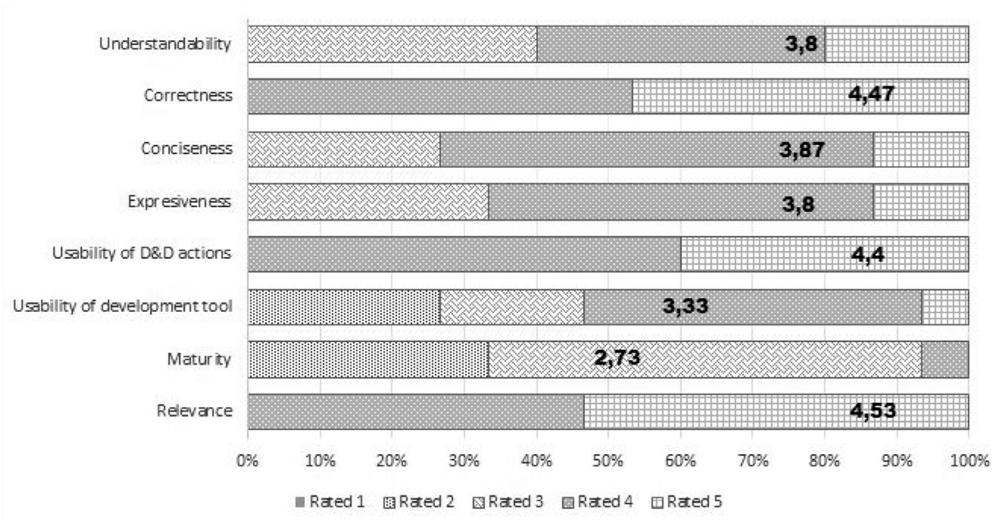

Fig. 9. Survey results.

stage, hence, the given score represents only experience with this single tool and language and is not focused on similarity-based evaluation.

The experiment also highlighted few technical issues related to certain implementation aspects of the solution. One of the issues, which the users found very annoying, was the abundance of duplicate *Connector* type elements in the experimental transformation patterns – those duplicates were being created after performing *copy/paste* operations with transformation patterns. Although this could have been avoided by properly developing transformation patterns, the users found the proposed proper course of actions quite tedious. We presume that this and other minor usability glitches could have had rather significant influence on the solution's usability evaluation scores.

6. Threats to validity

Although the experiment provided optimistic results, there are several threats to validity of those results that should be taken into consideration. In the assessment of the results of our experiment, we considered the following categories of threats to validity (Runeson et al., 2012): (a) threats to construct validity, (b) threats to internal validity, and (c) threats to external validity. Each group of these threats is discussed with regard to our experiment in the next sub-sections.

6.1. Threats to construct validity

Threats to construct validity mainly deal with possible research design problems, such as bias in experimental design, researcher expectations, and different understanding of concepts between researchers and participants of experiment. To mitigate construct validity threats, we ensured that all the participants of the experiment were competent with given modeling standards, worked on the same project dealing with M2M transformations, and were experienced users of the CASE tool MagicDraw. This allowed us to safely assume that all those people and us, designers of the experiment, talk the same language and have the same interpretation of relevant concepts. As it was mentioned in Section 5, all participants were given an introduction about various aspects of the approach; during the experiment itself, no interaction between participants and researchers or the participants themselves was allowed. Considering all the above mentioned preconditions and actions taken during the experiment, we assume that the identified threats to construct validity of the experiment results are minimal.

6.2. Threats to internal validity

Threats to internal validity consider the existence of internal factors, which may directly or indirectly affect experiment results. These factors may be affected by physical or psychological changes in participants during experiment, specialized knowledge and experience of participants acquired before experiment, ordering of subjects during the experimentation, etc.

For our experiment, we provided our participants with a relatively natural working environment (i.e. a spacious computer class with well-equipped working places, natural ventilation and water supply). The limited time frame for the experiment could be considered as, probably, one of the most realistic threats to internal validity; however, the given 4 h were considered as a comfortable time frame for the task. Nevertheless, one could assume that the results of the experiment could have been even better provided more time was given to accomplish the task. The awareness that the results of the experiment would be published, thus affecting the overall assessment of the developed approach, might also put unneeded psychological pressure onto participants; therefore, we did not disclose this fact to the class (even though many could presume it, taking into consideration all the circumstances). Due to our limited resources and realistic expectations towards the experiment itself, we did not deal with all the physical and psychological aspects, which are very specific to each individual (e.g. fatigue, boredom, ability to concentrate, motivation), in our experiment, even though they could have some effect on the performance results of certain participants.

