Model Transformation - Exercises

Exercise 1: Miscellaneous

1. Find more examples of model-to-text and model-to-model transformations from your previous developer/student experience. Classify them along the dimensions:
   - in-place vs. out-place
   - homogenous vs. heterogeneous
   - declarative or imperative
2. Would you consider the different components of a compiler (lexer, parser, optimizer, code generator, see also Fig. 1) as model transformations? Why or why not?

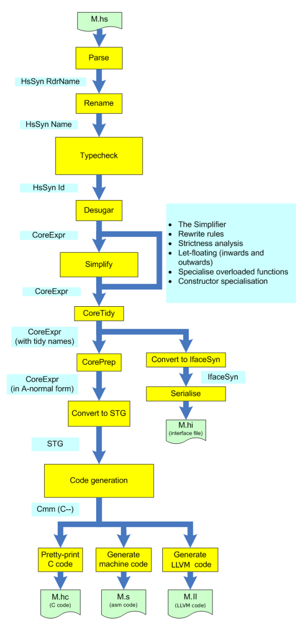
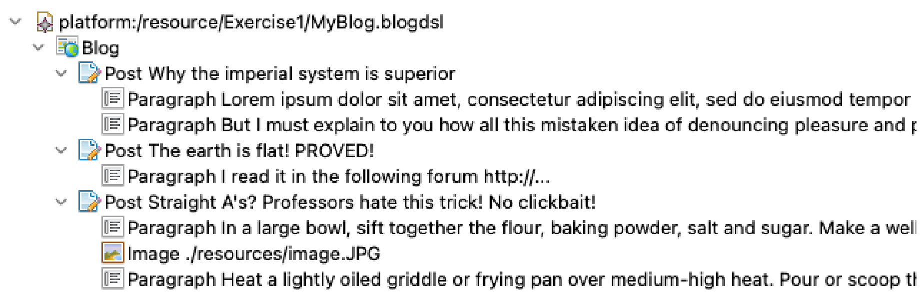


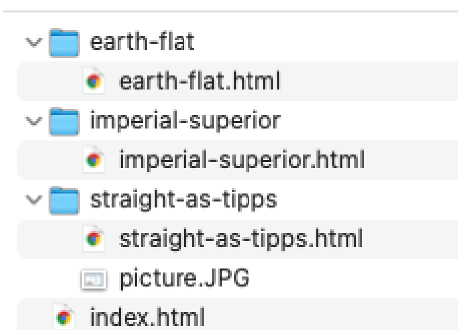
Figure 1: Haskell compiler pipeline

Exercise 2: Blog generator (m2t)

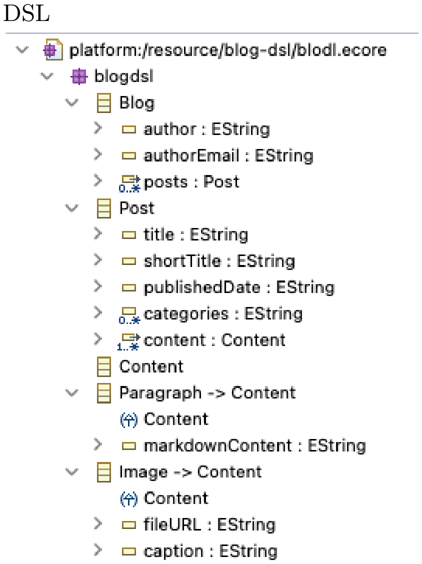
In this exercise, your task is to write a model-to-text transformation with EGL that provides a *web blog* generation functionality similar to tools like e.g. *Jekyll*[1]. The blog content is abstractly defined in a *BLOg Description Language (BLODL)* (Fig. 2a, which is defined by the metamodel in Fig. 2c. The resulting file structure (Fig. 2b) should contain a folder for each blog *post* containing the respective *html* file and referenced *images*[2]. The root should contain an html file with hyperlinks to all blog posts. Feel free to also add your own custom CSS styles into the generation.



(a) Blog DSL



(b) Resulting File Structure



(c) Blog DSL Metamodel

[1] `https://jekyllrb.com/`

[2] This will require you to copy files. For this recall that you can write EOL statements in EGX-scripts, which again allows to call arbitrary Java methods.

Exercise 3: Object-Relational-Mapping (m2m)

In this exercise, your task is to write a model-to-model transformation using ETL that performs an object-relational mapping between `Ecore` and the relational model [1]. Recall that an `.ecore` model is yet another model that is typed over the `Ecore` metamodel. An excerpt of the relevant concepts is shown in Fig. 3 (You do not need to consider methods and other more technical details.). To find the complete definition of the Ecore metamodel, open the *Plug-in Dependencies* of any project in the `metamodel-ws` workspace and then navigate to *org.eclipse.emf.ecore_...* → *model* → `Ecore.ecore`. A metamodel of the relational model, that we are using in this exercise is found in `metamodel-ws/relational-model/relational.ecore`.

