Controversy Corner

A survey of model transformation design patterns in practice

Kevin Lano\textsuperscript{a,b}, Shekoufeh Kolahdouz-Rahimi\textsuperscript{b}, Sobhan Yassipour-Tehrani\textsuperscript{a}, Mohammadreza Sharbaf\textsuperscript{b}

\textsuperscript{a} Department of Informatics, King's College London, London, UK
\textsuperscript{b} Department of Software Engineering, University of Isfahan, Iran

\section*{A R T I C L E  I N F O}

Article history:
Received 19 August 2017
Revised 26 February 2018
Accepted 1 March 2018

Keywords:
Model Transformations
Design Patterns
Empirical Software Engineering

\section*{A B S T R A C T}

Model transformation design patterns have been proposed by a number of researchers, but their usage appears to be sporadic and sometimes patterns are applied without recognition of the pattern. In this paper we provide a systematic literature review of transformation design pattern applications. We evaluate how widely patterns have been used, and how their use differs in different transformation languages and for different categories of transformation. We identify what benefits appear to arise from the use of patterns, and consider how the application of patterns can be improved. The paper also identifies several new patterns which have not previously been catalogued.

© 2018 Elsevier Inc. All rights reserved.

\section*{1. Introduction}

Design patterns have become a widely-used technique in software engineering, to support systematic software construction and the reuse of design solutions. Mainstream patterns, such as those defined in \citet{Gamma1994}, have become part of standard programming knowledge, and have been incorporated into programming languages and environments. Specialised patterns, for particular technical domains, have also been defined, for concurrent systems, for security, and for many other concerns. In the model transformations (MT) domain \citet{Carnecke2006, Bezivin2003, Iacob2008, Lano2014}, patterns have also been identified and formalised \citep{Bezivin2003, Iacob2008, Lano2014}. For example, the fundamental pattern Auxiliary Metamodel involves the introduction of auxiliary metamodel entity types and/or features to support transformation processing, such as the maintenance of traces or other information associated with the transformation execution \citep{Lano2014}. A specialised pattern is Auxiliary Correspondence Model, which uses auxiliary entity types and features to maintain a correspondence between source and target elements, to support bidirectional processing such as change propagation and model synchronisation (Fig. 1).

An example of Auxiliary Correspondence Model is the UmlToRel transformation in the ModelMofr QVT-R repository:

$$\text{transformation UmlToRel}(\text{uml:umlmm, rdbms:relmm})$$

$$\begin{align*}
\text{key umlmm::Class(name);} \\
\text{key umlmm::Attribute(name,class);} \\
\text{key relmm::Table(name);} \\
\text{key relmm::Column(name,table);} \\
\text{...}
\end{align*}$$

$$\text{top relation ClassToTable}$$

$$\begin{align*}
\text{c : Class ( name = n );} \\
\text{enforce domain uml} \\
\text{t : Table ( name = n );} \\
\text{...}
\end{align*}$$

The keys Class:: name and Table:: name are used to maintain a bidirectional correspondence between classes and tables.

We were interested in discovering how widely these and other patterns have been used in MT in practice, and if patterns were of clear benefit for MT development. We decided to perform a systematic literature review (SLR) \citet{Kitchenham2004}. We defined the SLR according to the PICOC criteria of \citet{Loniewski2010}:

- Population: Research papers presenting MT developments or case studies
- Intervention: MT design pattern usage
- Comparison: Analysis of the current state of MT pattern usage
- Outcome: MT pattern application
community than the terms Editing and Model synchronisation. Our main difference to Lucio et al. (2016) is that cases of a PIM to PSM mapping (such as the class diagram to relational database example Kleiner et al. (2013)) are considered as refinements, not translations. Likewise, mappings that map from a semi-formal to a formal language (eg., Syriani and Ergin, 2012) are usually considered semantic mappings, not translations. We use the term Translation only for cases of mappings from one language to another, which do not belong to a more specific category (migration, semantic mapping, etc).

The category of a transformation was assigned based on the following definitions:

1. **Refinement** – mapping from a higher abstraction level model to a lower-level model. This includes CIM to PIM and PIM to PSM mappings in the sense of the OMG’s Model-driven Architecture. The same as Refinement in Lucio et al. (2016), but subtracting the specialised category of code generation.
2. **Code generation** – mapping from a model to text or executable code. Corresponds to Synthesis in Lucio et al. (2016).
3. **Migration** – mapping from one language to another at the same level of abstraction. The same as Migration in Lucio et al. (2016).
4. **Analysis** – extracting information from a model as a view or other analysis result. Corresponds to Restrictive query and Analysis in Lucio et al. (2016).
5. **Refactoring** – Update-in-place transformations which restructure a model, retaining its conformance to the same or a closely-related metamodel. Corresponds to the Editing intent in Lucio et al. (2016).
6. **Semantic mapping** – maps a model m in one language to a formal representation in a language with a formal semantics, to support semantic analysis of m. Semantic definition and some cases of Translation in Lucio et al. (2016).
7. **Bidirectional (Bx)** – transformations which can be applied in either source-to-target or target-to-source directions, supporting model synchronisation and change-propagation. Model synchronisation in Lucio et al. (2016).
8. **Abstraction** – the inverse of refinement. The same as Abstraction in Lucio et al. (2016).

More unusual types of transformation, such as streaming, higher-order (HoT) or runtime transformations, were also encountered.

We adopt the classification of MT design patterns identified in Lano and Kolahdouz-Rahimi (2014), with the addition of Bidirectional patterns: MT patterns that address issues specific to bidirectional transformations (bx), such as model synchronisation and change propagation. The category of a pattern is assigned based on the primary purpose of the pattern.

The pattern categories are:

1. **Rule modularisation patterns** – these have the purpose to organise the structure of individual rules or the structure of dependencies and relationships between rules within a transformation. For example, the Map Objects before Links pattern separates rules that map model entity instances from rules that map links between the instances.
2. **Architectural patterns** – these aim to organise relationships between transformations, or to organise systems of transformations at the inter-transformation level to improve the modularity or processing capabilities of the system. For example, by using a pre-processing transformation to normalise models which are then provided to a subsequent transformation (Pre-Normalisation pattern).
3. **Optimisation patterns** – these are concerned with increasing the efficiency of transformation execution at the rule/individual
دریافت فوری متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات