Verification of the CD2RDBMS Transformation Case in Flora-2: VOLT 2015 Case Study Technical Report

Muzaffar Igamberdiev, Georg Grossmann, and Markus Stumptner

Advanced Computing Research Centre, School of IT and Mathematical Sciences
University of South Australia, Mawson Lakes, SA 5095, Australia
{firstname.lastname}@unisa.edu.au

Abstract. Model transformations enable models, which are the first class entities of MDE. Transformations are used to generate, refactor, synthesize, reverse engineer and simplify models among others. The accuracy of transformation will impact not only transformations themselves, but also the models. The correctness of transformation will be defined by its verification, which makes them critical for models. VOLT workshop has been addressing this important research area for several years. As a solution for the VOLT-2015 case on transformation between class diagrams and RDBMS models, we provide a verification approach, namely MOTIF, in Flora-2 language. With specifying models, transformations and verification in the same language, we aim at closing the gap between models and verification formalism.

Keywords: model transformation, verification, Flora-2

1 Introduction

Model transformations are means that connect abstract models to concrete (synthesis and reverse engineering), complex models to simplified (simplification and normalization), models written in one language to another (migration), models with certain operational quality to improved ones (optimization) and many more [17]. These means can serve as bridges when the source and target models reside in different technical spaces. In the role of enabling means and bridges the impact of transformations is critically important within the context of models, which was experienced in Model Driven Engineering (MDE) world in the last decade.

Consequently, transformation languages and tools are success factors of model transformations. Verification of model transformation is one of the success criteria of these languages and tools [17]. Verifying transformations is more complex than verifying models [1,12]. The Verification of Model Transformation Workshop (VOLT) has been addressing the verification from different perspectives. Within this year's VOLT 2015, we introduce a different verification approach for one of the specified cases using the Flora-2, a dialect of F-Logic with numerous

extensions. It is a single language framework for all artifacts involved in model transformation, namely models, transformations and verification properties.

The motivations to use a single language framework are:

- No need for the transformation from modeling environment to verification formalism. It closes the research gap in the intersection of MDE and validation and verification techniques [10].
- Smooth learning curve for users, e.g. modelers or domain engineers, since they learn a single language for modeling, transformation and verification.
- Existing models in other modeling languages (such as UML and EXPRESS) can benefit from reasoning features of MOTIF by transforming them into Flora-2. The difference from similar verification approaches [7,20,19] is that they transform a verification problem (derived from the underlying (meta-) models) to a particular verification formalism, where MOTIF transforms the underlying model and verifies properties on the original model.
- In MOTIF, verification results and diagnostics can be reported directly to users, without translating them into modeling formalism. User receives verification feedback in terms of modeling concepts, which he/she uses on a daily basis [10].
- A single language transparent to users, in a sense that they can see through models, transformations and verification properties from the perspective of the same language [10].
- It provides easier tool support. No need for the verification plugins or add-ons as modeling language extensions [10].

Besides verification based on both the source and target model properties, it verifies model transformation rules in transformation specification. The approach is called Model Transformation Verification in Flora-2 (MOTIF). Besides verifying correctness properties [18], MOTIF provides a flexible rule writing mechanism to address custom and domain specific verification properties.

The paper is organized as follows. Section 2 introduces the modeling of the CD2RDBMS case in Flora-2. The framework of transformation, verification rules and verification options are discussed in Section 3. The properties for the provided cases are verified in Section 4. Related work on verification of model transformation is analyzed in Section 5. Finally, Section 6 concludes and highlights future research.

2 UML Class Diagrams To Relational Database Management System Transformation Case (CD2RDBMS)

Initially, we give a brief introduction to Flora-2. Afterwards, we represent (encode) the source and target models of the transformation case in Flora-2. Models will be built from facts and eventually both the source and target models will be represented in the knowledge base, which will enable transformation, querying and verification.

Flora-2 stands for *F-Logic Translator* and is a dialect of F-Logic knowledge representation and ontology language [15], which has simple and expressive syntax with well-defined declarative and object-oriented frame-based semantics. It is a unified language of F-logic [14], HiLog [9], Transaction Logic [5,4] and defeasible reasoning [21]. It benefits from a natural way to do meta-programming in the style of HiLog, logical updates in the style of Transaction Logic, and a form of defeasible reasoning [15]. Flora-2 has observed several improvements to make the system more user-friendly and practical and it differs in many ways from the original version of F-logic, as described in [14], and from an earlier implementation, Flora-1.

