Electronic Communications of the EASST
Volume X (2010)

Proceedings of the
Second International Workshop on
Visual Formalisms for Patterns
(VFfP 2010)

A Generic Visual Language Technique for DSVL Model Refactoring to Patterns

Karen Li, John Hosking, and John Grundy

5 Pages

Guest Editors: Paolo Bottoni, Esther Guerra, Juan de Lara
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Generic Visual Language Technique for DSVL Model Refactoring to Patterns

Karen Li[1], **John Hosking**[1], **and John Grundy**[2]

[1] {k.li, j.hosking}@auckland.ac.nz
Departments of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand

[2] jgrundy@swin.edu.au
[2]Faculty of Information and Communication Technologies, Swinburne University of Technology, PO Box 218, Hawthorn, Victoria, Australia

Abstract: As the popularity of domain-specific visual languages (DSVLs) grows, many concerns have arisen regarding quality assurance and evolvability of their designs (meta-models) as well as their model instances. We address some aspects of automated DSVL model instance modification for quality improvement based on refactoring specifications. We propose a graph transformation based visual language approach for DSVL authors to specify at the meta-model level the matching and discovery of DSVL model smells and the application of pattern solutions in a DSVL meta-tool. As an outcome, DSVL users will be provided with pattern-based design evolution support for their domain models.

Keywords: Meta-tools, domain-specific visual languages, graph transformation, design patterns, refactoring, model-driven engineering

1 Introduction

As the popularity of DSVLs grows, many concerns have arisen regarding the quality of both DSVL designs and the domain models created by novice users using the DSVLs [Moo09, LB05]. Model quality assurance research is still in infancy, presenting limited outcomes associated with the areas of model measures, metrics, and transformations [RB09].

Refactoring [FB99, Ker05] is a mature technique integrated in most popular IDEs for evolutionary code design, allowing identifications of code smells (e.g. duplication and complexity) and addressing them with best practise solutions (design patterns) to improve code design quality. Equivalently, refactoring should be a desirable means to improve model quality at higher levels of abstraction by removing model smells (i.e. side-effects of models in MDA such as duplication, complexity, redundancy, incompleteness and inconsistency). However, very few modelling tools provide integrated automatic support for detecting model smells and invoking model refactoring accordingly. The state-of-the-art only supports limited types of models (mainly just UML) with pre-defined refactoring methods, currently lacking a generic way to express common but customisable smells and their linked refactoring solutions in DSVLs [MTM07, MRG09]. Our research aims to generalise a family of common model smells (antipatterns) and pattern solutions for improving DSVL modelling, and support generic but customisable refactoring specifications for reuse across different DSVL meta-model definitions in a meta-tool. This paper proposes integrating a graph transformation based technique into a DSVL meta-tool for pattern-based DSVL model refactoring specification.

2 Visual Specification of Model Refactoring

Various formalisms have been used to specify model refactoring [MTM07], and one that we are convinced is appropriate is fundamental graph transformation theory [Roz97]. This is because it presents an intuitive graphical computation paradigm as well as a natural fit for describing matching of model smells as left-hand side and pattern solutions as right-hand side transformation rules in our problem domain (UML-based approaches do not present such a natural visual linkage between smells and solutions), together with its ability to formulate effective validations of specifications through parsing graph grammars. Specification of anti-patterns and patterns using graph transformations has been an existing technique to support model evolution [BEK+06, ZKDZ07]. However, current solutions do not separate domain contexts from common transformation specifications, thus preventing reuse of specifications across different DSVLs [MTM07]. Our approach aims to provide such a separation, via a generic but configurable visual language. Our approach allows DSVL designers to define (with high-level reuse support) refactoring of model smells at the same level of abstraction as their DSVL meta-models. With our tool support, a refactoring specification generates code to be realised in a DSVL environment to inform users of detected smells, and provide commands to enable applications of pattern solutions on model instances. We have developed an extensible library of functional building blocks to be used in code generation for pattern matching based selection, insertion, deletion and update of model elements.

