Design Patterns for Model Transformations: Current research and future directions

K. Lano[1], S. Yassipour-Tehrani[1]

[1]Dept of Informatics, King's College London, Strand, London, UK

Abstract. There is increasing interest in the use of design patterns for model transformations, and a number of such patterns have been proposed. In this paper we survey previous work on transformation design patterns, discuss one pattern in detail, and identify priorities for future research.

1 Introduction

Design patterns have proved to be an effective concept to improve the quality of software systems, with classic patterns such as Iterator, Facade and MVC now part of the standard vocabulary of software developers and provided as essential elements of programming languages and of software development environments. Model transformation (MT) development should also be able to benefit from the standardised solutions and expertise embedded in design patterns: currently such development is often carried out in an ad-hoc manner because of the novel nature of the languages and processing involved.

Transformation development requires new or adapted specification and design patterns because of the specialised features of MT programming and languages:

- Transformations involve complex structured data (models) and navigation through and manipulation of such data.
- Transformations involve unconventional processing such as graph rewriting and partially-specified execution orders.
- Hybrid MT languages involve a mix of declarative and imperative language elements.
- Languages supporting bidirectional transformations (bx) involve complex processing to synchronise or transform models in either source to target or target to source directions.

2 Research into MT design patterns

The first works on MT design patterns identified language-specific patterns for the ATL [9] and QVT-R [19] languages:

ATL: In [2, 7] ATL-specific patterns were identified: Transformation Parameters; Multiple Matching; Object Indexing; Many-to-one; Many-to-many; Custom Tracing.

QVT-R: In [13] QVT-R patterns such as Enable conditions in only one Direction, and Simulation of Key were introduced. In [10] the use of marker relations in a Model Copying pattern for QVT-R was detailed.

In some cases, such patterns simply gave techniques for circumventing limitations of the specific MT languages.

Subsequently, work was carried out to identify language-independent MT design patterns, based on generalisations of the language-specific patterns or adaptions of classical design patterns for MT: in [1, 5–7, 11, 12, 14, 15, 17] such patterns were described. These included Auxiliary Metamodel, a generalisation of the ATL pattern Transformation Parameters, and Map Objects Before Links, a general strategy for model copying and migration transformations.

In [17] the patterns were grouped into five categories:

Rule modularisation patterns: Phased Construction; Structure Preservation; Entity Splitting/ Structure Elaboration; Entity Merging/Structure Abstraction; Map Objects Before Links; Parallel Composition/Sequential Composition; Auxiliary Metamodel; Construction and Cleanup; Recursive Descent; Replace Explicit Calls by Implicit Calls; Introduce Rule Inheritance.

Optimisation patterns: Unique Instantiation; Object Indexing; Omit Negative Application Conditions; Replace Fixed-point by Bounded Iteration; Decompose Complex Navigations; Restrict Input Ranges; Remove Duplicated Expression Evaluations; Implicit Copy.

Model-to-text patterns: Model Visitor; Text Templates; Replace Abstract by Concrete Syntax.

Expressiveness patterns: Simulating Multiple Matching; Simulating Universal Quantification; Simulating Explicit Rule Scheduling.

Architectural patterns: Phased Model Construction; Target Model Splitting; Model Merging; Auxiliary Models; Filter Before Processing.

Patterns for specific categories of transformation have also been identified:

Bidirectional transformation (bx) patterns: Auxiliary Correspondence Model; Cleanup before Construct; Unique Instantiation; Phased Construction for bx; Entity Merging/Splitting for bx; Map Objects Before Links for bx [18].

Data migration patterns: Migrate along Domain Partitions; Measure Migration Quality; Data Cleansing [20].

Auxiliary Correspondence Model is an inbuilt and essential language feature of TGG and QVT-R. Unique Instantiation is an inbuilt language feature of QVT-R and UML-RSDS, and Object Indexing and Omit Negative Application Conditions are also inbuilt into UML-RSDS.

Figure 1 (based on [17]) shows the generalisation and usage relationships between some of the above patterns.

