Model Transformation - Exercises

February 8, 2021

Exercise 1: Miscellaneous

Two of many examples of model transformations from my past experience:

Example: GUI Generator (SET GmbH)

The GUI-Generator is one of the proprietary MDE tools of the SET GmbH. Based on an XSD model, the tool will generate a GUI (Google Web Toolkit (GWT) - Java), which can be used to create instances (XML files) conforming to the XSD model. Like EMF Forms, there are predefined different UI-Elements for each datatype. Therefore, the generated UI is functional and has the same look and feel across the software products of the SET GmbH. Nevertheless, usability and user experience are not optimal. However, the same XSD model is also used to generate most of the persistence layer for the instances created by the UI.

Classification:

1. **out-place:** The XSD model does not change, and new source code representing a GUI to create instances of the model is generated.
2. **heterogeneous:** An XSD model conforming to the XSD metamodel is transformed into source code representing GUI in GWT (Java).
3. **declarative:** Reading the XSD model and generating the output code was implemented in Java by developers of the SET GmbH.

Example: Gymnastic elements generation

After a bit of work, I extracted all possible gymnastics elements from the official international regulation (PDF file) into a machine-readable CSV file. In each row, the name, difficulty, element group, and more information for an element are stated. I am using this CSV file (in fact, multiple files, one for each apparatus) to generate source code in typescript, which is used in the app to make the user pick from every possible element.

Classification:

1. **out-place:** The CSV file is untouched, and new source code is generated.
2. **heterogeneous:** An instance of a CSV/spreadsheet metamodel is transformed into source code representing a list of objects conforming to an object-oriented model in typescript.
3. **declarative:** Reading the CSV file and generating the output code was implemented in Java.

Compiler and model transformations

1. Lexer: Transforms text to text and tokens, so a text to model transformation is given.
2. Parser: Parsers transform text (and tokens) to another model such as an abstract syntax tree. So, they implement T2M transformations.
3. Optimizer: Creates a new model with the same structure and behavior as the old one but tries to minimize execution time, memory footprint, etc. In my opinion, it is a M2M transformation.
4. Code generator: Generating code is a classic M2T transformation.

Exercise 4: Bidirectional Transformations

- pairs of numbers `(Int,Int)` $\leftrightarrow$ their sum `(Int)`:
  $Get(\langle x, y\rangle) = x + y$,
  $Put(sum, complement) = \langle sum - complement, complement\rangle$
- `Person1` entities with `givenName` and `familyName` fields $\leftrightarrow$ `Person2` entities with `name` fields:
  $Get(person) = new\ Person2(person.givenName + \text{" "} + person.familyName)$
  $Put(person, delimiterIndex) = new\ Person1($
  $person.name.substring(0, delimiterIndex)$,
  $person.substring(delimiterIndex + 1))$
- sets {unordered,unique} $\leftrightarrow$ lists {ordered,non-unique}:
  $Get(list)$ = remove duplicate elements and forget order
  $Put(set, totalOrder, additionalElements) =$
  1. Create empty list
  2. Add elements from the set
  3. Add additional elements
  4. Sort elements according to the total order given by an ordering function

- java classes ↔ database schema:
  Requires put from db to java since operations and methods are typically not saved in a database schema.
  Requires put from java to db since there might be triggers and other things defined for a table.