An Overview of Selected Visual M2M Transformation Languages

Vladimir Dimitrieski*, Ivan Luković*, Slavica Aleksić*, Milan Čeliković*, Gordana Milosavljević*
* University of Novi Sad/Faculty of Technical Sciences, 21000 Novi Sad, Serbia
{dimitrieski, ivan, slavica, milancel, grist}@uns.ac.rs

***Abstract*—Although many model transformation languages currently exist, only a small number of them has a visual concrete syntax. As a good visual syntax usually allows better communication of ideas and easier understanding of specifications than textual syntax, we feel that visual transformation languages are unjustifiably neglected. In this paper we give an overview of most popular visual model transformation languages: AGG, AToM3, VMTS, Henshin, and QVT.**

I. INTRODUCTION

Model transformations are essential part of model driven software engineering (MDSE) [1]. This is due to the fact that models are in no way static entities. Many operations, such as refactoring, semantic mapping, and translation, are specified in a form of model transformations, thus defining model's dynamic behavior. As a dynamic model behavior represents an important part of MDSE, it has been discussed in many research papers. In these papers model transformations have been thoroughly defined and classified. However, to the best of our knowledge, only a small number of such papers are focused on transformation languages' visual syntax.

Whether to use visual or textual concrete syntax is an issue present from the beginning of MDSE. In the hands of an expert, textual syntax represents the means to specify models in a fast and precise way. At the same time, it allows easier versioning and usage of existing tools. On the other hand, such specification may be too verbose and long for humans to easily process. Additionally, such verbose specifications are often difficult to understand by inexperienced users of the modeling language, which can lead to a poor communication of ideas. Therefore, the principles of good visualization could be applied, as a good graphical notation may allow people that are unfamiliar with the textual syntax to easily recognize all ideas in behind the formal specifications [2].

Among transformation language nonfunctional requirements presented in [3], four requirements are directly related to languages' concrete syntax. Authors argue that the *understandability* of a transformation specification is greater for a language that uses a visual syntax than the one based on the textual one, since humans as visual beings are able to process images easier than text. Thus, *visualization* is another nonfunctional requirement that favors visual syntax. Furthermore, *verbosity* and *conciseness* are also in favor of good visual languages. Visual languages that are at a right level of verbosity and conciseness keep a diagram easy to understand and explain.

In a survey of Italian IT industry [4], only a small number of companies that are applying MDSE principles use model transformations in their project. In the light of aforementioned nonfunctional requirements, we feel that currently used tools rely too much on textual syntax. This often slows down the whole modeling process, as only a small number of modelers are proficient in these languages. Furthermore, they cannot easily communicate with their colleagues that are inexperienced in the transformation languages.

A good visual language would allow easier communication and more understandable specification. Hence, we present an overview of visual languages in order to provide modelers with useful details that may help them to select the most appropriate language for their problem domain. In this paper we present currently used visual transformation languages: AGG, AToM3, VMTS, Henshin, and QVT. In each of these languages, we have implemented a sample transformation from the class to the relational model. Although we have implemented the whole transformation, in this paper we present only the rule that transforms a non-abstract class to a table, by analyzing the following two questions: "*how matching patterns are specified?*" and "*how constraints are specified?*" By this, we may conclude about the level of satisfaction of the aforementioned nonfunctional requirements.

The paper is structured as follows. In Section 2, we present related work. In Section 3 we present the notions of graph and hybrid transformation approaches. In Section 4, we give an overview of some of the commonly used visual transformation languages with the focus on their visual syntax. Section 5 gives conclusions.

II. RELATED WORK

To the best of our knowledge, there are few papers that compare visual transformation languages. However, none of them focuses on visual syntax.

Taentzer et al. [5] present a comparison of four different graph-transformation approaches based on a common sample transformation. They are more focused on the expressiveness of these graph transformation languages, instead of their visual syntax. On the other hand, Lengyel et al. in [6] present their graph transformation tool VMTS with the focus on a specification of constraints. They offer a comparison of VMTS to other graph transformation languages from a point of constraint language's expressiveness. In addition to previous comparisons, Vara reviewed most of the modern model transformation languages [7]. Similarly, Czarnetcky et al. [8] presented an overview of transformation approaches and currently used model transformation languages. In contrast to all these works, we are focused here on the language visual syntax.

