Designing a Map of Mappings Visualization of QVT Relations using Petri-Nets

Ali Fatolahi, Stéphane S. Somé and Timothy C. Lethbridge

School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada

Abstract. QVT Relations are a popular standard for formalizing model-driven transformations. For the most part, QVT relations are expressed using the existing textual standard. A number of graphical approaches have been suggested for presenting QVT relations. Most of the existing approaches are either concerned with the graphical presentation of individual relations or deal with validation and execution. As a result, in many cases the visualization of relations does not lead to a better understanding of the set of relations as a whole and does not facilitate the design process of transformations. In this paper, we employ basic Petri-Nets for presenting a more comprehensible picture of QVT relations and a more flexible technique for designing such transformations. Our approach is not concerned with the accuracy or correctness of the graphical representation but with the understandability of the set of relations as a transformation chain.

1 Introduction

Model-driven development (MDD) suggests a paradigm shift in software engineering, where the ultimate goal is to remove the coding process. As an alternative to coding, MDD suggests automated transformations of models and code generation.The development of software-intensive systems could be seen as a repetitive process of creating models and transforming them to other models or code. Writing transformations for a model-driven method is an evolving field, which could be improved with more formal techniques and tools. Visualizing model-driven transformations is seen as an interesting part of such efforts that could help the design, verification and understandability of transformations. In this paper, a visualization technique is proposed for modeling the chain of transformations expressed using QVT relations [3] based on basic Petri-Nets.

Transformations as the chaining feature of model-driven development [2] play a pivotal role in implementing model-driven tools and techniques. Several formal languages for recording transformations exist. The OMG QVT standard is one of the most common techniques in this area. QVT relations are one of the options for specifying transformations based on QVT. A number of tools and techniques have been suggested in recent years addressing model-driven transformations using QVT relations; e.g. MediniQVT [12] and Wimmer et al. [4]. With large sets of QVT relations, however, it becomes difficult to obtain a high-level understanding of transformation paths without delving into the details of the relations. This hinders the maintainability and evolution of QVT relations. We propose a Petri-Net based technique that aims at providing a clearer

Fatolahi A., S. SomÃĩ S. and C. Lethbridge T.
Designing a Map of Mappings Visualization of QVT Relations using Petri-Nets.
DOI: 10.5220/0003025500350045
In *Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development (ICEIS 2010)*, page
ISBN: 978-989-8425-10-2

view of a set of QVT relations, which would result in more understandable mappings, with less details required to be considered.

A number of graphical approaches have been proposed to visualize QVT transformations in the literature such as the ones by Wimmer et al. [6] and Willink [13]. As expected from a graphical model, such approaches aim to help both users and developers of QVT relations to comprehend the transformations. However, in most cases the graphical approaches become as verbose as the textual code because they pay so much attention to all the details that appear in the textual format. Regardless of the original motivations of creating the visualization techniques, the difficulty of understanding their graphical representation of the transformations counteracts their formal strength. Our effort is to present a technique for visualizing QVT relations in such way that the designer can choose which details to hide, leading to greater understandability. Consequently, this would allow the QVT relations developer to be more concerned with the mappings of the main components instead of the relation details. Also, the user of the transformation could use our Petri-Nets in order to follow the rules of generating the target model from the source one and hence to find out the potential source of failures in the resultant model.

Our goal is to present a map of transformation mappings. As Freeman [19] suggests, a map serves as a medium of communication and its effectiveness counts on the quality of the displayed features. Also, Freeman [19] mentions that a map should render the information of interest clearly, rapidly, and without ambiguity. The incorporation of many details in a map of QVT relations means that there is less sparse space around every arbitrary piece of displayed information, which results in maps that are harder to read and difficulty following specific paths. Also as O'Conner and Robertson [20] put it when discussing geographical maps, there are choices to be made regarding how a map is oriented, and whether to include or disregard certain details; these decisions depend on the maps specific use or target. In the same way, the details to be inserted in a map of QVT relations and those to be hidden are a matter of the designer's choice. The benefit of our approach is that it enables the designer to select features to be shown and those to be hidden in different layers so that relations could be portrayed in different levels of detail to serve the variety of purposes for which the designers may need the map.