2.1. Generic notation

In our graph transformation based language, we specify a model smell as a left-hand side (LHS) and a pattern solution as a right-hand side (RHS) of a graph transformation rule, linking effectively when to apply a refactoring to the consequence of applying it. Both the LHS and RHS use the same node, edge and attribute notation, with nodes specifying participants and relationships, edges specifying role bindings of related participants, and attributes specifying additional pattern matching criteria or input prompt. LHS to RHS mappings are encoded conventionally using identical naming, numbering and colouring; mapped constructs represent structure preserving, unmapped LHS constructs are to be deleted, and unmapped RHS constructs are to be created in the transformation.

Our visual language includes the following node types for a generic refactoring specification:

1. Generic participants, represented by rectangular compartment shapes (with a compartment holding attribute specifications, collapsed by default) with labels encoding identification number and name, and a placeholder for a to-be bound DSVL meta-model context;
2. Generic participant relationships, represented by rounded rectangular compartment shapes with labels encoding identification number and name, and a placeholder for a to-be bound DSVL meta-model relationship context;
3. Generic implicit relationships, having a similar representation to the normal participant relationships, however with a distinctive presentation of a dashed dot border. They mainly

exploit attribute compartment fields for querying dynamic relation characteristics in a DSVL model instance, for example, the specification of equality of certain property values between related participants.

Edges are directed connectors between participants and relationships representing source and target role bindings. Attributes, as compartment members of a participant or relationship, specify pattern matching conditions that hold true or acquire input using simplified OCL expressions (to integrate our earlier DSVL constraint specification mechanism [LHG07]).

