YAMTL Solution to the TTC 2019 BibtexToDocBook Case

Artur Boronat
School of Informatics, University of Leicester, UK
aboronat@le.ac.uk

Abstract

Model-to-model transformations are often used to define consistency maintainers between metamodels. When these are executed incrementally, only the part of the model transformation that is affected by model updates is executed. The *TTC2019 BibTeX2DocBook* case proposes to evaluate mechanisms for avoiding the situation where updates, to DocBook models, introduce undesirable inconsistencies that are not valid according to the transformation. In this paper, we present the application of an inconsistency specification language, available within YAMTL, for locating when a model update yields illegal changes that break the consistency relation defined by a YAMTL model-to-model transformation. Moreover, a re-implementation of the reference ATL transformation in YAMTL is presented and compared against the most efficient transformation engine for ATL.

1 Introduction

Incrementality is an important optimization technique when transforming very large models, which avoids redundant computation. A transformation engine with support for incremental forward propagation of updates executes the transformation and then, when given an update to the source model, the changes are isolated and only the part of the transformation that is sensitive to them is re-executed.

YAMTL [Bor18] is a model transformation engine for very large models that enables the execution of model transformations both in batch mode and in incremental mode, without additional user specification overhead. It is implemented in Xtend, which transpiles to Java, and uses EMF as the (meta-)modeling front end. YAMTL implements a forward delta propagation procedure [Bor19] for executing model transformations in incremental mode that can handle documented change scenarios [BRVV12], i.e. documents representing a change to a given source model. Such documents are defined with the EMF change model [SBPM09], both conceptually and implementation-wise, guaranteeing interoperability with EMF-compliant tools. EMF is de facto implementation of the MOF standard [OMG16] for meta-modeling, in particular of *essential MOF (eMOF)*.

In this article we present the YAMTL solution to the *TTC19 Bibtex to Docbook Case* [GDH19], which has two parts: a transformation of documents from Bibtex to Docbook defining a consistency relation between source documents in Bibtex and target documents in Docbook; and a number of updates on the target document, which may invalidate the consistency relation. The first requirement is optional and consists in re-implementing the reference ATL transformation; the second requirement is mandatory and consists in finding out whether the target updates affect the validity of the consistency relation of the source Bibtex models against the mutated Docbook models, following a description of invalid updates on target models. In the mandatory requirement, the check should be possible without a full re-run of the transformation and inconsistencies should be reported clearly to the user. The solution is available at `http://bit.ly/ttc19-live-yamtl`.

In: A. Garcia-Dominguez, G. Hinkel and F. Krikava (eds.): Proceedings of the 12th Transformation Tool Contest, Eidhoven, Netherlands, 19-07-2019, published at http://ceur-ws.org

In the following, the YAMTL solution, starting with the mandatory requirement, and then following with the optional one. Finally, we discuss the main strengths of the approach.

2 Solution to Mandatory Requirement

YAMTL has been extended with an inconsistency specification DSL to filter out updates that are not valid w.r.t. the consistency relation specified by the transformation. This facility is generic and can be reused for any type of consistency relation. In this section, we describe how YAMTL's inconsistency specification language has been used to check for inconsistencies in updates without having to re-execute the transformation. Inconsistency reporting is considered in the next section, on the optional requirement.

2.1 Inconsistency Specification

An inconsistency specification consists of a list of possible inconsistency types. Each inconsistency type is declared by using the class, whose instances may affected by an update, the feature that may be updated, and a list of conflictive update types. Each conflictive update type is declared with an informative error, the type of update (`ADD`, `REMOVE`, `MOVE`, `UPDATE`), and an inconsistency guard that determines whether the update is conflictive or not. The inconsistency guard is optional. When none is required, the `TRIVIAL_CHECK` should be used. The inconsistency specification DSL allows for the definition of fine-grained inconsistency types, for feature bindings.

According to the acceptable mutations provided in [GDH19], the inconsistency specification of our solution, listed below, contains the following inconsistency types:

- Within `Section`s: swapping paragraphs breaks consistency (line 10) and paragraphs cannot be deleted (line 11).
- Within `Article`s: deleting sections is not allowed (line 16).
- Within `Sect1`s: deleting sections is not allowed (line 21), and paragraphs can only be added if they do not match one of the authors, titles, or journals in its `Sect1` (lines 24-28). The last inconsistency type is captured using an inconsistency guard, which checks whether there is a paragraph in `Sect1` whose contents match the contents of the paragraph being added.

```
1 class YAMTLSolution {
2   val static DocBook = DocbookPackage.eINSTANCE
3
4   val static TRIVIAL_CHECK = [EObject eObj, Object value | true] as (EObject,Object)=>boolean
5
6   @Accessors
7   val public static inconsistencySpec = #{
8     DocBook.section -> #{ 'paras' ->
9       #{
10        'Swapping paragraph' -> (YAMTLChangeType.MOVE -> TRIVIAL_CHECK),
11        'Deleting paragraph' -> (YAMTLChangeType.REMOVE -> TRIVIAL_CHECK)
12      }
13    },
14    DocBook.article -> #{
15      'sections_1' -> #{
16        'Deleting sections' -> (YAMTLChangeType.REMOVE -> TRIVIAL_CHECK)
17      }
18    },
19    DocBook.sect1 -> #{
20      'sections_2' -> #{
21        'Deleting sections' -> (YAMTLChangeType.REMOVE -> TRIVIAL_CHECK)
22      },
23      'paras' -> #{
24        'Adding an existing paragraph to Sect1' -> (YAMTLChangeType.ADD -> [ EObject eObj, Object value |
25          val sect1 = eObj as Sect1
26          val para = value as Para
27          sect1.paras.exists[it.content.startsWith(para.content)]
28        ] as (EObject, Object)=>boolean)
29      }
30    }
31  } as Map<EClass,Map<String,Map<String,Pair<YAMTLChangeType,(EObject,Object)=>boolean>>>>
32 }
```

2.2 Admissibility Tests

The YAMTL model transformation $\mathcal{T}$ between the `BibTeX` metamodel $\mathcal{M}_{BibTeX}$ and the `DocBook` metamodel $\mathcal{M}_{DocBook}$, can be regarded as the definition of a consistency relation $\mathcal{T}(\mathcal{M}_{BibTeX}, \mathcal{M}_{DocBook})$, and the predicate $\mathcal{T} \models (m_{BibTeX}, m_{DocBook})$ denotes that a source model m_{BibTeX} is consistent with a target model $m_{DocBook}$ according to $\mathcal{T}$.

An inconsistency specification defined over the target metamodel, $\mathcal{I}(\mathcal{M}_{DocBook})$, as presented in the previous section, defines the type of inconsistencies that are not allowed for target models. A target model update δ_t is admissible, $\mathcal{T} \models_{\mathcal{I}} \delta_t$, if and only if $\mathcal{I}(\mathcal{M}_{DocBook}) \not\models \delta_t$. That is, when it does not yield any inconsistency. In YAMTL, $\mathcal{T} \models_{\mathcal{I}} \delta_t$ is implemented with the expression

```
xform.admissibleChange(model, delta, iSpec)
```

which checks whether the `delta` for `model` is valid according to the inconsistency specification `iSpec` for the transformation `xform`. This is the operation that has been used to report whether a model update is consistent or not in the solution and it does not require the transformation `xform` to be re-executed.

3 Solution to Optional Requirement

The reference ATL transformation used to transform documents from Bibtex to Docbook was improved[1] by using rules that capture the four possible combinations of `title` and `journal` in `BibTeXEntry` objects using multiple rule inheritance in EMFTVM [WTCJ11]. EMFTVM matches those rules using a layered algorithm that uses a network of model constraints such that rules whose parent rules' matching conditions are not satisfied are not eligible to be matched, thus skipping rules during the matching phase.

In this section, the performance of the YAMTL implementation of the improved ATL transformation has been empirically evaluated using the TTC benchmark harness and compared against the analogous ATL transformation on EMFTVM. The traceability support built in the YAMTL transformation engine has been used to report inconsistencies according to the inconsistency specification presented in the previous section.

3.1 Transformation Re-Implementation in YAMTL

YAMTL also supports multiple rule inheritance [Bor18], benefitting from a thrifty application of rules as explained above, and the improved ATL transformation has been mirrored in an improved YAMTL transformation.[2] The results of both transformations have been checked for correctness and their outputs are identical but for the randomly generated identifiers. For obtaining performance results, we have executed the benchmark for models corresponding to size factors up to 10,000, for one single `mutantSet` and for one single `mutant`, as transformations are executed in batch mode, without propagation of updates. In addition, one thousand iterations were run for each input model and the results correspond to cold run times. Table 1 shows mean run times (in *ms.* unless stated otherwise) used by both tools. YAMTL takes about a third of the time used by EMFTVM as the size factor increases.

Size factor	ATL (EMFTVM)	YAMTL
10	352	84
100	456	143
1,000	766	297
10,000	3.6 s.	1.3 s.

Table 1: Model element cardinalities and run times in ms. (unless stated otherwise)

3.2 Reporting: Finding All Inconsistencies

The tool also provides the operation

```
xform.findInconsistenciesInChange(model, delta, iSpec, enableReport),
```

[1] `http://bit.ly/ttc19live-emftvm`

[2] The full solution included the original YAMTL transformation and the improved one was produced after the contest.

which finds all of the inconsistencies that are found in $\mathcal{T} \not\models_{\mathcal{I}} \delta_t$. The signature of the operation is the same as for `admissibleChange` but, in addition, it provides a boolean flag `enableReport`. If reporting is enabled, a graphical report is generated with all of the found inconsistencies, as shown in Fig. 1. For each inconsistency, the report includes the reason for the inconsistency (stating the description of the inconsistency, the object affected, the feature that was updated, the type of update and the value that was affected) and the transformation step that has been invalidated by the target model update (stating the rule that was applied, the excerpt of the source model m_{BibTeX} involved in the rule application, the excerpt of the target model $m_{DocBook}$ produced by the rule before the update, and the same excerpt of the target model after the update, i.e. $\delta_t(m_{DocBook})$). Graphical representation of models are obtained using PlantUML.

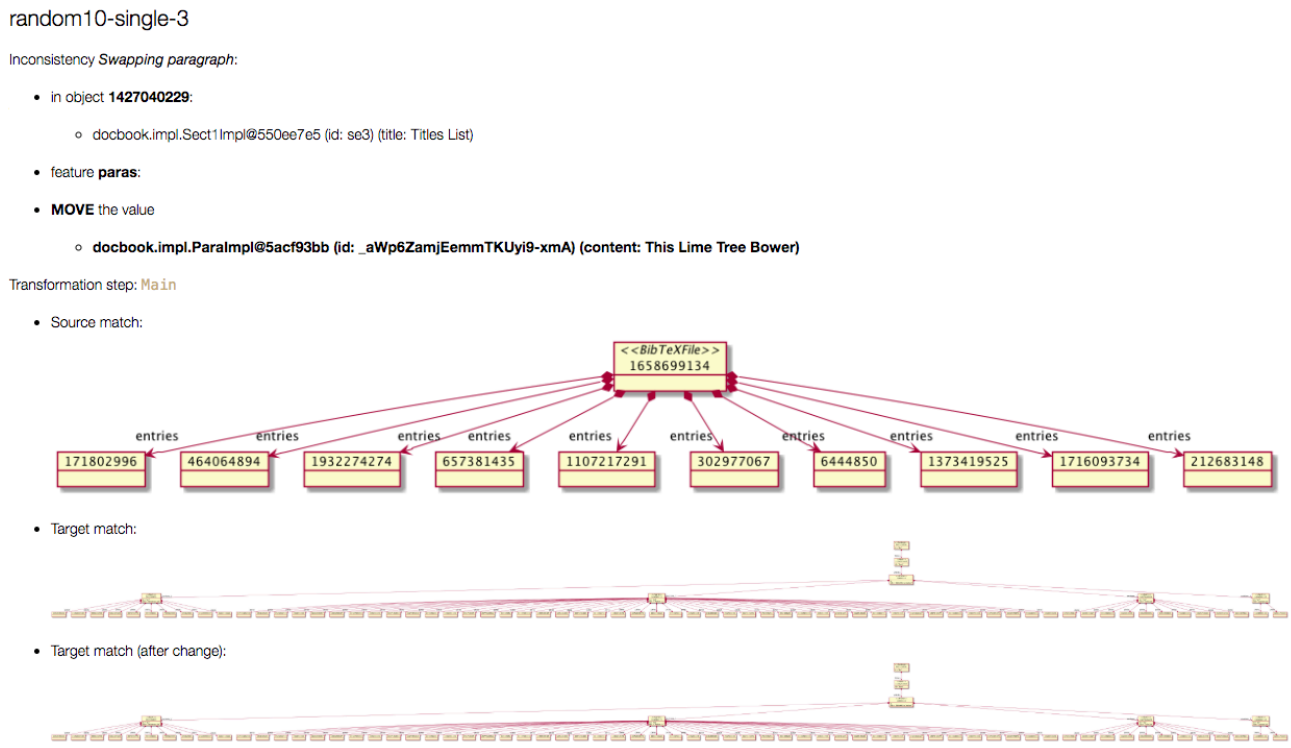

Figure 1: Outline of Solution

When reporting is enabled, the transformation `xform` must be executed by YAMTL in advance in order to provide traceability information from source to target models. Although the reporting feature is experimental, it serves as a proof-of-concept of the possibilities that YAMTL offers when informing modellers about inconsistencies that are created by target model updates, providing traceability to the affected part of the source model.

4 Conclusions

In this paper, we have discussed a full solution to the TTC'19 live case, including the use of YAMTL's inconsistency specification language for identifying inconsistencies in model updates (mandatory task) and a re-implementation of the ATL transformation (optional task) in YAMTL, which facilitates reporting inconsistencies when model updates are applied to the generated target model. Our solution provides the following strengths:

Granularity of Inconsistency Detection. Inconsistency specifications enable the identification of atomic model updates that lead to inconsistencies at binding level, including the possibility of defining complex checks that traverse the affected model using inconsistency guards. Inconsistencies in composite model updates are analysed by decomposing the model updates into atomic ones.

Reuse. Inconsistency specifications are defined at the metamodel level and the specification language itself is metamodel agnostic. Therefore it can be used for defining the language of admissible model updates for any consistency maintainer that defines a consistency relation between metamodels.

Reporting. The graphical reports generated by YAMTL enumerate all of the possible inconsistencies in model updates. YAMTL exploits the traceability links built during a transformation execution in order to illustrate the impact of a model update on a target model in the source model.

Performance. The goal of the optional part of the case consisted in providing an improvement of the original model transformation. The YAMTL transformation is analogous to the improved ATL transformation using multiple rule inheritance, and it has been shown that YAMTL executes the transformation in about one third of the time that EMFTVM uses to run the improved ATL transformation.

Some of the YAMTL features used in this case are experimental and their refinement is reserved for future work.

References

[Bor18] Artur Boronat. Expressive and efficient model transformation with an internal DSL of Xtend. In *MODELS 2018*, pages 78–88. ACM, 2018.

[Bor19] Artur Boronat. Offline delta-driven model transformation with dependency injection. In *FASE 2019*, volume 11424 of *LNCS*, pages 134–150. Springer, 2019.

[BRVV12] Gábor Bergmann, István Ráth, Gergely Varró, and Dániel Varró. Change-driven model transformations - change (in) the rule to rule the change. *Software and System Modeling*, 11(3):431–461, 2012.

[GDH19] Antonio García-Dominguez and Georg Hinkel. The TTC 2019 Bibtex to Docbook Case. In Antonio García-Dominguez and Georg Hinkel, editors, *Proceedings of the 12th Transformation Tool Contest (TTC@STAF)*, CEUR Workshop Proceedings. CEUR-WS.org, July 2019.

[OMG16] OMG. Meta Object Facility (MOF) 2.5.1 Core Specification, 2016.

[SBPM09] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[WTCJ11] Dennis Wagelaar, Massimo Tisi, Jordi Cabot, and Frédéric Jouault. Towards a General Composition Semantics for Rule-Based Model Transformation. In *MoDELS*, volume 6981, pages 623–637. LNCS, 2011.