Let us remind that at the beginning of the experiment each participant was provided with a certain set of formal transformation rules (textual specifications), and that those rules were of different level of complexity. Taking this into account, the order, in which those rules were to be approached, might also impact the overall result of each participant. To mitigate this threat, the participants were allowed to design their transformations in the order of their own preference.

Another potential threat to validity is the fact that some of the participants could have acquired certain specialized knowledge about the approach before the experiment, thus gaining certain advantage before other participants. Indeed, any participant had a natural opportunity to get some insights about the approach from one early research paper (Skersys et al., 2014), which was published before the start of the experiment. Other than that, the implemented solution and all related documentation were not accessible to any participant before the experiment.

6.3. Threats to external validity

External validity is related to generalizing things based on the results of an experiment. While our experimental findings can be generalized to a broader set of individuals, one should also consider the actual threats to such generalizations.

Arguably, the actual application of M2M transformation development techniques still remains a “prerogative” of academia researchers rather than industry practitioners – this is mainly due to the fact that practical tools for the development of M2M transformations are very few, moreover, they are not supported by commercial CASE tools. Even though the proposed solution aims to minimize this gap, our current experiment cannot provide the objective proof for such claim.

Another threat that should be taken into consideration is tool-dependency. Even though we claim that, besides MagicDraw, the approach could be implemented in other CASE tools as well, the experiment was run only with experienced MagicDraw users. Therefore, currently, it would be reasonable to make any kind of generalization only within the boundaries of the MagicDraw community. Yet once again, we claim that the conceptual design of the approach as well as its modular implementation architecture provides good premises to tailor the solution to other modern CASE tools, thus extending this generalization to include other CASE tools’ communities.

While our experiment considered users as both transformation designers and actual users of these transformations, a more extensive research on various features of the approach (e.g., usability) would also require experimentation with groups of users with different levels of knowledge and skills in modeling, model-driven transformations development, etc. The current experiment was targeted at the group of experienced users.

7. Related work

While M2M transformations and various contributions to this domain continuously remain a popular subject of discussion in numerous resources, research on model-driven development and application of visual model-based M2M transformations is still not very extensive. The Related Work section presents an overview of the analysis results covering this particular subset of M2M transformations, which are the most relevant to our research.

Visual model-based transformation languages rely on different graph transformation approaches. First introduced in Schürr (1994), Triple Graph Grammar (TGG) is a fundamental technique for numerous transformation techniques. It provides a formal framework to declaratively define model transformations and represent them as sets of left-hand-side (LHS) and right-hand-side (RHS) elements in forms of graphs; connections among source and target elements are also defined. The internal adequacy between LHS and RHS elements is specified in form of correspondence model, which defines the actual internal mapping logic. While there are some apparent similarities between principles of TGG and our approach, the latter does not use correspondence model – the mappings are defined directly as connections between corresponding source and target elements.

Under the direct influence of TGG technique, several well-known visual M2M transformation modeling and execution frameworks were introduced and extensively developed. Biermann et al. (2006) introduced an approach, which specified transformation rules based on Ecore metaelements with certain pre-conditions to define structural constraints. Later, Anjorin et al. (2011) described eMoflon framework for visual transformation modeling, which combined EMF and Eclipse technologies with a generic model transformation engine to support both graph transformations and TGG. This solution supports modeling of both unidirectional and bidirectional transformations in a form of story diagrams, as well as metamodeling aspect (Hildebrandt et al., 2013). CASE tool Enterprise Architect is used as a front-end implementation of eMoflon architecture – this marks one of the rare cases when a researched M2M transformations approach is defined outside the Eclipse framework and implemented within a commercially successful CASE tool. Moreover, the research also references the application of UML profiles as modular extensions to support other modeling languages – this feature closely resonates with the solution presented in this paper. However, the transformations in eMoflon are executed outside Enterprise Architect, using EMF2Fujaba and TGG Compiler, which is seen as a drawback from the perspective of our approach. EMorF (Klassen and Wagner, 2012) is another Eclipse EMF-based framework for the development of model-based M2M transformations, which supports TGG technique. It is implemented as an advanced tool providing means for visual modeling, including visual editor, palette and different views. The tool also provides model querying and traceability features. According to Klassen and Wagner (2012), EMorF also supports interactive and context sensitive rules execution, which supposedly is a similar feature supported by our M2M transformations approach as well; however, at the time of writing this paper, we were not able to practically verify the level of this similarity. TGG is also one of the most researched alternatives to OMG standard QVT (OMG, 2011); it was shown that transformation mappings between TGG and QVT-Core/QVT-Relations were possible to implement in TGG engine (Königs, 2008; Greenyer and Kindler, 2010).