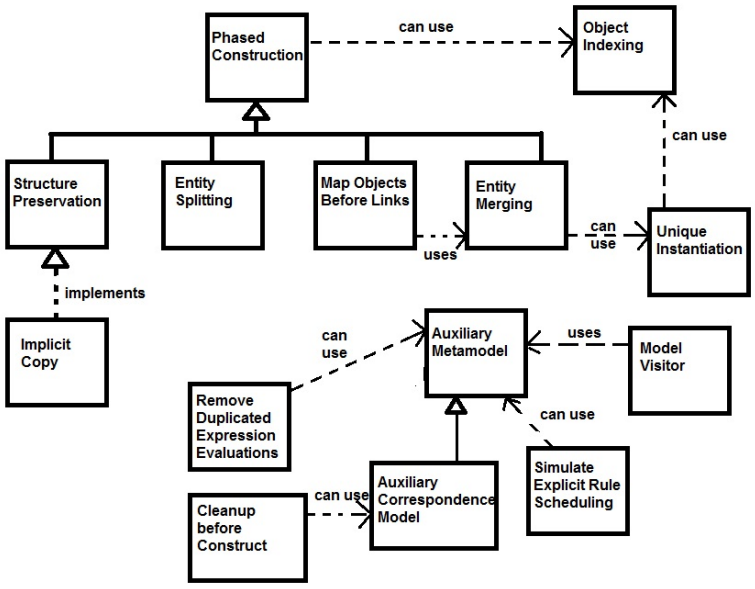

Fig. 1. Relationships between transformation design patterns

3 Pattern example: Cleanup before Construct

This pattern has been identified in several bx examples. The pattern defines a two-phase approach in both forward and reverse transformations associated with a bx with relation R: the forward transformation $R^{\rightarrow}$ first removes all elements from the target model n which fail to satisfy R for any element of the source m, and then constructs elements of n to satisfy R with respect to m. The reverse transformation $R^{\leftarrow}$ operates on m in the same manner.

Benefits: The pattern is an effective way to ensure the correctness of separate-models bx: all target elements which invalidate R are removed.

Disadvantages: There may be efficiency problems because for each target model element, a search through the source model for possibly corresponding source element(s) may be needed. Elements may be deleted in the Cleanup phase only to be reconstructed in the Construct phase: use of Auxilliary Correspondence Model [18] is an alternative strategy to avoid this problem, by enforcing that feature values should change in response to a feature value change in a corresponding element, rather than deletion and recreation of elements.

Related Patterns: This pattern is a variant of the *Construction and Cleanup* pattern of [17].

Examples: An example is the Composers bx [4]. Implicit deletion in QVT operates in a similar manner, but can only modify models (domains) marked as *enforced* [19]. In UML-RSDS, explicit cleanup rules $Cn^{\times}$ can be deduced from the construction rules Cn, for mapping transformations [16].

Assume that the transformation rules Cn are of the form:

$$S_i{\rightarrow}forAll(s \mid SCond(s) \Rightarrow T_j{\rightarrow}exists(t \mid TCond(t) \ \&\ P_{i,j}(s,t)))$$

for source entity types S_i and target entity types T_j. Then if identity attributes are used to define the source-target correspondence (so that target model element $t : T_j$ corresponds to source model element $s : S_i$ when the respective identity attributes are equal: $s.sId = t.tId$), then $Cn^{\times}$ can be expressed as the constraints (i):

$$T_j{\rightarrow}forAll(t \mid TCond(t) \ \&\ t.tId \notin S_i{\rightarrow}collect(sId) \Rightarrow t{\rightarrow}isDeleted())$$

which deletes t if there is no corresponding s with the same identity, and (ii):

$$T_j{\rightarrow}forAll(t \mid TCond(t) \ \&\ t.tId : S_i{\rightarrow}collect(sId) \ \&\ s = S_i[t.tId] \ \&\ not(SCond(s)) \Rightarrow t{\rightarrow}isDeleted())$$

which deletes t if such an s exists but it is not in the domain of the transformation (relation R). (ii) is omitted if $SCond$ is the *true* predicate.

In the case that $TCond(t)$ and $SCond(s)$ hold for corresponding s, t, but $P_{i,j}(s,t)$ does not hold, t should not be deleted, but $P_{i,j}(s,t)$ should be established by updating t (iii):

$$S_i{\rightarrow}forAll(s \mid s.sId : T_j{\rightarrow}collect(tId) \ \&\ t = T_j[sId] \ \&\ SCond(s) \ \&\ TCond(t) \Rightarrow P_{i,j}(s,t))$$

For a transformation τ, the cleanup transformation $\tau^{\times}$ has the above $Cn^{\times}$ constraints (i), (ii), (iii) as its postconditions, in the same order as the Cn occur in the *Post* of τ. The forward direction $\tau^{\rightarrow}$ of the bx is then $\tau^{\times}; \tau$.