III. Preliminaries on Model Transformations Approaches

Model transformations can be classified as Model-to-Model (M2M) or Model-to-Text (M2T) transformations. M2M transformations are the ones responsible for transforming one model into another. M2T transformations transform models to a textual file. M2T transformations are most commonly used in software industry in the process of code generation [9]. In this paper we focus on M2M transformations only. Currently, a number of M2M transformation paradigms exist. Czarnecki and Helsen [8] distinguish among seven of them: *direct-manipulation*, *structure-driven*, *operational, template-based*, *relational*, *graph-transformation-based,* and *hybrid* approaches.

In this paper we present graph and hybrid approaches. We consider graph approach as it is intended to be used in a visual way. Graph-transformation-based approaches are usually using visual concrete syntax to represent most important transformation concepts. Graph transformation consists of applying an iterative rule to a graph. Each application of a rule transforms a graph by replacing its part by a new graph. For this purpose, each rule contains a left-hand side (LHS) and a right-hand side (RHS). The application of a rule to a graph replaces each occurrence of the LHS in a graph by the RHS. Additionally, the notions of applicability conditions (ACs) and negative applicability conditions (NACs) are used to limit the application of these replacements. Together, they can be considered as means of constraint specification. Therefore, such languages usually have three separate modeling parts in their visual representations: RHS, LHS, and NACs/ACs. In the reminder of this subsection, visual graph-transformation languages are described.

Hybrid approaches may combine all other approaches in a single transformation specification. These approaches can be combined as separate components or, in a more fine-grained fashion, at the level of individual rules. We present them here mainly because they are a de facto standard to specify transforming models. Of all hybrid approaches, only Query/View/Transformation (QVT) [10] offers a graphical syntax in support for transformation specifications.

IV. Overview of Visual Model Transformation Approaches

To analyze various visual transformation approaches, we have specified a transformation of a class diagram to a relational model. For each transformation approach, we show only a rule which specifies transformation of a non-abstract class to a table. Only non-abstract classes are to be transformed and a name of the table should be the same as the class name. Our selection of such example has been motivated mainly by its simplicity, since our goal is to overview just a visual syntax, without going into details about transformation specifications or underlying transformation mechanics.

As we need a class and relational meta-models to participate in the transformation, we have implemented them in each of presented environments. Figure 1 and Figure 2 depict these meta-models implemented in Ecore which is a part of Eclipse Modeling Project [11].

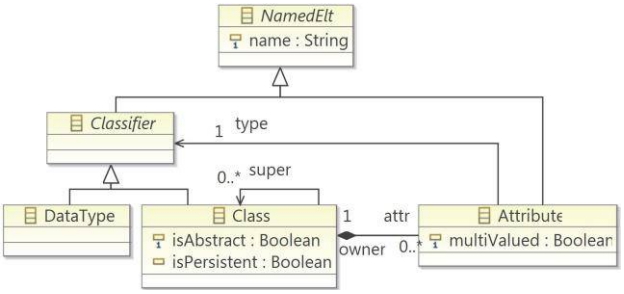

Figure 1 Simplified class meta-model

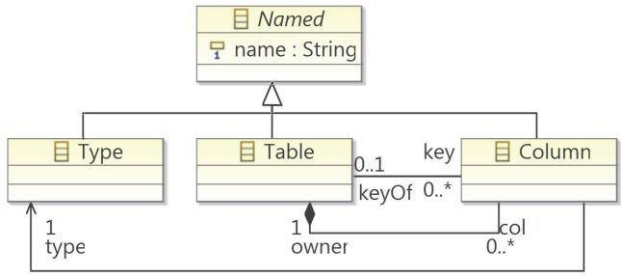

Figure 2 Simplified relational meta-model

In the following subsections we overview the following graph-based visual transformation languages: AGG, AToM3, VMTS, and Henshin. The only visual hybrid language presented is QVT. In the end, we present a summary of our observations.

