Verification of the CD2RDBMS Transformation Case in Flora-2.

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 play a key role in model-driven development. They are used to generate, refactor, synthesize, reverse engineer and simplify models among others. The accuracy of transformations will impact not only transformations themselves, but also the models, the first class entities of MDE. Verification of correctness properties ensures the quality of both transformations and 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. By specifying models, transformations and verification in the same language, we aim at closing the research 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 [12]. These means can serve as bridges when the source and target models reside in different technical spaces. 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 [12]. Verifying a transformation is more complex than verifying a model [8]. 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 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 benefits of a single language framework are: (1) integration of modeling and verification formalism, (2) a smooth learning curve for users, (3) a benefit

from reasoning features by transforming existing models into Flora-2, (4) a direct verification feedback to users, (5) a single language transparency for users in a sense that they can see through models, transformations and verification properties from the perspective of the same language, (6) the easier tool support, since modeling and verification are performed in the same framework, without a need for transformations and verification extensions (e.g. plugin, add-ons) for modeling language/environment [6,9].

Two verification options are provided based on: (1) both the source and target model properties, and (2) model transformation rules in transformation specification. The approach is called Model Transformation Verification in Flora-2 (MOTIF). Besides verifying correctness properties [13], MOTIF provides a flexible rule writing mechanism to address custom and domain specific verification properties.

Due to lack of space, we provide main arguments in this paper and a detailed discussion can be found in the relevant technical report [9]. The paper is organized as follows. Section 2 introduces the modeling of the UML Class Diagrams to Relational Database Management System (CD2RDBMS) transformation 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 The CD2RDBMS transformation case

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:

1. Non-persistent classes and non-top classes must not be transformed into a corresponding table.
2. All persistent top classes must be transformed into a corresponding table
3. Column duplicates are forbidden in the output models, i.e., there should not be two columns with the same name in one table.

These properties will be verified by relevant verification rules in Section 4. Additionally, MOTIF verifies different correctness properties other than the provided ones. The interested reader can refer to the example verification rules [9]. The source and target meta-models for the case are displayed in Figure 1. The uniqueness of table names is assumed and verified [9]. The multiplicity is represented with asterisk (*) symbol in Figure 1. We represent the meta-models in Flora-2.

Flora-2 stands for *F-Logic Translator* and is a dialect of F-Logic knowledge representation and ontology language [10], which has simple and expressive syntax with well-defined declarative and object-oriented frame-based semantics. These characteristics make it practical to apply on model transformations. The knowledge base is formed from facts that are represented as `a[prop{min:max}*=>b]`.

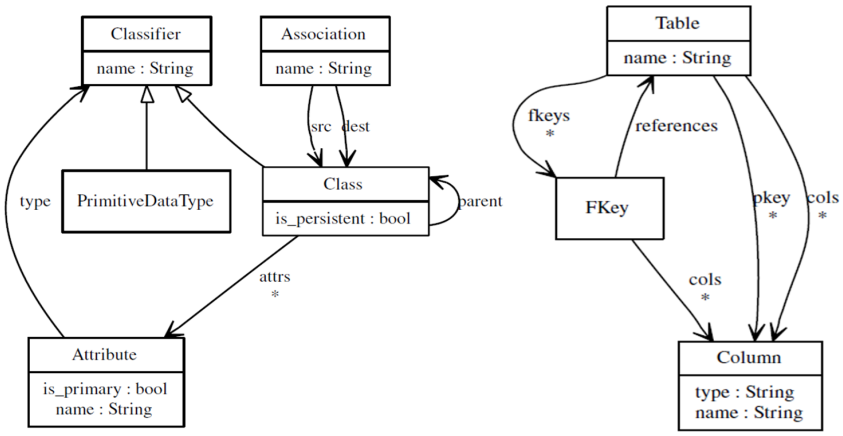

Fig. 1: Class and RDBMS meta-model [2]

It means an object '`a`' has an inheritable (denoted by `*=>`) property '`prop`' with value '`b`'. The cardinality constraints of a property can be defined in property signature between lower (`min`) and an upper (`max`) bounds. The operators `::` and `:` represent generalization and classification relationships respectively.

```
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

```
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 meta-models have been represented as Flora-2 knowledgebases, shown in Listings 1 and 2 respectively [1]

[1] The primitive datatypes (e.g. `_boolean`) are defined with prefixed underscore symbol in Flora-2.

