View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by King's Research Portal

King's Research Portal

DOI:
10.1016/j.jss.2018.03.001

Document Version
Peer reviewed version

Link to publication record in King's Research Portal

Citation for published version (APA):
Lano, K., Kolahdouz-Rahimi, S., Yassipour-Tehrani, S., & Sharbaf, M. (2018). A survey of model transformation design patterns in practice. Journal of Systems and Software, 140, 48-73. https://doi.org/10.1016/j.jss.2018.03.001

Citing this paper
Please note that where the full-text provided on King's Research Portal is the Author Accepted Manuscript or Post-Print version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version for pagination, volume/issue, and date of publication details. And where the final published version is provided on the Research Portal, if citing you are again advised to check the publisher's website for any subsequent corrections.

General rights
Copyright and moral rights for the publications made accessible in the Research Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

•Users may download and print one copy of any publication from the Research Portal for the purpose of private study or research.
•You may not further distribute the material or use it for any profit-making activity or commercial gain
•You may freely distribute the URL identifying the publication in the Research Portal

Take down policy
If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Apr. 2019

A Survey of Model Transformation Design Patterns in Practice

Kevin Lano, Sobhan Yassipour-Tehrani,
Dept. of Informatics
King's College London, London, UK
Email: { kevin.lano, sobhan.yassipour_tehrani }@kcl.ac.uk
Shekoufeh Kolahdouz-Rahimi, Mohammadreza Sharbaf
Dept. of Software Engineering
University of Isfahan, Iran
Email: { sh.rahimi, m.sharbaf }@eng.ui.ac.ir

***Abstract*—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.**

Keywords: Model Transformations; Design Patterns; Empirical Software Engineering.

I. 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 [72], 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, patterns have also been identified and formalised [24], [102], [145]. 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 [145]. 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 (Figure 1).

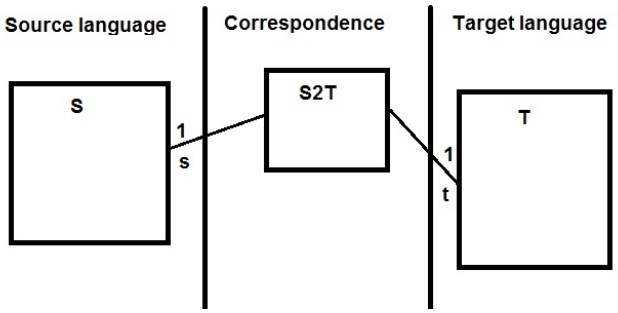

Figure 1. Auxiliary Correspondence Model pattern

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

```
transformation UmlToRel(uml:umlmm, rdbms:relmm)
{ key umlmm::Class{name};
  key umlmm::Attribute{name,class};
  key relmm::Table{name};
  key relmm::Column{name,table};

  ...

  top relation ClassToTable
  { n : String;

    enforce domain uml
      c : Class { name = n };
    enforce domain rdbms
      t : Table { name = n };
  }
  ...
}
```

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) [118]. We defined the SLR according to the PICOC criteria of [155]:

- 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: Identification of benefits of MT pattern use, and potential for increased usage
- Context: MT specification and design.

In Section II we define our research questions and the classifications used for transformations and transformation patterns. In Section III we describe the sources and procedure used for the survey, and in Section IV we give the results with respect to each research question. Section V gives a detailed analysis of the results, Section VI considers threats to validity, and Section VII describes related work.

II. Research questions

The research questions we asked were:

Q1. Design pattern usage: How often are model transformation design patterns used in practice in MT development? Which categories of patterns and which individual patterns are used? Is the use explicit (recognised as a specific design decision) or not? Are different categories of patterns used for particular categories of transformations?

Q2. Design patterns benefits: What perceived benefits were obtained from using patterns? Is there evidence for these benefits?

Q3. Gaps and novel patterns: Are there cases where patterns should have been applied? Are there necessary patterns which have not yet been formalised?

Q4. Trends: Has there been a change in the adoption of MT patterns over time? Have papers cataloguing MT patterns been influential for MT developers?

Q5. MT languages: Do different MT languages differ in their capabilities for expressing and using MT patterns?

The term *model transformation design pattern* is defined in [102] as "A general repeatable solution to a commonly-occurring model transformation problem". Adapting the definition of *design pattern* from [72] we could also define the concept as "descriptions of transformation rules and transformations that are customised to solve a general model transformation design problem in a particular context".

We adopt a classification of transformation kinds similar to the transformation *intents* of [156]. We grouped transformations into categories based upon the main divisions of transformation implementation types which arose in the surveyed cases. In this survey we are concerned with the structure and techniques used in the implementations of transformation cases, therefore our classifications differ from [156]. The difference can be seen in a paper such as [238], which has a Migration intent, but uses a Refactoring transformation to perform migration. For our purposes, it is considered a refactoring.

In contrast to [156] we use the terms *Refactoring* and *Bidirectional* as these are more commonly used in the MT community than the terms *Editing* and *Model synchronisation*. Our main difference to [156] is that cases of a PIM to PSM mapping (such as the class diagram to relational database example [119]) are considered as refinements, not translations. Likewise, mappings that map from a semi-formal to a formal language (eg., [212]) 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 [156], but subtracting the specialised category of code generation.
2) **Code generation** – mapping from a model to text or executable code. Corresponds to *Synthesis* in [156].
3) **Migration** – mapping from one language to another at the same level of abstraction. The same as *Migration* in [156].
4) **Analysis** – extracting information from a model as a view or other analysis result. Corresponds to *Restrictive query* and *Analysis* in [156].
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 [156].
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 [156].
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 synchronization* in [156].
8) **Abstraction** – the inverse of refinement. The same as *Abstraction* in [156].

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 [145], 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 transformation level. For example by caching of rule results to avoid repeated computations (Rule Caching pattern).
4) **Expressiveness** – these patterns are concerned with providing extended processing capabilities for a transformation, by simulating these capabilities using existing facilities. For example, simulating a universal quantification $X \rightarrow forAll(P)$ matching condition by $not(X \rightarrow exists(not(P)))$ (Simulate Universal Quantification pattern).
5) **Bidirectional (Bx)** – these are concerned with aspects specific to bidirectional processing: change-propagation and model synchronisation. For example, maintaining an explicit source-target correspondence relation as an auxiliary model (Auxiliary Correspondence Model pattern).
6) **Model-to-text/Concrete syntax** – these are concerned with aspects specific to the concrete syntax of models, including code or text generation from models. For example, writing text emission rules using a combination of literal text and model element expressions (Text Templates pattern).
7) **Classical/external** – patterns from the GoF book [72] or from the patterns community external to MT. For example, the classical State or Visitor patterns.

We considered that bx and model-to-text patterns do form separate categories because management of bidirectionality and concrete syntax involve specialised problems which therefore require specialised patterns. On the other hand, Auxiliary Metamodel and Unique Instantiation are somewhat different in character from the other rule modularisation patterns, and could be considered separately in a category of fundamental patterns.

It is interesting to consider whether particular categories of patterns are predominately used for particular categories of transformations (eg., if optimisation patterns tend to be used for refactoring transformations). We will address this question as part of Q1. Details of the particular patterns considered in this paper are given in the appendix.

III. Survey method

We used a combination of manual and automated search to identify transformation cases. These were then filtered to remove irrelevant papers, and the remaining papers were inspected in detail to identify which patterns (if any) were used in the cases.

A. Sources and selection criteria

We surveyed papers from four specific sources: The SoSyM journal (from 2003-2016); the Transformation Tool Contest (TTC) from 2010-2016; the ICMT conference from 2008-2016, the MODELS conference from 2005-2016. These sources were chosen as they were known to be the leading conference and journal sources for MT publications. We also included the 82 transformation case papers from [22] in our initial sources. Note that the selection criteria of [22] are similar to ours, although they use ECMFA as an additional specific source, in addition to SoSyM, ICMT and MODELS, and they omit TTC.

In addition, we performed a search in Research Gate[1] using the search string

Model transformation

The reason for using such a general search string is that we wish to estimate the prevalence of pattern use in MT development. Using a more specific string, such as *model transformation AND design pattern*, would distort this estimate. From this search only peer-reviewed papers published in conferences and journals were selected for further consideration: PhD theses and other unpublished materials were excluded at this point, together with papers not written in English. Only papers between 2000 and 2016 were considered.

Papers were only considered if they satisfied either of these conditions:

- Contains sufficient samples of source code of the transformation to determine if patterns were used in the case.
- Provides sufficiently detailed description of the transformation to determine if any patterns were used.

The King's College authors were responsible for gathering all of the potential cases from these sources, and for classifying the cases as (i) irrelevant for the SLR due to the absence of any transformation case study or of any adequate information about a transformation case according to the above conditions; (ii) containing sufficient information about a transformation case to evaluate the transformation, but not using any patterns; (iii) containing a transformation case with a pattern use. The final category was subdivided based on whether the pattern use was explicit or not. The proportion of transformation cases using a pattern is then $\frac{(iii)}{(ii)+(iii)}$.

The overall number of papers produced from the initial searches was over 1000. These were scanned by the two reviewers to initially decide on their inclusion or exclusion. Only papers which included some transformation specifications or designs from transformation developments, industrial or academic, were then selected for further consideration. In cases where exclusion/inclusion was not clear, the reviewers discussed the papers and reached agreement. Resulting from this process, we obtained 289 papers suitable for analysis. We then extended our survey by cascading the references from papers in category (iii). This produced a further 304 papers for analysis, of which 110 were in category (ii) and 120 in category (iii). Table I gives the aggregated numbers of papers considered in the initial and second rounds.

Source	(i) No case	(ii) No pattern	(iii) Pattern	All
Initial search	1062	181	108	1351
Cascaded	74	110	120	304
Total	1136	291	228	1655

TABLE I. FINAL SELECTION STATISTICS

The 228 category (iii) papers are included in the references of this paper. The 291 excluded papers (category (ii) papers) are listed in the supplementary material, and at www.nms.kcl.ac.uk/kevin.lano/mtdpsurx.pdf.

We did not analyse further the 10 papers from category (iii) that are focussed on cataloguing patterns ([24], [49], [102], [132], [110], [141], [142], [143], [145], [147]). These are considered separately in Table XVI.

B. Data collection

The following information was extracted for each surveyed paper: (i) which patterns were used; (ii) whether a use was explicitly identified as a design decision or not; (iii) what benefits, if any, were expected or obtained from use of the patterns; (iv) the MT language(s) used; (v) what was the category of the transformation (refinement; refactoring; migration, etc); (vi) the date of the case publication (year).

The patterns used in the papers were identified (for previously-documented patterns) according to the criteria given in the appendix. In all cases the specifications or code of the transformations were examined in detail. We decided to identify new patterns if there was a clearly defined specification or design structure expressed in a transformation case, which was not an example of a known pattern.

Tables XXVIII, XXIX, XXX, XXXI, XXXII in the Appendix give the initial extracted data for each analysed case in category (iii). Pattern usages are listed for each case, an explicit use of a pattern is marked by *. The category of transformation and the language(s) used to implement the transformation are listed. In some cases the MT language used was not defined in the paper (eg., [232]) or the transformation was implemented in a 3GL/custom language. In these cases we have marked the language entry as *none*. We also indicate which (if any) of the transformation catalogue papers are cited in the case.

Table II shows the distribution of transformation categories across the 218 surveyed cases. In some aspects this distribution is similar to that found in [22], with the categories of Semantic Map/Definition (12% in [22]) and Analysis (8%) similar in extent to our results. Our category of Bidirectional transformations corresponds partly to Model Composition (9%) in [22]. Bidirectional transformations have become an area of significant research activity in recent years, hence the relatively large proportion of bx cases in our study compared with [22].

Evaluating the 82 cases of [22] using our transformation classifications gives: Refinement 33%; Semantic mapping 15%; Translation 11%; Code generation 9%; Refactoring 7%; Migration 6%; Abstraction 6%; Bidirectional 6%; others 7%.

Category	Number of cases	Percentage (of 218)
Refinement	56	26%
Bidirectional	38	17%
Refactoring	30	14%
Migration	27	12%
Code generation	20	9%
Analysis	17	8%
Semantic map	15	7%
Others	15	7%
Total	218	

TABLE II. SLR TRANSFORMATION CATEGORIES

IV. RESULTS

Concerning research question 1, from the search results we identified 519 papers that concerned MT developments. Of these, 228 (44%) contained details of MT pattern usage in the developments. In 109 cases (21%) at least one pattern use was explicitly recognised as a design decision. In 119 cases (23%) no pattern use was explicit. Figure 2 shows the extent of MT pattern usage.

Table III identifies the frequencies of use of different categories of patterns in the surveyed cases, and the number of different patterns used in each category. The percentages are taken of the 218 analysed category (iii) papers, and do not add to 100% because some papers used several patterns, from different categories. On average, a paper uses patterns from 1.3 categories (the total of Table III divided by 218), which suggests that our pattern categories correspond to separate use cases for patterns in practice.

Figure 3 shows the percentage of analysed category (iii) papers using each pattern category.

In terms of the percentages of cases (Table III) and the number of individual pattern usages (Table IV), there is a preponderance of

[1]This database was used because of its very broad scope, and because it directly provides full-text versions of papers in many cases.

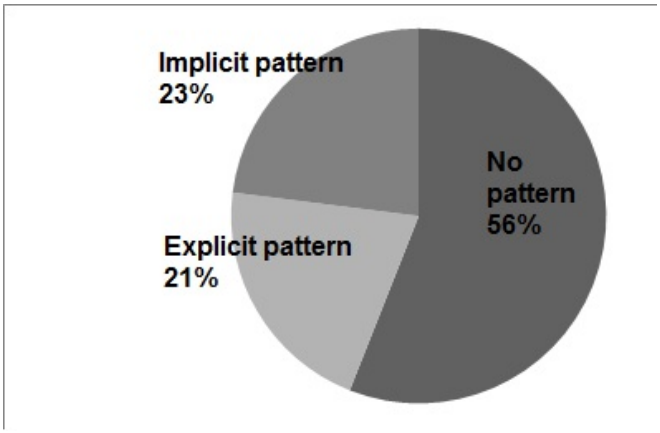

Figure 2. Extent of MT pattern usage

Category	*Patterns*	*Papers*	*Percentage (of 218)*
Rule Modularisation	14	128	58%
Architectural	13	46	21%
Optimisation	6	35	13%
Expressiveness	4	14	6%
Bidirectional	4	45	21%
Model-to-text	2	15	7%
External (GoF, etc)	4	6	3%
Total	47	289	

TABLE III. NUMBER/PERCENTAGES OF PAPERS USING EACH PATTERN CATEGORY

Modularisation and Architectural patterns: 79% of cases used such a pattern, and 66% of pattern uses were of these categories.

Category	*Total*	*Percentage* (of 363)
Rule Modularisation	189	52%
Architectural	52	14%
Bidirectional	44	12%
Optimisation	42	12%
Expressiveness	15	4%
Model-to-text	15	4%
External (GoF, etc)	6	2%
Total	363	

TABLE IV. NUMBER OF PATTERN USES IN EACH PATTERN CATEGORY

Architectural patterns have become more widely used in recent years, perhaps due to the increasing scale of transformations, and the use of systems of transformations (31 of the 46 papers using architectural patterns date from 2011 or later). Optimisation patterns have had more limited use, in part perhaps because these patterns are not well-known, and because some need specific MT language support. Other kinds of pattern, such as model-to-text, are specialised to particular forms of transformation.

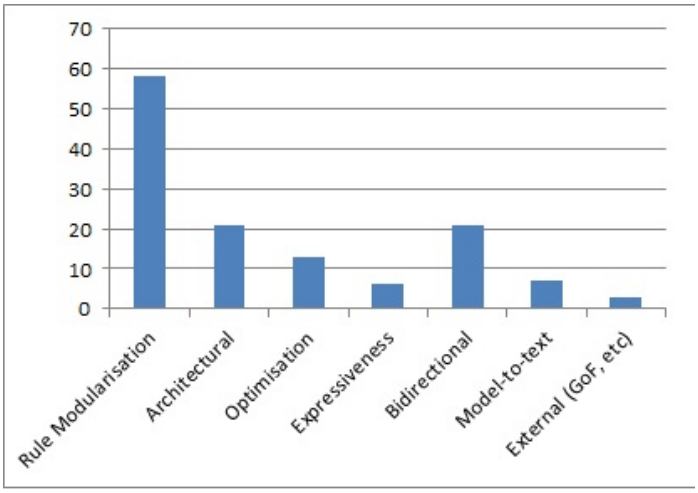

Figure 3. Frequency of use of MT pattern categories

A. The extent and type of pattern usage

Tables V, VI, VII, VIII show the number and type of applications of individual rule modularisation, architectural, optimisation and expressiveness design patterns in the surveyed papers that contain patterns. The main categories of transformation that use the pattern are identified, together with the main MT languages of these cases. Graph transformation languages are listed as GT, the Triple Graph Grammar language is listed separately as TGG. Apparent benefits of using the pattern in the cases are also given.