A. AGG

AGG [12] is a tool environment for algebraic graph transformation systems. Graphs in AGG are defined by a type graph on different abstraction levels, and as such they may be attributed by any kind of Java objects. Graph transformations can be equipped with arbitrary computations on these Java objects described by Java expressions. The AGG environment consists of a graphical user interface comprising several visual editors, an interpreter, and a set of validation tools.

In AGG, two visually separated parts of a rule specification exist: LHS and RHS patterns. Before specifying any graphs, including aforementioned patterns, node and edge types must be specified. These are main building blocks of any graph and may be provided with different colors and shapes. This allows easier visualization and idea communication by modelers as they are able to emphasize important parts of their models, i.e. graphs. Node coloring is particularly helpful in case of big diagrams to differentiate between source and target modeling elements. Additionally, labeling elements with numbers allows modeler to denote whether a LHS element is deleted from a model or not.

Constraints are modeled in a form of ACs. AGG allows a specification of negative, positive, and general application conditions. All three of them are specified in a visual way. For that purpose dedicated part of the tool's workspace is designed for a specification of ACs. Nevertheless, the specification is performed in the same way as for patterns. Furthermore, in order to set an execution order of its rules, AGG allows rules to be put in layers that are executed sequentially. Unfortunately, this type of constraint could not be directly set in the diagram.

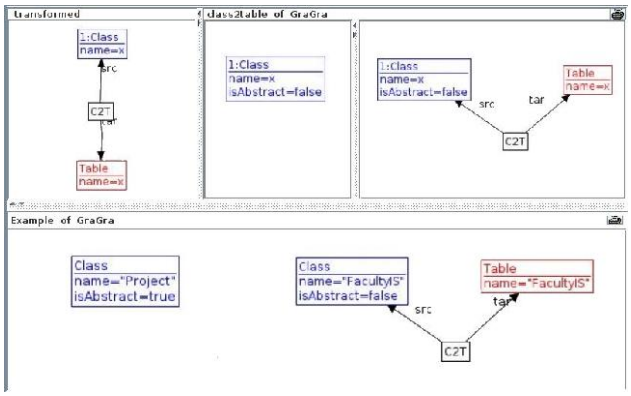

Figure 3 AGG class2table rule specification and example model

In Figure 3, an AGG rule specification is depicted. In the bottom part of the AGG editor, an example of a model is presented after an execution of the transformation. The transformation rule itself is presented in the top part of the figure. Rule specification comprises three visually separated compartments in which a modeler may specify AC, LHS, and RHS respectively. This visual separation, offers a great insight in a transformation execution as a modeler is able to see a rule specification while executing it step-by-step. In order to terminate the transformation from a class to a table, in this example we have specified NAC to be equal to the RHS. Thus, if a class is associated with a table by a C2T node, that class is not to be considered again. C2T node represents a kind of traceability links that help a modeler to trace which target element is created from which source element. Furthermore, values from source to target elements are passed by variables. In this example, we have passed a value of *name* attribute from a class to a table using a variable *x*. In order to preserve an element in a model, element should be labeled with a same number on both LHS and RHS. In the example presented in Figure 3, a class from the LHS would be present in the target model, as there is a class in the RHS with the same label number. If those numbers are different, original class is to be deleted and a new one is to be created.

B. AToM3

A Tool for Multi-formalism and Meta-Modeling (AToM3) [13] is one of the first tools providing an integrated meta-modeling environment. It allows a definition of meta-models, concrete syntax, and model transformations. Additionally, it allows simulation of user defined models' transformation in a step-by-step mode. Such simulation is of a big help to transformation developers, as it allows them to debug their transformations' execution. The abstract syntax induced by the meta-model provides definitions of all entities, their attributes, possible connections and constraints amongst them. As users of a modelling environment are more often working with concrete syntax of transformed models, all transformations are defined at the level of that concrete syntax.