3 Transformation and verification rules

This section introduces transformation and verification rules in 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 and verification framework

A framework for model transformation consists of a transformation engine that executes a rule at a time from a set of transformation (verification) rules, i.e. transformation specification. All aspects of transformation (models, transformation specification/rules and their execution) are specified within MOTIF.

A transformation rule is composed of three building blocks: pre-conditions, target manipulation and post-conditions. The pre-conditions and patterns that should be matched in a source model are addressed in the source model section. 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-condition section contains the conditions that should be satisfied after the rule's application.

Syntactically, a rule in Flora-2 consists of `head :- body.` statements, which means if the `body` is true then the `head` is true as well. The predicates at the `head` section can take an arbitrary number of parameters. These parameters are used to pass values to the body to assess certain conditions. The body can have conditions to verify a property.

```
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

Listing 3 illustrates a transformation rule to transform a `Class` to a `Table`. The rule head is a predicate with four arguments. The first two are for the source and target model elements, `Class` and `Table` respectively. The last two indicates the source and target modules; modules are used to separate models in Flora-2. Variables are indicated by being prefixed with a question mark, while the percentage prefix on the predicate name disables the cashing of results of the rule since the rule body modifies the knowledge base. When the rule is executed, it queries for classes in the source module that are persistent and top. It then retrieves the name of the class and inserts a table into the target module with the same name. The result of calling the rule through the predicate in the head is all the class objects that satisfied the query and the new table objects that were created.

Similarly, verification rules are also built using the same framework. 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.

3.2 Verification options

Transformation process involves three artifacts: the source (meta-)model, the target (meta-)model and the transformation specification itself. Verification considers the information received from these artifacts. 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. Specifically 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 can be used to query the transformation rule within the transformation specification, to check whether it contains a mapping from a persistent class to a table. Both options are supported by MOTIF.

4 Implementation of the verification rules in Flora-2

We will demonstrate both verification options in the context of the three rules. 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 accepts four parameters for class and table instances and the source and target modules respectively (Listing 4). A particular model element of class or table can be provided as a parameter to verify against the property. All existing violations for any class and table will be retrieved if first two parameters are not provided. This feature makes Flora-2 beneficial to use a single rule to verify all model elements in the knowledge base.

```
1  non_persistent_non_top_classes(?C, ?T, ?SrcM, ?TargetM ):-
2          ?C[is_persistent->false] : Class@?SrcM,
3          ?C[parent->?_Y]@?SrcM,
4          ?C[name->?_CNAME]@?SrcM,
5          ?T[name->?_CNAME] : Table@?TargetM,
```

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` has any parent (`?_Y`) model element, which means `?C` is a non-top element. As a next step, the rule checks whether `name` properties of both class and table instance (`?C` and `?T` respectively) are bound to the same variable (`?_CNAME`), which triggers the violation

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

Similarly, the rule 2 uses the similar body structure with a few differences in Listing 5. The rule fetches persistent classes (line 2 in Listing 5), which are top elements (line 3). Afterwards, lines 4 and 5 indicate that if the class name (`?_CNAME`) does not match with any name of table instances (`forall(?_T)`) then the rule reports a violation with the name of the unmatched class (`?_CNAME`).