Petri-Nets are recognized as a formal language for describing discrete systems. They are well suited for the specification of systems composed of different states and involving concurrency. A variety of tools, techniques and extensions to Petri-Nets have been provided in the past. In this paper, basic Petri-Nets are used to model sets of QVT relations. Considering every relation as a transformation from a source model to a target model, the semantics of Petri-Nets are well suited to describe sets of QVT relations. Places carry tokens, which can be understood as pieces of information; in our approach, elements, domains and variables are represented by places. Transitions are decision-making points, so are in our approach are used to visualize guards, constrains and domain patterns. The resultant net will steer the input tokens throughout different places ending with the desired target elements. Such a net would provide a global map of the chain of transformations that is suitable for understanding the transformations and would facilitate the design and maintenance of transformations. Using our mapping of QVT relations and Petri-Nets, one may design the whole transformation using Petri-

Nets from scratch or obtain the equivalent Petri-Net of a pre-written transformation for comprehension purposes.

The rest of this paper is organized as follows. Section 2 presents an introduction to QVT relations. Section 3 defines the mapping between Petri-Nets and QVT relations. Section 4 provides detailed examples of transforming QVT relations to Petri-Nets and vice versa. Section 5 reviews the related work. Finally, Section 6 concludes this paper.

2 QVT Relations

Relations are one of the languages proposed by the OMG as part of the QVT standard for model transformations. Code Sample 1 defines the syntax of a relation R that transforms *typed_model_1* to *typed_model_2*, referred to as the source and target domains respectively. The when clause indicates a pre-condition that must hold before the matching of the source and the target. Respectively, the *where* clause states a post-condition. The patterns used for defining source and target domains act as preconditions as well; that is, in order to run a transformation from a to b, a pre-requirement would be a successful pattern matching of domain a as the source element within the source domain *domain_1*.

Code Sample 1 - QVT Relations Syntax [3]

```
Relation R {
  Var <R_variable_set>
  [checkonly | enforce] Domain:<typed_model_1>
    <domain_1_variable_set> {
    <domain_1_pattern> [<domain_1_condition>] }
  [checkonly | enforce]
  Domain:< typed_model_n><domain_n_variable_set> {
    <domain_n_pattern> [<domain_n_condition>] }
  [when <when_variable_set> <when_condition>]
  [where <where_condition>]
}
```

A QVT relation may be a top relation; that is a relation that the transformation starts with. Non-top relations can only be called within other relations as a part of their *when* or *where* sections. Code Sample 2 shows an example of *ClassToTable* relation from the QVT official document. *Code Sample 2 - Class to Table QVT relation [3]*

```
top relation ClassToTable {
  cn, prefix: String;
  checkonly domain uml c:Class {namespace=p:Package {},
                  kind='Persistent', name=cn};
  enforce domain rdbms t:Table {schema=s:Schema {},
         name=cn, column=cl:Column {
name=cn+'_tid', type='NUMBER'},
   key=k:Key{name=cn+'_pk',column=cl}
  };
  when { PackageToSchema(p, s);}
  where { prefix = '';AttributeToColumn(c, t, prefix);}
}
```

Code Sample 2 describes a relation that maps a UML class to a table of a relational database. The transformation proceeds from classes to tables . In order for the transformation to be valid, it is necessary that the *PackageToSchema* relation hold. This relation maps a package to a database schema, meaning that the table must belong to the schema generated as the result of mapping the owning package of the source class; such mapping is required to guarantee the uniqueness of the created tables. The object pattern defined for the domain *c* indicates that the transformation looks for classes within the source model that are *persistent* and have a name. The transformation will then map every single such class to database tables with the same name as source classes and a default column as the primary key. Afterwards, the transformation will call another relation to map the attributes of the class to columns of the table. Figure 1 presents a subset of a meta-model for QVT relations that are composed of the elements required to specify QVT relations.