All patterns, LHS, and RHS are specified using a concrete syntax of transformed models unlike other presented approaches where generic icons are used. This allows transformation developers to visualize their transformations and communicate their ideas in a more suitable way. The usage of concrete syntax in transformation specifications is unique among all presented tools, since all others provide specifying transformations using a generic abstract syntax instead of a concrete one. The usage of a concrete syntax is enabled by the AToM3's ability to import arbitrary formalisms, i.e. meta-models, in its pattern editor. Therefore, it allows a modeler to use the same concepts, buttons, and icons for transformation rule specification as the ones used for model specification. Furthermore, this concrete syntax approach is also very helpful while running simulations on models specified in the AToM3 environment. Therefore, each step of transformation is visualized and each change in the model is animated at the level of a concrete syntax.

In AToM3, generalized ACs may be used as well as pre-conditions and post-conditions to events. Constraints may be defined by either a semantic or visual way. For each component, a constraint on its attributes may be defined. AToM3 allows a modeler to define whether a LHS pattern element must have a specific value assigned to its property or any value is permitted for that property. Additionally, a global rule pre-condition and post-condition may be specified. They are specified in Python programming language and may be used to control the execution of the whole rule. For example, post-condition is often used to specify a rule termination condition based on the evaluation of all previously created elements. Actions may also be specified. They are also specified using Python, and they represent an imperative section of the rule. Whenever the rule is executed, the action code is run, thus allowing more control over a flow of transformation. All of the aforementioned constraints are not specified directly on a transformation diagram, but through several dialogs. This lowers the ability to see a whole transformation in one diagram.

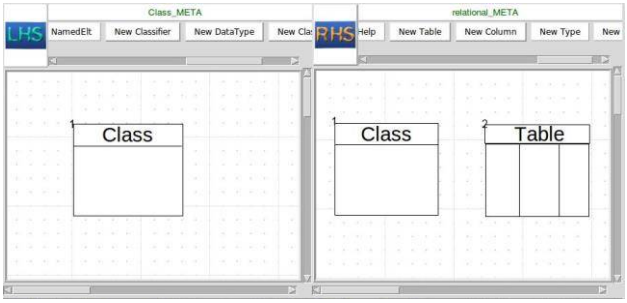

Figure 4 AToM3 class2table rule specification

Figure 4 depicts a transformation rule which adds one table to a model for each class that is specified. All icons in this rule are previously specified as concrete syntax icons for Class and Table meta-elements. Similarly to AGG, AToM3 uses layers to define the application order of the transformation rules. Labels with numbers are used to distinguish between elements that are present in both LHS and RHS. In this example, class element is present in both patterns and is labeled with number 1 in the top-left corner of Class element. It allows the environment to decide whether to delete element from LHS after the rule execution. The same number in our example means that a class remains in the model while new table instance is added.

C. VMTS

The Visual Modeling and Transformation System (VMTS) [14] is a graph-based, domain-specific modeling environment. The system provides a graphical interface for defining and transforming domain specific languages. VMTS allows visual definition of meta-models and generation of domain specific modeling languages. Additionally, it allows a definition of

model transformations using either a visual language or a textual, template-based language.

The visual specification comprises LHS and RHS connected together in a rule, where connection holds the definition of an operation that is executed. As it is a graph-based language, all patterns are specified using nodes and edges. Nodes are specified by selecting a meta-element that is to be used, as well as an operation over the element. Allowed operations are *delete*, *modify*, *create*, and *no process*. As the operation is selected, the node's visual representation changes color. Deleted elements are red, modified are gray, created are green, and not processed ones are white. This behavior is very similar to the one of AGG. The only difference is that in AGG arbitrary color may be assigned to an element, while in VMTS all colors are predefined. Furthermore, VTMS is the only presented environment that visually differentiates between various element relations. Relations of four types may be placed on a transformation diagram. *Matched relation* represents an edge of source or target model. It is depicted by a solid arrow. *Matched reference* represents a reference from one node to another and *replaced relation* is used when a relation needs to be replaced in a target model. These relations are represented with gray dashed arrow and yellow solid arrows. Similarly, *replaced reference* is used when a reference needs to be replaced and it is represented with a yellow dashed arrow.

After transformation rules are specified, transformation's control flow is also modeled. It is represented by a directed graph, the nodes of which are the *rule containers*. Rule containers represent a rewriting rule that is used for a transformation of source to target graph. Special visual syntax is used for a specification of the control flow. Each flow has a start and an end state. Each execution follows an execution flow comprised of rule containers connected with flow edges.

In contrast to AGG and AToM3, which use Java and Python languages, VMTS uses Object Constraint Language (OCL) [15] for constraint specifications. OCL offers more declarative constructs than other programming languages used for the same purpose. Constraints may be specified for each transformation's meta-element. Constraints are not placed directly on a transformation diagram. Instead, they are specified in the element's property editor. Hence, it is not always easy to comprehend a whole diagram without going into details of each element.

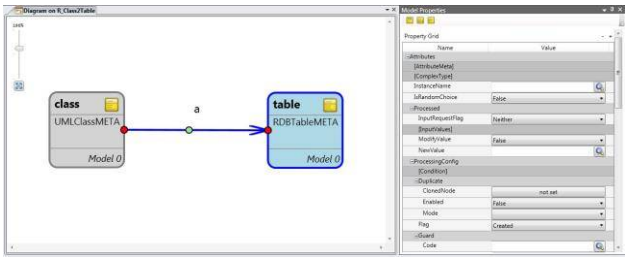

Figure 5 VMTS class2table rule specification

In Figure 5, a VMTS class to table transformation rule is depicted. Transformation editor is divided into two areas. On the left hand side, transformation is specified using visual symbols for nodes and edges. In this example, a class is being transformed to a table. As a table is created in a target model, it is represented with a blue color. The right area of the visual editor is a property view. It shows all specified constraints and other element properties. Figure 5 depicts a property view of the selected relation between the class and the table nodes.

Figure 6 VMTS transformation flow specification

Figure 6 depicts the transformation flow for our example. After the execution is initiated, *CreateTable* rule is executed. *CreateTable* is the only rule that is executed in this example. After it is successfully executed, transformation is terminated.

D. Henshin

Henshin [16] and its predecessor Transformation generation (Tiger) [17] are transformation languages and tools that best fit into graph-transformation approach. In contrast to other presented tools, they are developed as the Eclipse environment plugins. Their main goal is to support various kinds of EMF model modifications such as refactoring, introduction of design patterns and other modeling patterns. Therefore, they represent an in-place transformation approach, operating directly on EMF models. In contrast to Tiger, Henshin offers extended set of declarative language constructs. Tiger's basic concept of transformation rules is enriched by powerful application conditions and flexible attribute computations based on Java or JavaScript. The Henshin tool environment consists primarily of a fast transformation engine, several visual editors, and a state space. Since these transformation concepts are based on graph transformation concepts, it is possible to translate the rules to AGG, where they might be further analyzed.

Visual syntax is somewhat different from previously presented graph-transformation languages. It is developed in the Eclipse environment and utilizes Graphical Editing Framework's (GEF) look and feel. Similarly to VMTS, RHS, LHS, and ACs are all a part of the same rule specification. In order to alleviate some problems regarding readability of such diagrams, the tool uses colors and annotations to distinguish between different patterns. Elements that are created in a RHS only are annotated with *create* label and are colored in green, by default. Elements from LHS that are not deleted in target model are annotated with *preserve* label. Elements for deletion are annotated with *delete* label, and colored in red. The colors may also be manually set, allowing a modeler to further emphasize on some elements.

In addition to a rule specification, transformation unit is also visually specified. Transformation unit represents a collection of sequentially executed rules with a goal to transform a source to a target model. They are visually represented as linear execution flows of already specified rules. These flows are similar to the ones in VMTS language. Notable difference is that Henshin's flows are specified in the same diagram as the rest of the rules. This may be particularly useful in perceiving a big picture of a transformation.