There are 363 pattern applications in the 218 analysed papers, with some papers including uses of several patterns (on average, 1.7 patterns are used per paper). 14 different Modularisation patterns are used, 13 Architectural patterns, 6 Optimisation patterns, 4 Bx patterns and 4 Expressiveness patterns.

Table IX shows the uses of bidirectional patterns, and Table X the model-to-text patterns uses.

External/classical patterns are used in six cases: State [107], Visitor [187], [195], [207], Observer [186] and Sliding Window [54].

The most frequently used individual patterns are: (i) Entity Splitting (vertical), 37 uses; (ii) Structure Preservation, 33 uses; (iii) Auxiliary Metamodel and Auxiliary Correspondence Model, 27 uses each, (v) Factor Expression Evaluations, 25 uses; and (vi) Transformation Chain with 22 uses. Overall, 47 different patterns and 20 of the 29 patterns described in [145] occur in transformation development cases, including all of the 11 rule modularisation patterns of [145].

Different patterns tend to be used for different categories of transformation, with Entity Splitting (vertical) more common in cases of refinement transformations, rather than in migrations, which instead use Entity Split (horizontal) and Structure Preservation. Code generators more commonly use Transformation Chain and Text Templates, whilst refactorings are the most frequent users of Replace Collection Matching and Construction and Cleanup. Factor Expression Evaluation is used frequently for Refactorings and Analysis transformations, perhaps because complex OCL expressions arise in these types of transformation, as in the case of [138].

Table XI shows the extent of correlation of pattern types and transformation types. This reveals that bx transformations are poorly supported by optimisation or architectural patterns. It also shows that architectural patterns are used broadly across all other categories of transformations. As expected, bidirectional patterns are predominately used by bx cases, and model-to-text patterns by code generation transformations.

There is considerable variation within some categories of patterns. For example, Entity Splitting (v) is most used by refinement transformations, whilst Structure Preservation is most used by migrations. Table XII shows the patterns which are used in at least 10% of the cases in each of the main transformation categories.

The frequency of use of patterns (the number of pattern usages per case) also differs across different categories of transformation (Table XIII). The gap in usage rates between bx and semantic mappings and other categories may be because there are relatively few patterns specifically for semantic mapping or bx transformations.

While some patterns are very specialised to certain kinds of transformation (eg., Lens, Text Templates) others are used more generally. Structure Preservation occurs widely across a range of transformation types (7 categories of transformation use it), although it particularly

	Refinement	*Refactoring*	*Bx*	*Migration*	*Code gen.*	*Analysis*	*Sem. Map*	*Abstraction*
Rule Mod.	66	22	18	40	10	7	11	7
Architectural	13	5	1	6	13	5	4	1
Optimisation	11	11	1	4	1	11	1	1
Expressiveness	5	4	2	0	0	3	0	0
Bidirectional	5	2	28	0	2	1	5	0
Model-to-text	3	3	0	1	7	0	0	0

TABLE XI. MT PATTERN CATEGORY USES IN MT CATEGORIES

Pattern	*MT Categories*	*Languages*	*Benefits*
Entity Splitting (vertical) 37 cases	Refinement (19) Semantic Map (5) Refactoring (5) Code generation (2) Migration (2) Bidirectional (2) HoT (1) Abstraction (1)	ATL (14) QVT-R (6) GT (3) UML-RSDS (3) ETL (2) TGG (2)	Elaborates structure from source to target
Structure Preservation 33 cases	Migration (12) Refinement (10) Bidirectional (6) Refactoring (3) Runtime (1) Semantic map (1)	ATL (14) QVT-R (7) UML-RSDS (3) GT (3) TGG (2) ETL (1)	Ensures copying of data
Auxiliary Metamodel 27 cases	Refinement (11) Migration (4) Analysis (3) Abstraction (2) Refactoring (2) Bx (1), Code gen (1) Semantic map (1)	GT (8) ATL (5) QVT-R (2) TGG (1) ETL (1)	Improves clarity, flexibility, modularisation
Recursive Descent 20 cases	Refinement (7) Migration (5) Analysis (3) Abstr. (2) Refact (1) Bx, Code gen (1)	ATL (6) QVT-R (5) ETL (2) QVT-O (1)	Functional decomposition
Map Objects Before Links 15 cases	Refinement (9) Migration (4) Refact; Bx (1)	ATL (8) UML-RSDS (3) GT; QVT-R (1)	Avoids circularity in processing
Introduce Rule Inheritance 14 cases	Refinement (4) Migration (4) Refactoring (2) Code gen; Bx (1) HoT; Sem map (1)	ATL (5) TGG (3) QVT-R, GT (1) QVT-O (1)	Factor common rule parts
Sequential Composition 11 cases	Semantic mapping (3) Refactoring (2) Code generation (2) Bx; Analysis (1) Trans; Mig (1)	GT (5) TGG (2)	Modularisation, Efficiency, Correctness
Entity Merging 10 cases	Refinement (2) Migration (2) Abstract. (2) Refact., Bx (1)	ATL (6) QVT-R (2) ETL (1) QVT-O (1)	Organises instance data integration
Construction and Cleanup 7 cases	Refactoring (5) Migration (1) Bidirectional (1)	UML-RSDS (4) GT (2) QVT-R (1)	Modularisation
Phased Construction 5 cases	Refinement (3) Code generation (1) Migration (1)	UML-RSDS (1) QVT-O (1)	Modularisation, Correctness
Entity Splitting (horizontal) 4 cases	Migration (3) Bidirectional (1)	ATL (2) GT (1) UML-RSDS (1)	Distinguishes cases in source data
Unique Instantiation 3 cases	Bidirectional (2) Code generation (1)	QVT-R (1) TGG (1)	Avoids duplicating instances
Replace Explicit calls by Implicit 2 cases	Migration (1) Model merging (1)	ETL (1)	Flexibility
Fixed-point Iteration 1 case	Refinement (1)	GT (1)	Organises fixpoint processing
Total 189	Refinement (66) Migration (40) Refactoring (22) Bidirectional (18) Semantic map (11) Code generation (10) Analysis (7) Abstraction (7) Model Merging (5)	ATL (60) GT (25) QVT-R (25) UML-RSDS (15) TGG (10) ETL (8) QVT-O (3)	

TABLE V. RULE MODULARISATION PATTERN USES

Pattern	*MT Categories*	*Languages*	*Benefits*
Transformation Chain 22 cases	Code generation (8) Refinement (7) Refactoring (3) Analysis (2) Bx (1) Model merge (1)	ATL (7) TGG (2) QVT-R (2) ETL, GT (1) QVT-O (1) UML-RSDS (1)	Compose transformations in sequence
Pre-Normalisation 7 cases	Refinement (3) Analysis (1) Migration (1) Semantic map (1)	GT (2) QVT-O (1) QVT-R (1)	Simplifies main transformation task
Factor Code Generation 6 cases	Code generation (4) Refinement (2)	ATL (2) QVT-O (1)	Modularity, extensibility
Transformation Inheritance 4 cases	Migration (2) Higher-order (1) Abstraction (1)	ATL (3) QVT-O (1)	Reuse, factorisation
Intermediate Language 3 cases	Semantic map (2) Code generation (1)	UML-RSDS (1)	Factors transformation chains
Generic Transformations 3 cases	Refactoring (2) Analysis (1)	ATL (2)	Reuse, generality
Adapter Transformations	Migration (1)		Reuse, generality
Localised MT	Refinement (1)		Modularisation
Filter Transformation	Analysis (1)	UML-RSDS (1)	Simplifies main transformation
Post-Normalisation	Semantic map (1)		Separation of concerns
Data Cleansing	Migration (1)		Simplifies main transformation
Migrate along Domain Partitions	Migration (1)		Task decomposition
Parallel Transformations			Concurrent execution
Total 52	Code generation (13) Refinement (13) Migration (6) Refactoring (5) Analysis (5) Semantic map (4) Bidirectional (1)	ATL (14) QVT-O (4) GT (3) QVT-R (3) UML-RSDS (3) TGG (2) ETL (1)	

TABLE VI. ARCHITECTURAL PATTERN USES

Pattern	*MT Categories*	*Languages*	*Benefits*
Factor out Expression Evaluations 25 cases	Refinement (7) Analysis (6) Refactoring (6) Code gen, Mig. (1) Bx; Abstr (1)	ATL (14) UML-RSDS (3) QVT-R (2) QVT-O (1)	Factors specification, may optimise execution
Rule Caching 8 cases	Refinement (4) Analysis (2) Refact, Mig. (1)	ATL (1) QVT-O (1) UML-RSDS (1)	Avoids recomputation
Restrict Input Ranges 3 cases	Analysis (2) Refactoring (1)	UML-RSDS (2)	Reduces search space
Replace Collection Matching 3 cases	Refactoring (3)	UML-RSDS (2)	Reduces search space
Implicit Copy 2 cases	Migration (2)		Avoids explicit copying
Replace Fixed-point by Bounded Iteration	Analysis (1)	UML-RSDS (1)	Efficiency
Total 42	Refactoring (11) Analysis (11) Refinement (11) Migration (4) Code gen; Bx (1)	ATL (15) UML-RSDS (9) QVT-O (2) QVT-R (2)	

TABLE VII. OPTIMISATION PATTERN USES

Pattern	*MT Categories*	*Languages*	*Benefits*
Simulate Explicit Rule Scheduling 8 cases	Refinement (3) Bidirectional (2) Refactoring (1) Analysis (1)	QVT-R (6) GT (1)	Enforces rule execution order
Collection Matching 3 cases	Refactoring (2) Refinement (1)	GT (2)	Directly match collections
Simulate Collection Matching 2 cases	Analysis (1) Refinement (1)	GT (1) UML-RSDS (1)	Individual matching for collection matching
Simulate Univ. Quantification 2 cases	Refactoring (1) Analysis (1)	GT (2)	Express $\forall$-formulae using $\exists$
Total 15	Refinement (5) Refactoring (4) Analysis (3) Bidirectional (2)	GT (6) QVT-R (6) UML-RSDS (1)	

TABLE VIII. EXPRESSIVENESS PATTERN USES

Pattern	*MT Categories*	*Languages*	*Benefits*
Auxiliary Correspondence Model 27 cases	Bidirectional (13) Refinement (5) Semantic map (3) Code generation (2) Refactoring (2)	TGG (15) GT (5) ATL (2) QVT-R (2) UML-RSDS (1)	Synchronise models; change-propagation
Lens 14 cases	Bidirectional (13) Semantic map (1)	GT (1)	Source-view consistency
Active Operations 2 cases	Bidirectional (1) Semantic map (1)	GT (2)	Change-propagation
Three-way Merge	Bidirectional (1)	QVT-R (1)	Multiple model synchronisation
Total 44	Bidirectional (28) Semantic map (5) Refinement (5) Refactoring (2) Code generation (2) Analysis (1)	TGG (15) GT (8) QVT-R (3) ATL (2) UML-RSDS (1)	

TABLE IX. BIDIRECTIONAL PATTERN USES

occurs in migration or refinement cases. Auxiliary Metamodel is a fundamental pattern which is also used widely (9 categories of transformation). In refinements it is used for tracing purposes and to store composite information during transformation processing. It is also used within several other patterns. For example, to simulate the matching $s : Set(E)$ of collections of elements, an auxiliary entity type *ESet* can be introduced whose instances reference partial collections of E instances. The collections are incrementally accumulated from E instances by auxiliary rules [91]. Single-instance matching can then be used on *ESet* instances (in contrast, if a collection-matching facility is provided in an MT language, then auxiliary entity types are not needed [85]).

It is unusual for MT patterns to be explicitly quoted or formally described (as, for example, using the pattern template of [72]). The term 'transformation design pattern' is rarely used in the analysed

Pattern	*MT Categories*	*Languages*	*Benefits*
Text Templates 11 cases	Code generation (7) Refinement (2) Migration (1) Higher-order (1)	ATL (3) ETL (1) QVT-O (1) GT (1)	Simplifies text production
Replace Abstract by Concrete Syntax 4 cases	Refactoring (3) Refinement (1)	GT (2) QVT-R (1)	Simplifies rule specifications
Total 15	Code generation (7) Refinement (3) Refactoring (3) Migration (1) Higher-order (1)	ATL (3) GT (3) ETL (1) QVT-R (1) QVT-O (1)	

TABLE X. MODEL TO TEXT PATTERN USES

Category	*Pattern usages*
Refinement (56 cases)	Entity Splitting v (19) Auxiliary Metamodel (11) Structure Preservation (10) Map Objects before Links (9) Recursive Descent (7) Transformation Chain (7) Factor Expression Evaluation (7) Auxiliary Correspondence Model (5) Introduce Rule Inherit (4) Rule Caching (4)
Bx (38 cases)	Lens (13) Auxiliary Correspondence Model (13) Structure Preservation (6)
Refactoring (30 cases)	Factor Expression Evaluation (6) Entity Split v (5) Construction and Cleanup (5) Structure Preservation (3) Replace Collection Match (3) Transformation Chain (3) Replace Abstract by Concrete (3)
Migration (27 cases)	Structure Preservation (12) Recursive Descent (5) Intro Rule Inherit (4) Map Objects before Links (4) Auxiliary Metamodel (4) Entity Split h (3)
Code Generation (20 cases)	Transformation Chain (8) Text Templates (7) Factor Code Generation (4) Entity Split v (2) Auxiliary Correspondence Model (2) Sequential Composition (2)
Analysis (17 cases)	Factor Expression Evaluation (6) Recursive Descent (3) Auxiliary Metamodel (3) Restrict Input Ranges (2) Rule Caching (2) Transformation Chain (2)
Semantic Mapping (15 cases)	Entity Splitting v (5) Sequential Composition (3) Auxiliary Correspondence Model (3) Intermediate Language (2)

TABLE XII. PATTERN APPLICATIONS IN DIFFERENT TRANSFORMATION CATEGORIES

Category	*Pattern usages*	*Cases*	*Pattern usages per case*
Migration	51	27	1.88
Refinement	103	56	1.84
Code Generation	33	20	1.65
Analysis	27	17	1.59
Refactoring	47	30	1.57
Semantic Mapping	21	15	1.4
Bx	50	38	1.31

TABLE XIII. FREQUENCY OF PATTERN APPLICATIONS IN DIFFERENT TRANSFORMATION CATEGORIES

cases. An exception is [65], which describes a behavioural pattern, Fixed-point Iteration, and gives examples of this.

Many of the pattern usages in the development cases appear to be ad-hoc and without explicit recognition of the pattern. For example, in [138], we used Replace Collection Matching implicitly, to implement a matching condition of the form "v is a set of classes such that" using an individual element matching. The rule before applying the

pattern appeared like this:

Entity ::
$v \subseteq specialisation$ & $a : v.specific.ownedAttribute$ &
$v \neq specialisation$ &
$v \rightarrow forAll(s \mid$
  $s.specific.ownedAttribute \rightarrow exists(b \mid$
    $b.name = a.name$ & $b.type = a.type))$ &
$v.size > 1 \Rightarrow$
  $Entity \rightarrow exists(e \mid e.name = name + \text{"_2_"} + a.name$ &
    $a : e.ownedAttribute$ &
    $e.specialisation = v$ &
    $Generalization \rightarrow exists(g \mid$
      $g : specialisation$ & $g.specific = e))$ &
    $v.specific.ownedAttribute \rightarrow select($
      $name = a.name) \rightarrow isDeleted()$

This rule matches strict subsets v of the subclasses of a class *self*, such that all elements of v all have a common-named and -typed attribute. It then creates a class e as a new superclass of the v classes and moves the duplicated attribute up to e.

Applying Replace Collection Matching replaces the collection matching for v by a normal instance matching for a, with v derived from a:

Entity ::
$a : specialisation.specific.ownedAttribute$ &
$v = specialisation \rightarrow select($
  $specific.ownedAttribute \rightarrow exists(b \mid$
    $b.name = a.name$ & $b.type = a.type))$ &
$v \neq specialisation$ &
$v.size > 1 \Rightarrow$
  $Entity \rightarrow exists(e \mid e.name = name + \text{"_2_"} + a.name$ &
    $a : e.ownedAttribute$ &
    $e.specialisation = v$ &
    $Generalization \rightarrow exists(g \mid$
      $g : specialisation$ & $g.specific = e))$ &
    $v.specific.ownedAttribute \rightarrow select($
      $name = a.name) \rightarrow isDeleted()$

The optimisation is based on swapping the order of the two input variable definitions, one for a and one for v, so that the value of v becomes determined by the choice of a.