3 Mapping Petri-Nets to QVT Relations

According to Peterson [1], a Petri-Net may be defined as a triple *N=(P,T,F)* where *P* is a set of places, *T* is a set of transitions, disjoint from P and *F* is a flow relation *F ¡= (P * T) U (T * P)* for the set of arcs. Places may be loaded with tokens that indicate the inputs through outgoing arcs and outputs from incoming ones. The number of incoming tokens must equal the number of outgoings tokens for a transition to proceed. Arcs occur from places to transitions and from transitions to other places. Once the sources of all incoming arcs to a transition are marked with tokens, the transition may fire, resulting in the marking of the target places of the outgoing transitions with generated tokens. In this paper, we use basic Petri-Nets in which, transitions are simple transitions and arcs are normal, which means they do not perform any special functions.

Figure 2 displays a simple meta-model that specifies basic Petri-Nets. In Figure 2, arcs can be outgoing or incoming from/to nodes that are either places or transitions. The constraint attached to the element *Arc* indicates that an arc may only connect a place to a transition or vice versa; in other words there would never be an arc from a place to a place or from a transition to another transition. This is formalized using the OMG's Object-Constraint Language (OCL). The attribute *tokens* of class *Arc* indicates the multiplicity of the arc; that is the number of tokens that transit through the arc at every step of the simulation.

In order to map QVT relations to Petri-Nets, a reformulation of transformations is made as follows: A transformation is defined as a set of relations. A relation is a tuple *R= (Pre, DP, Post). Pre* is a set of constraints *when1, when2, ...* used in the *When* section of the relation. In the same way, *Post* is a set of conditions *where1, where2, ...* utilized in the *Where* section of the relation. The set *DP* is composed of two subsets *SD :sd1, sd2, ...* and *TD:td1, td2, ...*, with *SD* a set of source domains and *TD* a set of target domains. Elements of Pre and Post can be relations or other well-formed constraints. A relation validated as a part of the *Pre* or *Post* sets, carries a set of parameters *P:p1, p2, ...*, where every p_i matches a variable of either sd_j or td_k. We would refer the matching domain as dp_i.

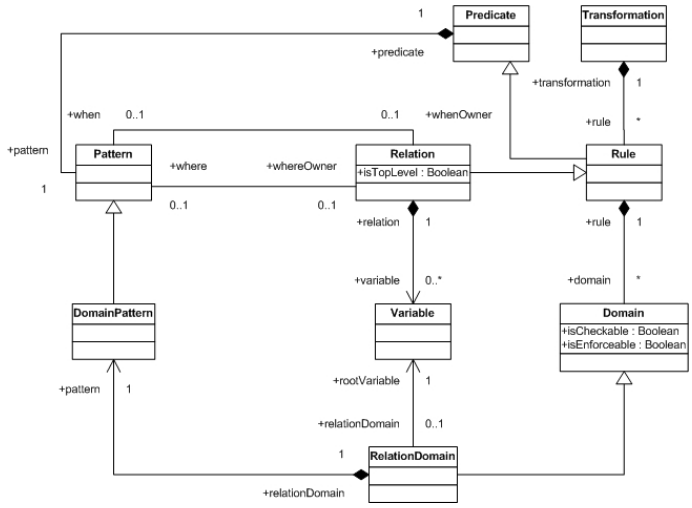

Fig. 1. A Simple Meta-Model for QVT relations obtained from [3].