Patterns may be annotated with the AC and NAC constructs. These conditions are either logical expressions or ACs which are extensions of the original graph structure. A rule may be applied to a host graph only if all application conditions are fulfilled. All ACs are shown in the same rule diagram with other transformation pattern elements. Application conditions

are colored in blue and distinguished from other elements by annotation labels, such as *forbid*.

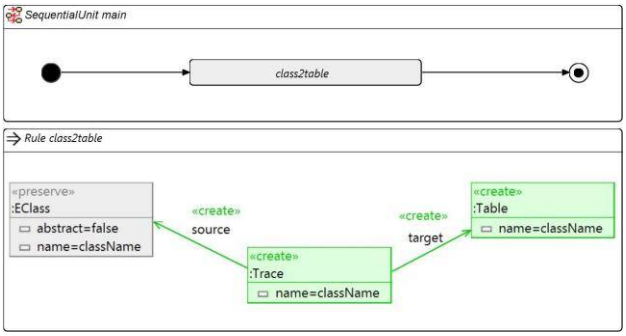

Figure 7 Henshin class2table rule and transformation unit specification

The bottom part of Figure 7 depicts a class2table rule specified in Henshin. A class is transformed into a table and a traceability link is created. The class is preserved in the target model, while the table is created and its name is set to correspond to the class name. Aforementioned node characteristics are specified with annotations above the node names. Passing an argument is done by variables in the same way as in AGG or AToM3. In this example, a variable named *className* is used for passing a value of a class name to a table name. For each table that is created from a class, a trace is created. It allows a modeler to see which tables are created from which class. This is very important in a case of very large number of classes in a source model as it allows a modeler to know which table is created from which class.

E. QVT

Query/View/Transformation (QVT) is a hybrid transformation language supported by the Object Management Group (OMG). QVT is of a hybrid declarative/imperative nature, with the declarative part being split into two-level architecture: QVT *Relational* and QVT *Core*. A user-friendly Relations meta-model and concrete syntax supports complex object pattern matching and object template creation. As diagrammatic notations have been a key factor in the success of their Unified Modeling Language (UML), OMG specified a visual syntax of QVT Relational to complement its textual syntax [10]. However, currently there are no QVT tools that allow a specification of transformations in a visual way. All visual representations are generated from textual specifications.

QVT notation is similar to the notation of UML class diagrams enriched with additional elements for transformation specification. For each meta-element specified in any pattern, its attributes are shown. Attribute values are also shown in a visual representation. If an attribute is assigned a variable instead of a raw value, the variable may be used in the opposite pattern to pass a value to another attribute.

OCL constraints may be specified by attaching a note to an object or pattern. Such note is shown within a diagram. However, the OLC expression is shown only after opening the note. Other constraints, like when and where clauses are also shown in the visual representation of a rule. *Where* and *when* boxes are shown at the bottom of a rule, if their respective clauses exist in its specification.

Figure 8 depicts the class to table transformation specification using the visual syntax of QVT Relational. Transformation is represented by a hexagon in the middle of the diagram. It divides the diagram into two separate parts, i.e. domains. A name of each domain is given on each side of the transformation symbol. In the example, LHS domain is named *uml* while RHS domain is named *rdbms*. Capital letters shown in the transformation depict whether the transformation in that direction is enforced or check-only.

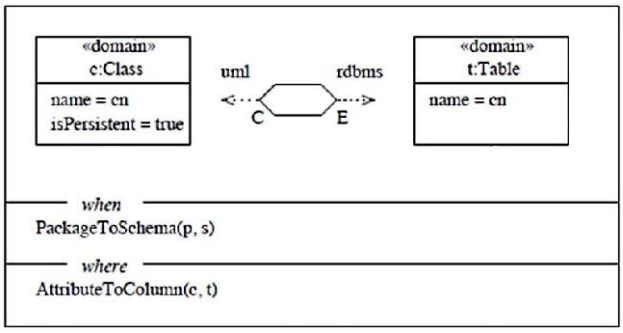

Figure 8 QVT class2table rule specification

When clause that specifies the conditions under which the relationship needs to hold is presented in the first box below the transformation rule. It specifies that this rule must hold when package to schema transformation is executed, as a schema must contain all tables that are previously created. *Where* clause that specifies the condition that must be satisfied by the model elements that are being related. In our example, before creating all of the tables, table's columns must be created from attributes. *Where* clause is specified in the second box below the rule specification.