The pattern had not been recognised when [138] was written, and it was only after encountering other examples of this situation that we identified it as a pattern (called Avoid Collection Matching in [149]). Subsequently, other examples were recognised, as in the *FoldEntryAction* rule of [182], which acts on a set of transitions using single-element matching on their common target state. Likewise, Simulate Collection Matching was used implicitly in [140] to incrementally construct sets of elements with a given property, before being codified in [149]. The terminology of patterns differs between authors, and especially between the model transformation and graph transformation communities. For example, phasing (Sequential Composition) [51] is termed *layering* in graph transformation languages [94], [173], and one approach (retyping) for structural preservation is termed *relabelling* in GT [31]. Map Objects before Links is also known as Entities before Relations [166]. The specific approach to Structure Preservation described in [79] is known as the Marker Relation idiom. There is not yet agreement on how to represent patterns, with [145] using an ATL-style textual notation, and [64], [166] using graph transformations. Some patterns from [145] need to be reconsidered, for example the Entity Splitting pattern seems to be clearly subdivided into two distinct subpatterns, (i) a 'horizontal' version, where different instances of one source entity type are mapped to different target entity types (as in the mapping of Pseudostates in [136], or of family members in [97]), and (ii) a 'vertical' version (Structure Elaboration) where each source instance is mapped to multiple target instances. The former variant is more suited to migration cases, and the latter to refinements.

We consider that it would be beneficial if there was wider awareness of MT patterns, and an agreed terminology, to make the use of such solutions to specification/design problems more systematic. A library of pattern application examples would help MT developers in this respect. Further research is needed to refine and improve the definitions of known patterns, to fit these more closely to actual practice.

It is unusual for MT developers to present alternative specification/design approaches for transformations. An exception is in [159], where QVT-R transformations for a UML to RDB mapping and for hierarchical to non-hierarchical state machine mappings are presented, first in the conventional Recursive Descent manner, and then in improved versions using Entity Splitting and computation of the closure of a model association to avoid recursive rule invocations. Again, a well-known repertoire of MT patterns would assist in the description and selection of alternative MT designs.

B. Benefits of MT pattern use

Concerning research question 2, precise evaluation of the benefits of pattern applications is unusual, and there has been no systematic analysis of these benefits. Improved execution times from the application of optimisation patterns are reported in [145], and improved scalability is shown in quantitative terms in [140]. However, some of these patterns are specific to UML-RSDS and are difficult to express in other MT languages. Quantitative comparisons of transformations with and without patterns are given in [217]. Positive results for the customisability achieved by the Factor Code Generation pattern is described in [71]. Efficiency improvements from Sequential Composition/layering are shown in [94]. Efficiency improvements from the use of Active Operations are shown in [113]. Quantitative evaluation of different transformation styles and languages is carried out in [10], including evaluation of the effect on performance of the removal of duplicated expression evaluations by caching. Generally, where evaluation has been carried out, pattern use does show the expected benefits. However there is a lack of systematic evaluation and comparison of transformation versions with and without patterns, and this will require further empirical research.

We have summarised the apparent or stated benefits of particular patterns in Tables V, VI, VII, VIII, IX, X and XIV.

C. Occurrences of novel patterns

Concerning research question 3, the survey also uncovered several patterns which had not previously been documented in pattern collections (ie., in the papers of Table XVI). Table XIV summarises these patterns and their benefits/costs. There are 55 applications of these patterns in total. The first 10 patterns are architectural patterns. Both Localised Model Transformations and Migrate along Domain Partitions aim to divide a single large transformation operating on an entire source model, into smaller partial transformations operating on submodels. They are related to the Phased Model Construction architectural pattern of [145]. Factor Code Generation has been widely used, and is a special case of Transformation Chain in which a model-to-model transformation precedes a model-to-text transformation. Pre-Normalisation and Data Cleansing are also specialisations of Transformation Chain, and are related to the Filter Transformation pattern of [145]. Post-Normalisation is also a specialisation of Transformation Chain. Intermediate Language is a technique for factorising multiple

transformation chains using a common intermediate representation. Where models from multiple alternative source languages may be mapped to two or more target languages, the intermediate representation reduces the number of transformations needed (Figure 4). Instead of N*M individual transformations from N sources to M targets, only N transformations to the intermediate language (IL) are needed, and M from the IL. This pattern is related to the Auxiliary Models pattern of [145]. Parallel Transformations aims to execute a transformation

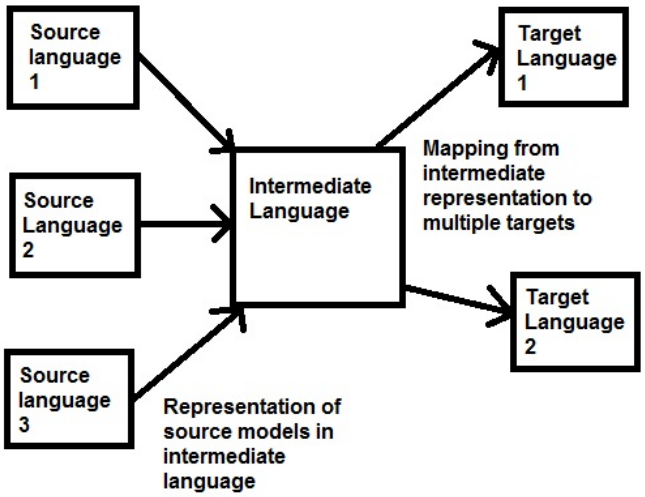

Figure 4. Intermediate Language pattern

in parallel on partitions of an input model, to increase processing capacity.

Transformation Inheritance aims to factor out common aspects of two or more similar transformations by means of an inheritance mechanism. It is a transformation-level version of Introduce Rule Inheritance from [145].

Lens [70] is a specialised bx pattern, for a restricted category of asymmetric bx transformations where the source to target mapping defines the target model $t : TL$ as a view or abstraction $get(s)$ containing partial information from the source model $s : SL$ [70], [23], [59]. A dual map $put : SL \times TL \rightarrow SL$ propagates view changes to the source, such that $put(s, get(s)) = s$ and $get(put(s,t)) = t$. Three-way Merge is a bx pattern for the management of independent updates of two views [241], it is related to Lens. Active Operations uses an Observer-style mechanism to incrementally propagate data changes from one model to another, to maintain an active source-target data correspondence (Active Expressions could be a more appropriate name for this pattern).

Replace Collection Matching has been described above. Rule Caching is an optimisation pattern related to Unique Instantiation. It enforces that transformation rules or operations are not re-executed if applied to elements that they have already processed. Instead the value or object they previously computed is returned for the element. Fixed-point Iteration defines rule structures for iterative computations.

The transformation structures of Table XIV were identified as patterns because they are specific structures of rules or of transformations, organised to solve particular specification or design problems. They therefore satisfy the definitions of MT pattern given in Section II. For example, Factor Code Generation addresses the problem of the inflexibility (and poor analysability) of a specification language-to-programming language transformation that directly generates program text.

Areas where patterns would be useful, but where we found few patterns, are in the parallelisation of transformation execution, the handling of large models (eg., by adaption of Big Data techniques such as Map/Reduce for transformations) and techniques for enhancing the genericity of transformations and their independence from specific metamodels. In order to decouple transformations from the metamodels they operate on, variants of the classical Facade, Adapter and Proxy patterns are probably needed. There are relatively few patterns specialised for semantic mapping, code generation, analysis or bx transformations.

D. Trends and influences

Concerning research question 4, Table XV identifies the changing usage of MT patterns over time. It shows the number of papers per year featuring MT design patterns, the total number of MT cases, and the percentages per year featuring patterns. Similarly to [22], we found a rise in the total number of papers using model transformations from 2005 to 2010, followed by a dip, however in our results we found a renewed increase to a higher sustained level in 2014–2016.

Year	*With patterns*	*Without*	*Total*	*% with*
2000	1	0	1	100%
2001	0	1	1	0%
2002	0	5	5	0%
2003	1	7	8	12%
2004	0	11	11	0%
2005	3	13	16	19%
2006	8	14	22	36%
2007	11	13	24	46%
2008	17	15	32	53%
2009	9	23	32	28%
2010	27	31	58	46%
2011	21	28	49	43%
2012	14	20	34	41%
2013	26	18	44	59%
2014	27	37	64	42%
2015	29	29	58	50%
2016	25	26	51	49%

TABLE XV. PERCENTAGES OF MT PAPERS USING PATTERNS PER YEAR

Considering ranges of years shows that there has been an overall trend for increasing use of transformations, and relatively increasing use of transformation patterns. For example, in the last four years (2013-2016) there were 217 MT papers with 107 using patterns (49%), compared to the preceding four years (2009-2012) with 71 of 173 papers with patterns (41%), and the period 2005-2008 with 39 of 94 papers (41%). For 2000–2004 only 2 of 26 papers used patterns (8%).

The per-year graph (Figure 5) shows these trends visually.

Table XVI lists the papers that catalogue MT patterns, and gives their citation counts. Despite the high number of citations for the top four of these papers, it is still unusual to find explicit reference in MT cases to the patterns that these papers define. Only 18 of the 218 cases cite any of these papers.

Paper	*Year*	*Citations*
Iaco et al. [102]	2008	62
Lano et al. [141]	2013	39
Lano & Kolahdouz-Rahimi [145]	2014	36
Bezivin et al. [24]	2003	34
Kurtev et al. [132]	2006	27
Johannes et al. [110]	2009	21
Cuadrado et al [49]	2008	20
Lano et al [142]	2014	10
Lano et al [143]	2013	9
Lano et al [147]	2015	0

TABLE XVI. MT DESIGN PATTERN PAPERS

Tool support for MT design patterns is in its early stages. The UML-RSDS tool incorporates optimisation patterns into its design

Papers	*New pattern*	*Summary*	*Benefits*	*Costs*
[67]	Localised Model Transformations	Decompose complex transformations into chains of cohesive small transformations operating on metamodel subsets. Related to Implicit Copy.	Increases modularity, verifiability, flexibility, changeability	Model and metamodel management
[73]	Adapter Transformations	Use pre/post processing transformations to adapt a transformation to evolved metamodels.	Supports reuse, evolution.	Effort needed to define adapters
[2], [169], [219] [92], [208] [71]	Factor Code Generation into Model-to-Model and Model-to-Text Transformations	Map to metamodel of target code language, and define text generation from this.	Modularisation, flexibility	Model and metamodel management
[7], [9], [127] [184], [168] [174], [88]	Pre-Normalisation	Simplify source model data to simplify main transformation processing	Modularisation, efficiency	Additional processing step
[56]	Post-Normalisation	Normalise/simplify transformation result	Modularisation, separates mapping and refactoring aspects	Additional processing step
[232]	Migrate along Domain Partitions	Migrate largely independent domain model parts separately	Modularisation	
[196]	Data Cleansing	Clean legacy/source data before transforming it. Related to Filter Trans, Pre-Normalisation.	Simplify main transformation by pre-processing.	Additional processing step
[146], [222] [240]	Intermediate Language	Replace N*M transformations by N+M via intermediate metamodel	Reduce redundancy, increase flexibility	Needs careful design of intermediate language
[46]	Parallel Transformations	Parallelise transformations to process large models	Increases capacity, reduces execution time	Needs coordination, model splitting, merging
[127], [217] [228], [229]	Transformation Inheritance	Reuse/adapt a transformation by specialisation/superposition	Avoids duplication	May reduce cohesion
[23], [27], [59], [70], [158] [98], [99], [164], [227], [233] [84], [128], [243], [244]	Lens	Asymmetric bx, target model computed as view of source	Source-view consistency	Restricted applicability
[112], [113]	Active Operations	Use incremental expression evaluation for change-propagation	Change-propagation	Needs specific language support
[241]	Three-way Merge	Consistently combine 2 independent updates of a model	Multiple model synchronisation	Complex semantics
[121], [138], [182]	Replace Collection Matching	Replace $s : Set(T)$ input range by $x : T$ & $s = f(x)$	Efficiency	Reduces understandability
[6], [74], [75], [148], [180] [126], [193], [239]	Rule Caching	Cache rule result to avoid re-execution. Cf.: ATL *unique lazy* rules.	Efficiency	Increased memory usage
[65]	Fixed-point Iteration	Pattern for computing result using incremental approximation steps.	Organises fixpoint processing	

TABLE XIV. NEW MT DESIGN PATTERNS INTRODUCED IN SURVEYED PAPERS

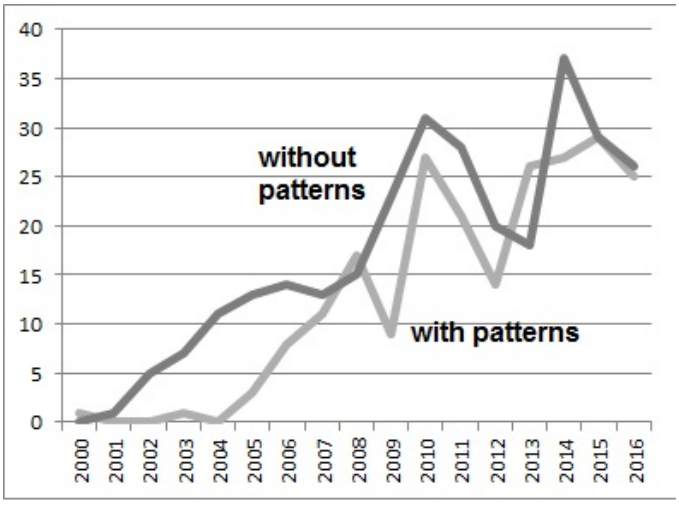

Figure 5. MT pattern cases per year

synthesis algorithm [149]. The work of [166] describes a language and tool to recognise occurrences of MT design patterns. The use of 'bad smells' to detect optimisation flaws in transformation specifications is described in [215], this approach could be used to identify places where optimisation patterns (and particularly Restrict Input Ranges) should be introduced. Because of the variability and need for creative choices in the application of patterns, fully automated selection and application of patterns is unlikely to be possible. Instead, advisory tools which can detect relevant patterns and advise transformation developers on their use, would be more appropriate. These are likely to be MT language-specific because of the differences between pattern support in different MT languages (see the following section).

E. Patterns in different MT languages

Concerning research question 5, a wide range of languages were used to implement transformations in the surveyed papers, including custom languages and programming languages. Over 73 different languages were used in the 218 patterns cases, indicating the lack of standardisation in the MT field. The most popular were ATL with 48 cases, QVT-R with 24, TGG with 23 and UML-RSDS with 10. Apart from TGG, other graph transformation languages such as VIATRA, Groove, MotTif, Henshin, AGG and GrGen are used in 32 cases in total. ETL is used in 5 cases and QVT-O in 7. Table XVII shows the languages with the main categories of transformations that used these languages, with the number of cases in each category. Table XIX shows in detail the individual patterns used in each language. Patterns with three or more uses in a language are listed.

Table XVIII shows the use of different pattern categories in different MT languages. This reveals a lack of optimisation patterns for graph transformations, which are in contrast the most likely language

MT Language	*Transformation categories*
ATL	Refinement (18)
	Migration (12)
(48 cases)	Analysis (5)
	Refactoring (4)
Graph Transformation	Refactoring (10)
	Refinement (8)
(32 cases)	Semantic map (4)
	Analysis (4)
QVT-R	Refinement (8)
	Bidirectional (7)
(24 cases)	Refactoring (4)
TGG	Bidirectional (14)
(23 cases)	Code Generation (3)
UML-RSDS	Refactoring (4)
(10 cases)	Migration (2)
QVT-O	Refinement (2)
(7 cases)	
ETL	Refinement (3)
(5 cases)	

TABLE XVII. TRANSFORMATION CATEGORIES IN MT LANGUAGES

(with QVT-R) to use expressiveness patterns. The bx languages QVT-R and TGG also seem poorly supported by optimisation patterns.

MT Language	*Patterns used*
ATL	Factor Expression Evaluation (14)
	Entity Splitting v (14)
(94 usages)	Structure Preservation (14)
	Map Objects before Links (8)
	Transformation Chain (7)
	Recursive Descent (6)
	Entity Merging (6)
	Introduce Rule Inherit (5)
	Aux. Meta (5); Text Templ. (3)
	Transformation Inheritance (3)
Graph Transformation	Auxiliary Metamodel (8)
	Auxiliary Correspondence Model (5)
(45 usages)	Sequential Composition (5)
	Entity Split v (3)
	Structure Preservation (3)
QVT-R	Structure Preservation (7)
	Entity Splitting v (6)
(40 usages)	Sim. Explicit Rule Sched (6)
	Recursive Descent (5)
UML-RSDS	Construction + Cleanup (4)
	Structure Preservation (3)
(29 usages)	Entity Split v (3)
	Map Objects before Links (3)
	Factor Expr. Evaluations (3)
TGG	Auxiliary Correspondence Model (15)
(27 usages)	Intro Rule Inherit (3)
ETL	Entity Split (v) (2)
(10 usages)	Recursive Descent (2)

TABLE XIX. DESIGN PATTERN USAGES IN MT LANGUAGES

The frequency of usage of patterns in different languages in the SLR cases (the number of pattern usages in the languages divided by the number of cases) is shown in Table XX.