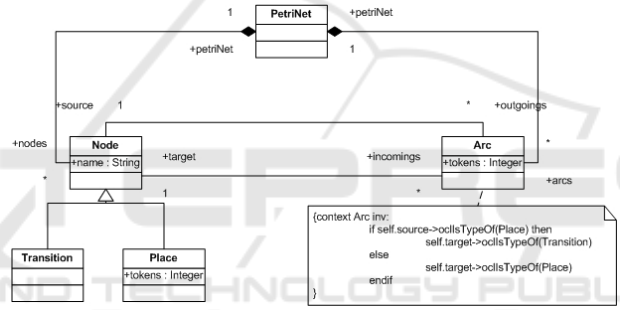

Fig. 2. Basic Petri-Nets Meta-Model.

Each of the elements of *Pre*, *DP* and *Post* may bind several variable or domains whose values are either consumed for the purpose of checking a constraint or produced as a result of an enforced expression. In our basic mapping, each element of *Pre*, *DP* and *Post* is mapped to a transition while variables/domains are mapped to places. Other considerations should be taken in order to ensure the set Pre is checked before the validation of the *DP*, while the set *Post* is to be checked after the validation of the *DP*. Validating each of those sets is represented by firing all the transitions conforming the members of the sets in the equivalent Petri-Net. In the rest of this section, we will first review an example and next we will formalize the mapping rules.

3.1 An Example

Figure 3 shows the *class_table* relation of the Code Sample 2. Places have been assigned to the domains *class* and *table*. Since the relation checks the validity of another relation, *package_schema*, the model verifies the validation of the relation *package_schema* prior

to finalizing the validation of the relation *class_table*. Therefore, two transitions are defined to transform the domains *class* and *table* to their constituent variables; the two transitions supply the domains *package* -of the domain pattern *class*- and *schema* -of the domain pattern *table*- to the relation *package_schema*.

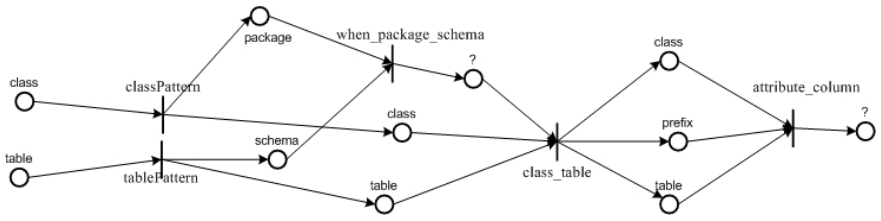

Fig. 3. Modeling the *class_table* Relation of the Code Sample 2.

Transitions with the postfix 'Pattern' generate output to places with the same name as the ones that supplied the inputs. For example, the transition *classPattern* has two places labeled *class* as both input and output. The pattern of every domain encompasses the variable representing the domain itself. Thus, this style of modeling is used, in order to assert that the *class* variable validated in the relation *class_table* is the same as the one of which, the variable *package* is chosen to be submitted to the when relation *package_schema*.

Assuming that we are not aware of the structure of other relations, the output from the transition *when_package_schema* is indicated with a question mark; in any case the model reflects the fact that the relation *class_table* would not run unless both domain patterns are checked as well as the relation *package_schema*. Once all three conditions are validated the where section is checked, which requires the validation of the two domains *class* and *table* as well as the variable *prefix*. Again the output from the relation *attribute_column* is only provided anonymously assuming we are not aware of its content. Figure 3 shows that our approach can model every relation individually.

3.2 Mapping Pre-conditions

Consider the relation *rel1*. Mapping every instance of a *when* relation $when_i$ that consumes the parameter p_i of the domain pattern dp_i would be as follows: Two places are added to fulfill two instances of dp_i. One is required to supply the required parameter submitted to $when_i$; the second instance is eventually used to validate the relation. A place is added to represent an instance of p_i to be sent to $when_i$. A fourth place is required to indicate the result of the validation of $when_i$; this place is an arbitrary place, which is only added to indicate that the relation is successfully added.

Three transitions are added. The first transition *dp_Pattern* is added to represent the checking of the domain pattern dp_i, this transition accepts an input from the first place of dpi and fires two outputs ending at the second instance of the dp_i as well as the place of the parameter p_i.The second transition simply supports the validation of wheni accepting an input from pi and firing an output to the arbitrarily added place, which will in turn be used by the third transition. The third transition represents the relation *rel1*.