F. Summary

The only presented language that uses concrete syntax of transformed models is AToM3. The usage of concrete syntax allows a modeler to use same concepts as the ones used while specifying models. Therefore, using concrete syntax makes the specification easier and transformations more understandable. Additionally, it allows very good visualization of transformation execution simulation. Other tools use generic icons to represent meta-elements participating in a transformation. In order to ease modeler's task while working with generic icons, AGG, Henshin, and VMTS employ different colors of such elements. In these tools, colors denote different operations and states of such elements. Henshin and AGG allow a modeler to choose arbitrary colors for their elements, while VMTS colors are predefined for each operation executed on some element. Additionally, AGG provides different shapes for elements. Therefore, a modeler may choose not only the color of a specific element but also its shape. This further improves diagram readability.

QVT's syntax is similar to other OMG languages, such as UML. Transformation characteristics are shown in the center of the diagram and are easily readable. Downside of its visual syntax is that QVT transformations may be difficult to read if there are many elements present in a diagram. They do not deploy any differentiation mechanisms like applying different colors or shapes for elements. Similarly, Henshin's diagrams may be overcrowded as it encourages so called multi-transformation rules. Multi-transformation rules allow one rule to represent several different rules in one compact way. This may lead to the mainly overcrowded rule specifications. In

contrast to Henshin, QVT improves readability by a total separation of LHS and RHS.

Henshin and VMTS allow explicit specifications of the transformation execution flows. As VMTS tool has a specific editor for this purpose, a modeler must constantly switch editors in order to specify execution flow and transformation rules. On the other hand, Henshin allows a specification of both rules and flows in the same editor. This greatly improves the speed of specifying transformations, as well as their understandability.

All languages provide constraints specification over transformation elements. However, only QVT and VMTS use OCL as a language for constraint specifications. This is of a great importance, as OCL is a de facto standard provided by a variety of tools for the constraint specification. In all presented languages, constraints may be specified on each element but not all tools provide specifications directly on the diagram. AGG and AToM3 only support labeling of elements in order to differentiate between element instances. On the other hand, VMTS tool shows a property view of the selected element, but it is overcrowded with various properties which make it hard to read. On the other hand, AGG allows an explicit visual modeling of its applicability condition. This is very useful for transformation debugging as a modeler is able to execute transformation step-by-step having the whole rule specification visible including specified constraints.

V. CONCLUSIONS

In this paper we give a brief overview of five transformation approaches: AGG, AToM3, VMTS, Henshin, and QVT. We focus our attention to their visual syntax as it directly influences understandability, visualization, verbosity, and conciseness of transformation approaches. We have asked two questions: "*how matching patterns are specified?*" and "*how constraints are specified?*" These questions helped us to focus on the two most important aspects when choosing a transformation approach and appropriate tooling. These two aspects are: (i) specification of patterns containing transformed elements and (ii) specification of pattern constraints.

From the literature overview, we conclude that there is a lack of information about visual syntaxes of transformation approaches. Therefore, a modeler has to try each transformation language before a selection of the most suitable one. Our aim is to assist a modeler in this process by providing an overview of common approaches. Based on our analysis, we conclude that Henshin provide the most complete visual syntax of all presented approaches. It is concise and easily understandable. Henshin's editors are the most compact ones allowing a definition of both rules and execution flows in a single diagram. Additionally, Henshin is an Eclipse plug-in. It is an advantage, as Eclipse is a widely used tool by the modeling community.