A change-propagation version τ^{Δ} of τ can also be defined, to operate on source model increments δm (finite collections of deletion, creation and modification updates). τ^{Δ} is $\tau^{\times}; \tau$ where the rules of $\tau^{\times}$ are only applied for deletion (i) and modification (ii), (iii) updates in δm, and the rules of τ are only applied for modification and creation updates in δm.

4 Future research directions and challenges

Because of the novelty of the MT field, there is as yet insufficient evidence for the effectiveness of most of the identified patterns. Work is needed to survey applications of MT patterns in practice and to evaluate their utility. Work is also needed to identify appropriate classifications for MT patterns, to identify connections between patterns, and useful compositions of patterns. Techniques

and guidelines for the selection of MT patterns need to be developed and incorporated into MT engineering environments. Patterns for new types of transformation, such as transformations at runtime or streaming transformations, are also needed. Finally, research into transformation anti-patterns and techniques for the identification of 'bad smells' in transformation specifications is needed.

References

1. A. Agrawal, A. Vizhanyo, Z. Kalmar, F. Shi, A. Narayanan, G. Karsai, *Reusable Idioms and Patterns in Graph Transformation Languages*, Electronic notes in Theoretical Computer Science, pp. 181–192, 2005.
2. ATL Design Patterns, https://wiki.eclipse.org/ATL/Design_Patterns, 2015.
3. I. Bayley, H. Zhu, *On the composition of design patterns*, QSIC 2008, IEEE Computer Society, 2008, pp. 27–36.
4. J. Cheney, J. McKinna, P. Stevens, J. Gibbons, *Towards a repository of bx examples*, EDBT/ICDT 2014, 2014, pp. 87–91.
5. J. S. Cuadrado, F. Jouault, J. G. Molina, J. Bezivin, *Optimization patterns for OCL-based model transformations*, MODELS 2008, vol. 5421 LNCS, Springer-Verlag, pp. 273–284, 2008.
6. K. Duddy, A. Gerber, M. Lawley, K. Raymond, J. Steel, *Model transformation: a declarative, reusable pattern approach*, 7th International Enterprise Distributed Object Computing Conference (EDOC '03), 2003, pp. 174–185.
7. J. Bezivin, F. Jouault, J. Palies, *Towards Model Transformation Design Patterns*, 1st European Workshop on Model Transformations, 2005.
8. V. Bollati, J. Vara, A. Jimenez, E. Marcos, *Applying MDE to the (semi-)automatic development of model transformations*, Information and Software Technology, 2013.
9. Eclipsepedia, *ATL User Guide*, http://wiki.eclipse.org/ATL/ User_Guide_-_The_ATL_Language, 2014.
10. T. Goldschmidt, G. Wachsmuth, *Refinement transformation support for QVT relational transformations*, FZI, Karlsruhe, Germany, 2011.
11. M. E. Iacob, M. W. A. Steen, L. Heerink, *Reusable model transformation patterns*, Enterprise Distributed Object Computing Conference Workshops, 2008, pp. 1–10, doi:10.1109/EDOCW.2008.51.
12. J. Johannes, S. Zschaler, M. Fernandez, A. Castillo, D. Kolovos, R. Paige, *Abstracting complex languages through transformation and composition*, MODELS 2009, LNCS 5795, pp. 546–550, 2009.
13. J. Kiegeland, H. Eichler, *Medini-QVT*, http://projects.ikv.de/qvt, 2014.
14. K. Lano, S. Kolahdouz-Rahimi, *Model transformation design patterns*, ICSEA 2011, IARIA, 2011, pp. 263–268.
15. K. Lano, S. Kolahdouz-Rahimi, *Optimising Model-transformations using Design Patterns*, MODELSWARD 2013.
16. K. Lano, *The UML-RSDS Manual*, www.dcs.kcl.ac.uk/staff/kcl/uml2web/umlrsds.pdf, 2014.
17. K. Lano, S. Kolahdouz-Rahimi, *Model Transformation Design Patterns*, IEEE Transactions in Software Engineering, 2014.
18. K. Lano, S. Kolahdouz-Rahimi, *Model transformation design patterns for bidirectionality*, FSEN 2015.
19. OMG, *MOF 2.0 Query/View/Transformation Specification v1.1*, 2011.
20. M. Wagner, T. Wellhausen, *Patterns for data migration projects*, www.tngtech.com, 2011.