The TTC 2019 Live Case: BibTeX to DocBook (v1.0)

Antonio García-Domínguez
Aston University
B4 7ET, Birmingham, United Kingdom
a.garcia-dominguez@aston.ac.uk

July 15, 2019

The initial transformation of a model into another model is only the first step. After the creation of the target model, it may be manually changed and the consistency with the source model may be lost or obscured. Ideally, transformation tools should have a way to check the degree of consistency between the source model and the current version of the destination model. This case presents such a scenario for a small transformation, with an automated mutation tool which will introduce changes that may or may not impact consistency. The aim of this case is to evaluate the speed and verbosity of the inter-model consistency checking in the state of the art.

1 Introduction

This live case is based on the original ATL Zoo [2] BibTeX to DocBook transformation, where a simplified version of the BibTeX reference manager's data model (shown in Figure 1) is transformed to a simplification of the DocBook document typesetting tool (shown in Figure 2).

The transformation is quite simple:

- From the root BibTeXFile, a DocBook is created. The DocBook has a Book with an Article titled "BibTeXML to DocBook", which has in turn four Sect1 instances: "References List", "Author List", "Titles List" and "Journals List".
- Any BibTeX Author is listed in the "Authors List" as a Para.
- Any BibTeXEntry is listed in the "References List", including its unique identifier and any available information.

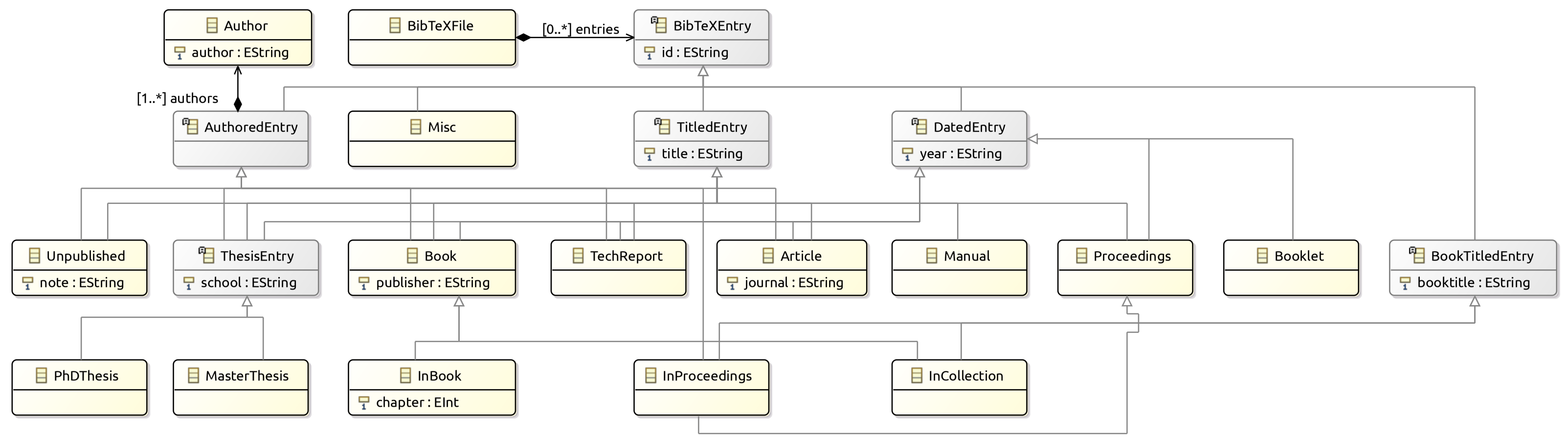

Figure 1: Class diagram for the input BibTeXML metamodel

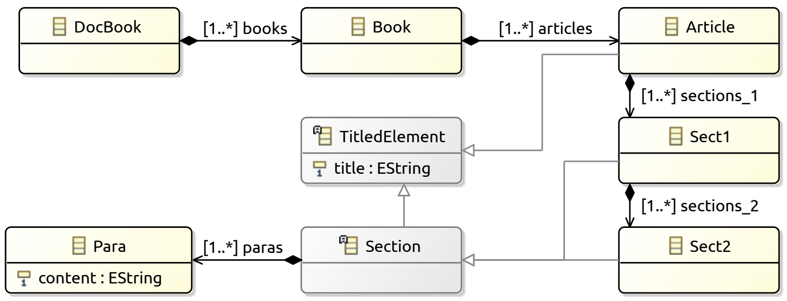

Figure 2: Class diagram for the target DocBook metamodel

- For each TitledEntry, a Para with its title is added to the “Titles List”
- For each BibTeX Article, a Para with its journal name is added to the “Journals List”.
- The “Author List”, “Titles List”, and “Journals List” are all sorted lexicographically in ascending order.

The transformation is somewhat different in that it involves some sorting. The original ATL-based implementation worked well for instances with 10, 100 and 1000 entries, but seems to struggle with instances with 10000 entries.

Still, a more interesting problem is the case when the target model continues to be worked on after the initial transformation. Suppose that we gave the DocBook document to an editor, which would reorganise sections, add paragraphs in the middle with further commentary and perhaps extend some of the text itself. The task would be to ensure that these manual editions have not impacted the consistency of the DocBook model with the original BibTeX model.

Ideally, the transformation tool or the editing environment would give some infrastructure to tackle this. However, for many tools, the approach seems to be only feasible through the creation of an external consistency checker. This case is to evaluate the current state of the art in out-of-the-box after-the-fact consistency checking. To do so, the case provides a generator which can produce source models of arbitrary size, a repackaged version of the original ATL transformation, and a mutator which can make a number of changes to the model, some of which may impact consistency.

All resources for this case are available on Github[1]. Please follow the link in the footnote and create a pull request with your own solution.

The rest of the document is structured as follows: Section 2 describes the structure of the live case. Section 3 describes the proposed tasks for this live case. Section 4 mentions the benchmark framework for those solutions that focus on raw performance. Finally, Section 5 mentions an outline of the initial audience-based evaluation across all solutions, and the approach that will be followed to derive additional prizes depending on the attributes targeted by the solutions.

[1] https://github.com/TransformationToolContest/ttc2019-live

2 Case Structure

The case is intended to review the different approaches for checking after-the-fact inter-model consistency between a BibTeX model and a DocBook model. The process is roughly as follows:

1. The BibTeX model is generated randomly to a certain size, by the included *generator* in the `models/generator.jar` JAR. The generator uses a Java port of the Ruby Faker[2] library to produce random data.

   The generator is simple to use, if you wish to produce larger models:

   ```
   java -jar generator.jar destination.bibtex size [seed]
   ```

   `destination.bibtex` is the path to the file to be produced, and `size` is an integer with the number of desired entries. The random `seed` is an optional integer that will produce consistent results across reruns.

   A number of random models (sizes 10, 100, 1000 and 10000) have been produced already and are included in the case resources.

2. The BibTeX model is transformed automatically to DocBook by the repackaged version of the original ATL transformation in the `bibtex2docbook.jar` JAR. This transformation deals well with models up to 1000 entries, but seems to struggle with larger models. It can be used like this:

   ```
   java -jar bibtex2docbook.jar in.bibtex out.docbook
   ```

3. The DocBook model is edited, in this case with the automated *mutator* in the `models/mutator.jar` JAR. It can be invoked with the following command:

   ```
   java -jar mutator.jar source.docbook \
     dirPrefix nMutants [nMutationsPerMutant=1] [seed=time]
   ```

   The mutator will operate on a DocBook file, creating a set of folders whose path will start with the specified prefix, adding `-N` from 1 to `nMutants`. Each folder will contain the mutated DocBook model, as well as a *change model* explaining what was done to the model: Figure 3 shows that such models have a ModelChangeSet as the root, with a number of ModelChange instances of various types. The DocBook model will have gone through a number of random mutations according to a seed: if unspecified, the seed will be based on the current system time.