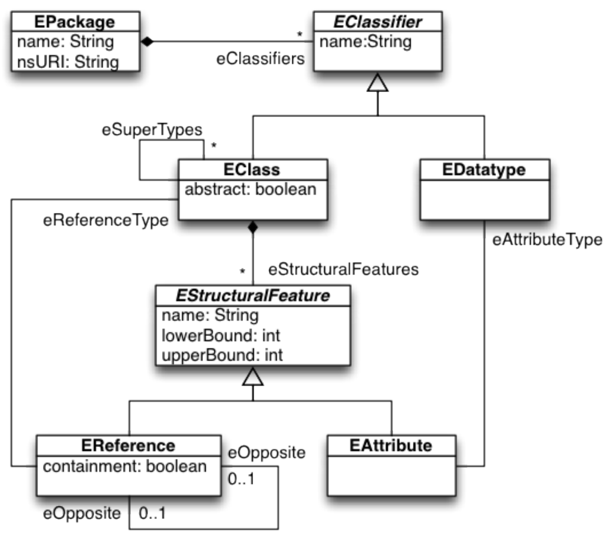


Figure 3: Ecore metamodel (simplified)

It is recommended to study both metamodels carefully before you begin and develop your solution step-wise.

1. Start with a `Schema` for every `EPackage` and for every `EClass` in the package you generate a corresponding `Table` that contains a single numeric *id* `Column` with a `PrimaryKey` constraint on it. The name of the `Table` should correspond to the `EClass`, i.e. identical, capitalized or etc.

2. In the next iteration, create a `Column` in each `Table` for every `EAttribute` of the corresponding `EClass`[3]. `EDataType`s translate as follows:

   - `EString` → `Varchar(4000)`

[3]You may require to use the ETL-method `equivalent()` here

- `ELong/EInteger,EShort,EByte` → `Number(32,0)`
- `EFloat/EDoubke` → `Number(32,4)`
- `EBoolean` → `Number(1,0)`
- `EEnum` → `Number(2,0)`

3. In the third iteration, the task is to translate `EReference`s to `ForeignKey`s. Here, you have to pay attention to multiplicities: A many-to-one relation from a table/class A to table/class B is realized by adding a new numeric column to A that has a foreign key to the primary key of B. Many-to-many relations, require to create a new *junction* table with two columns pointing with foreign keys to the respective primary keys of A and B.

4. In the final iteration, inheritance must be handled. Implement all three strategies [3]

   - Single Table Inheritance (table per inheritance hierarchy)[4],
   - Class Table Inheritance (table per class)[5],
   - Concrete Table Inheritance (table per concrete class)[6].

   and design a mechanism such that the user can configure which inheritance mapping strategy should be used[7].

[4] `https://martinfowler.com/eaaCatalog/singleTableInheritance.html`
[5] `https://martinfowler.com/eaaCatalog/classTableInheritance.html`
[6] `https://martinfowler.com/eaaCatalog/concreteTableInheritance.html`
[7] There are many possibilities, one may chose to use `EAnnotation`s, an external configuration file etc.

Exercise 4: Bidirectional Transformations

1. Let A and B be two models where A has 5 and B has 4 elements. What is the maximum number of elements in the trace-model [2] of a transformation between A and B if we assume that there cannot be two trace-links referring to the same pair of objects taken from an A and B?

2. Below, you will find a list of transformations. For each case you will have to decide in what direction which can define a GET (derivation) and in which direction you require PUT (back propagation). Maybe it is also sometime possible to define a GET in both directions or you even require a PUT in both directions? If you require a PUT, clearly state *what* additional information you require from the *source*.

   - pairs of numbers `(Int,Int)` ↔ their sum `(Int)`
   - `Person` entities with `givenName` and `familyName` fields ↔ `Person` entities with `name` fields.
   - sets {unordered,unique} ↔ lists {ordered,non-unique}
   - an `Ecore` model ↔ an XSD file
   - java classes ↔ database schema
   - state machines ↔ hierarchical state machines
   - C-code ↔ Python-code

References

[1] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, June 1970.

[2] Nikolaos Drivalos, Dimitrios S. Kolovos, Richard F. Paige, and Kiran J. Fernandes. Engineering a DSL for Software Traceability. In Dragan Gašević, Ralf Lämmel, and Eric Van Wyk, editors, *Software Language Engineering*, Lecture Notes in Computer Science, pages 151–167, Berlin, Heidelberg, 2009. Springer.

[3] M. Fowler. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch.* Addison-Wesley Signature Series (Fowler). Pearson Education, 2012.