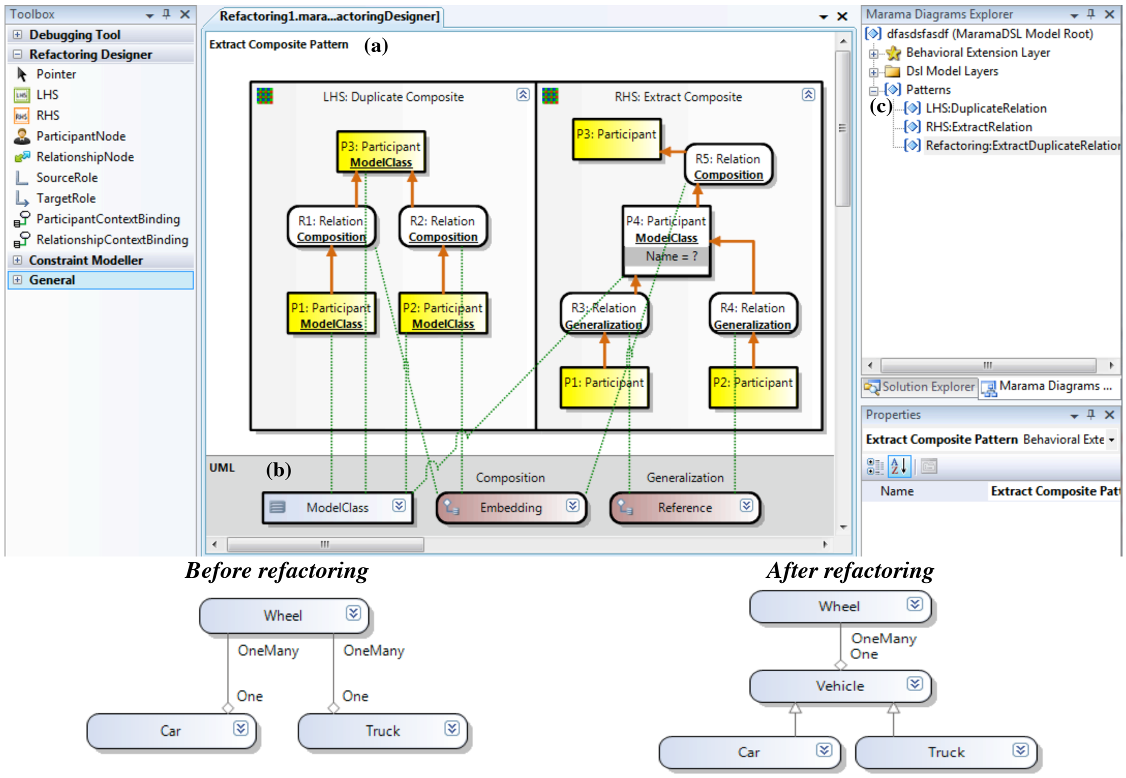

Figure 1. A DSVL refactoring pattern specification environment (with an example showing a generic Extract Duplicate Relation refactoring pattern customised for UML Extract Composite)

2.2. Reuse and customisation

A refactoring is specified using a linked view provided in our proof-of-concept meta-tool, MaramaDSL. As shown in Figure 1, the view exploits parallel orthogonal layered representations for separate but easy to bind generic refactoring pattern specifications (a) and DSVL meta-model contexts (b). MaramaDSL provides filtering capabilities for adding in interested potential DSVL meta-model elements for pattern participation. The context binding of a DSVL meta-model is visually supported via green dotted lines across the two layers connecting elements in the DSVL meta-model with their participations in the refactoring pattern specification layer. The simple linking mechanism allows easy domain customisation of generic pattern components. While we promote maximum reuse of generic specifications, such context bindings can effectively restrict the runtime domain model element types to be involved in pattern matching and refactoring. The context binding links can be concealed at individual pattern element level for diagram clutter management. Context bindings are supplemented by a dual text encoding on a pattern element (via underlined text in the bound pattern element) to ease context navigations.

MaramaDSL provides support for high-level separate and holistic reuse of model smell definition, pattern solution specification, and the overall refactoring transformation. It allows a whole specification or LHS/RHS to be saved context-free (with all context bindings removed), appearing in an explorer window (c); it can then be accessed and drag-dropped from there for direct reuse and binding with other DSVL meta-models. Accessed pattern specifications can also be easily adapted for reuse in a variant way, e.g. modify or remove any existing participant or relationship, or add elements to meet specific needs.

We provide two examples here illustrating the usage of our visual notation. The example in Figure 1 is the specification of a generic Extract Duplicate Relation refactoring pattern customised for a UML Extract Composite use case. It defines that if a duplicate Composition relationship (R1 and R2) holds from two source ModelClasses (P1 and P2) to a target ModelClass (P3), the transformation will extract a Composition relationship (R5) for presence between a new super ModelClass (P4) (created from Generalizations of P1 and P2) and the target ModelClass. The example in Figure 2 shows the specification of a generic Pull Up Common Element refactoring pattern customised for a UML Pull Up Feature use case. It defines that if two Attributes held in two ModelClasses share the same name and type (queried in an implicit relationship), the common Attribute should be pulled up to a super ModelClass.