Besides the arbitrarily added place, this last transition requires its own domain patterns as well, which is in this case dp_i .

The above mechanism ensures that the relation runs only if the when relation is satisfied, otherwise the required input from the transition representing wheni would not be supplied. Consequently, the validation of the domain pattern dp_i fails as expected.

3.3 Mapping the Domain Patterns

Mapping of the domain patterns is a straight forward process, which maps every sd_i and td_j to a place in the equivalent Petri-Net. There should also be a transition representing the relation. The net is complete with two sets of arcs. In the first set, one arc carries input from one of the sd_is to the transition. In the second set, outputs are carried from the transition to every td_js across an arc.

3.4 Mapping Post-conditions

In order to simulate the post-conditions $where_i$ that consume the parameter p_i of the domain pattern dp_i, a transition representing $where_i$, is added. Also, a transition that represents the pattern of dp_i is added. A place representing dp_i would supply the input to this transition. The output would end at the places that represent any p_i, which are in turn fed into the transition of $where_i$.

4 An Example

In this section, we review the usage of our Petri-Net based approach in the well-known example of *uml_rdbms* relations from [3]. Figure 4 presents a Petri-Net that is equivalent to the *uml_rdbms* transformations of the [3]. There are two start points in Figure 4, which relate to two top relations *class_table* and *association_foriegnKey*. The simulation process may start at either of the two relations. The arcs incoming to the transition *class_table* are assigned with the multiplicity of 2. This is an alternative way of applying a precondition. The relation *package_schema* is a precondition to the relation *class_table*. Both relations, however, carry the source and target domains, *class* and *table*. Thus, two tokens at both places must be present in order for the transition *class_table* to fire. The allocation of multiplicity is automatically specified when the mappings are performed from QVT relations to Petri-Nets. This becomes a designer's decision when the mappings are performed from Petri-Nets to QVT relations.

The transition *attribute_column* is followed by three transitions representing three postconditions in the relation *attribute_column*. These are *ComplexAttribute_Column*, *SuperAttribute_Column* and *PrimitiveAttribute_Column*. The reader will notice that at any time only one of these may fire. A random choice is assumed. A more precise design may impose conditions that decide exactly which transition is chosen based on the provided models so that the simulation of the net using the same input model would always trace through the same path. Our approach, is however, concerned with the path of the transformation and the understandability of the map of mappings.