```
1  persistent_top_classes(?C, ?SrcM, ?TargetM) :-
2          ?C[is_persistent->true] : Class@?SrcM,
3          \+(?C[parent->?_Y])@?SrcM,
4          ?C[name->?_CNAME]@?SrcM,
5          forall(?_T)^(\+(?_T[name->?_CNAME]) : Table@?TargetM).
```

Listing 5: A verification rule for property 2.

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 ones.

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

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, Listing 6). It only queries target RDBMS model, since we don't need any information from the source model to verify this property. It fetches two different (line 3) columns from an instance of table (line 2) and then checks whether they are not equal (lines 4 and 5).

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 of `clause(head,body)` statement as illustrated in Listing 7. Line 1 represents a query and the rest (lines 3-7) show 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.

5 Related work

Different criteria, such as language, transformation related properties, level of formality, tooling, transformation languages, verification formalism, have been

explored to categorize model transformation properties and verification techniques [13,5]. 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 a single language framework, where at least transformation and verification are performed in the same language.

Approach	**Correctness properties**	**MT language**	**Verification approach /language**	**A single language framework**
UMLtoCSP constraint programming [4,3]	satisfiability, lack of constraint redundancies & subsumptions, liveliness	UML/OCL to Constraint Satisfaction Problem (CSP)	ECLiPSe constraint programming system	NO
UML/OCL Boolean satisfiability [15]	consistency of system states & redundancy of OCL constraints	UML models, OCL >SAT instances	SAT solver	NO
CARE [14]	conformance	Xtend	Answer Set Programming(ASP)	NO
Language independent MT verification [11]	termination, single inheritance, name conflicts and others	Transformation spec meta-model is applied on ATL	an intermediate represent, lang independent framework	NO
UML/OCL Validator [7]	transf model consistency, property preservation	transformation models	USE model validator	NO
MOTIF	conformance, completeness & inconsistencies	Flora-2	Flora-2	YES

Table 1: Model transformation verification approaches

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 [4], UML/OCL to SAT encoding [15] and Xtend [14] are examples for such transformations. Similarly existing models in other modeling languages (such as UML and EXPRESS) can benefit from reasoning features of Flora-2. The language independent framework [11] uses an interesting approach to define a common transformation meta-model, which can be used to verify different model transformation properties. Another approach [7] uses transformation models [1] for transformations and to check properties by applying USE model validator. These verification approaches (Table 1) use different languages for modeling and verification. On the other hand, MOTIF uses the same Flora-2 language for modeling, transformation and verification. It is a single language framework, where models and transformations can be built, transformed and verified, as illustrated in the last column of Table 1.

6 Conclusion and Future work

In this paper we introduced a novel approach, namely MOTIF, to verify model transformation properties in the context of the CD2RDBMS case in Flora-2 language. Two verification options were considered and implemented. 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 domain.

Acknowledgments This research was partially funded by the Data to Decisions Cooperative Research Centre (D2D CRC).

References

1. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model transformations? Transformation Models! In: Proc. MODELS 2006. pp. 440–453. Springer (2006)
2. Bézivin, J., Rumpe, B., Schürr, A., Tratt, L.: Model transformations in practice workshop. In: Satellite Events at MoDELS 2005. pp. 120–127. Springer (2006)
3. Cabot, J., Clarisó, R., Riera, D.: UMLtoCSP: A tool for the formal verification of UML/OCL models using constraint programming. In: Proc. ASE '07. pp. 547–548. ACM (2007)
4. Cabot, J., Clarisó, R., Riera, D.: On the verification of UML/OCL class diagrams using constraint programming. Journal of Systems and Software 93, 1–23 (2014)
5. Calegari, D., Szasz, N.: Verification of model transformations: a survey of the state-of-the-art. ENTCS 292, 5–25 (2013)
6. Dubois, C., Famelis, M., Gogolla, M., Nobrega, L., Ober, I., Seidl, M., Völter, M.: Research questions for validation and verification in the context of model-based engineering. In: MoDeVVa@ MoDELS. pp. 67–76 (2013)
7. Gogolla, M., Hamann, L., Hilken, F.: Checking transformation model properties with a UML and OCL model validator. In: Proc. VOLT'14 (2014)
8. Goos, G.: Compiler verification and compiler architecture. ENTCS 65(2), 1 (2002)
9. Igamberdiev, M., Grossmann, G., Stumptner, M.: Verification of the CD2RDBMS transformation case in Flora-2: VOLT 2015 case study technical report. Tech. rep., Knowledge and Software Engineering Lab, University of South Australia (2015)
10. Kifer, M., Yang, G., Wan, H., Zhao, C., Kuznetsova, P., Liang, S.: Flora-2: User's Manual (2013)
11. Lano, K., Kolahdouz-Rahimi, S., Clark, T.: Language-Independent Model Transformation Verification. In: Proc. VOLT'14 (2014)
12. Mens, T., Gorp, P.V.: A taxonomy of model transformation. In: GraMoT 2005. pp. 125–142. ENTCS 152 (2006)
13. Rahim, L.A., Whittle, J.: A survey of approaches for verifying model transformations. SoSyM 14(2), 1003–1028 (2013)
14. Schönböck, J., Kusel, A., Etzlstorfer, J., Kapsammer, E., Schwinger, W., Wimmer, M., Wischenbart, M.: CARE – a constraint-based approach for re-establishing conformance relationships. In: APCCM 2014. pp. 19–28. ACS (2014)
15. Soeken, M., Wille, R., Kuhlmann, M., Gogolla, M., Drechsler, R.: Verifying UML/OCL models using boolean satisfiability. In: Proc. DATE 2010. pp. 1341–1344