Besides the approaches based on TGG, there exists a number of approaches, which proposed different solutions for the development of model-based M2M transformations. VMT approach (Sendall et al., 2003) introduced a simple visual declarative language for specification, composition and reuse of model transformation rules to select, create, modify and remove model elements; more advanced transformations could be additionally specified using OCL constraints. UMLX (Willink, 2003) was another early attempt to define transformations using UML class diagrams. It supported cardinality, preservation and removal operations, OCL constraints for more complex transformation rules, as well as transformation inheritance and evolution/traceability features. GReAT approach (Agrawal et al., 2006) combined OCL-enriched UML class diagrams to specify the domains of transformations and dataflow-like explicit sequencing of transformation rules. MOLA language (Kalnins et al., 2005) is an imperative approach providing a structured flowchart-based graphical modeling language. It also provides support for loop elements to enable modeling of iterative transformations. VIATRA (Varro and Balogh, 2007) introduced an advanced declarative multilevel transformation language based on abstract state machines; the approach combined visual rules and pattern-based paradigms to perform graph transformations. Its next generation, VIATRA2 (later, renamed to VIATRA again), provided an advanced reactive event-driven transformation engine with its own graph-based language, rule-based design framework and querying engine, which could be reused in different scenarios (Eclipse Foundation, 2016b). Even though this project is still supported, at the time of writing this paper, we could not find any actual implementations on VIATRA framework, which could be compared to our approach. From the point of view of our solution, Henshin framework (Arendt et al., 2010) seems quite relevant. Here, transformations are visually depicted as transformation rules with LHS and RHS elements; UML metamodel concepts are used for the specification of transformation models. Transformation rules support delete, preserve (for source elements), create, preserve (for target elements), and forbid (constraint) operations, which can be executed in parallel. Yet, LHS and RHS elements do not have clear visual representations in different building blocks of transformation model, which, when compared with our solution, might not appear so intuitive and convenient for M2M transformation designers. Another drawback of Henshin framework is no support for model traceability at the elements level (Biehl, 2010), which is seen by many as a crucial feature of any modern model-driven development approach.

UML profiling is one of the core requirements for any M2M transformation approach used in any modern MDA CASE tool. In a CASE tool, UML profiles are used to implement additional modeling languages, including such OMG standards as BPMN (OMG, 2014), SoaML (OMG, 2012), SBVR (OMG, 2013; Skersys et al., 2015b) or any other custom-made domain specific language (No Magic, Inc., 2016). Advanced modeling tools, such as MagicDraw or Enterprise Architect, provide the required capabilities for the development of UML profiles. One of the applications of UML profiles for M2M transformations was presented in van Gorp et al. (2009). According to van Gorp et al., their approach could increase the interoperability, reuse, alignment, integration and portability among different transformation tools, including the ones overviewed earlier in this section (e.g., eMoflon, GReAT, MOLA). Positioned among other UML/OCL based metamodeling tools and extensions,

For more discussion on different M2M transformation approaches (including text-based M2M languages), one could refer to Czarnecki and Helsen (2003) and Mens and van Gorp (2006); some practical implementation aspects of model-based M2M transformations are examined and discussed in the survey by Rose et al. (2012). Hilde-brandt et al. (2013) and Leblebici et al. (2014) also provide extensive formal and technical comparison of several TGG-based frameworks.

8. Conclusions and future work

Model-based M2M transformations are an important topic to be explored in the domain of model-driven system development. Nevertheless, the approaches of model-driven development of model-based M2M transformations have not yet been discussed extensively in academic community.

In this paper, we present an alternative approach and implementation of user-interacted partial M2M transformations based on drag-and-drop actions. One of the key features of the approach is the model-driven development and customization of M2M transformations, which increases the flexibility, extensibility and usability features of this capability when used in a CASE tool environment. This approach comes with relatively simple and intuitive visual transformation modeling language, which, while being conceptually simple, can be used to design powerful transformations composed of tuples of related/contained model elements. In terms of practical application, the complexity of such transformations is restricted only by certain limitations of the current implementation.