   The mutator has a number of predefined *mutation operators* that will modify the model:

[2] https://github.com/DiUS/java-faker

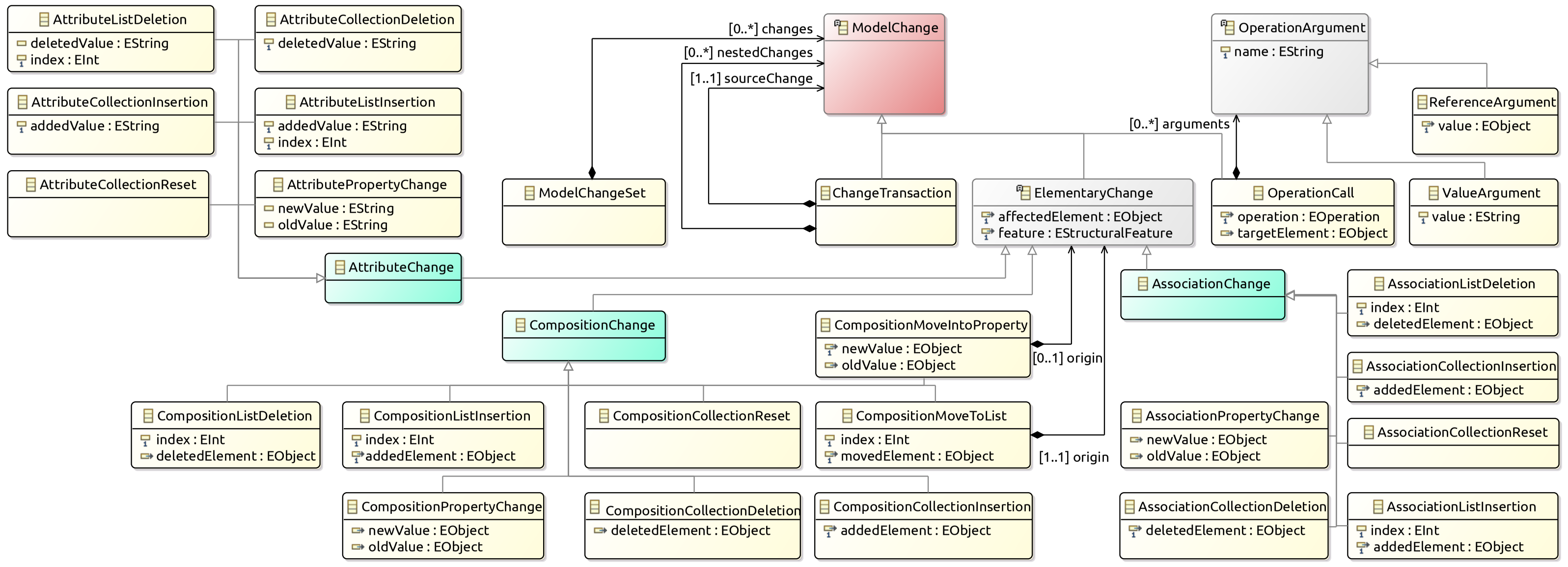

Figure 3: Class diagram for the Changes metamodel

- Swapping paragraphs: this should break consistency in terms of sorting, but it will not result in missing information.
- Swapping sections should not break consistency.
- Deleting paragraphs/sections should break consistency.
- Appending text to a paragraph should not break consistency.
- Adding a new paragraph: unless it happens to match one of the authors, titles, or journals in its SECT1, it should not break consistency.

The mutated models have been created already. There are three sets of mutated models from the generated 10/100-entry models: one with a single mutation, one with two mutations, and one with three mutations.

4. A *consistency checker* would take any combination of the previous artifacts (source BibTeX, fresh DocBook, mutated DocBook, change model) and make a judgment about whether the mutated DocBook is still consistent or not.

   If issues are found, it should point to the element in the source model which lacks a proper mapping on the other side, or the element in the target model which is not mapped correctly from the source model (e.g. it is not sorted anymore).

3 Task Description

There is an optional task and a mandatory task in this case:

- The optional task is to re-implement or improve the original transformation itself, in a way that lends itself better to after-the-fact consistency checking. Your transformation tool may have better support for this, or ATL could be made to deal better with larger versions of this model.

- The mandatory task is to check for the consistency of the source BibTeX against the mutated DocBook models in the `models` directory, and report this information as efficiently and clearly as possible to the user. To evaluate the efficiency of your approach, you must use the benchmarking framework further detailed in Section 4. Ideally, this should be possible without a full re-run of the transformation.