These characteristics make it practical to apply to model transformations. Its dynamic module system makes the language extensible and flexible. Source and target model semantics can be compactly expressed and queried, the constraints can be checked, and verification rules can be applied.

Flora-2 represents higher-order and object-oriented concepts both syntactically and semantically [14]. Its reasoning power is enhanced by defeasible reasoning, which is a form of non-monotonic logical reasoning where some rule instances can be defeated by other rule instances. The defeated rules do not need to be satisfied by the intended model of the knowledge base [15]. It helps when one works with big industrial models with a large number of transformation and verification rules in the rule base.

The knowledge base is formed from facts that are represented as `a[prop{min:max}->b]`. It means an object '`a`' has the property '`prop`' with value '`b`'. The cardinality constraints of a property can be defined between lower (`min`) and an upper (`max`) bounds. The property can be inherited by `*=>` operator. The operators `::` and `:` represent generalization and classification relationships respectively.

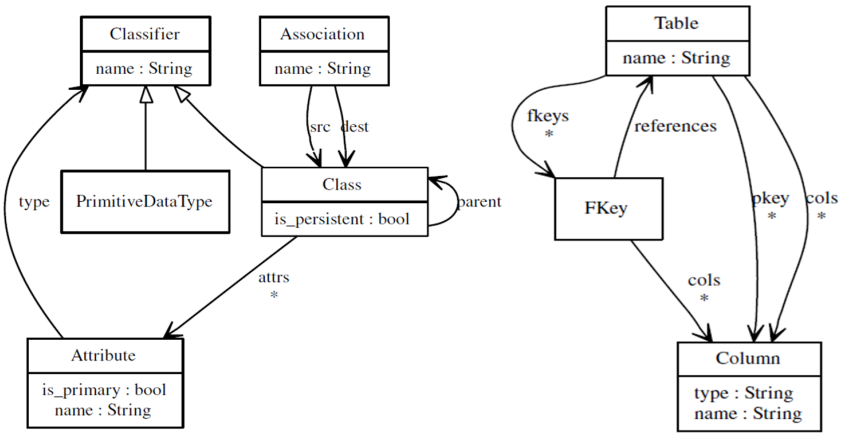

Fig. 1: Class and RDBMS meta-model [3]

The transformation case considers the classical model transformation between UML class diagrams and relational database schemata. Three properties should be verified with respect to the correspondences between the two modeling languages. These properties will be introduced and verified by relevant verification rules in Section 4. Both the source and target meta-models (Figure 1) are represented in Flora-2.

```
1  classifier[name*=>_string].
2  primitiveDataType :: classifier.
3  class :: classifier.
4  association[name*=>_string].
5  association[src*=>class].
6  association[dest*=>class].
7  class[is_persistent*=>_boolean].
8  class[parent*=>class].
9  class[attrs{0:*}*=>attribute].
10 attribute[is_primary:_boolean].
11 attribute[name*=>_string].
12 attribute[type*=>primitiveDataType].
```

Listing 1: Class meta-model in Flora-2

Listing 1 illustrates Class meta-model in Flora-2. The `classifier`, `association` and `attribute` model elements have a property called `name` (lines 1,4 and 11). Type of the `name` property is `_string`. The primitive datatypes (e.g. `_string`) are defined with prefixed underscore symbol in Flora-2. The `primitiveDataType` and `class` are subclasses (denoted by `::`) of `classifier`. The association ends `src` and `dest` that refer to `class` are represented with the relevant properties to `class` model element (lines 5-6). Similarly, `class` refers to `class` itself and `attribute` model element with `parent` and `attrs` properties respectively (lines 8-9). The cardinality constraint for `attrs` is defined between zero and infinity (line 9, `{0:*}`). The same multiplicity is represented with asterisk (*) symbol in Figure 1. The boolean properties are defined with `is_persistent` and `is_primary` properties in `class` and `attribute` model elements respectively (lines 7 and 10). Finally `attribute` and `primitiveDataType` model elements are connected with `type` property (line 12).