Language	*Cases*	*Uses*	*Frequency of pattern use*
UML-RSDS	10	29	2.9
ETL	5	10	2
ATL	48	94	1.96
QVT-R	24	40	1.67
QVT-O	7	10	1.42
GT	32	45	1.4
TGG	23	27	1.17

TABLE XX. FREQUENCY OF PATTERN USE PER LANGUAGE

The hybrid languages with many features, such as ETL and UML-RSDS, perhaps score highly because they provide means to support many different patterns. More restricted languages such as QVT-R and graph transformation languages have less support for a wide range of patterns.

ATL is the most widely-used MT language. It provides good support for Entity Splitting, by means of rules with one `from` clause and multiple `to` clauses (for the vertical splitting pattern). Standard mode ATL already has inbuilt a Map Objects Before Links execution strategy (target instances are created in an initialisation phase for all source instances that satisfy top-level rule application conditions, then the links between target instances and their referenced instances are established in the main execution phase). Specifiers can explicitly use the pattern by using a *resolveTemp* call to lookup previously-mapped target objects to link to other target objects. Rule inheritance has only been present in more recent versions of ATL, and is infrequently used [134]. Unique lazy rules and helper attributes directly implement rule caching. Helper attributes, functions and `using` (let) variables can be used to factor out expression evaluations [239], and helper attributes can be used to define auxiliary metamodel features (but not entity types). The order of execution of rules within a transformation cannot be directly controlled, which limits the use of the Phased Construction or Sequential Composition patterns. There is no implicit copying facility in standard mode ATL, so that Structure Preservation rules must be used to copy source elements to the target. It is a unidirectional language, and external mechanisms are needed to support bx capabilities, as in [159]. Likewise, genericity [53] and composition of transformations [216] are facilities external to the core ATL language. ATL has a restricted update-in-place mode, so that its application for refactoring transformations is limited (Table XVII). To provide additional insight into the use of patterns in ATL we examined 53 of the 103 ATL Zoo transformations [18] and identified that the patterns supported by intrinsic facilities of the language (such as Map Objects before Links) are often used, whilst optimisation and refactoring patterns are less-often applied (Table XXI). There is a consistent trend for larger transformations to use more patterns than smaller ones, hence the rate of pattern usages in the repository is higher than in the SLR cases, which are often relatively short illustrative examples. The situation regarding patterns in the ATL Zoo cases is otherwise similar to that in the SLR ATL papers, with the most popular patterns in both sets of cases being Factor out Expression Evaluations, Entity Splitting (vertical), and Structure Preservation.

Pattern	*Used in cases*	*Number of cases*
Factor Exp. Eval.	1, 2, 4, 6, 7, 9-14, 18-23, 25-27, 29-34, 36-43, 45-48, 51-55	42
Structure Preservation	1, 2, 3, 6, 9, 10, 12-14, 17, 18, 22-25, 27-31, 34, 35, 37-39, 41-43, 48-50, 52, 53	33
Entity Split (vertical)	1, 3, 5, 6, 9-13, 20, 22, 25, 28, 29, 31, 34-39, 41-47, 51, 54, 55	31
Map Obj. Before Links	1, 3, 5, 6, 9-13, 35-41, 43	17
Trans. Chain	1, 7, 9, 10, 13, 22, 29, 31, 34, 36, 39, 41, 42	13
Entity Split (horizontal)	1, 9, 11-13, 19, 22, 31, 36, 40, 41, 44	12
Recursive Descent	2, 11, 12, 25, 32-34, 42, 44, 46, 47, 55	12
Rule Caching	2, 11, 12, 25, 34, 42, 46, 47, 55	9
Entity Merging	4, 6, 10, 42, 46, 48, 49	7
Factor Code Gen.	29, 39	2
Text Templates	29, 39	2
Aux. Metamodel	17, 20	2
Object Indexing	24	1
Pre-Normalisation	41	1

TABLE XXI. MT DESIGN PATTERNS IN ATL ZOO

The rate of pattern usages in the ATL zoo cases was 3.47. It is interesting to note that although the use of helper operations to factor out expression evaluations is the most common pattern in ATL, there are still many cases in the published transformations where the pattern could be further applied and duplicated expressions are still apparent, eg., in cases 2, 20, 21, 22, etc. In case 2, in helper ClassExistInLib() there is a substantial duplicated expression. This is an example of the code/design smell *Duplicate Code* [211]. Excessive use of helper functions (instead of transformation rules)

	ATL	*GT*	*QVT-R*	*TGG*	*UML-RSDS*	*QVT-O*	*ETL*
Rule Mod.	60	25	25	10	15	3	8
Architectural	14	3	3	2	3	4	1
Optimisation	15	0	2	0	9	2	0
Expressiveness	0	6	6	0	1	0	0
Bidirectional	2	8	3	15	1	0	0
Model-to-text	3	3	1	0	0	1	1

TABLE XVIII. MT PATTERN CATEGORY USES IN MT LANGUAGES

to define functionality can however hinder the comprehensibility of a transformation, producing a specification similar in style to a functional program. Rule Inheritance would simplify a number of cases where there are similar rules with slight variations, eg., in cases 1, 4, 6, 13, etc. In case 1, for example, rules AntProject2Maven and AntProject2MavenWithoutDescription are very similar and could be expressed as two specialisations of one rule. Likewise for several rules of the MOF to UML and UML to MOF transformations. It was noted that Structure Preservation rules are used in many cases to explicitly copy entities, because Implicit Copy is not available in ATL. The list of cases is given in Table XXII.

Case	*Name*
1	Ant to Maven
2	Assertion modification
3	ATL to BindingDebugger
4	ATL to Problem
5	ATL to Tracer
6	BibTeXML to DocBook
7	Book to Publication
8	CatalogueModelTransformations
9	Class to Relational
10	Code Clone Tools to SVG
11	CPL to SPL
12	Disaggregation
13	DSL to EMF
14	EliminateRedundantInheritance
17	EMF2KM3
18	EquivalenceAttributesAssociations
19	Families to Persons
20	Feature models, BIS to BPMN
21	Geometrical transformations
22	Grafcet to PetriNet
23	IEEE1471-2000 to MoDAF-AV
24	Introduce Primary Key
25	Introducing an Interface
26	Java source to Table
27	KM3 to SimpleClass
28	List refactoring
29	Make to Ant
30	Make partial role total
31	Maven to Ant
32	Measure to Table
33	Measure to XHTML
34	Measuring model repositories
35	Metah2Acme
36	MDL to GMF
37	MOF to UML
38	Monitor to Semaphore
39	Microsoft Office Excel Extractor
40	Microsoft Office Excel Injector
41	MySQL to KM3
42	OCL to R2ML
43	PathExpression to/from PetriNet
44	Ports
45	Public2Private
46	R2ML to OCL
47	R2ML to WSDL
48	Remove association classes
49	Replace association by foreign key
50	Replace inheritance by association
51	SimpleClass to SimpleRDBMS
52	Tree to List
53	UML to Java
54	UML to MOF
55	WSDL to R2ML

TABLE XXII. ATL CASES

Graph transformation (GT) languages include VIATRA [207], GrGen [31] and AGG [85]. AGG supports collection matching. GrGen supports a form of Structure Preservation via *relabelling* or retyping of elements. Limitations in the expressiveness of GT languages have been compensated by the use of patterns such as Auxiliary Metamodel to add control over rule execution orders and strategies using flag variables or traces [114], [204].

Triple Graph Grammar (TGG) is a GT language oriented towards bx definition, and it includes the Auxiliary Correspondence Model pattern as a built-in mechanism. Multiple source elements can be related to multiple target elements, so that Entity Splitting and Entity Merging are both supported.

QVT-R is a bidirectional MT language. Its rules permit multiple source and target domains, so that Entity Splitting and Entity Merging are both directly supported. QVT-R supports rule and transformation inheritance. A common style of QVT-R specification is Recursive Descent, with top-level rules calling non-top rules, which may then in turn call other non-top rules. Partial rule ordering is supported by using a *when* clause of a rule to require that certain mappings have been established prior to execution of the rule. This mechanism is also used for Map Objects before Links. Relation precedence can also be enforced by specifying conditions in *when* clauses, as in case *r*12 below. Implicit copying is not supported, and update-in-place functionality can be complex to express. Curiously, QVT-R is more often used for unidirectional transformations than for bx, in the surveyed papers and cases, perhaps due to semantic issues for bx application using QVT-R [29]. We examined further cases of QVT-R specification in the examples given for the ModelMorf tool plugin for Eclipse (cases 1 to 11, 21) or from the Medini tool examples (cases 12 to 20):

1) Hierarchical to flat state machines
2) Abstract (class) to concrete (class)
3) Class model to class model
4) DNF
5) DNF_bbox
6) Hstmtostm*
7) Mi2Si
8) SeqtoStm
9) seqtostmct
10) UMLtoRDBMS
11) UMLtoRel*
12) sampleTransformation*
13) Shapes*
14) Learn debugging*
15) AImpliesBProblem
16) r1
17) r1_not_toplevel
18) r1_incarnation_by_where
19) r12
20) r3
21) relToCore

Bidirectional transformations are indicated with *. Table XXIII shows the patterns used in these cases.

The rate of pattern usage in the QVT-R repository cases is 2.95. In both the SLR and repository cases, Structure Preservation is

Pattern	*Used in cases*	*Number of cases*
Struc. Pres.	1-8, 10-21	20
Recursive Descent	1, 3-11, 15, 17, 18, 21	14
Map Obj. Before Links	1, 8, 10-14, 16-19, 21	12
Factor Exp. Eval.	1, 6, 10, 11, 21	5
Aux. Corr. Model	6, 11, 12, 15, 16	5
Entity Split (vertical)	10, 12, 20, 21	4
Simulate Rule Sched.	4, 5	2

TABLE XXIII. MT DESIGN PATTERNS IN QVT-R

extensively used to explicitly copy source to target elements. Although QVT-R does provide a means to define auxiliary helper functions for transformations, this is less extensively used than in ATL to factor out duplicated expressions. In general, ATL seems to facilitate a wider range of specification styles than QVT-R, and clearer expression of patterns.

UML-RSDS is a hybrid MT language which supports rule ordering, and has inbuilt support for Object Indexing and other optimisation patterns. It does not support rule or transformation inheritance, or collection matching. If rules are specified as operations, then operation inheritance can be used to provide rule inheritance. Operations can also be cached, to provide rule caching similar to ATL `unique lazy` rules. UML-RSDS has direct support for update-in-place transformations, and transformations may be composed within the language. Implicit Copy is supported by the UML-RSDS tools. UML-RSDS supports a wide range of specification styles, and 16 different patterns were used in the 10 SLR UML-RSDS cases, compared to 16 for ATL from 48 papers and 16 for QVT-R from 24 (Table XXVI).

ETL [122] is a MT language similar to ATL, but with stronger support for update-in-place transformations, and a more procedural orientation. Table XXIV surveys patterns used in the ETL transformations from the Eclipse ETL repository. Again, the lack of Implicit Copy facilities leads to frequent use of Structure Preservation rules, although Implicit Copy is available in the related Flock language. Rules in ETL have a single input and possibly many outputs, so that Entity Splitting is supported. Rules can be cached. Rule inheritance is supported, as is a form of transformation inheritance/superposition. The rate of pattern usage in the ETL repository examples is 3.09.

Pattern	*Used in cases*	*Number of cases*
Struc. Pres.	1-3, 7, 9-11	7
Map Obj. Before Links	1, 5, 6, 7, 10, 11	6
Factor Exp. Eval.	6, 7, 10, 11	4
Rule Caching	1, 7, 11	3
Entity Merging	1, 6, 10	3
Recursive Descent	5, 7, 10	3
Entity Split (vertical)	6, 8	2
Rule Inheritance	1, 10	2
Phased Construction	4, 6	2
Entity Split (horizontal)	1	1
Trans. Chain	10	1

TABLE XXIV. MT DESIGN PATTERNS IN ETL

The ETL cases were:

1) Flowchart2HTML
2) CopyFlowchart
3) CopyOO
4) Flowchart2XML
5) int2out
6) OO2DB
7) Rss2Atom
8) Tree2Graph
9) UML2XSD
10) MDD-TIF
11) Argouml2Ecore

Table XXV shows how ATL, ETL, QVT-R and UML-RSDS differ in the way they support patterns.

The range of usage of patterns in different languages in the SLR cases (the number of different patterns used in the languages, divided by the number of cases) is shown in Table XXVI.

Language	*Cases*	*Patterns*	*Variety of pattern use*
ETL	5	8	1.6
UML-RSDS	10	16	1.6
QVT-O	7	11	1.57
QVT-R	24	16	0.66
GT	32	20	0.63
TGG	23	8	0.35
ATL	48	16	0.33

TABLE XXVI. VARIETY OF PATTERN USE PER LANGUAGE

We can conclude that the choice of MT language does have a significant effect on the application of MT patterns. Whilst simple patterns, such as Structure Preservation, can be used in any MT language, more complex patterns may be difficult to use in some MT languages. Additionally, the semantics of language features such as rule inheritance differs from one language to another [230]. A significant impediment is the lack of language support for a phasing mechanism [51] in some MT languages, which affects the use of rule modularisation patterns that rely on rule orderings (Sequential Composition, Phased Construction, Construction and Cleanup). Such patterns are not only useful to modularise large transformations, but can also improve transformation efficiency. Support for architectural patterns involving composition of transformations (Transformation Chains, Pre-Normalisation, etc) is also lacking in some MT languages, and is only provided by external facilities. Some facilities, such as implicit copying of elements for migration, are not supported in the main MT languages. The lack of consistency between different MT languages impairs transformation reuse to the extent that language-independent transformation designs using patterns cannot necessarily be equivalently expressed in different MT languages.

V. ANALYSIS

In this section we try to draw some conclusions based on the results for each research question.

A. Design patterns usage

Concerning Q1, we found that patterns were used in 44% of the 519 surveyed papers, although only in 21% was the usage made explicit. This relatively low usage is probably due to the low level of awareness of design patterns amongst MT developers in general (hence the 'accidental' use of patterns in 23% of cases). The low numbers of publications and citations concerning MT patterns (Table XVI) also supports this view. The low awareness of patterns means that it is rare to find combinations of related patterns being used together (eg., Map Objects before Links used with Object Indexing). The problem of awareness of patterns applies even in cases where an MT language provides direct support – such as the capability of defining helper functions and rule generalisations in ATL. In surveying the ATL Zoo we found several cases where these facilities could have been further used to simplify transformations, but were not optimally applied. In particular, manual inspection for duplicated expressions (with more than 10 tokens) in the zoo cases 29, 31, 37, 39, 40, 41 and 43 found occurrences of identical clones in each of these (2, 7, 7, 2, 3, 7 and 8 different clones in each of these cases, respectively).

The type of patterns used show a predominance of modularisation and architectural patterns, mainly concerned with improving transformation structure at an infra- or inter-transformation level. There is surprisingly low use (13% of pattern uses) of optimisation patterns, probably due both to low awareness of these patterns, and to low support for optimisation patterns in MT languages. For example, to

ATL	*ETL*	*QVT-R*	*UML-RSDS*	*Supports*
Helpers, `using` clauses, `let` expressions	Helpers	Functions	Auxiliary operations, let expressions	Factor out Expression Evaluations
Helper attributes		Keys	Use case attributes, auxiliary classes/ features	Auxiliary Metamodel
		Keys	Identity attributes	Auxiliary Correspondence Model
Rule inheritance	Rule inheritance	Rule overriding	Operation inheritance	Introduce Rule Inheritance
Called rules, `lazy` rules	Lazy rules	Non-top rules	Operations	Recursive Descent
`unique lazy` rules, helper attributes	Cached rules, operations, *equivalent*()	check-before-enforce	Cached operations	Rule Caching
Execution strategy, `resolveTemp`	Instance mapping by *equivalent*() before linking	*when* clause to enforce prior object mapping	Instance lookup by primary key, explicit rule ordering.	Map Objects before Links
External script code	Epsilon ANT script	External script	Use case ≪*include*≫	Transformation Chain

TABLE XXV. LANGUAGE ELEMENTS SUPPORTING PATTERNS IN ATL, ETL, QVT-R AND UML-RSDS

use Restrict Input Ranges, an MT language needs to define a definite order of evaluation of rule application test conditions, as in UML-RSDS [149] or Henshin [215]. The use of phasing/layering, which is both an optimisation and a modularisation mechanism, is limited in the most popular MT languages (ATL, ETL and QVT-R) by their lack of direct language support for this facility.