Another distinctive feature of our solution is that it can be applied to any graphical modeling language that is based on UML metamodel. The solution has been already experimented with four modeling languages: UML, BPMN, SoaML and SBVR. CASE tool MagicDraw supports UML profiles for BPMN (OMG, 2014) and SoaML (OMG, 2012); also, UML profile for SBVR was introduced in Skersys et al. (2015b).

Besides flexibility and extensibility features, the solution also possesses characteristic of reuse, which comes in a form of specialized M2M transformation libraries that can be exchanged among different modeling projects and also directly reused in the development of new transformation libraries. The already developed transformation libraries can be further extended adding new transformations. Also, even though current solution works as an extension (plug-in) to the CASE tool MagicDraw, it was built on a generic framework, which can be utilized to develop other internal CASE tool-specific implementation components on heterogeneous environments.

The experiment and survey results also confirmed the relevance and acceptance of the solution among modeling practitioners, both transformation developers and actual users of those transformations. The results of the experiment and survey provided certain suggestions regarding potential areas of improvement of our solution. Near future plans involve further improvements of M2M transformations metamodel and DSL engine itself to bring even more flexibility and usability features to the transformations development and application. Feedback from the respondents has also provided some insights towards further improvement of the advanced text processing features. It also highlighted the need for more advanced constructs to develop more sophisticated transformations. Introducing additional aspects of so called ecological validity to the settings of the next phase of experimenting could provide even more in-depth feedback about our solution's performance in real-life situations.

Another interesting direction of our research would be extending the approach to support *full* M2M transformations – premises for this look quite promising. Such new solution could serve as an alternative to the existing developments of full M2M transformations supported by CASE tools.

References

Agrawal, A., Karsai, G., Neema, S., Shi, F., Vizhany, A., 2006. The design of a language for model transformations. Softw. Syst. Model. 5, 261–288.

Anjorin, A., Lauder, M., Patzina, S., Schürr, A., 2011. eMoflon : leveraging EMF and professional CASE tools. Tagungsband der INFORMATIK 2011, vol. 192.

Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G., 2010. Henshin: advanced concepts and tools for in-place EMF model transformations. In: Model Driven Engineering Languages and Systems, vol. 6394, pp. 121–135.

Biehl M. Literature Study on Model Transformations. Technical Report ISRN/KTH/MMK, Royal Institute of Technology (2010).

Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E., 2006. Graphical definition of in-place transformations in the eclipse modeling framework. In: Model Driven Engineering Languages and Systems, vol. 4199, pp. 425–439.

Czarnecki, K., Helsen, S., 2003. Classification of model transformation approaches. In: Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, vol. 45, pp. 1–17.

Greenyer J., Kindler E., Comparing relational model transformation technologies: implementing query / view / transformation with triple graph grammars, (2010), pp. 21–46.

Hildebrandt, S., Lambers, L., Giese, H., Rieke, J., Greenyer, J., Schäfer, W., Lauder, M., Anjorin, A., Schürr, A., 2013. A survey of triple graph grammar tools. Electron. Commun. EASST 57.

Kalnins, A., Barzdins, J., Celms, E., 2005. The model transformation language MOLA. In: Model Driven Architecture, vol. 3599, pp. 62–76.

Klassen, L., Wagner, R., 2012. EMorF - a tool for model transformations. Electron. Commun. EASST 54.

Königs, A, 2008. Model Integration and Transformation - A Triple Graph Grammar-based QVT Implementation PhD thesis. echnische Universität, Darmstadt.

Leblebici, E., Anjorin, A., Schürr, A., Hildebrandt, S., Rieke, J., Greenyer, J., 2014. A comparison of incremental triple graph grammar tools. Electron. Commun. EASST 67.

Mens, T., Van Gorp, P., 2006. A taxonomy of model transformation. Electron Notes Theor. Comput. Sci. 152, 125–142.

No Magic, Inc. MagicDraw – architecture made simple. www.nomagic.com (2016).

No Magic, Inc.: UML profiling and DSL. User Guide, v 18.1 (2015).

Object Management Group (OMG). Meta object facility (MOF) 2.0 Query/View/Transformation, v1.1, OMG. www.omg.org/spec/QVT/1.1/ (2011).

Object Management Group (OMG). Model-driven architecture – MDA guide. OMG spec. v. 1.0.1. www.omg.org/cgi-bin/doc?omg/03-06-01 (2003).