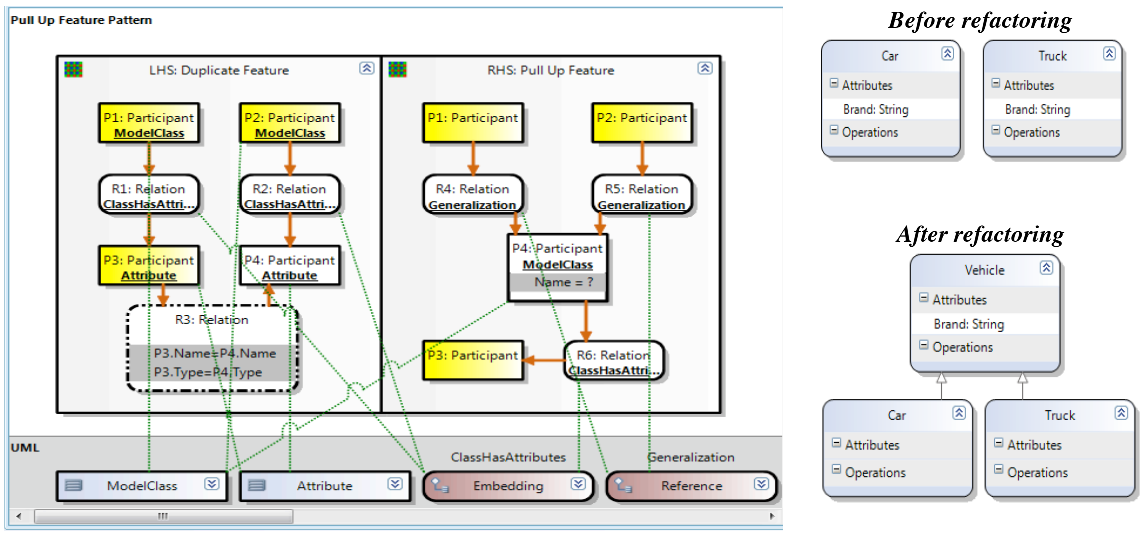

Figure 2. Generic Pull Up Common Element pattern customised for UML Pull up Feature

2.3. Ongoing work

Our initial visual language does not yet cope with scale up of pattern matching and transformation taking into account overlapping of multiple matched occurrences, collections of objects, and chains/hierarchies with arbitrary numbers of participants and relationships. These need addressing. We are going to incorporate an additional control flow graph for dealing with dependencies and conflicts among multiple refactorings at a high-level of abstraction. We are yet to define validation of refactoring models for completeness and correctness as well as behaviour preservation. We are looking to integrate an existing graph grammar parser as the backend for this purpose, and design additional visual feedback to notify users of the validation. We are also designing dynamic visualisation of pattern matching and transformation with the inclusion of helpful annotation, playback and rollback features.

3 Conclusion

Model refactoring should be generic and reusable across DSVLs, in a similar way that code refactoring has been applied across different programming languages and platforms. We propose adding into a DSVL meta-tool a generic and reusable specification technique for DSVL authors to define model refactoring methods as a means to support DSVL users to evolve their model instances. Our graph transformation based visual language approach is to be evolved for this purpose.

Bibliography

[BEK+06] E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer, E. Weiss. EMF Model Refactoring based on Graph Transformation Concepts. In Proc. of the Third Workshop on Software Evolution through Transformations: Embracing the Change (SeTra2006), 2006.

[FB99] M. Fowler, K. Beck. Refactoring: improving the design of existing code. Reading, MA: Addison-Wesley, 1999.

[Ker05] J. Kerievsky. Refactoring to patterns. Boston: Addison-Wesley, 2005.

[LB05] F. Leung, N. Bolloju. Analyzing the Quality of Domain Models developed by Novice Systems Analysts. In Proc. of the 38th Hawaii International Conference on System Sciences, 2005.

[LHG07] N. Liu, J. Hosking, J. Grundy. MaramaTatau: Extending a Domain Specific Visual Language Meta Tool with a Declarative Constraint Mechanism. In Proc. of VL/HCC 2007.

[Moo09] D.L. Moody. The “Physics” of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE TSE 2009.

[MRG09] M. Mohamed, M. Romdhani, K. Ghedira. Classification of model refactoring approaches. Journal of Object Technology 8(6), 2009.

[MTM07] T. Mens, G. Taentzer, D. Müller. Challenges in Model Refactoring. In Proc. of 1st Workshop on Refactoring Tools, University of Berlin, 2007.

[RB09] J. Rech, C. Bunse. Model-driven software development: integrating quality assurance. Hershey: Information Science Reference, c2009.

[Roz97] G. Rozenberg. Handbook of graph grammars and computing by graph transformation. World Scientific, c1997.

[ZKDZ07] C. Zhao, J. Kong, J. Dong, K. Zhang. Pattern-based design evolution using graph transformation. JVLC 18(4), pp. 378-398, 2007.