Identically, RDBMS meta-model is represented in Flora-2 in Listing 2. Flora-2 uses properties both to define type signatures and relate to other model elements. The examples for the former can be seen in `_string` properties in lines 1,7 and 8 (Listing 2). The rest of the lines are examples of relating to other model elements by means of properties (lines 2-6). The primary keys consists of a subset of columns (line 3). The multiplicities of attributes are defined between zero and infinity(`*`) in lines 2-4 and 6. Uniqueness of table names is assumed and it is verified in Section 4.

```
1  table[name*=>_string].
2  table[fkeys{0:*}*=>fKey].
3  table[pkey{0:*}*=>column].
4  table[cols{0:*}*=>column].
5  fKey[references*=>table].
6  fKey[cols{0:*}*=>column].
7  column[type*=>_string].
8  column[name*=>_string].
```

Listing 2: RDBMS meta-model in Flora-2

The knowledge base has been built from both class (Listing 1) and RDBMS meta-models (Listing 2). In the next section we will extend the knowledge base with transformation rules to enable transformation from source (UML class diagrams) to target(RDBMS) models.

3 Transformation and verification rules

This section introduces rules, particularly transformation and verification rules within Flora-2. First, we present a framework to design transformation and verification rules. Later, we consider verification options for model transformations.

3.1 A model transformation framework and verification rules

A framework for model transformations in Flora-2 is displayed in Figure 2. All aspects (models, transformations and their execution) are specified within Flora-2. It involves source and target models both in Flora-2. The transformation between them is realized by a transformation engine, which uses different transformation rules to transform from source to target.

The transformation framework is expressed via F-Logic rules. The rule consists of a head and a body part (e.g. `rule_head(p1,p2,...) :- body`). The head consists of a name and an arbitrary numbers of parameters of a transformation rule. These parameters are used in the body to assess conditions. The body comprises of source (pre-conditions), target and post-conditions sections (Figure 2). Pre-conditions and patterns that should be matched in a source model are addressed in the source model section of the transformation rule. The facts that should be added to the target model, as a result of match in the source model, take place in the target model section, while the post-transformation section contains conditions that should be satisfied after the rule's application. Different transformation rules can be applied on the same knowledge base. The negation is represented by "\+" sign.

```
1  %TransformClass2Table(?CLASS, ?TABLE, ?SrcModule, ?TargetModule) :-
2    ((?CLASS[is_persistent->true,name->?NAME]:class@?SrcModule;
3    \+(?CLASS :: ?_Y))),
4    insert{(?TABLE[name->?NAME] : table)@?TargetModule}.
```