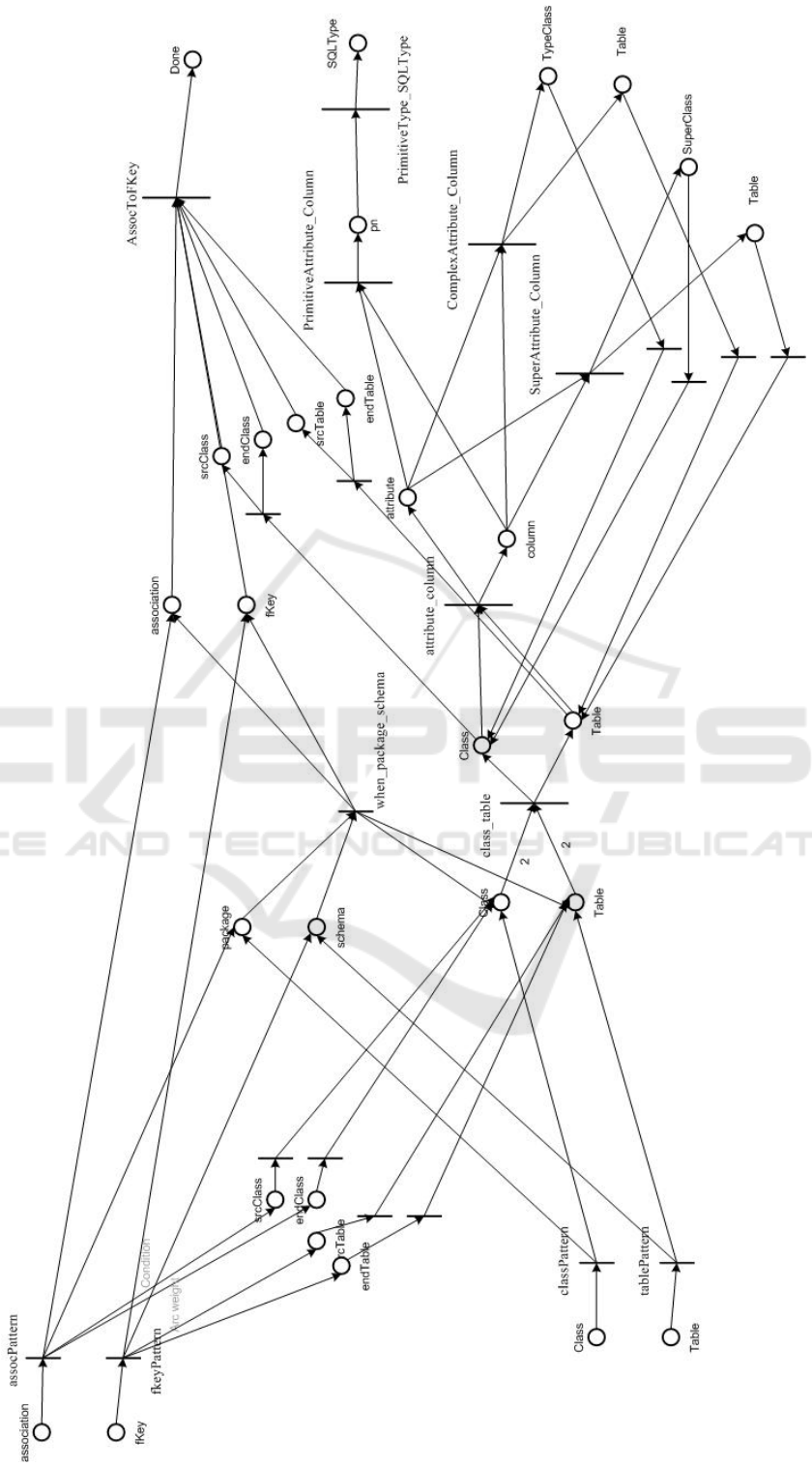

Fig. 4. The equivalent Petri Net of *uml_rdbmstransformation* of the [3].

The transition *primitiveAttribute_Column* is followed by the transition *PrimitiveType-ToSQLType*, which is the only transition with a single place as input in the net. This transition is a simulation of a function carrying the same name in the original transformation. This function accepts one input and results in one output as any other function. A higher-level net could be created by avoiding certain details of the transformations such as the *package_schema*, which is only used for uniqueness purposes and does not play a key role in understanding the general path of the mappings.

5 Related Work

Several existing graphical approaches for the specification of QVT relations are based on Petri-Nets. These approaches often use Petri-Nets as a technique for validation, debugging and simulation of QVT relations. Wimmer et al. [4],[5],[6],[11] have presented one of the most comprehensive works in this area in a series of consistently related efforts for the formalization of colored Petri-Nets mainly for debugging and validation of the QVT relations. Wimmer et al.'s work in [4] and [5] present and develop a debugging environment for QVT relations using Petri-Nets. Wimmer et al.'s work in [11] is rather focused on the formalisms of the tokens amongst the Petri-Nets as a representation of modeling elements and instances. Wimmer et al.'s work in [6] targets the same objective as our approach. The idea is for the colored Petri-Nets to act as a map amongst the set of transformations but the output is once again very close to the authors' other works, which tends to generate very rich models that are suitable for validation/verification purposes but too illustrative for a high-level understanding of the transformations. A successor in the series by Kusel et al. [9] is an effort to bridge QVT relations to a model transformation language based on Petri-Nets named TROPIC introduced by Reiter et al. [10].