There is a range of complexity and abstraction level in MT patterns. At the simplest level they are close to refactorings and specification idioms: Factor out Expression Evaluations (related to the Extract Helper refactoring for ATL [239]); Structure Preservation; Entity Splitting; Recursive Descent. Such patterns require only a low level of effort to incorporate into a specification, and are widely used in different transformation types and in different MT languages. They constitute 33% of the pattern uses in the SLR cases (119 out of 363 uses). They were 71% of the ATL Zoo pattern uses, and 69% of the QVT-R repository pattern uses.

More complex patterns, requiring more significant design decisions and effort in analysis and organisation of a transformation specification, include: Auxiliary Metamodel, Phased Construction, Entity Merging, Map Objects before Links, Construction and Cleanup, Introduce Rule Inheritance, Unique Instantiation, Restrict Input Ranges, Replace Fixed-point by Bounded Iteration. A specifier has more options in how these are used, and their introduction may more impact on a specification than the more localised patterns.

Finally, highly complex patterns requiring substantial effort include architectural patterns, which may require the construction of intermediary metamodels within a transformation chain (eg., Factor Code Generation, Intermediate Language), or careful construction of auxiliary transformations (Adapter Transformations; Filter Transformation; Pre-Normalisation; Post-Normalisation) or decisions on how to divide responsibility between transformations in a chain (Localised Model Transformations).

Guidelines for the selection of patterns are given in [145]. Based on the SLR, we can refine the guidelines to suggest the use of certain patterns preferentially for given categories of transformation. For example, Construction and Cleanup and Replace Collection Matching were predominately used in refactoring transformations in the surveyed cases, and Factor Code Generation was used only in refinement and code generation transformations. This suggests that they are lower-priority patterns for other transformation categories. Tables XI and XII indicate that, for developing a code generation transformation, a specifier should look firstly at architectural patterns, and specifically at Transformation Chain, whilst for a refactoring, optimisation and rule modularisation patterns are more relevant, particularly Factor Expression Evaluation and Construction and Cleanup. More precisely, in cases of a pattern being used, a refinement transformation has a 34% likelihood of using Entity Splitting (v), whilst a bx transformation has a 34% likelihood of using Auxiliary Correspondence Model.

B. Design pattern benefits

Regarding MT pattern benefits, Q2, the widespread use of patterns shows that their benefits are perceived by many MT developers, however detailed comparative and quantitative analysis of benefits remains to be done, particularly the evaluation of improvements to transformation comprehensibility and maintainability by the use of patterns [183]. The incorporation of some patterns into MT languages (even if unconsciously) also provides evidence that they are perceived as beneficial.

C. Gaps and novel patterns

Regarding Q3, the survey uncovered 16 patterns which had not previously been explicitly documented as patterns in MT pattern collections. Ten of these patterns are architectural, three are bidirectional patterns (Lens, Active Operations and 3-way Merge), two are optimisation patterns (Replace Collection Matching, Rule Caching), and one is a modularisation pattern (Fixed-point Iteration). The architectural patterns provide advanced techniques for the organisation of systems of transformations, and four of these patterns have been applied several times in practice. Nevertheless, there remain gaps where additional patterns would be useful, especially for the architectural organisation and optimisation of bx (Table XI), for the genericity of transformations wrt metamodels, and for big data processing.

D. Trends

Regarding Q4, we found a trend for increasing pattern use over time, corresponding closely to the general growth in transformation use (Figure 5). In the earlier years of the surveyed range we came across numbers of papers which concerned transformation tasks, but which did not use model transformations, perhaps due to the lack of MT languages or awareness of MT at that time. As in the survey of [22], we observed a substantial increase in the use of MT up to 2010. Subsequently, MT are still widely used, but often appear as only one element in a paper, reflecting a shift in research focus away

from the construction of a transformation as being the main subject of a paper. It is clear that the field of MT is not dwindling, instead MT is being applied in a wider range of application areas (such as streaming transformations, combination with search-based software engineering, and with big data analysis). Specialised MT workshop series, such as BX, VOLT and AMT have been initiated in recent years.

The level of influence of papers concerned with defining and cataloguing patterns still seems quite low (they are infrequently cited in the surveyed MT development cases, and in less than 10% of the papers using patterns). Dedicated support forums (eg., on stackoverflow.com) and inbuilt pattern advisors in MT tools (eg., for Eclipse) may be a more effective means of propagating knowledge of patterns to practitioners, compared to academic publications.

E. MT languages and patterns

Regarding Q5, we found that the main MT languages all support the use of patterns, but with significant differences (Tables XVIII and XIX). While ATL has good support for many modularisation patterns, it does not support Sequential Composition or Phased Construction, and it has relatively poor support for architectural or bx patterns. QVT-R also has similar limitations for rule sequencing and architectural patterns.

We examined transformation repositories for ATL, QVT-R and ETL and found widespread use of MT patterns in these larger cases. Table XXVII summarises the frequency of use of patterns in different MT language repositories.

Language	*Patterns per case*	*Different patterns used*
ATL	3.47	14
ETL	3.09	11
QVT-R	2.95	7

TABLE XXVII. PATTERN USE IN MT LANGUAGE REPOSITORIES

Stronger support for patterns may require enhancement of MT languages. We can suggest three significant improvements that could be made in this direction:

1) Definition of *Rule groups*: a group of rules representing a layer or phase [51], [94], whose execution may be ordered before or after that of other rule groups. UML-RSDS partly provides this facility via use cases: a subtransformation with a set of postconditions can represent a rule group within a larger transformation, and can be composed into the main transformation via use case $\ll include \gg$. ETL defines a *pre* section which can be applied before other rules. Apart from supporting phasing, rule groups could also be used for other forms of modularisation, for example, to put together all rules which process the same source entity type, if there are several of these.
2) Ordering of rule application condition tests [149], [215], to facilitate optimisation based on more efficient evaluation orders, particularly restricting the range of source elements which need to be evaluated for matches. Eg., the tests

   $e1 : E1$ *and* $e2 : e1.r$ *and* $e2 : E2$

   are potentially more efficient than

   $e1 : E1$ *and* $e2 : E2$ *and* $e2 : e1.r$

3) Capabilities to compose transformations within the language, so avoiding the need for external facilities such as scripts to coordinate the execution of transformation chains, etc.

VI. THREATS TO VALIDITY

The main threats to the validity of the study are: incompleteness, publication bias, inaccuracy in classification and in data analysis.

Incompleteness may arise by the omission of relevant cases through incorrect selection procedures. We have tried to avoid this by considering both a focussed selection of published case studies of MT, from specialised MDE conferences and journals, and a general survey of the Informatics research domain using a broadly-based research database. This approach has been used in previous MT surveys, such as [22]. We obtained more cases than [22] because we covered a wider span of years, and a wider range of sources. We also included the cases of this survey in our initial selection. Unlike [22] we also carried out snowballing of references from the initial collection of papers to also consider the papers that they cite. This process potentially reduces the effect of publication bias arising from the restricted range of the initial sources. We found that the process broadened the range of cases considered in terms of MT languages and widened the span of years in which cases occurred. Our survey became more inclusive of cases that used graph transformation languages, in particular.

We compared our results to analysis of public repositories of ATL, QVT-R and ETL transformations. The SLR and repository analysis results were generally consistent, which provides evidence that for these languages at least, the SLR cases are representative of transformation cases in general.

It is possible that mistakes have been made in recognising the presence of patterns in the selected papers, we have tried to minimise this problem by having two independent reviewers for each paper. In many published transformation cases there were insufficient details of the transformation provided to determine if patterns had been used, so it is possible that some positive examples have been overlooked.

Mistakes in classification (of transformation category or of the type of pattern used) may also arise, again we cross-checked this analysis between independent reviewers. In a number of cases there were some discrepancies between reviewers and these were resolved by discussion.

Analysis results were double-checked by the main author re-examining each pattern case paper, to reduce the number of errors in analysis. The analysis presented in this expanded version of the paper is generally consistent with the smaller initial survey reported in ICMT '16. This gives us some confidence in the consistency of our data analysis procedures.

VII. RELATED WORK

To our knowledge, there is no other survey of MT design patterns in practice. In [145], some examples of applications of the proposed patterns are cited, to provide evidence for existence of the patterns. Kusel et. al. [133], [134] survey MT reuse mechanisms, including generic and higher-order transformations, but do not explicitly consider patterns. They conclude that reuse mechanisms in MT languages lack consistency, and that reuse in practice in MT developments is still quite low. Regarding ATL, they also note the low use of rule and transformation inheritance in the ATL zoo [134], and the high use of functions for factorisation. A study of MT developments in general is given in [22], this identified 82 MT cases, which we included in our survey sources. This study also identified a lack of validation in the surveyed cases. In contrast to [22], we found a general increase in the number of publications that include a model transformation, up to 2016.

VIII. CONCLUSIONS AND FUTURE WORK

We have shown that MT design pattern use is quite widespread, however this use is often unconscious and unsystematic. We identified

also a trend towards increasing use of patterns, with very few explicit uses of transformation patterns before 2005. With one exception, all the papers which formally define MT patterns also date from after 2005.

We also identified some new patterns which had not previously been formally recognised as MT patterns. We identified that different MT languages vary significantly in their support for patterns and in how patterns can be expressed in the languages.

Not considered here are MT anti-patterns, or MT structures/solutions to be avoided (see, eg., [92]). This is an important area, but is not so well developed as MT patterns, so we do not include these in this paper. Likewise, the related concept of MT idioms [4] is not considered.

Overall, we can conclude that although MT patterns represent a useful tool for the construction of high quality transformations, awareness of existing patterns needs to be raised, and improved tool support and documentation of MT patterns is needed. There are areas where further patterns appear to be necessary but have not yet been identified. Greater uniformity between the specification facilities of different MT languages would be beneficial, as would enhanced language facilities to enable the use of patterns.

REFERENCES

[1] V. Acretoaie, et al., *Transparent MT*, ICMT 2015.

[2] J. Agirre et al., *A flexible MDSD process for component based embedded control systems*, III Jornadas de Computacion Empotradas JCE, SARTCO, 2012.

[3] J. Agirre et al., *Evolving legacy MT to aggregate non-functional requirements*, MODELSWARD 2015.

[4] A. Agrawal, A. Vizhanyo, Z. Kalmar, F. Shi, A. Narayanan, G. Karsai, *Reusable Idioms and Patterns in Graph Transformation Languages*, GraBaTs 2004, Electronic notes in Theoretical Computer Science, pp. 181–192, 2005.

[5] D. Akehurst, S. Kent, O. Patrascoiu, *A relational approach to defining and implementing transformations between metamodels*, SoSyM vol. 2, no. 4, 2003, pp. 215–239.

[6] D. Akehurst et al., *SiTra: simple transformations in Java*, MODELS 2005 Workshops, 2006.

[7] B. Al-Batram et al., *Semantic clone detection for MBD of embedded systems*, MODELS 2011.

[8] E. Alves, P. Machado, F. Ramalho, *Automatic generation of built-in contract test drivers*, SosyM, vol. 13, no. 3, 2014.

[9] M. van Amstel, M. van den Brand, Z. Protic, T. Verhoeff, *Transforming process algebra models into UML state machines*, ICMT 2008.

[10] M. van Amstel, S. Bosems, I. Kurtev, L. Pires, *Performance in model transformations: experiments with ATL and QVT*, ICMT 2011, LNCS 6707, pp. 198–212, 2011.

[11] A. Anjorin et al., *Complex attribute manipulation in TGGs with constraint-based programming techniques*, BX 2012.

[12] A. Anjorin, M. Lauder, *A solution to the Flowgraphs case study using Triple Graph Grammars and eMoflon*, TTC 2013.

[13] A. Anjorin et al., *A systematic approach and guidelines to developing a TGG*, BX 2015.

[14] A. Anwar et al, *Towards a generic approach for model composition*, ICSEA 2008.

[15] A. Anwar et al., *A rule-driven approach for composing viewpoint-oriented models*, JoT vol. 9, 2010.

[16] T. Arendt et al., *Henshin: Advanced concepts and tools for in-place EMF model transformations*, MODELS 2010, LNCS 6394, 2010.

[17] C. Atkinson et al., *Enhancing classic transformation languages to support multi-level modeling*, Sosym vol. 14, 2015.

[18] ATL Zoo, www.eclipse.org/atl/atlTransformations, accessed November 30th, 2016.

[19] T. Baar, J. Whittle, *On the usage of concrete syntax in MT rules*, PSI 2006.

[20] Z. Balogh, D. Varro, *Model transformation by example using inductive logic programming*, SoSyM, 2009.

[21] B. Barroca et al., *DSLTrans: a Turing incomplete transformation language*, SLE 2010.

[22] E. Batot, H. Sahraoui, E. Syriani, P. Molins, W. Sboui, *Systematic mapping study of model transformations for concrete problems*, Modelsward 2016, pp. 176–183.

[23] G. Bergmann, C. Debreceni, I. Rath, D. Varro, *Query-based access control for secure collaborative modelling using bidirectional transformations*, MODELS 2016.

[24] J. Bezivin, F. Jouault, J. Palies, *Towards Model Transformation Design Patterns*, ATLAS group, University of Nantes, 2003.

[25] E. Biermann, C. Ermel, G. Taentzer, *Formal foundation of consistent EMF model transformations*, SoSyM (2012) 11: 227–250.

[26] P. Bocciarelli, A. D'Ambrogio, *A model-driven method for enacting the design-time QoS analysis of business processes*, Sosym, 13: 573–598, 2014.

[27] A. Bohannon et al., *Boomerang: resourceful lenses for string data*, POPL '08, 2008.

[28] K. Born, S. Schulz, D. Struber, S. John, *Solving the CRA case with Henshin and a Genetic Algorithm*, TTC 2016.

[29] J. Bradfield, P. Stevens, *Enforcing QVT-R with mu-calculus and games*, FASE 2013.

[30] T. Buchmann, F. Schwagerl, *Using meta-code generation to realize higher-order MT*, ICSOFT 2013.

[31] S. Buchwald, E. Jakumeit, *A GrGen.NET solution of the model migration case*, TTC 2010.

[32] P. Buneman, et al., *UnQL: a query language and algebra for semistructured data based on structural recursion*, The VLDB Journal, 2000.

[33] E. Burger, O. Schneider, *Translatability and translation of updated views in ModelJoin*, ICMT 2016.

[34] L. Burgueno, et al., *Static fault localisation in MT*, IEEE Trans. SE., vol. 41, 2014.

[35] F. Buttner, et al., *On validation of ATL transformation rules by transformation models*, MoDeVVa 2011.

[36] F. Buttner, et al., *Verification of ATL transformations using transformation models and model finders*, ICFEM 2012.

[37] F. Buttner et al., *On verifying ATL transformations using 'off-the-shelf' SMT solvers*, MODELS 2012.

[38] F. Buttner et al., *Checking MT refinement*, ICMT 2013.

[39] J. Cabot, R. Clariso, E. Guerra, J. de Lara, *Verification and validation of declarative model-to-model transformations through invariants*, JSS, 2009.

[40] J. Cabot et al., *Synthesis of OCL pre-conditions for GT rules*, ICMT 2010.

[41] D. Calegari, A. Delgado, *Rule chains coverage for testing QVT-R*, AMT 2013.

[42] V. de Castro, J. Vara, E. Marcos, *Model transformations for service-oriented web applications development*, MDWE 2007.

[43] Z. Cheng et al., *A sound execution semantics for ATL via translation validation*, ICMT 2015, LNCS 9152, 2015.

[44] Z. Cheng, M. Tisi, *Towards incremental deductive verification for ATL*, VOLT 2016.

[45] A. Cicchetti, B. Meyers, M. Wimmer, *Abstract and concrete syntax migration of instance models*, TTC 2010.

[46] C. Clasen et al., *Transforming very large models in the cloud*, MDE on and for the Cloud, 2012.

[47] V. Cosentino, M. Tisi, F. Buttner, *Analysing flowgraphs with ATL*, TTC 2013.

[48] J. Cuadrado et al., *RubyTL: a practical extensible transformation language*, ECMDA-FA 2006.

[49] 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.

[50] J. Cuadrado, J. Molina, *Approaches for MT reuse: factorization and composition*, ICMT 2008.

[51] J. Cuadrado, J. Molina, *Modularisation of model transformations*

through a phasing mechanism, SoSyM vol. 8, no. 3, 2009, pp. 325–345.

[52] J. Cuadrado et al., *Generic MT: write once, reuse everywhere*, ICMT 2011.