Listing 3: Class2Table transformation rule

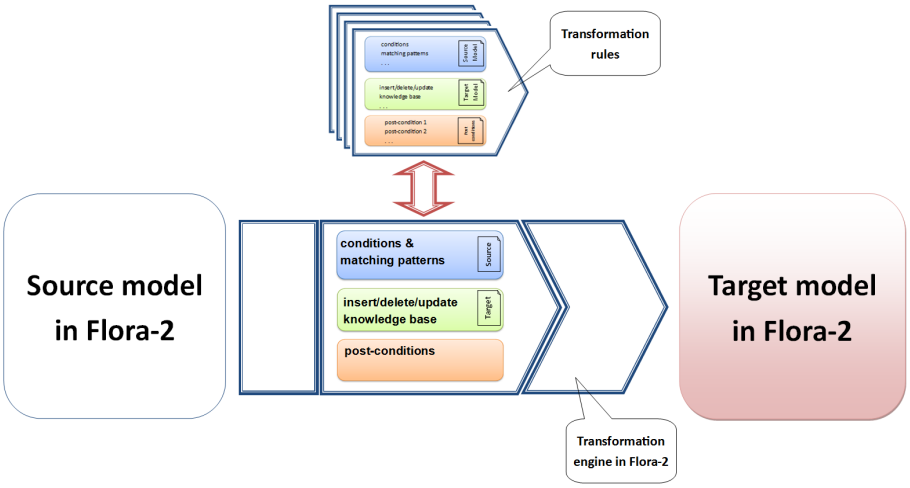

Fig. 2: A model transformation framework in Flora-2

Listing 3 illustrates a transformation rule to transform a class to a table. The rule contains its name (`%TransformClass2Table`) and two arguments for class and table instances and two arguments for the source and target modules (line 1). The prefixed percentage (`%`) in a rule name indicates verification rule (predicate) uses tabling(cashing), a technique that enhances top-down query evaluation with a mechanism that remembers the calls made previously in the process. The source and target models are organized in two different Flora-2 modules. The rule fetches all instances (`?CLASS`) of `class` which are persistent (line 2) and top class (line 3 - `?CLASS` should have no superclass). Later it stores their name in the `?_NAME` variable for further use (line 2). The "*don't care*" variables (e.g. `?_NAME`) are anonymous variables which are not printed out and used to store intermediate values. The names of the matched instances from the source model are used to insert `table` instances with the same name in the target model (line 3).

Similarly, verification rules are also built using the same framework (see Figure 2). The difference is that a verification rule has no target model manipulation section, their pre-conditions and post-conditions are merged, and their scope covers both models and transformation specification. The verification rule outputs results which indicate whether properties are satisfied and can point out specific model elements that caused the verification error. The concept will be exemplified by verifying correctness properties in the next section.

3.2 Verification options

Transformation process involves three artifacts: the source (meta-)model, the target (meta-)model and the transformation specification itself. Verification process considers the information received from these artifact sources to verify a specific correctness property. In this sense, transformation can be verified in two

ways: (1) based on (the information of) the properties of the source and target model and (2) based on (the information obtained from) the transformation specification, particularly on the transformation rules. Particularly in our case, for example, the former option can be used to verify whether the target model contains a table with the same name as a persistent class in the source model. The latter option is to query the transformation rule within the transformation specification, that it contains a mapping from a persistent class to a table. Both options are supported by MOTIF. It can query and verify both the source and target model at the same time, since they reside at the same knowledge base (see Section 2).

In the case of verification based the source and target model, one must execute the transformation to be able to query the target model, which can be costly in terms of time and effort. On the other hand, verification based on the transformation rules can detect verification errors before the actual execution of the transformation. It can benefit by saving time and effort of transformation execution. Additionally, it can verify and prevent potential errors during the design of transformation. In the next section we provide implementation for both verification options.

4 Implementation of verification rules in Flora-2

This section starts with a brief introduction to rule definition in Flora-2 and then it will proceed with implementation and discussion of verification properties of the provided and other example cases to demonstrate verification features of the framework.

In the context of Flora-2 verification is performed by defining verification rules and executing them on the knowledge base. The verification rules in Flora-2 consists of `head:-body.` statements, which means when the `body` is true, the `head` is true as well. The head can take an arbitrary number of arguments. The body can have conditions to verify a property. Both the source and target models can be queried to verify a model transformation property. If variable `?X`, an argument of a rule is bound by the query, then all possible bindings are retrieved as a result set. That means it can fetch all instances of a specific (meta-)model element or of a particular violation. It can be used for the sake of metrics of a transformation rule.

4.1 Verification of the CD2RDBMS case

We will demonstrate both verification options in the context of the three rules, which were provided in the CD2RDBMS case. First, we illustrate verification rules that use the source and target models to verify the properties. Afterwards, we will give an example for verification based on transformation rules. All rules return violations of the properties.

Property rule 1. Non-persistent classes and non-top classes must not be transformed into a corresponding table.

The rule is illustrated in Listing 4. The rules accept four parameters for class and table instances and the source and target modules respectively. A particular model element can be provided as an argument to verify against the property. Additionally, if the rule is executed without providing values to the arguments (`?C, ?T`), all existing violations for any class and table will be retrieved. This feature makes Flora-2 practical to use a single rule to verify all model elements in knowledge base.

```
1  non_peristent_non_top_classes(?C, ?T, ?SrcM, ?TargetM ):-
2          ?C[is_persistent->false] : class@?SrcM,
3          ?C :: ?_Y@?SrcM,
4          ?C[name->?_CNAME]@?SrcM,
5          ?T[name->?_TNAME] : table@?TargetM,
6          ?_CNAME = ?_TNAME.
```

Listing 4: A verification rule for property 1.