Object Management Group (OMG): Semantics of business vocabulary and business rules (SBVR) v.1.2, OMG Doc. No.: formal/2013-11-04 (2013).

Object Management Group (OMG). Service oriented architecture modeling language (SoaML). OMG spec. v. 1.0.1. www.omg.org/spec/SoaML/ (2012).

Object Management Group (OMG). UML profile for BPMN processes. OMG spec. v.1.0. http://www.omg.org/spec/BPMNProfile/1.0/ (2014).

Rose, L.M., Herrmannsdoerfer, M., Mazanek, S., Van Gorp, P., Buchwald, S., Horn, T., Kalnina, E., Lano, K., Schätz, B., Wimmer, M., 2012. Graph and model transformation tools for model migration. Softw. Syst. Model. 13, 323–359.

Runeson, P., Höst, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons.

Schürr, A., 1994. Specification of graph translators with triple graph grammars. In: Proceeding of WG'94 Workshop on Graph-Theoretic Concepts in Computer Science, pp. 151–163.

Sendall S., Perrouin G., Guelfi N., Biberstein O. Supporting Model-to-Model Transformations: The VMT approach. Technical Report, 2003.

Skersys, T., Pavalkis, S., Lagzdinyte-Budnike, I., 2014. Model-driven approach and implementation of partial model-to-model transformations in a CASE tool. In: Proceedings of Information and Software Technologies: 20th International Conference, ICIST 2014, Druskininkai, Lithuania, October 9-10, pp. 260–271.

Skersys, T., Pavalkis, S., Nemuraite, L., 2015a. Implementing semantically rich business vocabularies in CASE tools. In: Theodore, E.S., Charalambos, T. (Eds.), AIP Conference Proceedings: Proceedings of the International Conference on Numerical Analysis and Applied Mathematics (ICNAAM-2014), 22–28 September 2014, Rhodes, Greece, Vol. 1648. AIP Publishing, Melville, NY, pp. 1–4. ISSN 0094-243XArticle 310002.

Skersys, T., Danenas, P., Pavalkis, S., 2015b. Model-driven development of partial Model-to-Model transformations in a CASE tool. In: Theodore, E.S. (Ed.), AIP Conference Proceedings: Proceedings of the International Conference on Numerical Analysis and Applied Mathematics (ICNAAM-2015), 23–29 September 2015. Rhodes, Greece. AIP Publishing, Melville, NY.

Sparx Systems. Enterprise architect – UML modeling and lifecycle tool suite. www.sparxsystems.com.au (2016).

The Eclipse Foundation. ATL – a model transformation technology. www.eclipse.org/atl/ (2016a).

The Eclipse Foundation. Viatra project. http://www.eclipse.org/viatra/ (2016b).

Van Gorp, P., Keller, A., Janssens, D., 2009. Transformation language integration based on profiles and higher order transformations. In: Software Language Engineering, vol. 5452, pp. 208–226.

Varro, D., Balogh, A., 2007. The VIATRA2 transformation framework model transformation by graph transformation. Sci. Comput. Program 68, 187–207.

Willink, E.D., 2003. UMLX: a graphical transformation language for MDA. In: Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, pp. 13–24.

Tomas Skersys is a scientific researcher at the Center of Information Systems Design Technologies and an Associate Professor at the Department of Information Systems (Kaunas University of Technology). His research interests and practical experience cover various aspects of business process management and model-driven development of information systems. On these topics, he published several articles in high rated academic journals and in a number of international conferences. He is also a co-editor of three books of international conferences published by Springer Verlag.

Paulius Danenas is a junior scientific researcher at the Center of Information Systems Design Technologies, Kaunas University of Technology in Lithuania. He obtained a PhD degree in Informatics from Vilnius University in 2013. His research interests cover various topics in software engineering and model-driven development, as well as artificial intelligence, machine learning, business intelligence and decision support systems (particularly for business and financial domains). He has published several papers in high rated academic journals and in a number of international conferences.

Rimantas Butleris is a Director of the Center of Information Systems Design Technologies and a full-time Professor at the Department of Information Systems (Kaunas University of Technology). His main area of research is requirements engineering. In his career, he was a co-author of more than seventy research papers and also participated in more than fifteen national and international research projects; in many of those he was a leading researcher. Up to the year 2013, Rimantas Butleris was a long-time program chair and co-chair of the International Conference on Information and Software Technologies (ICIST).