[53] J. Cuadrado, E. Guerra, J. de Lara, *A component model for model transformations*, IEEE TSE, vol. 7, no. 7, 2013.

[54] J. Cuadrado, J. de Lara, *Streaming model transformations: Scenarios, challenges and initial solutions*, ICMT 2013.

[55] J. Cuadrado, E. Guerra, J. de Lara, *Quick fixing ATL model transformations*, MODELS 2015.

[56] A. Cunha, A. Garis, D. Riesco, *Translating between Alloy specifications and UML class diagrams annotated with OCL*, SoSyM vol. 14, 2015, pp. 5–25.

[57] K. Czarnecki, S. Helsen, *Feature-based survey of model transformation approaches*, IBM System Journal, 45(3), pp. 621–645, 2006.

[58] A. Demuth, et al., *Constraint-driven modelling through transformation*, SoSym vol. 14, no. 2, 2015.

[59] Z. Diskin, Y. Xiong, K. Czarnecki, *From state- to delta-based bidirectional model transformations*, ICMT 2010.

[60] J. Dong et al., *QVT-based MT for design pattern evolutions*, IASTEAD IMSA, 2010.

[61] J. Dyck et al., *Towards the automatic verification of behaviour preservation at the transformation level for operational MT*, AMT 2015.

[62] H. Ehrig, et al., *Information preserving bidirectional MT*, FASE 2007.

[63] C. Eickhoff, T. George, S. Lindel, A. Zundorf, *SDMLib solution to the MovieDB case*, TTC 2014.

[64] H. Ergin, E. Syriani, *A unified template for MT design patterns*, PAME, 2015.

[65] H. Ergin, E. Syriani, *Identification and application of a model transformation design pattern*, ACMSE '13, 2013.

[66] H. Ergin, E. Syriani, *AToMPM solution for the IMDb case study*, TTC 2014.

[67] A. Etien, A. Muller, T. Legrand, R. Paige, *Localised model transformations for building large-scale transformations*, SoSyM vol. 14, no. 3, 2015, pp. 1189–1213.

[68] M. Didonet Del Fabro, P. Valduriez, *Towards the efficient development of MT using model weaving and matching transformations*, Sosym vol. 8, 2009.

[69] F. Fleurey, et al., *MDE for software migration in a large industrial context*, MODELS 2007.

[70] J. Foster, M. Greenwald, J. Moore, B. Pierce, A. Schmitt, *Combinators for bi-directional tree transformations*, ACM Trans. Prog. Lang. Sys., 29(3), 2007.

[71] M. Funk, A. Nysen, H. Lichter, *From UML to ANSI-C: an Eclipse-based code generation framework*, RWTH, 2007.

[72] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[73] K. Garces, J. Vara, F. Jouault, E. Marcos, *Adapting transformations to metamodel changes via external transformation composition*, SoSyM (2014) 13: 789–806.

[74] A. Garcia-Dominguez, D. Kolovos, *Models from Code or Code as a Model?*, OCL 2016.

[75] C. Gerking, C. Heinzemann, *Solving the Movie database case with QVT-O*, TTC, 2014.

[76] H. Giese, S. Glesner, J. Leitner, W. Schafer, R. Wagner, *Towards verified model transformations*, MODEVA 2006.

[77] H. Giese, R. Wagner, *From model transformation to incremental bidirectional model synchronisation*, SoSyM vol. 8, no. 1, 2009, pp. 21–43.

[78] H. Giese, S. Hildebrant, S. Neumann, *Model synchronisation at work: keeping SysML and AUTOSAR models consistent*, ICMT 2010.

[79] T. Goldschmidt, G. Wachsmuth, *Refinement transformation support for QVT relational transformations*, ENCS, 2011.

[80] S. Gottmann et al., *Correctness and completeness of generalised model synchronisation based on TGGs*, AMT 2013.

[81] S. Gottmann et al., *Towards the propagation of model updates*, BX 2016.

[82] J. Greenyer, E. Kindler, *Comparing relational MT technologies*, Sosym vol. 9, 2010.

[83] J. Greenyer, J. Rieke, *Applying advanced TGG concepts*, AGTIVE 2011.

[84] H. Grohne et al., *Formalising semantic bidirectionalism with dependent types*, EDBT/ICDT 2014.

[85] R. Gronmo, B. Moller-Pedersen, *From sequence diagrams to state machines by graph transformation*, ICMT 2010. Also in JoOT, vol. 10, 2011.

[86] R. Gronmo, S. Krogdahl, B. Moller-Pedersen, *A collection operator for graph transformations*, SoSyM (2013) 12: 121–144.

[87] E. Guerra, J. de Lara, F. Orejas, *Inter-modelling with patterns*, SoSyM (2013) 12: 145–174.

[88] E. Guerra, J. de Lara, D. Kolovos, R. Paige, O. Marchi dos Santos, *Engineering model transformations with transML*, SoSyM (2013) 12: 555–577.

[89] E. Guerra, J. de Lara, *Colouring: execution, debug and analysis of QVT-R transformations through coloured Petri nets*, SoSyM (2014) 13: 1447–1472.

[90] E. Guerra, M. Soeken, *Specification-driven MT testing*, Sosym, vol. 14, no. 2, 2015.

[91] C. Heinzemann, J. Suck, R. Jubeh, A. Zundorf, *Topology analysis of car platoons merge with FujabaRT*, TTC 2010.

[92] Z. Hemel, L. Kats, D. Groenewegen, E. Visser, *Code generation by model transformation: a case study in transformation modularity*, SoSyM (2010) 9: 375–402.

[93] F. Hermann, N. Nachtigall, B. Braatz, S. Gottmann, T. Engel, *Solving the FIXML2Code case study with HenshinTGG*, TTC 2014.

[94] F. Hermann, S. Gottmann, N. Nachtigall, H. Ehrig, B. Braatz, G. Morelli, A. Pierre, T. Engel, C. Ermel, *Triple graph grammers in the large for translating satellite procedures*, ICMT 2014.

[95] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, Y. Xiong, S. Gottmann, T. Engel, *Model synchronisation based on triple graph grammars*, SoSyM (2015), 14: 241–269.

[96] A. S-B. Herrera et al., *An OCL-based Bridge from Concrete to Abstract Syntax*, OCL 2015.

[97] S. Hidaka, M. Tisi, J. Cabot, Z. Hu, *Feature-based classification of bidirectional transformation approaches*, SoSyM 15: 907–928, 2016.

[98] G. Hinkel, *Change propagation in an internal MT language*, ICMT 2015.

[99] M. Hoffmann, et al., *Symmetric Lenses*, POPL' 2011.

[100] T. Horn, *Solving the class diagram restructuring transformation case with FunnyQT*, TTC 2013.

[101] H. Hoyos, J. Chavarriaga, P. Gomez, *Solving the FIXML case study using Epsilon and Java*, TTC 2014.

[102] 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.

[103] M. Igamberdiev et al., *Verification of the CD2RDBMS Transformation Case in Flora-2*, VOLT 2015.

[104] P. Inostroza, et al., *Tracing program transformation with string origins*, ICMT 2014.

[105] P. Inostroza, T. van der Storm, *The TTC 2014 FIXML case: Rascal solution*, TTC 2014.

[106] P. Inostroza, T. van der Storm, *The TTC 2014 Movie database case: Rascal solution*, TTC 2014.

[107] M. Iqbal, A. Arcuri, L. Briand, *Environment modeling and simulation for automated testing of soft real-time embedded software*, SoSyM 14: 483–524, 2015.

[108] B. Izso, A. Hegedus, G. Bergmann, A. Horvath, *PN2SC case study: an EMF-IncQuery solution*, TTC 2013.

[109] A. Jimenez, et al., *Using ATL to support MDD of RubyTL transformations*, MtATL, 2011.

[110] 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.

[111] F. Jouault, I. Kurtev, *Transforming models with ATL*, MODELS 2005 Workshops, LNCS vol. 3844, 2006.

[112] F. Jouault, O. Beaudoux, *On the use of active operations for incremental Bidirectional Evaluation of OCL*, OCL 2015.

[113] F. Jouault, O. Beaudoux, *Efficient OCL-based Incremental Transformations*, OCL 2016.

[114] S. Jurack, J. Tietje, *Solving the TTC 2011 Reengineering case with Henshin*, TTC 2011.

[115] L. Kapova et al., *Evaluating maintainability with code metrics for model-to-model transformations*, Research into Practice, Springer, 2010.

[116] R. Khadka et al., *WSCDL to WSBPEL: A Case Study of ATL-based transformation*, MtATL 2011.

[117] Y. Khan, M. Al-Attar, *Using model transformation to refactor use case models based on antipatterns*, Inf. Syst. Front. (2016), 18: 171–204.

[118] B. Kitchenham, *Procedures for defining systematic reviews*, 2004.

[119] M. Kleiner, M. Del Fabro, D. De Santos, *Transformation as search*, ECMFA 2013, LNCS 7949, pp. 54–69, 2013.

[120] N. Koch, *Transformation techniques in the MDD process of UWE*, ICWE '06, ACM, 2006.

[121] S. Kolahdouz-Rahimi et al., *Evaluation of MT approaches for model refactoring*, Sci. Comp. Prog., vol. 85, 2014.

[122] D. Kolovos, R. Paige, F. Polack, *The Epsilon Transformation Language*, ICMT 2008.

[123] D. Kolovos, et al., *Update transformations in the small with the Epsilon wizard language*, JoT, 2007.

[124] D. Kolovos et al., *Implementing the interactive television applications case study using Epsilon*, MDDTIF 2007.

[125] D. Kolovos et al, *Merging models with the Epsilon Merging Language (EML)*, MODELS 2006.

[126] D. Kolovos et al., *The Epsilon pattern language*, MiSE 2017.

[127] A. Kraas, *Realizing model simplifications with QVT-O*, OCL 2014.

[128] M. Kramer, K. Rakhman, *Automated inversion of attribute mappings in bx*, BX 2016.

[129] F. Krikava, *Solving the CRA case with SIGMA*, TTC 2016.

[130] T. Kuhne et al., *Explicit transformation modelling*, MODELS 2009 Workshops, LNCS 6002, 2009.

[131] G. Kulcsar, E. Leblebici, A. Anjorin, *A solution to the FIXML case study using Triple Graph Grammars and eMoflon*, TTC 2014.

[132] I. Kurtev, K. Van den Berg, F. Joualt, *Rule-based modularisation in model transformation languages illustrated with ATL*, Proceedings 2006 ACM Symposium on Applied Computing (SAC 06), ACM Press, pp. 1202–1209, 2006.

[133] A. Kusel, J. Schonbock, M. Wimmer, G. Kappel, W. Retschitzegger, W. Schwinger, *Reuse in model-to-model transformation languages: are we there yet?*, SoSyM vol. 14, no. 2, 2015.

[134] A. Kusel, J. Schonbock, M. Wimmer, W. Retschitzegger, W. Schwinger, G. Kappel, *Reality check for MT reuse: the ATL transformation zoo case study*, AMT 2013.

[135] M. Kuznetsov, *UML model transformation and its application to MDA technology*, Prog. Computer Soft., vol 33, 2007.

[136] K. Lano, S. Kolahdouz-Rahimi, *Model migration transformation specification in UML-RSDS*, TTC 2010.

[137] K. Lano, S. Kolahdouz-Rahimi, *Solving the TTC 2011 model migration case with UML-RSDS*, TTC 2011.

[138] K. Lano, S. Kolahdouz-Rahimi, *Case study: class diagram restructuring*, TTC 2013.

[139] K. Lano, *Solving the Petri-nets to Statecharts transformation case with UML-RSDS*, TTC 2013.

[140] K. Lano, S. Yassipour-Tehrani, *Solving the TTC 2014 Movie Database Case with UML-RSDS*, TTC 2014.

[141] K. Lano, S. Kolahdouz-Rahimi, *Constraint-based specification of model transformations*, Journal of Systems and Software, vol. 88, no. 2, February 2013, pp. 412–436.

[142] K. Lano, S. Kolahdouz-Rahimi, I. Poernomo, J. Terrell, S. Zschaler, *Correct-by-construction synthesis of model transformations using design patterns*, SoSyM, vol. 13, no. 2, May 2014, pp. 412–436.

[143] K. Lano, S. Kolahdouz-Rahimi, *Optimising Model-transformations using Design Patterns*, MODELSWARD 2013.

[144] K. Lano et al., *A framework for MT verification*, FACS, 2014.

[145] K. Lano, S. Kolahdouz-Rahimi, *Model-transformation Design Patterns*, IEEE Transactions in Software Engineering, vol 40, 2014.

[146] K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani, *Analysis of hybrid MT language specifications*, FSEN 2015.

[147] K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani, *Patterns for Specifying Bidirectional Transformations in UML-RSDS*, ICSEA 2015.

[148] K. Lano, S. Yassipour-Tehrani, *Solving the Class Responsibility Assignment Case with UML-RSDS*, TTC 2016.

[149] K. Lano, *Agile model-based development using UML-RSDS*, CRC Press, 2016.

[150] K. Lano, S. Yassipour-Tehrani, *Verified bidirectional transformations by construction*, VOLT, 2016.

[151] M. Lawley, J. Steel, *Practical declarative Model Transformation with Tafkat*, MODELS 2005, LNCS 3844, 2006.

[152] E. Leblebici, *Towards a graph grammar based approach to inter-model consistency checks*, BX 2016.

[153] L. Lengyel, et al., *Model transformation with a visual control flow language*, Int. Journal of Computer and Information Eng., vol. 2, no. 2, 2008.

[154] D. Li, X. Li, V. Stolz, *FIXML to Java, C# and C++ transformations with QVTR-XSLT*, TTC 2014.

[155] G. Loniewski, E. Insfran, S. Abrahao, *A systematic review of the use of requirements engineering techniques in model-driven development*, MODELS 2010, Springer, pp. 213–227.

[156] L. Lucio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. Selim, E. Syriani, M. Wimmer, *Model transformation intents and their properties*, SoSyM (2016) 15: 647–684.

[157] K. Ma et al., *A relational approach to MT with QVT-R supporting model synchronization*, Journal Univ. Comp. Sci., vol 17, 2011.

[158] N. Macedo et al., *Composing least-change lenses*, BX 2013, ECEASST vol. X, 2013.

[159] N. Macedo, A. Cunha, *Least-change bidirectional model transformation with QVT-R and ATL*, SoSyM (2016) 15: 783–810.

[160] A. di Marco, S. Pace, *Model-driven approach to Agilla agent generation*, IWCMC 2013, IEEE.

[161] S. Markovic, T. Baar, *Refactoring OCL annotated UML class diagrams*, SoSyM vol. 7, no. 1, 2008, pp. 25–47.

[162] S. Markovic, T. Baar, *Semantics of OCL specified with QVT*, SoSyM vol. 7, no. 4, 2008, pp. 399–422.

[163] N. Matragkas et al., *A traceability-driven approach to MT testing*, AMT 2013.

[164] K. Matsuda et al., *Bidirectionalisation transformation based on automatic derivation of view complement functions*, ICFP '07, 2007.

[165] M. McGill, B. Cheng, *Test-driven development of a MT with Jemtte*, Research Gate, 2008.

[166] C. Mokaddem, H. Sahraoui, E. Syriani, *Towards rule-based detection of design patterns in model transformations*, SAM 2016, LNCS vol. 9959, pp. 211–225, 2016.

[167] T. Moreira et al., *Generating VHDL source code from UML models of embedded systems*, WPPL 2010.

[168] O. Muliawan, P. van Gorp, A. Keller, D. Janssens, *Executing a standard compliant transformation model on a non-standard platform*, IEEE Int. Conf. Software Testing Verification and Validation Workshop, 2008

[169] P. Muller, F. Fondement, F. Fleurey, M. Hassenforder, R. Schnekenburger, S. Gerard, J-M. Jezequel, *Model-driven analysis and synthesis of textual concrete syntax*, SoSyM vol. 7, no. 4, 2008, pp. 423–441.

[170] N. Nachtigall et al., *Towards Domain Completeness for Model Transformations Based on Triple Graph Grammars*, VOLT 2016.

[171] S. Nalchigar et al., *Towards a catalog of non-functional requirements for MT*, AMT 2013.

[172] A. Narayanan, G. Karsai, *Verifying MT by structural correspondence*, GT-VMT 2008.

[173] C. Natschlager, F. Kossak, K. D. Schewe, *Deontic BPMN: a powerful extension of BPMN with a trusted model transformation*, SoSyM (2015) 14: 765–793.

[174] A. Nayak, D. Samanta, *Synthesis of test scenarios using UML activity diagrams*, SoSyM vol. 10, no. 1, 2011, pp. 63–89.

[175] B. Oakes, et al., *Fully verifying transformation contracts for declarative ATL*, MODELS 2015.