Other work in this area shares similar objectives. De Lara and Guerra [7] present a framework for formalization of QVT relations using colored Petri-Nets. The objective of this work is to alleviate the process of running QVT relations as well as their debug and analysis. The approach provides the capability of generating a high-level view of the transformations in which, every relation is shown using a single transition only. Kappel et al. [8] present an effort for modeling the atomic mapping operators using a universal definition to be used in multiple tools and environments. The approach is based on Petri-Nets. Reiter et al. [10] suggest a Petri-Nets based approach towards executing QVT relations. The approach explores different aspects of bridging QVT relations, a declarative language, as well as imperative languages concepts that are easier for execution and tracing.

Other work exists that is not based on Petri-Nets. Such approaches are usually dedicated to presenting a visualization of the QVT relations without any specific concern. Thus the idea is to present a graphical equivalent of the QVT relations. However, the result does often suggest a clearer or even smaller version of the transformations compared to the textual format, hence they remain as just another way of presenting QVT relations and yet as verbose as the textual format. A good example is UMLX [13], a graphical transformation language concerned with the presentation of transformations. This notation, which is partly used in the official QVT document is also accepted and

used by other researchers such as Mazn et al. [14] and Blanco et al. [15]. However, the size and complexity of the generated graphs using this notation is not smaller or clearer than the textual format. Another example of such approaches is the graphical presentation framework QVT_VMTS built using the eclipse plugin VMTS [16] . These approaches are mostly suitable for the visualization of individual relations and do not scale up for presenting a global picture of transformations as a set of relations.

In summary, the two distinctive features of our work are: 1) Providing an all-in-one picture of all the transformations in one frame i.e. a map of mappings and 2) The capability of the models to be expressed at different levels of details.

Amongst the related work, Wimmer et al.'s [6] approaches the same goal as our first point above. Thus, the ultimate model, being originally rich, does not serve the purpose of understandability. The reason seems to be that our technique is concerned with comprehensibility while their work is rather focused on the formall correctness. De Lara and Guerra [7] approach point 2 above. One important difference is that our technique allows the designer to build upon as many levels of details as equired while De Lara and Guerra's work restricts the high-level view to only one predefined level.

6 Conclusions

We have presented a technique for using basic Petri-Nets as a visualization technique for QVT relations. The technique is focused on providing a comprehensible picture of the transformations as a map of mappings. We presented the rules for mapping QVT relations to Petri-Nets such that the resultant Petri-Net projects a traceable path of the QVT relations. Our nets may not qualify for formal validation and execution of the relations but prepare a venue for design and understanding of the relations so that they can be easily traced in order to verify if the expected mappings occur.

The mappings presented in this paper have been successfully implemented in MediniQVT, which accepts Eclipse-based meta-models of both QVT relations and Petri-Nets generated using GMF [17]. Both the meta-models and the transformations have been tested upon different sets of transformations from different sources. Certain examples are the uml_rdbms transformations of the OMG's QVT document, three different sets of transformations of a model-driven web development method [18] and the set of transformations designed for mapping QVT relations to Petri-Nets as prescribed in this paper.

Future work will examine and formalize several patterns that may occur in QVT relations in order to present a more complete coverage of the automated transformations that map QVT relations to Petri-Nets and vice versa. Also, as presented in Section 5, it is possible to obtain different nets equivalent to a set of QVT relations based on the designer's viewpoint as well as the level of required abstraction. This feature needs to be formally defined in the future and to be formally related to the elements of the Petri-Nets that are placed in different layers.

References

1. James L. Peterson. Theory and the Modeling of Systems. Prentice-Hall, N.J., 1981.

2. Pierantonio, A. Vallecillo, A. Selic, B. and Gray, J.: Special Issue on Model Transformation. Sci. Comput. Program. 68(3): 111-113 (2007)
3. OMG, MOF QVT Final Adopted Specification, November 2005
4. Wimmer, M. Kappel, G. Schönböck, J. Kusel, A. Retschitzegger, W. and Schwinger, W. A Petri Net based Debugging Environment for QVT Relations Proceedings of the 24th International Conference on Automated Software Engineering (ASE 2009), IEEE, pp. 1-12, 2009
5. Wimmer, M. Kusel, A. Schoenboeck, J. Kappel, G. Retschitzegger, W. and Schwinger, W. Reviving QVT Relations: Model-Based Debugging Using Colored Petri Nets. In MoDELS '09: Proceedings of the 12th international conference on Model Driven Engineering Languages and Systems (2009) Pages 727-732.
6. Wimmer, M. Kusel, A. Reiter, T. Retschitzegger, W. Schwinger, W. Kappel, G. Lost in Translation? Transformation Nets to the Rescue! Information Systems: Modeling, Development, and Integration (2009), pp. 315-327.
7. Juan de Lara, Esther Guerra, Formal Support for QVT-Relations with Coloured Petri Nets. In Model Driven Engineering Languages and Systems (2009), pp. 256-270.
8. Kappel, G. Kargl, H. Reiter, T. Retschitzegger, W. Schwinger, W. Strommer, M. and Wimmer, M. A Framework for Building Mapping Operators Resolving Structural Heterogeneities, in Information Systems and e-Business Technologies (UNISCON'2008), Springer, pp. 158-174, 2008
9. Kusel, A. Schwinger, W. Wimmer, M. Retschitzegger, W. Common Pitfalls of Using QVT Relations - Graphical Debugging as Remedy. In: Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems. Pages 329-334
10. Reiter, T. Wimmer, M. and Kargl, H. Towards a runtime model based on Colored Petri Nets for the execution of model transformations. In Proc. of 3rd Workshop on Models and Aspects @ ECOOP'07, Berlin, 2007.
11. Wimmer, M. Kusel, A. Schönböck, J. Reiter, T. Retschitzegger and W. Schwinger, W. Lets's Play the Token Game – Model Transformations Powered By Transformation Nets; Vortrag: International Workshop on Petri Nets and Software Engineering, Paris, France; 22.06.2009 - 23.06.2009; in: Proc. of the International Workshop on Petri Nets and Software Engineering PNSE'09, Universit Paris 13, (2009), S. 35 - 50.
12. mediniQVT Trac, projects.ikv.de/qvt, 3 May 2008
13. Willink, E. D. UMLX: A graphical transformation language for MDA (2003) In Proc. of OOPSLA 2003.
14. Mazón, J.-N. Trujillo, J. and Lechtenbörger, J. A Set of QVT Relations to Assure the Correctness of Data Warehouses by Using Multidimensional Normal Forms. Conceptual Modeling - ER 2006. Pages 385-398.
15. Blanco, C. De Guzmán, I. G.-R. Medina, E. F. Trujillo, J. and Piattini, M. Automatic Generation of Secure Multidimensional Code for Data Warehouses: An MDA Approach. Lecture Notes In Computer Science; Vol. 5332 Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008.Pages: 1052 - 1068
16. VMTS_QVT, avalon.aut.bme.hu/ tihamer/research/vmts/qvt/vmts_qvt.html, December 2009
17. GMF - Graphical Modeling Framework, www.eclipse.org/modeling/gmf, April 2009
18. A. Fatolahi, S. S. Somé , T. C. Lethbridge. TR-2008-02 Automated Generation of Abstract Web Applications using QVT Relations August 2008. Available from www.site.uottawa.ca/eng/school/publications/techrep/2008/FatohaliSomeLethbridge.pdf
19. Freeman, Herbert, Automated Cartographic Text Placement. White paper. Available from www.maptext.com/ProductLiterature/Freeman-White-Paper-041027.pdf
20. O'Connor, J.J. and E.F. Robertson, The History of Cartography. Scotland: St. Andrews University, 2002.