Solutions can focus on efficiency, conciseness, or clarity of presentation to the user. Solutions that can operate straight from the definition of the transformation (i.e. without a separate consistency checker) would be preferred.

4 Benchmark Framework

If focusing on performance, the solution authors should integrate their solution with the provided benchmark framework. It is based on that of the TTC 2017 Smart Grid case [1], Listing 1: `solution.ini` file for the reference Epsilon solution

```
[build]
default=true
skipTests=true

[run]
cmd=JAVA_OPTS="-Xms4g" java -jar epsilon.jar
```

and supports the automated build and execution of solutions. For this specific case study, the visualisation of the results is currently disabled.

The benchmark consists of three phases:

1. **Initialization**, which involves setting up the basic infrastructure (e.g. loading metamodels). These measurements are optional.
2. **Load**, which loads the input models.
3. **Run**, which runs the consistency checking, finding a number of consistency violations in the mutated DocBook model.

4.1 Solution requirements

Solutions should be forks of the main Github project[3], and should be submitted as pull requests.

Each solution wishing to use the benchmarking framework should print to the standard output a line with the following fields, separated by semicolons (";"):

- **Tool**: name of the tool.
- **MutantSet**: set of mutants used ("single", "double" or "triple").
- **Source**: base name of the input BibTeX model (e.g. "random10.bibtex").
- **Mutant**: integer starting at 1, identifying the mutant model within this set.
- **RunIndex**: index of the run of this combination of tools and inputs.
- **PhaseName**: name of the phase being run.
- **MetricName**: the name of the metric. It may be the used **Memory** in bytes, the wall clock **Time** spent in integer nanoseconds, or the number of consistency **Problems** found in the mutated DocBook model.

[3] https://github.com/TransformationToolContest/ttc2019-live

To enable automatic execution by the benchmark framework, solutions should add a subdirectory to the `solutions` folder of the benchmark with a `solution.ini` file stating how the solution should be built and how it should be run. As an example, the `solution.ini` file for the reference solution is shown on Listing 1. In the `build` section, the `default` option specifies the command to build and test the solution, and the `skipTests` option specifies the command to build the solution while skipping unit tests. In the `run` section, the `cmd` option specifies the command to run the solution.

The repetition of executions as defined in the benchmark configuration is done by the benchmark. For 5 runs, the specified command will be called 5 times, passing any required information (e.g. run index, or input model name) through environment variables. Solutions must not save intermediate data between different runs: each run should be entirely independent.

The name and absolute path of the input model, the run index and the name of the tool are passed using environment variables `Tool`, `MutantSet`, `SourcePath`, `Mutant`, `MutantPath`, and `RunIndex`. Solution authors are suggested to study the reference solution on how to use these values to run their transformation.

4.2 Running the benchmark

The benchmark framework only requires Python 3.3 to be installed. Furthermore, the solutions may imply additional frameworks. We would ask solution authors to explicitly note dependencies to additional frameworks necessary to run their solutions.

If all prerequisites are fulfilled, the benchmark can be run using Python with the command `python scripts/run.py`. Additional options can be queried using the option `--help`. The benchmark framework can be configured through the `config/config.json` file: this includes the input models to be evaluated (some of which have been excluded by default due to their high cost with the sample solution), the names of the tools to be run, the number of runs per tool+model, and the timeout for each command in milliseconds.

5 Evaluation

The evaluation will operate on several dimensions:

- How efficient is the approach in time and space (memory)? The reference ATL solution struggled with large models, and the reference solution has not been designed with performance in mind.
- Is consistency checking directly supported by the transformation approach? Many tools lack this capability, though it might be interesting as an additional execution mode if the target model has seen manual changes since it was generated.
- How informative and accessible is the feedback that can be provided by the approach?

References

[1] Georg Hinkel. The TTC 2017 Outage System Case for Incremental Model Views. In *Proceedings of the 10th Transformation Tool Contest*, volume 2026, pages 3–12, Marburg, Germany, July 2017. CEUR-WS.org.

[2] Guillaume Savaton. BibTeXML to DocBook, ATL Transformations. https://www.eclipse.org/atl/atlTransformations/#BibTeXML2DocBook, February 2006. Last accessed on 2019-07-15. Archived on http://archive.is/HdoHM.