An NMF solution to the TTC 2019 Live Case

Georg Hinkel

Am Rathaus 4b, 65207 Wiesbaden, Germany
georg.hinkel@gmail.com

Abstract

This paper presents a solution to the BibTex to Docbook (live) case at the Transformation Tool Contest (TTC) 2019. We demonstrate how the flexible execution strategies of NMF Synchronizations can be used to obtain an incrementally maintained analysis of inconsistencies between models.

1 Introduction

In many applications, systems are modeled multiple times with various aspects in mind. Because these aspects of a system share commonalities in their understanding of a system, such as its principle structure, these models tend to have a semantic overlap. To ensure consistency, a set of rules is often used to check and enforce that these models fit to each other [1].

However, it is not feasible to resolve inconsistencies automatically, for instance if models of multiple levels of abstraction are involved: An entity present in a more abstract model often cannot be automatically added in a more detailed model as usually additional input is required. In such cases, it is necessary to temporarily allow inconsistencies between the models and to explicitly identify and manage those inconsistencies. For some inconsistencies, it is possible to resolve them automatically, for others it is not.

In the TTC 2019 BibTex to DocBook benchmark [2], the task was to identify inconsistencies between models of a bibliographic reference and a model of a book, if the latter model faces changes. Those changes are either available as mutated DocBook models or as operational change sequences describing which operations had been performed on the target model. Solutions were asked to analyze the correspondences, i.e. whether new change sequences have arisen.

In this paper, we present a solution to this benchmark using synchronization blocks and their implementation in NMF Synchronizations [3]. NMF Synchronizations allows us to obtain descriptions of change sequences straight from the specification of the consistency specification. Because inconsistencies can be repaired automatically towards the target DocBook model, the same specification can also be used to transform a BibTex model into a corresponding DocBook model. Both the transformation and also the analysis for inconsistencies can be run incrementally, i.e. the engine automatically attaches to the source and target model and notifies new inconsistencies or resolves them automatically.

2 Synchronization Blocks and NMF Synchronizations

Synchronization blocks are a formal tool to run model transformations in an incremental (and bidirectional) way [3]. They combine a slightly modified notion of lenses [4] with incrementalization systems. Model properties and methods are considered morphisms between objects of a category that are set-theoretic products of a type (a set of instances) and a global state space Ω.

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

A (single-valued) synchronization block $\boldsymbol{S}$ is an octuple $(A, B, C, D, \Phi_{A-C}, \Phi_{B-D}, f, g)$ that declares a synchronization action given a pair $(a, c) \in \Phi_{A-C} : A \cong C$ of corresponding elements in a base isomorphism Φ_{A-C}. For each such a tuple in states (ω_L, ω_R), the synchronization block specifies that the elements $(f(a, \omega_L), g \nearrow (b, \omega_R)) \in B \times D$ gained by the lenses f and g are isomorphic with regard to Φ_{B-D}.

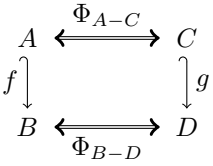

Figure 1: Schematic overview of unidirectional synchronization blocks

A schematic overview of a synchronization block is depicted in Figure 1. The usage of lenses allows this declarations to be enforced automatically and in both directions. The engine simply computes the value that the right selector should have and enforces it using the PUT operation. Similarly, a multi-valued synchronization block is a synchronization block where the lenses f and g are typed with collections of B and D, for example $f : A \hookrightarrow B*$ and $g : C \hookrightarrow D*$ where stars denote Kleene closures.

Synchronization Blocks have been implemented in NMF Synchronizations, an internal DSL hosted by C# [3], [5]. For the incrementalization, it uses the extensible incrementalization system NMF Expressions [6]. This DSL is able to lift the specification of a model transformation/synchronization in three quite orthogonal dimensions:

- **Direction:** A client may choose between transformation from left to right, right to left or in check-only mode
- **Change Propagation:** A client may choose whether changes to the input model should be propagated to the output model, also vice versa or not at all
- **Synchronization:** A client may execute the transformation in synchronization mode between a left and a right model. In that case, the engine finds differences between the models and handles them according to the given strategy (only add missing elements to either side, also delete superfluous elements on the other or full duplex synchronization)

The check-only mode has been implemented very recently and the BibTex to DocBook case has been the first actual usage. In this mode, the engine will try to match the given model and report inconsistencies. This direction is compatible with incremental execution. However, one-way change propagation is not supported[1] (must be two-way or disabled) and the synchronization is required to be bidirectional.

This flexibility makes it possible to reuse the specification of a transformation in a broad range of different use cases. Furthermore, the fact that NMF Synchronizations is an internal language means that a wide range of advantages from mainstream languages, most notably modularity and tool support, can be inherited [7].

3 Solution

When creating a model transformation with NMF Synchronizations, one has to find correspondences between input and target model elements and how they relate to each other. The first correspondence is usually clear and is the entry point for the synchronization process afterwards: The root of the input model is known to correspond to the root of the output model, in our case the `BibTeXFile` element should correspond to the `DocBook` element.

For the isomorphism from a `BibTex` file to a `DocBook`, we need to override the creation of target models. If the `DocBook` element is missing, it should be created with the required sections right from the start. For this, the initialization syntax as depicted in Listing 1 can be used.

To synchronize the contents of a BibTexFile with the contents of a DocBook, we need to specify synchronization blocks accordingly. As an example, the synchronization of the BibTex entries with the paragraphs in the references list is depicted in Figure 2. Because the DocBook does not have a direct reference to the references list, we created a helper method to identify the references list.

[1]The problem here is that the internal DSL of NMF Synchronizations allows a write-only interface for one-way synchronization blocks. Thus, an incrementally maintained list of inconsistencies can only take changes in the source model into account and could therefore produce misleading results which is why the feature has not been implemented, yet. For a bidirectional transformation, this is not a problem, because the list of inconsistencies can always be kept consistent.

```
protected override DocBook CreateRightOutput(...) {
return new DocBook() {
  Books = {
    new TTC2019.LiveContest.Metamodels.Docbook.Book() {
      Id = "book",
      Articles = {
        new TTC2019.LiveContest.Metamodels.Docbook.Article() {
          Title = "BibTeXML to DocBook",
          Sections_1 = {
            new Sect1() { Id = "se1", Title = "References List" },
            new Sect1() { Id = "se2", Title = "Authors list" },
            new Sect1() { Id = "se3", Title = "Titles List" },
            new Sect1() { Id = "se4", Title = "Journals List" }
          }
        }
      }
    }
 }
};
}
```

Listing 1: Creating the default output model for the DocBook metamodel

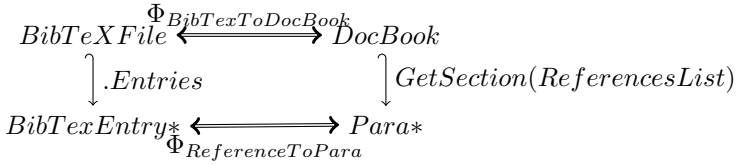

Figure 2: Synchronization block to synchronize entries with the paragraphs in the references list

The implementation for the synchronization blocks from Figure 2 is depicted in Listing 2. In particular, line 1 defines the isomorphism $\phi_{BibTexToDocBook}$, lines 3-5 implement the synchronization block from Figure 2.

NMF Synchronizations uses the incrementalization system of NMF Expressions, which allows to incrementalize also more complex queries. However, queries only have a readonly interface, which is why the C# compiler throws an error if we tried to use it for a bidirectional synchronization. To aid this problem, our solution uses a mocked class `PseudoCollection` that mimics a collection interface on top of a query but throws runtime exceptions if the collection is attempted to be modified. With this trick, the synchronization blocks for the remaining correspondence criteria can be implemented as in Listing 3.

In particular, lines 1-7 specify the synchronization of authors that appear in any of the authored entries with the paragraphs in the authors list. For this, we just iterate over all entries that are authored entries, select all their autors, do a deduplication and sort the result by the author name. Lines 9-13 specify the synchronization of titles in a very similar way. Last, lines 15-21 synchronize the journal names with the paragraphs in the corresponding section.

These synchronization blocks all reference other synchronization blocks responsible for how the paragraphs are laid out exactly. In particular, the synchronization rule `ReferenceToPara` has an instantiating rule[2] for each concrete type in the BibTex model.

Furthermore, the synchronization rules can (and in general should) specify when NMF Synchronizations should establish a correspondence link between two existing model elements. This is in particular required if a

[2]NMF Synchronizations – as NMF Transformations – supports a form of transformation rule superimposition. This concept is called transformation rule inheritance as it is realized through inheritance of the respective transformation rule or synchronization rule. In contrast, an instantiating rule is a child rule that maps a part of the input space (usually denoted through a subtype of the input type) to a part of the output space (again, usually denoted through a subtype of the output type), cf. [8].

```
public class BibTexToDocBook : SynchronizationRule<BibTeXFile, DocBook> {
    public override void DeclareSynchronization() {
       SynchronizeMany(SyncRule<ReferenceToPara>(),
         bibTex => bibTex.Entries,
         docBook => GetSection(docBook, "References List"));
    }
}
```

Listing 2: Definition of synchronization blocks from Figure 2 in NMF Synchronizations

```
SynchronizeMany(SyncRule<AuthorToPara>(),
  bibTex => new PseudoCollection<IAuthor>(
      bibTex.Entries.OfType<IAuthoredEntry>()
            .SelectMany(entry => entry.Authors)
            .Distinct()
            .OrderBy(a => a.Author_)),
  docBook => GetSection(docBook, "Authors␣list"));

SynchronizeMany(SyncRule<TitledEntryToPara>(),
  bibTex => new PseudoCollection<ITitledEntry>(
      bibTex.Entries.OfType<ITitledEntry>()
            .OrderBy(en => en.Title)),
  docBook => GetSection(docBook, "Titles␣List"));

SynchronizeMany(SyncRule<JournalNameToPara>(),
  bibTex => new PseudoCollection<string>(
      bibTex.Entries.OfType<Bibtex.IArticle>()
            .Select(article => article.Journal)
            .Distinct()
            .OrderBy(journal => journal)),
  docBook => GetSection(docBook, "Journals␣List"));
```

Listing 3: Synchronization blocks to synchronize the authors in the authors list, the titles in the titles list and the journals in the journals list.

```
public override bool ShouldCorrespond(IBibTeXEntry left, IPara right, .. context) {
  return right.Content != null && right.Content.StartsWith($"[{left.Id}]");
}
```

Listing 4: Specification when a paragraph should be considered corresponding to a BibTex entry

synchronization is applied to existing models.

In our case, we could establish a correspondence link in case the article starts with the Id of the entry in square brackets as implemented in Listing 4. This allows to synchronize elements with custom global identifiers.

The solution offers two ways to execute it. In the batch mode, the synchronization is run in check-only mode without change propagation, loading the reference `BibTex` model and the mutated `DocBook` model. The implementation is depicted in Listing 5.

Alternatively, the incremental version synchronizes the initial `BibTex` model with the initial `DocBook` model, but with change propagation set to two-way as depicted in Listing 6. Afterwards, the solution loads the target model changes and applies them. Because the changes are propagated as they are applied, the collection of inconsistencies is already maintained.

Besides the synchronization engine also supports a greenfield transformation (i.e. where no target model exists when the transformation is started), the solution is configured to run the initial synchronization in a brownfield setting both in incremental and batch mode. In the batch mode, only inconsistencies between the source model and existing target model are collected, in the incremental setting the synchronization engine collects inconsistencies (there are none as the source model and initial target model are consistent) and installs change propagation hooks to get notified when further changes cause new inconsistencies or resolve existing ones.

In the solution, the inconsistencies are only counted. As an alternative, the changes can be inspected to find out what these inconsistencies actually look like and offer functions to resolve them on either left or right model.

4 Evaluation

We noted that the original solution produced different results in batch or incremental mode. The reason for this was that the definition of correspondence between authors and paragraphs being that a correspondence should be established if the content of the paragraph is the name of the author. However, this is sensitive to changes that modify the contents of the paragraph. While the incremental execution is aware that the paragraph

```
var context = transformation.Synchronize(ref bibTex, ref docBook,
    SynchronizationDirection.CheckOnly,
    ChangePropagationMode.None);
Report("Run", context.Inconsistencies.Count);
```

Listing 5: Running the synchronization in batch mode

```
var context = transformation.Synchronize(ref bibTex, ref initialDocBook,
    SynchronizationDirection.CheckOnly,
    ChangePropagationMode.TwoWay);
var changes = repository.Resolve(...).RootElements[0] as ModelChangeSet;
Report("Load");
changes.Apply();
Report("Run", context.Inconsistencies.Count);
```

Listing 6: Running the synchronization incrementally

corresponds to the author and therefore reports a single inconsistency (that the content of the paragraph no longer matches the name of the author), the batch execution reports two inconsistencies: One that a paragraph has no corresponding author and another that an author has no corresponding paragraph. The reason for this is that the batch mode does not have the history and is not aware that the paragraph was corresponding to the author before.

This behavior can be easily fixed by actually using the unique identifiers that happen to exist in the metamodel in question as they can be used to identify the *same* paragraph even though its contents have changed. Doing so, also the batch execution detects that the contents of the paragraph have changed. The default behavior for matching elements is currently not taking identifiers into account, because NMF Synchronizations is independent of the model representation of NMF.

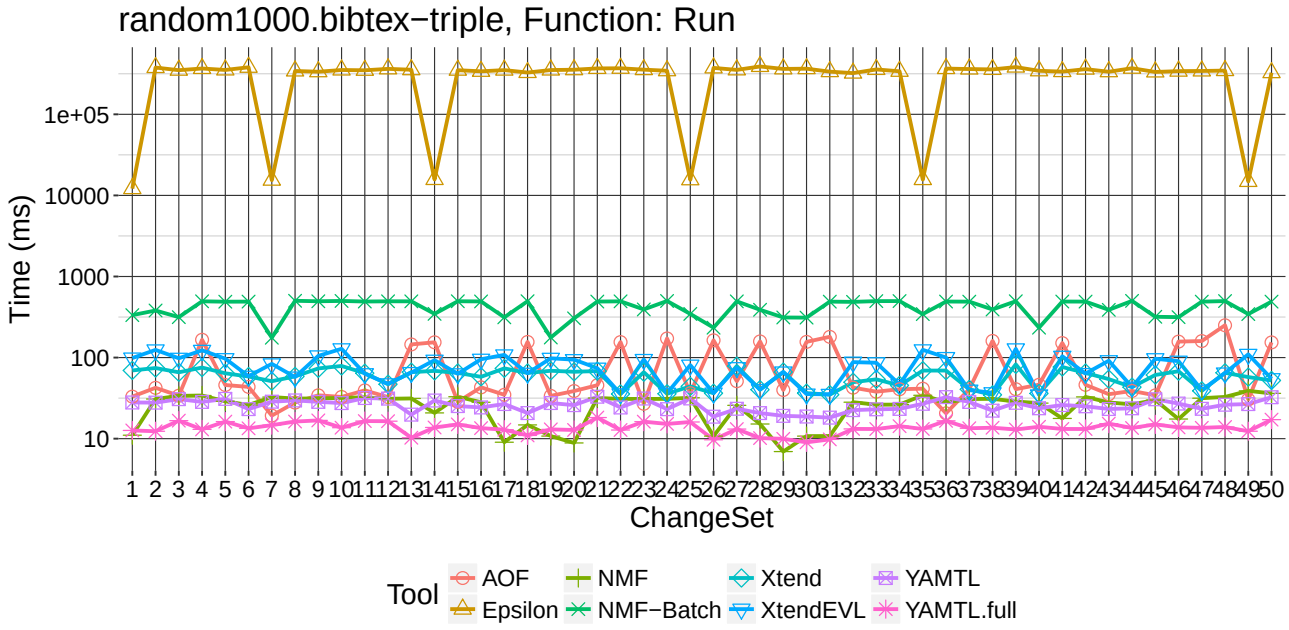

Figure 3: Performance results of the solutions at the TTC 2019

Figure 3 shows the performance results of the solutions submitted to the TTC 2019 for the largest input model provided with triple changes. These results were produced through a Google Cloud Compute Engine c1-standard-4 machine with 16GiB of RAM and 30GB SSDs, using a Docker image produced through the Dockerfile in the root of the benchmark repository. For NMF, two versions are depicted. The NMF version uses the incremental change propagation of inconsistencies. In contrast, the NMF Batch solution runs a complete consistency check on the source model and modified target model, reflecting a scenario where a description of changes is not available.

The results show that the batch version is significantly slower than competing solutions, which is clear, given that the entire modified target model has to be considered and correspondence relations have to be established. The incremental solution, however, is among the fastest solutions, indicating that it is about as fast as solutions that basically analyze the target model changes manually.

At the TTC 2019, the NMF solution was the only solution that was able to entirely use the same consistency specification for transforming the source model to the target model and to identify inconsistencies afterwards. Further, the NMF solution has won the best integration award and the first place in the audience award.

5 Conclusion

We think that the NMF solution highlights the advantages model transformations based on synchronization blocks can offer in terms of flexibility. A single specification of consistency relationships between the `BibTex` model and the `DocBook` model suffices to transform an existing `BibTex` model to a `DocBook` model or identify inconsistencies between an existing source and an existing target model. Furthermore, both the transformation and the consistency check can be run incrementally, i.e. the synchronization engine attaches to the source (or target) model and reports or resolves inconsistencies as they occur.

References

[1] M. E. Kramer, "Specification languages for preserving consistency between models of different languages," PhD thesis, Karlsruhe Institute of Technology (KIT), 2017, 278 pp.

[2] A. Garcia-Dominguez and G. Hinkel, "The TTC 2019 Live Case: BibTeX to DocBook," in *Proceedings of the 12th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2019) federation of conferences*, ser. CEUR Workshop Proceedings, CEUR-WS.org, 2019.

[3] G. Hinkel and E. Burger, "Change Propagation and Bidirectionality in Internal Transformation DSLs," *Software & Systems Modeling*, 2017.

[4] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, "Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 29, no. 3, 2007.

[5] G. Hinkel, "Change Propagation in an Internal Model Transformation Language," in *Theory and Practice of Model Transformations: 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015. Proceedings*, Springer International Publishing, 2015, pp. 3–17.

[6] G. Hinkel, R. Heinrich, and R. Reussner, "An extensible approach to implicit incremental model analyses," *Software & Systems Modeling*, 2019.

[7] G. Hinkel, T. Goldschmidt, E. Burger, and R. Reussner, "Using Internal Domain-Specific Languages to inherit Tool Support and Modularity for Model Transformations," *Software & Systems Modeling*, pp. 1–27, 2017.

[8] G. Hinkel, "An approach to maintainable model transformations using an internal DSL," Master's thesis, Karlsruhe Institute of Technology, 2013.