First, the rule retrieves non-persistent (line 2) and non-top (line 3) instances of `class`. Line 3 indicates whether `?C` is a subclass of any (`?_Y`) model element, which means non-top element. As a next step, the rule compares name (`?_CNAME` in line 4) of the matched instance (`?C`) with name of a table instance (line 5). The names must be equal (line 6), to find the violation of the verification property.

Property rule 2. All persistent top classes must be transformed into a corresponding table

Similarly, the rule 2 uses the same body with few differences in Listing 5.

```
1  persistent_top_classes(?C, ?T, ?SrcM, ?TargetM) :-
2          ?C[is_persistent->true] : class@?SrcM,
3          \+(?C :: ?_Y)@?SrcM,
4          ?C[name->?_CNAME]@?SrcM,
5          ?T[name->?_TNAME] : table@?TargetM,
6          ?_CNAME \= ?_TNAME.
```

Listing 5: A verification rule for property 2.

Identically the rule named as `persistent_top_classes` with four arguments. This time it fetches persistent classes (line 2 in Listing 5), which are top elements (line 3). Top elements are queried as elements which has no super classes (`\+` indicates negation). When the names of matched (class) elements are not equal to the names of table instances(line 6), it indicates a violation. The violated instances will be printed out.

Property rule 3. Column duplicates are forbidden in the output models, i.e., there should not be two columns with the same name in one table.

This verification rule checks for column duplicates, which is demonstrated in Listing 6. It has the simpler condition than the previous rules.

```
1  no_column_duplicates(?T,?TargetModule) :-
2          ?T : table[cols->?COL1, cols->?COL2]@?TargetModule,
3          ?COL1[name->?_C1]@?TargetModule,
4          ?COL2[name->?_C2]@?TargetModule,
5          ?_C1 = ?_C2.
```

Listing 6: A verification rule for property 3.

Similarly, it starts with its name and two arguments for table instance and the target module (line 1). It only queries target RDBMS model, since we don't need any information from the source model to verify this property. It fetches any two columns from instances of `table` (line 2) and compares their names (lines 3-5 in Listing 6). The violation occurs when these names are equal, which means column duplicates will be found.

Verification rules in Flora-2 for the case illustrates that scope of rules can cover both the source and target models. Listing 7 demonstrates a query (not a rule like in previous three listings), which enables the possibility of verification based on transformation rules.

```
1  ?- clause{%TransClass2Table(?_,?_),((?_:Class@?_, ?X), ?Y)}.
2
3  ?X = (${?_h5888[is_persistent->true]@_h5886},
4  ${?_h5888[name->?_h5913]@_h5886})
5  ?Y = ${insert{?_h5940[name->?_h5913]@_h5938?_h5940:Table@_h5938}}
6
7  1 solution(s) in 0.0000 seconds
```

Listing 7: Querying the Class2Table transformation rule

The rule base of Flora-2 can be queried by means `clause(head,body)` statement as illustrated in Listing 7. Line 1 represents a query and the rest (lines 3-7) shows the result of the query. The `head` and `body` uses variables and patterns to match the verification rule (line 1). We query for the transformation rule which was demonstrated in the previous section(see Listing 3). The transformation rule (from Class to Table) is queried to find out whether the persistent classes (lines 3-4) are transformed to relevant tables (line 5). The `name` variable `?_h5913` (lines 4-5) is used for both class instance(line 4) and table instance(line 5), which indicates a (persistent) class instance will be transformed to a table instance with the same name. By means of such queries, it verifies the validity of the transformation rule before its execution.

4.2 Verification of other correctness properties

Uniqueness of table names can be verified as demonstrated in Listing 8.

```
1  unique_table_name(?T) :-
2          ?T[name->?_N1] : Table,
3          ?_T[name->?_N2] : Table,
4          ?_N1 = ?_N2.
```

Listing 8: A verification rule to check uniqueness of table names

The verification rule finds violations of the unique name from the knowledge base. First, it finds an instance of `table` with a name property (line 2), which is equal to (line 4) the name of another table (line 3, Listing 8). Similarly, other artifacts (e.g. `class`) can be verified for unique names.

```
1  unique_artifact_name(?T, ?A) :-
2          ?T[name->?_N1] : ?A,
3          ?_T[name->?_N2] : ?A,
4          ?_N1 = ?_N2.
```