[176] B. Oakes, et al., *Full contract verification for ATL using symbolic execution*, Soft. Sys. Model, July 2016.

[177] S. Peldszus el al., *Java refactoring case using eMoflon*, TTC 2015.

[178] S. Peldszus et al., *Incremental co-evolution of Java programs based on bidirectional GT*, PPPJ 2015.

[179] R. Perez-Castillo, I. Garcia-Rodriguez de Guzman, M. Piattini, *Implementing business process recovery patterns through QVT transformations*, ICMT 2010.

[180] J. von Pilgrim, *Ecore2GenModel with Mitra and GEF3D*, TTC 2010.

[181] E. Planas et al., *Two Basic Correctness Properties for ATL Transformations: Executability and Coverage*, MtATL 2011.

[182] I. Potters, *Rule-based update transformations and their application to model refactorings*, SoSyM vol. 4, no. 4, 2005, pp. 368–385.

[183] L. Prechelt, *An experiment on the usefulness of design patterns*, Univ. Karlsruhe Tech report 9/1997, 1997.

[184] T. Rahmani, D. Oberle, M. Dahms, *An adjustable transformation from OWL to Ecore*, MODELS 2010.

[185] M. Rahmouni, S. Mbarki, *MDA-based ATL transformation to generate MVC2 web models*, IJCSIT vol 3., 2011.

[186] I. Rath et al., *Live MT driven by incremental pattern matching*, ICMT 2008.

[187] A. Razavi, K. Kontogiannis, *Partial evaluation of MT*, ICSE 2012.

[188] J. Reimann, M. Seifert, U. Afmann, *On the reuse and recommendation of model refactoring specifications*, SoSyM (2013) 12: 579–596.

[189] A. Rentschler et al., *Remodularising legacy MT with automatic clustering techniques*, AMT 2014.

[190] J. Rivera et al., *Orchestrating ATL model transformations*, MtATL 2009.

[191] H. H. Rodriguez, D. Kolovos, *Declarative Model Transformation Execution Planning*, OCL 2016.

[192] R. Romeikat, S. Roser, P. Mullender, B. Bauer, *Translation of QVT Relations into QVT Operational Mappings*, ICMT 2008.

[193] L. Rose, D. Kolovos, R. Paige, F. Polack, *Migrating activity diagrams with Epsilon Flock*, TTC 2010.

[194] L. Rose, D. Kolovos, R. Paige, F. Polack, S. Poulding, *Epsilon Flock: a model migration language*, SoSyM (2014) 13: 735–755.

[195] S. Rottger, S. Zschaler, *Tool support for refinement of non-functional specifications*, SoSyM vol. 6, no. 2, 2007, pp. 185–204.

[196] A. Ruping, *Transform! – Patterns for data migration*, EuroPLoP 2010.

[197] J. Santos, A. Moreira, J. Araujo, M. Goulao, *Increasing quality in scenario modelling with MDD*, 7th Int. Conf. on Quality of Information and Communications Technology, 2010.

[198] I. Sasano et al., *Toward bidirectionalisation of ATL with GRoundTram*, ICMT 2011.

[199] B. Schatz, *UML model migration with PETE*, TTC 2010.

[200] G. Selim et al., *Specification and verification of graph-based MT properties*, ICGT 2014.

[201] G. Selim et al., *Finding and fixing bugs in MT with formal verification*, AMT@MoDELS, 2015.

[202] G. Selim, S. Wang, J. Cordy, J. Dingel, *Model transformations for migrating legacy deployment models in the automotive industry*, SoSyM (2015), 14: 365–381.

[203] O. Semerath et al., *Incremental backward change propagation of view models by logic solvers*, MODELS 2016.

[204] W. Smid, A. Rensink, *Class diagram restructuring with Groove*, TTC 2013.

[205] M. dos Santos Soares, J. Vrancken, *A metamodeling approach to transform UML 2.0 sequence diagrams to Petri nets*, IASTED 2008.

[206] H. Song, G. Huang, F. Chauvel, W. Zhang, Y. Sun, W. Shao, H. Mei, *Instant and incremental QVT transformation for runtime models*, MODELS 2011, LNCS 6981, pp. 273–288.

[207] D. Stein, G. Szarnyas, I. Rath, *Java refactoring: a VIATRA solution*, TTC 2015.

[208] K. Stenzel, N. Moebius, W. Reif, *Formal verification of QVT transformations for code generation*, SoSyM vol. 14, no. 2, 2015.

[209] P. Stevens, *Bidirectional MT in QVT: semantic issues and open questions*, Sosym vol. 9, 2010.

[210] P. Stevens, *A simple game-theoretic approach to checkonly QVT relations*, Sosym, vol. 12, no. 1, Feb. 2013.

[211] G. Suryanarayana, G. Samarthyam, T. Sharma, *Refactoring for software design smells: managing technical debt*, Morgan Kaufmann, 2014.

[212] E. Syriani, H. Ergin, *Operational semantics of UML activity diagrams*, MoDRE 2012.

[213] G. Szarnyas, O. Semerath, B. Izso, C. Debreceni, *Movie database case: an EMF-INCQUERY solution*, TTC 2014.

[214] G. Taentzer, et al., *MT by graph transformation: a comparative survey*, MT in Practice, MODELS 2005.

[215] M. Tichy, et al., *Detecting performance bad smells for Henshin MT*, AMT 2013.

[216] M. Tisi, J. Cabot, *Combining transformation steps in ATL chains*, 2nd Workshop on MT Composition, 2011.

[217] M. Tisi, J. Cabot, F. Jouault, *Improving higher-order transformations support in ATL*, ICMT 2010.

[218] M. Tisi, et al., *Lazy execution of model-to-model transformations*, MODELS 2011.

[219] V. Torres, P. Giner, V. Pelechano, *Developing BP-driven web applications through the use of MDE techniques*, SoSyM vol. 11, no.4, 2012, pp. 609–631.

[220] J. Troya, A. Vallecillo, *A rewriting logic semantics for ATL*, JOT, 2011.

[221] J. Troya et al., *Towards systematic mutations for and with ATL MT*, ICSTW 2015.

[222] B. Vanhooff et al., *Towards a transformation chain modelling language*, Embedded Computer Systems workshop, 2006.

[223] B. Vanhooff, Y. Berbers, *Breaking up the transformation chain*, OOPSLA 2012.

[224] D. Varro et al., *Road to a reactive and incremental MT platform*, SoSoym vol. 15, 2016.

[225] A. Vieira, F. Ramalho, *Static Analyzer for Model Transformations*, MtATL 2011.

[226] T. Vogel, et al., *Incremental model synchronisation for efficient runtime monitoring*, MODELS 2009 workshops, LNCS vol. 6002, 2010.

[227] J. Voigtlander, et al., *Combining syntactic and semantic bidirectionalism*, ICFP '10, 2010.

[228] D. Wagelaar, *Composition techniques for rule-based MT languages*, ICMT 2008.

[229] D. Wagelaar et al., *Module Superposition: a composition technique for rule-based MT languages*, SoSyM vol. 9, 2009.

[230] D. Wagelaar et al., *Towards a general composition semantics for rule-based MT*, MODELS 2011.

[231] D. Wagelaar, *ATL solution to train benchmark case*, TTC 2015.

[232] M. Wagner, T. Wellhausen, *Patterns for data migration projects*, EuroPLoP 2010.

[233] M. Wang, J. Gibbons, N. Wu, *Incremental Updates for Efficient Bidirectional Transformations*, ICFP '11, 2011.

[234] D. O. Wendell et al., *An MDE approach for automatic code generation from UML/MARTE to OpenCL*, Computing in Science and Engineering, vol. 15, 2012.

[235] E. Willink, *Optimised declarative transformation: first Eclipse QVTC results*, BigMDE 2016.

[236] E. Willink, *Local optimisations in Eclipse QVTC and QVTr*, EXE 2016.

[237] M. Wimmer, A. Schauerhuber, M. Strommer, W. Schwinger, G. Kappel, *A semi-automatic approach for bridging DSLs with UML*, DSM '07, 2007.

[238] M. Wimmer, et al., *On using in-place transformations for model co-evolution*, MtATL, 2010.

[239] M. Wimmer, et al., *A Catalogue of Refactorings for model-to-model transformations*, Journal of Object Technology, vol. 11, no. 2, 2012.

[240] M. Woodside, et. al., *Transformation challenges: from software models to performance models*, Sosym vol. 13, no. 4, 2014.

[241] Y. Xiong, H. Song, Z. Hu, M. Takeichi, *Synchronising concurrent model updates based on bidirectional transformation*, SoSyM vol. 12, no. 1, 2013, pp. 89–104.

[242] A. Yie, R. Casallas, D. Deridder, D. Wagelaar, *Realizing model transformation chain interoperability*, SoSyM (2012) 11: 55–75.

[243] T. Zan et al, *BRUL: a putback-based bx library for updatable views*, BX 2016.

[244] Z. Zhu et al., *BiYacc: roll your parser and reflective printer into one*, BX 2015.

[245] S. Zschaler, S. Yassipour-Tehrani, *Mapping FIXML to OO with aspectual code generators*, TTC 2014.

APPENDIX

The following were used as criteria to identify if particular transformation patterns had been used in the surveyed papers.

Auxiliary Metamodel The transformation uses additional entity types and/or features which are not defined in the source or target metamodels.

Auxiliary Correspondence Model The transformation uses identity attributes or other auxiliary data to maintain correspondences between corresponding source and target elements.

Collection Matching A rule application matches collections of source elements instead of individual instances.

Construction and Cleanup A transformation has separate phases for the construction and destruction of model elements.

Entity Merging Two rules each create/update elements of the same target entity type, using different source entity types or elements. Or one rule combines the data of two source elements into one target element.

Entity Splitting (Horizontal) Two rules (with disjoint application conditions) map (instances of) the same source entity type to instances of different target entities.

Entity Splitting (Vertical) A rule maps instances of one source entity type to a group of two or more instances of possibly different target entity types. The target instances can be explicitly linked by associations, or implicitly linked by key attribute values. The target group can also be created by separate rules with the same source entity and application conditions, but different target instances.

Factor Code Generation A code generation transformation creates elements of a program language model, instead of directly generating code text. The text is generated from the program model by a separate transformation.

Factor out Expression Evaluations A function/query operation or rule-scope variable is used to compute/hold the value of an expression and is referenced from within its scope.

Filter Transformation A transformation in a transformation chain is preceded by a pre-processing transformation which filters out elements from source models if they are not relevant for the main transformation.

Generic Transformations A transformation τ has a type/predicate or transformation-valued parameter which can be used to adapt τ for different uses.

Implicit Copy A transformation uses an implicit mechanism to copy source model data to the target model, without the need for explicit copying rules.

Introduce Rule Inheritance A rule is defined as a specialisation of another rule.

Map Objects before Links One rule r maps instances ex of a source entity type E to instances fx of a target entity type F prior to a rule r' which maps association end features of E to corresponding features of F. r' receives ex and fx as parameters or looks them up using a mechanism such as Object Indexing.

Object Indexing Entity type instances are looked-up by means of an index or key value.

Omit Negative Application Conditions Rule application conditions are removed if they are redundant with other conditions, in particular NACs which are contradictory with the positive application conditions are removed.

Phased Construction A transformation or rule set has only non-cyclic data dependencies. At least one case occurs where a rule r precedes a rule r' in execution order and writes a target entity type T which r' reads.

Recursive Descent Transformation rules use invoked subordinate rules or operations to carry out part of their effect.

Replace Abstract by Concrete Syntax A rule is specified in terms of the concrete syntax of source/target languages, instead of their abstract syntax.

Replace Explicit calls by Implicit A mechanism such as the *equivalent*() operator of ETL is used to implicitly invoke rules instead of making explicit calls.

Replace Fixed-point by Bounded Iteration A rule that both reads and writes the same entity type or feature is modified so that this circular data-dependence is avoided, eg., by referring to f@pre instead of f.

Restrict Input Ranges A rule application condition restricts a source element to be in a subset of a source entity type, based on preceding application conditions of the rule. Alternatively the context of a rule can be changed to reduce the range of element searches. For example, if there is a 1-many association from E to F, with respective rolenames x and r, then a rule

$$E :: \quad v : r \ \& \ Cond(v, self) \ \Rightarrow \ Post(v, self)$$

can be more efficiently expressed as:

$$F :: \quad Cond(self, x) \ \Rightarrow \ Post(self, x)$$

Sequential Composition Two rules, or two groups of rules, have a sequential execution order defined by the transformation.

Simulate Collection Matching Rules incrementally add matched elements to collections *sx.elements* referenced by auxiliary variables sx : *ESet* until *sx.elements* meets the original condition for being matched as a collection.

Match conditions s : $Set(E)$ *and* $\theta(s)$ in the original collection-matching rule are replaced by sx : *ESet and* $\theta(sx.elements)$.

Simulate Explicit Rule Scheduling Rule application conditions are used to enforce rule execution orders, by blocking execution of the rule until other rules have executed. Cf. `when` conditions in QVT-R.

Simulating Universal Quantification A logical condition $not(X \rightarrow exists(not(P)))$ is used in a rule.

Structure Preservation Separate source and target models. A group of rules (with no application conditions) each map one source entity type to one target entity type, with no source entity types used in two or more rules, and no target entity types used in two or more rules. Attribute values of the source entities are copied without change to corresponding target entity attributes.

Text Templates Text-generation rules use text templates which combine fixed text with variable text derived from source element data.

Transformation Chain The transformation involves the sequential composition of two or more subtransformations.
Transformation Inheritance One transformation is defined as a specialisation of another.
Unique Instantiation A rule or operation avoids re-creating a target element, if a target element satisfying the rule specification with respect to given source elements already exists.

Case	*Patterns*	*Category*	*Language*	*Year*	*Cites MT DP papers*
[1]	Repl. Abstract by Concrete syntax	Refactoring	VMTL	2015	none
[2]	Factor Code gen*, Text Templates* Trans. Chain*	Code gen	ATL/Xpand	2012	none
[3]	Entity Split (v), Intro Rule Inherit	HoT	ATL	2015	none
[5]	Auxiliary Metamodel	Refinement	KMF/ToolGen	2003	none
[6]	Rule Caching, Rec. Descent Intro. Rule Inherit	Refinement	SiTra	2006	none
[7]	Pre-normalisation, Seq. Comp	Analysis	GT	2011	none
[8]	Entity split v Recursive Descent Factor Expr. Eval.	Code generation	ATL	2014	none
[9]	Pre-Norm*, Trans Chain*	Refinement	ASF+SDF	2008	none
[11]	Aux. Corr. Model	Bidirectional	TGG	2012	none
[12]	Aux. Corr. Model*	Analysis	TGG	2013	none
[13]	Intro Rule Inherit, Aux corr model	Bidirectional	TGG	2015	none
[14]	Entity Merge, Aux meta	Model merging	ATL	2008	none
[15]	Trans. Chain Aux meta, Ent merge	Model merging	ATL	2008	none
[16]	Seq. Composition*	Refactoring	Henshin	2010	none
[17]	Factor Expr. Eval Recursive Descent Entity Split v	Refinement	ATL	2015	none
[19]	Repl. Abstract by Concrete syntax*	Refactoring	QVT-R	2006	none
[20]	Map Objects before Links Aux. Corr model*	Refinement	ILP	2009	none
[21]	Sequential Composition	Refactoring	DSLTrans	2010	none
[23]	Lens*	Bidirectional	Viatra	2016	none
[25]	Collection Match	Refactoring	EMF	2012	none
[26]	Recursive Descent	Migration	QVT-R	2014	none
[27]	Lens*	Bidirectional	Boomerang	2008	none
[28]	Transformation Chain* Constr. and Cleanup*	Refactoring	Henshin	2016	none
[29]	Construction and Cleanup	Bidirectional	QVT-R	2013	none
[30]	Text Templates*	Code generation	ATL/Acceleo	2013	none
[31]	Auxiliary Meta Text Templates* Entity Split h* Relabelling (Str. Pres)*	Migration	GrGen	2010	none
[32]	Rec. Descent	Analysis	UnQL	2000	none
[33]	Auxiliary Meta*	Bidirectional	ModelJoin, xPand	2016	none
[34]	Intro. Rule Inherit	Refinement	ATL	2014	none
[35]	Struc. Pres.	Refinement	ATL	2011	none
[36]	(i) Struc. Pres, Ent. Merge Map Obj. before Links	Refinement	ATL	2012	none
	(ii) Aux. Meta, Ent split v.	Semantic map	ATL	2012	none
[37]	Struc. Preservation	Migration	ATL	2012	none
[38]	Recursive Descent	Refinement	QVT-R	2013	none
[39]	Auxiliary Meta*	Refinement	TGG, QVT-R	2009	none
[40]	Replace Abstract by Concrete syn.	Refactoring	GT	2010	none
[41]	Sim explicit rule Sched, Rec descent	Refinement	QVT-R	2013	none
[42]	Auxiliary Meta Entity Split v	Refinement	GT	2007	none
[43]	Struc. Pres.	Migration	ATL	2015	none
[44]	Entity Merge Struc. Preservation	Migration	ATL	2016	none
[45]	Entity Split h* Entity Merge* Structure Pres.	Migration	ATL/ Java	2010	none
[46]	Parallel Transformations*	–	none	2012	none
[47]	Transformation Chain* Factor Expr. Eval.	Analysis	ATL	2013	none
[48]	Ent. Split (v)	Refinement	RubyTL	2006	none
[50]	Seq. Composition*, Aux. Meta Phased Cons.	Code generation	RubyTL	2008	none
[51]	Phased Cons.* Entity split v	Refinement	Eclectic, RubyTL	2009	[132]
[52]	Generic Trans.*	Analysis	ATL	2011	[24]
[53]	Generic Trans*	Refactoring	ATL	2013	none
[54]	Sliding Window* Factor Expr. Eval	Streaming	Eclectic	2013	none
[55]	Structure Pres. Entity Merge Factor Expr. Eval	Refactoring	ATL	2015	none
[56]	Post-normalisation*	Semantic mapping	Haskell	2015	none
[58]	Factor Expr. Eval	Analysis	CDM	2015	none
[60]	Intro Rule Inherit, Rec. Descent	Migration	MTF	2010	none
[59]	Lens*	Bidirectional	none	2010	none
[61]	Aux Corr Model, Seq Comp	Translation	GT	2015	none