In our future work, we plan to perform a larger comparative study that includes other visual transformation languages such as PROGRESS, GReAT, and VIATRA. Additionally, we have notified a lack of hybrid transformation approaches with visual syntax as they often have only a textual syntax to be used by a modeler. This may lead to hardly understandable specifications, especially for people inexperienced in these languages. Therefore, we plan to develop a visual transformation language that would allow modelers to easily specify transformations at a platform independent (PIM) level. Afterwards, such specifications are to be transformed into platform specific specifications in a hybrid transformation language. Therefore, such visual PIM-level transformation language would provide a single visual syntax for transformation specification. Additionally, platform specific specification would allow transformations to be executed on a desired platform.

ACKNOWLEDGMENT

The research presented in this paper was supported by Ministry of Education and Science of Republic of Serbia, Grant III-44010: Intelligent Systems for Software Product Development and Business Support based on Models.

REFERENCES

[1] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *Softw. IEEE*, vol. 20, no. 5, pp. 42–45, 2003.

[2] I. Dejanovic, M. Tumbas, G. Milosavljevic, and B. Perisic, "Comparison of Textual and Visual Notations of DOMMLite Domain-Specific Language.," in *ADBIS (Local Proceedings)*, 2010, pp. 131–136.

[3] S. Nalchigar, R. Salay, and M. Chechik, "Towards a Catalog of Non-Functional Requirements for Model Transformations," in *Proceedings of the Second Workshop on the Analysis of Model Transformations (AMT 2013)*, Miami, FL, USA, 2013.

[4] F. Tomassetti, M. Torchiano, A. Tiso, F. Ricca, and G. Reggio, "Maturity of software modelling and model driven engineering: a survey in the italian industry," in *Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference on*, 2012, pp. 91–100.

[5] G. Taentzer, K. Ehrig, E. Guerra, J. De Lara, L. Lengyel, T. Levendovszky, U. Prange, D. Varró, and S. Varró-Gyapay, "Model transformation by graph transformation: A comparative study," in *Proc. Workshop Model Transformation in Practice, Montego Bay, Jamaica*, 2005.

[6] L. Lengyel, T. Levendovszky, G. Mezei, and H. Charaf, "Model transformation with a visual control flow language," *Int. J. Comput. Sci. IJCS*, vol. 1, no. 1, pp. 45–53, 2006.

[7] J. M. Vara Mesa, "M2DAT: a Technical Solution for Model-Driven Development of Web Information Systems," 2009.

[8] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Syst. J.*, vol. 45, no. 3, pp. 621–645, 2006.

[9] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. San Rafael: Morgan & Claypool Publishers, 2012.

[10] "Query, View, and Transformation (QVT)." [Online]. Available: http://www.omg.org/spec/QVT/1.1/PDF/. [Accessed: 17-Dec-2013].

[11] "Eclipse Modeling Project (EMP)." [Online]. Available: http://projects.eclipse.org/projects/modeling. [Accessed: 04-Jan-2014].

[12] G. Taentzer, "AGG: A tool environment for algebraic graph transformation," in *Applications of Graph Transformations with Industrial Relevance*, Springer, 2000, pp. 481–488.

[13] J. de Lara and H. Vangheluwe, "AToM3: A Tool for Multi-formalism and Meta-modelling," in *Fundamental Approaches to Software Engineering*, R.-D. Kutsche and H. Weber, Eds. Springer Berlin Heidelberg, 2002, pp. 174–188.

[14] T. Levendovszky, L. Lengyel, G. Mezei, and H. Charaf, "A systematic approach to metamodeling environments and model transformation systems in VMTS," *Electron. Notes Theor. Comput. Sci.*, vol. 127, no. 1, pp. 65–75, 2005.

[15] "Object Constraint Language (OCL)." [Online]. Available: http://www.omg.org/spec/OCL/2.3.1/PDF/. [Accessed: 30-Dec-2013].

[16] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: advanced concepts and tools for in-place EMF model transformations," in *Model Driven Engineering Languages and Systems*, Springer, 2010, pp. 121–135.

[17] E. Biermann, C. Ermel, L. Lambers, U. Prange, O. Runge, and G. Taentzer, "Introduction to AGG and EMF Tiger by modeling a conference scheduling system," *Int. J. Softw. Tools Technol. Transf.*, vol. 12, no. 3–4, pp. 245–261, 2010.