Listing 9: A verification rule to check uniqueness of any model element name

The `table` element in Listing 8 is substituted by a second argument `?A` in Listing 9. It represents any model element. It means the verification rule (9) checks all model elements in knowledge base. This demonstrates how easy it is to generalize verification rules in Flora-2. A set of similar general rules can bring verification mechanisms that benefit users to apply the same correctness rule to any underlying domain model.

```
1  ?- ?errors = setof {?_error |
2          unique_artifact_name(?_T,?_A),
3          ?_error=[?_T,?_A]},
4          ?errors.length@_basetype=?N_violations.
```

Listing 10: Number of verification errors(violations)

The number of verification violations can be counted as illustrated in the query in Listing 10. The `?errors` counts members of set of violations (line 1). The query uses the verification rule (line 2) described in Listing 9.

5 Related work

Different criteria have been explored for categorizing model transformation properties and verification techniques. They are classified based on technical approaches (theorem proving, model checking, testing), language and transformation related properties, level of formality, tooling, transformation languages and others [18,8].

In our case, verification of model transformations has been analyzed from four perspectives (see Table 1): properties, model transformation language, verification approach/language and support for single language framework, where at least transformation and verification are performed in the same language.

Several approaches have addressed verification of correctness properties, in particular satisfiability (of certain axioms) [7] and conformance [19]. Similarly, MOTIF also supports verification of conformance, satisfiability and meta-model completeness properties.

As shown in Table 1, some approaches use transformations to transform a model to a formalism where solvers and constraint checkers can be used to verify correctness properties. UMLtoCSP [7], UML/OCL to SAT encoding [20] and Xtend [19] are examples for such transformations. They use constraint programming, SAT solver and ASP programming for verification of properties respectively. While they transform the verification problem (extracted from the original (meta-)models), MOTIF transforms the original model and verify properties on

Approach	Correctness properties	MT language	Verification approach /language	Single language framework
UMLtoCSP constraint programming [7,6]	weak and strong satisfiability, lack of constraint redundancies & subsumptions, liveliness of a class	UML/OCL to Constraint Satisfaction Problem (CSP), UMLtoCSP	ECLiPSe constraint programming system	NO
UML/OCL Boolean satisfiability [20]	consistency of system states & redundancy of OCL constraints	UML models, OCL constrains are encoded as SAT instances	SAT solver	NO
CARE [19]	conformance	Xtend	Answer Set Programming(ASP)	NO
Language independent MT verification [16]	termination, single inheritance, name conflicts and others	Transformation specification metamodel is applied on ATL	a common intermediate representation, language independent framework	NO
UML/OCL Model Validator [11]	transformation model consistency, property preservation	transformation models	USE model validator	NO
MOTIF	conformance, completeness, inconsistencies [13]	Flora-2	Flora-2	YES

Table 1: Model transformation verification approaches

it. The language independent framework [16] uses an interesting approach to define a common transformation meta-model, which can be used to verify different model transformation properties. The transformation languages can be mapped to the the meta-model, particularly ATL mapping is provided [16]. Another approach [11] uses transformation models [2] for transformations and to check properties by applying USE model validator. MOTIF does not need to transform to other formalism, since it uses the same Flora-2 language for modeling, transformation and verification. In other words, MOTIF is a single language framework, where models and transformations can be built, transformed and verified, as illustrated in the last column of Table 1. The verification rules can be directly applied on both models and transformations.

6 Conclusion and Future work

In this paper we introduced an alternative approach, namely MOTIF, to verify model transformation properties in the context of CD2RDBMS case in Flora-2 language. Two verification options were considered: (1) based on source and target and (2) based on model transformation rules. Both options have been implemented in Flora-2. The added value behind this proposal is two-fold. Firstly, it uses a single language framework for modeling, transformation, querying and verification, which addresses the gap between models and verification formalism. Secondly, it allows to query transformation rules to verify the properties, which enables (design-time) verification of transformation before their actual execution.

This research will serve as a base for future work to apply verification rules on larger industry models, such as the ISO 15926 standard from the engineering do-

main. Furthermore, we plan to elaborate towards the direction of transformation rule based verification.