TABLE XXVIII. INITIAL DATA EXTRACTION (1)

Case	*Patterns*	*Category*	*Language*	*Year*	*Cites MT DP papers*
[62]	Aux. Corr. Model*	Bidirectional	TGG	2007	none
[63]	Auxiliary Meta.*	Analysis	SDMLib	2014	none
[65]	Fixed-point Iter.*	Refinement	MoTif	2013	[102]
[66]	Auxiliary Meta*	Analysis	MoTif	2014	none
[67]	Localised MT*	Refinement	LTDesigner	2015	none
[68]	Intro. Rule Inherit.* Ent. Split (v)	Migration	ATL	2009	none
[69]	Text Templates	Code Generation	MIA-Transformation	2007	none
[70]	Lens*	Bidirectional	none	2007	none
[71]	Factor Code Gen.*	Code generation	xTend/xPand	2007	none
[73]	Adapter Trans.*	Migration	AML	2014	none
[74]	Rule Caching	Analysis	EVL	2016	none
[75]	Rule Caching* Factor Expr. Eval.	Analysis	QVT-O	2014	none
[76]	Entity Split v Aux. Corr Model	Code generation	TGG	2006	none
[77]	Auxiliary Corr. Model*	Bidirectional	TGG	2009	none
[78]	Aux cor model*, Uniq. Instant.	Bidirectional	TGG	2010	none
[79]	Structure Pres.* Sim. Explicit Rule Sched.*	Refinement	QVT-R	2011	none
[80]	Aux Corr Model	Bidirectional	TGG	2013	none
[81]	Aux Corr Model	Bidirectional	TGG	2016	none
[82]	Aux. Corr. Model Structure Pres. Sim. Explicit Rule Sched	Bidirectional	QVT-R, TGG	2010	none
[83]	Intro Rule Inherit*	Semantic map.	TGG	2011	none
[84]	Lens*	Bidirectional	Agda	2014	none
[85]	Auxiliary Meta Collection Match* Simulate Coll. Match Replace Abst. By Concrete	Refinement	GT	2010	none
[86]	Collection Match*	Refactoring	GT	2013	none
[87]	Aux Corr Model*	Bidirectional	TGG	2013	none
[88]	Transformation Chain* Pre-Normalisation* Entity Split v Auxiliary Meta*	Refinement	QVT-R, ETL	2013	[24], [102]
[89]	Aux. Corr. Model*	Semantic map.	QVT-R	2014	none
[90]	Entity Split v	Refinement	PaMoMo	2015	none
[91]	Simulate Coll. Match	Analysis	FujabaRT	2010	none
[92]	Factor Code Gen.*	Code generation	Stratego	2010	none
[93]	Aux. Corr. Model* Transformation Chain* Sequential Composition	Code generation	TGG	2014	none
[94]	Transformation Chain* Sequential Comp.	Bidirectional	TGG	2014	none
[95]	Aux. Corr Model*	Bidirectional	TGG	2015	none
[96]	Rec. Descent, Factor Expr Eval	Abstraction	OCL	2015	none
[97]	Aux Corr Model Entity Split h	Bidirectional	ATL, TGG	2016	none
[98]	Lens*	Semantic map.	NMF	2015	none
[99]	Lens*	Bidirectional	none	2011	none
[100]	Factor Expr. Eval*	Refactoring	FunnyQT	2013	none
[101]	Transformation Chain* Unique Instantiation	Code generation	Epsilon/Java	2014	none
[103]	Struc. Pres.	Refinement	MOTIF	2015	none
[104]	Text Templates	Code Generation	Rascal	2014	none
[105]	Transformation Chain	Code generation	Rascal	2014	none
[106]	Restrict Input Ranges*	Analysis	Rascal	2014	none
[107]	State*	Refinement	MOFScript	2015	none
[108]	Auxiliary Meta	Refactoring	EMF-IncQuery	2013	none
[109]	Struc pres, Ent split v Factor Expr Eval	Refinement	ATL	2011	none
[111]	Factor Expr. Eval.	Refinement	ATL	2006	none
[112]	Active Operations*	Bidirectional	EMF	2015	none
[113]	Active Operations*	Refinement	Viatra	2016	none
[114]	Aux. Meta*	Abstraction	Henshin	2011	none

TABLE XXIX. INITIAL DATA EXTRACTION (2)

Case	*Patterns*	*Category*	*Language*	*Year*	*Cites MT DP papers*
[115]	Rec. Descent	Refinement	QVT-R	2010	none
[116]	Trans. Chain*, Ent. Split v, Factor Expr. Evaluation	Refinement	ATL	2011	none
[117]	Entity Split v	Refactoring	ATL	2016	none
[119]	Aux. Corr. Model*	Refinement	ATL/CP	2013	none
[120]	Structure Pres. Entity Split v Sim. Explicit Rule Sched.	Refinement	QVT-R	2006	none
[122]	Rec. Descent Repl. Explicit calls by Implicit	Migration	ETL	2008	none
[121]	(1) Replace Coll. Match, Constr. and Cleanup (2) Constr. and Cleanup (3) Struc. Pres	Refactoring	UML-RSDS GrGen QVT-R	2014	none
[123]	Factor out Expression Evaluations*	Refactoring	EWL	2007	none
[124]	Ent. Split v, Rec. Descent Struc. Pres., Ent. Merge Text Templates	Refinement	ETL, EGL	2007	none
[125]	Replace Explicit by Implicit	Model merging	EML	2006	none
[126]	Rule Caching	Refinement	EPL	2016	none
[127]	Trans Inherit, Rec. Descent Pre-normalisation	Abstraction	QVT-O	2014	[102]
[128]	Lens*	Bidirectional	EMF/Java	2016	none
[129]	Transformation Chain*	Refactoring	SIGMA	2016	none
[130]	Aux. Corr. Model	Semantic mapping	AToM3	2009	none
[131]	Intro. Rule Inherit	Code generation	eMoflon/TGG	2014	none
[135]	Struc. Pres, Ent Split v	Refinement	Custom	2007	none
[136]	Phased Cons. Entity Split h Structure Pres.	Migration	UML-RSDS	2010	[132]
[137]	Const. + Cleanup* Entity Split v Map Objs before Links	Migration	UML-RSDS	2011	none
[138]	Restrict Input Ranges Replace Coll. Match Factor Expr. Eval	Refactoring	UML-RSDS	2013	none
[139]	Structure Pres. Entity Split v Map Objects before Links* Construction + Cleanup*	Refactoring	UML-RSDS	2013	[141]
[140]	Factor Expr. Eval* Repl. Fixed Point by Bounded* Restr. Input Ranges* Filter Transformation* Sim. Coll. Match	Analysis	UML-RSDS	2014	[145]
[144]	Map Obj. Before Links*	Refinement	UML-RSDS	2014	none
[146]	Intermediate Language	Semantic mapping	UML-RSDS	2015	[145]
[148]	Transformation Chain* Construction + Cleanup* Factor Expr. Eval* Rule Caching*	Refactoring	UML-RSDS	2016	[141]
[150]	Structure Pres. Ent Split v Aux Corr Model*	Bidirectional	UML-RSDS	2016	[141], [145]
[151]	Aux. Metamodel*, Rec. Descent	Refinement	Tefkat	2005	none
[152]	Aux. Corr. Model	Bidirectional	TGG	2016	none
[153]	Seq. Composition	Refactoring	VMTS	2008	none
[154]	Transformation Chain*	Code generation	QVTR-XSLT	2014	none
[157]	Map Objs before Links, Sim. Explicit rule Sched. Factor Expr. Eval	Bidirectional	QVT-R	2011	none
[158]	Lens*	Bidirectional	none	2013	none
[159]	Entity Merge Entity Split v Recursive Descent Structure Pres.	Bidirectional	QVT-R, ATL	2016	none
[160]	Text Templates	Code generation	Acceleo	2013	none
[161]	Intro. Rule Inheritance* Factor Expr. Eval.	Refactoring	QVT-R	2008	none
[162]	Entity Split v	Semantic mapping	QVT-R	2008	none
[163]	Aux Metamodel	Refinement	ETL	2013	none
[164]	Lens	Bidirectional	Haskell	2007	none
[165]	Factor Expr. Eval., Map Objs. Before Links	Refinement	ATL	2008	none
[166]	Map Objects before Links	Refinement	Henshin	2016	[102], [145]
[167]	Text Templates	Code generation	XML/VTL	2010	none
[168]	Pre-Normalisation	Semantic mapping	MotMot (GT)	2008	none
[169]	Text Templates Factor Code Gen*	Code generation	Sintaks/ TCSSL	2008	none

TABLE XXX. INITIAL DATA EXTRACTION (3)

Case	*Patterns*	*Category*	*Language*	*Year*	*Cites MT DP papers*
[170]	Aux. Corr. Model, Ent. Split v Struc. Pres	Refinement	TGG	2016	none
[171]	Ent split (v)	Refinement	ATL	2013	[132]
[172]	Aux. Corr. Model, Ent Split (v)	Semantic mapping	GReAT (GT)	2008	none
[173]	Sequential Composition Entity Split v	Semantic map.	GT	2015	none
[174]	Transformation Chain* Pre-Normalisation*	Refinement	ITM generator	2011	none
[175]	Map Objs. Before Links	Migration	ATL	2015	none
[176]	Recursive Descent	Migration	ATL	2016	none
[177]	Aux. Correspondence Model	Refactoring	eMoflon/TGG	2015	none
[178]	Aux. Corr. Model*	Refactoring	TGG	2015	none
[179]	Entity Merge Auxiliary Metamodel Entity Split v	Abstraction	QVT-R	2010	none
[180]	Intro. Rule Inheritance Rule Caching Auxiliary Metamodel	Refinement	Mitra	2010	none
[181]	Ent split v, Factor Expr Eval	Refinement	ATL	2011	none
[182]	Replace Coll. Match Entity Split v	Refactoring	SMW/Python	2005	none
[184]	Auxiliary Metamodel Pre-Normalisation	Migration	none	2010	none
[185]	Ent split v, Map objs before links	Refinement	ATL	2011	none
[186]	Observer*	Reactive	none	2008	none
[187]	Visitor*	Higher-order	QVT-O	2012	none
[189]	Ent. Merge	Abstraction	QVT-O	2014	none
[188]	Generic Trans.	Refactoring	EMFText	2013	none
[190]	Transformation Chain*	Analysis	ATL	2009	[132]
[191]	Struc. Pres., Aux. Meta	Migration	QVT-C	2016	none
[192]	Entity Split v Phased Constr.	Refinement	QVT-R, QVT-O	2008	none
[193]	Structure Pres Rule Caching* Implicit Copy* Map Objects before Links	Migration	Flock	2010	none
[194]	Implicit Copy*	Migration	Flock	2014	none
[195]	Visitor*	Refinement	Custom/XML	2007	none
[196]	Data Cleansing*	Migration	none	2010	none
[197]	Auxiliary Meta*	Refinement	EMF	2010	none
[198]	Structure Preservation	Bidirectional	ATL	2011	none
[199]	Recursive Descent	Migration	PETE	2010	none
[200]	Seq. Composition*	Migration	DSLTrans	2014	none
[201]	Seq. Composition*	Semantic map	DSLTrans	2015	none
[202]	Entity Split v Map Obj. before Links Factor Expr. Eval	Migration	ATL	2015	none
[203]	Aux. Corr. Model	Bidirectional	Viatra	2016	none
[204]	Entity Split v Auxiliary Metamodel Sim. Explicit Rule Sched.	Refactoring	Groove	2013	none
[205]	Entity Split v	Semantic map	none	2008	none
[206]	Structure Pres Sim. Explicit Rule Sched.	Runtime	QVT-R	2011	none
[207]	Sim. Univ. Quantification* Visitor*	Refactoring	VIATRA	2015	none
[208]	Intro. Rule Inherit. Factor Code Gen* Text Templates	Refinement	QVT-O	2015	none
[209]	Unique Instantiation	Bidirectional	QVT-R	2010	none
[210]	Structure preservation	Bidirectional	QVT-R	2013	none
[212]	Structure Pres. Entity Split v Phasing (Seq. Comp.)*	Semantic map	T-Core/MoTif	2012	none
[213]	Sim. Univ. Quant.*	Analysis	EMF-INCQUERY	2014	none

TABLE XXXI. INITIAL DATA EXTRACTION (4)

Case	*Patterns*	*Category*	*Language*	*Year*	*Cites MT DP papers*
[214]	Aux. Meta, Aux. Corr. Model	Refinement	GT	2005	none
[216]	Transformation Chain* Structure Pres. Map Objects before Links	Refinement	ATL	2010	[49]
[217]	Text Templates* Factor Expr. Eval. Transformation Inherit.*	Higher-order transformation	ATL	2010	[102]
[218]	Map obj. Before Links, Rec. Descent	Refinement	ATL	2011	none
[219]	Factor Code Gen.*	Refinement	ATL, MOFScript	2012	none
[220]	(1) Aux. Metamodel Factor Expr. Eval., Rec. Desc. (2) Ent. Split (v) Factor Expr. Eval.	Analysis Refactoring	ATL	2011	none
[221]	Entity Split (v)	Refinement	ATL	2015	none
[222]	Trans. Chain*, Intermediate Language	Code Generation	none	2006	none
[223]	Trans. Chain*, Phased Cons.	Refinement	none	2012	none
[224]	Aux. Metamodel*	Refinement	Viatra	2016	none
[225]	Intro Rule Inherit*, Struc. Pres	Migration	ATL	2011	none
[226]	Aux. Corr. Model*, Struc. Pres.	Bidirectional	TGG	2009	none
[227]	Lens	Bidirectional	Haskell	2010	none
[228]	Trans. Inheritance*, Struc. Preservation	Migration	ATL	2008	none
[229]	Trans. Inherit.*, Struc. Pres.	Migration	ATL	2009	[132]
[230]	Intro. Rule Inherit.*	Refactoring	SimpleGT	2011	none
[231]	Recursive Descent Factor Expr. Eval	Analysis	ATL	2015	none
[232]	Migrate along Dom. Partitions*	Migration	none	2010	none
[233]	Lens*	Bidirectional	Haskell	2011	none
[234]	Trans. Chain*	Code generation	QVT-O	2012	none
[235]	Ent. Split v	Refinement	QVT-C	2016	none
[236]	Rec. Descent	Refactoring	QVT-R	2016	none
[237]	Auxiliary Meta Structure Pres. Intro. Rule Inherit*	Migration	ATL	2007	none
[239]	Map Objects before Links, Rule Caching* Factor Expre. Evaluations*	Refinement	ATL	2012	[49]
[240]	Intermediate Language*	Semantic map	PUMA	2014	none
[241]	3-way Merge*	Bidirectional	QVT-R	2013	none
[242]	Entity Split v Aux. Corr. Model* Transformation Chain*	Refinement	ATL	2012	none
[243]	Lens*	Bidirectional	Haskell	2016	none
[244]	Lens*	Bidirectional	Haskell	2015	none
[245]	Transformation Chain*	Code generation	EOL/ETL/EGL	2014	none

TABLE XXXII. INITIAL DATA EXTRACTION (5)