References

1. Benoit Baudry, Trung Dinh-Trong, Jean-Marie Mottu, Devon Simmonds, Robert France, Sudipto Ghosh, Franck Fleurey, and Yves Le Traon. Model transformation testing challenges. In *ECMDA workshop on Integration of Model Driven Development and Model Driven Testing.*, 2006.
2. Jean Bézivin, Fabian Büttner, Martin Gogolla, Frédéric Jouault, Ivan Kurtev, and Arne Lindow. Model transformations? Transformation Models! In *Model driven engineering languages and systems*, pages 440–453. Springer, 2006.
3. Jean Bézivin, Bernhard Rumpe, Andy Schürr, and Laurence Tratt. Model transformations in practice workshop. In *Satellite Events at the MoDELS 2005 Conference*, pages 120–127. Springer, 2006.
4. Anthony J Bonner and Michael Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133(2):205–265, 1994.
5. Anthony J Bonner and Michael Kifer. A logic for programming database transactions. In *Logics for databases and information systems*, pages 117–166. Springer, 1998.
6. Jordi Cabot, Robert Clarisó, and Daniel Riera. UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models Using Constraint Programming. In *Proceeding ASE '07*, pages 547–548, New York, NY, USA, 2007. ACM.
7. Jordi Cabot, Robert Clarisó, and Daniel Riera. On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*, 93:1–23, 2014.
8. Daniel Calegari and Nora Szasz. Verification of model transformations: a survey of the state-of-the-art. *Electronic Notes in Theoretical Computer Science*, 292:5–25, 2013.
9. Weidong Chen, Michael Kifer, and David S Warren. Hilog: A foundation for higher-order logic programming. *The Journal of Logic Programming*, 15(3):187–230, 1993.
10. Catherine Dubois, Michalis Famelis, Martin Gogolla, Leonel Nobrega, Ileana Ober, Martina Seidl, and Markus Völter. Research questions for validation and verification in the context of model-based engineering. In *MoDeVVa@ MoDELS*, pages 67–76. Citeseer, 2013.
11. Martin Gogolla, Lars Hamann, and Frank Hilken. Checking Transformation Model Properties with a UML and OCL Model Validator. In *Proc. 3rd Int. STAF2014 Workshop Verification of Model Transformations (VOLT2014)*, 2014.
12. Gerhard Goos. Compiler verification and compiler architecture. *Electronic Notes in Theoretical Computer Science*, 65(2):1, 2002.
13. Muzaffar Igamberdiev, Georg Grossmann, and Markus Stumptner. An implementation of multi-level modelling in F-logic. In *Proc. of the Workshop on Multi-Level Modelling (MULTI14) co-located with MoDELS 2014*, volume 1286 of *CEUR*, pages 33–42, 2014.
14. Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843, 1995.
15. Michael Kifer, Guizhen Yang, Hui Wan, Chang Zhao, Polina Kuznetsova, and Senlin Liang. Flora-2: User's Manual. *Flora*, 2:4, 2013.

16. K Lano, S Kolahdouz-Rahimi, and T Clark. Language-Independent Model Transformation Verification. In *Proc. 3rd Int. STAF2014 Workshop Verification of Model Transformations (VOLT2014)*, 2014.
17. Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. In *GraMoT 2005*, ENTCS 152, pages 125–142, 2006.
18. Lukman Ab Rahim and Jon Whittle. A survey of approaches for verifying model transformations. *Springer SoSyM*, 2013.
19. Johannes Schönböck, Angelika Kusel, Jürgen Etzlstorfer, Elisabeth Kapsammer, Wieland Schwinger, Manuel Wimmer, and Martin Wischenbart. CARE – A Constraint-Based Approach for Re-Establishing Conformance Relationships. In *APCCM 2014*, CRPIT Vol.154, pages 19–28. ACS, 2014.
20. Mathias Soeken, Robert Wille, Mirco Kuhlmann, Martin Gogolla, and Rolf Drechsler. Verifying UML/OCL Models Using Boolean Satisfiability. In *Proceedings DATE '10*, pages 1341–1344, 3001 Leuven, Belgium, Belgium, 2010.
21. Hui Wan, Benjamin Grosof, Michael Kifer, Paul Fodor, and Senlin Liang. Logic programming with defaults and argumentation theories. In *Logic Programming*, pages 432–448. Springer, 2009.