A Multi-level Transformation from Conceptual Data Models to Database Scripts using Java Agents

Ahmad F Subahi and Anthony J H Simons

Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello, Sheffield S1 4DP, United Kingdom

{A.Subahi, A.Simons}@dcs.shef.ac.uk

Abstract. This paper presents a framework for model transformation, organised around Java agents. Internally, the agents are hierarchically composed to build each translation step, offering a fine-grained control over the transformation. Externally, a linear composition of translation steps is used to create a multi-level, forwards transformation from high-level models to executable code, in which intermediate representations reduce multi-language dependency. The approach is illustrated with a database generation example.

Keywords. Model-driven engineering, model transformation, composition of model transformations, automatic code generation, *ReMoDeL*

1 Introduction

Model transformation plays a significant role in supporting Model-Driven Engineering (MDE) tasks, such as refactoring and refining models (transformation, translation), and in synchronizing and weaving together different views of a system (merging) [1, 2]. Model transformation is defined as a process of converting one model into another using transformation rules acting on models at different levels of abstraction [3, 4]. A goal is to identify modular and reusable transformation components, which generalise some of the transformation tasks, but which can be deployed in different contexts, such that effort is not wastefully duplicated [2]. This will lead to more comprehensible and maintainable transformation modules [5].

The composition of model transformations has emerged as an important research topic in its own right. Tasks include designing a suitable orchestrating mechanism for controlling the execution of the decomposed transformations, in order to produce a single consistent result [6]. This paper reviews different approaches to composing model transformations in section 2, contrasting these with our own approach, which embodies two kinds of composition. Firstly, our transformations are executed by collaborating Java agents, each with a focus on a different level of detail in the input models. Secondly, we demonstrate a linear composition of transformation stages, in a two-phase forward model transformation approach. Our architecture is described in section 3 and illustrated in section 4 with examples from a case study.

2 Composition of Model Transformations

The composition task is influenced by the diversity of models and transformations to be composed. For instance, one approach to composition simply links several pre-designed model transformations, whether they are expressed using one, or several, languages/metamodels; and executed by one or several tools [7, 8]. This is a heterogeneous approach. In contrast, Hidaka et al. [2] develop a homogeneous compositional framework for graph-based model transformations, by extending the graph query language UnQL, which operates on graphs encoded in the JSON format. Transformation steps consist of extending, replacing or deleting subgraphs. Larger transformations integrate several steps via intermediate graph representations.

Composition is also influenced by what designers perceive to be the basic units of modularity. Typically, whole rules are the units of modularity, converting source to target elements in a single step; however more complex problems require finer-grained units [9], breaking each rule down into atomic CRUD actions upon target models [10]. Bespoke compositions of these actions can be maintained in a single translation module, which helps to maintain consistency and modifiability, which is not possible in the whole-rule approach, which splits a single complex mapping over several rules. At the other extreme, Wagelaar [11, 12] proposes a technique for composing rule transformations, called module superimposition. This internally composes the rules from two or more model transformations into a new, single transformation. Module superimposition has been implemented for the Atlas Transformation Language (ATL) [7] and the QVT Relations [13] language.

Our work adopts the direct-manipulation approach [3, 6] to model transformation, using Java for the rules and XML for the models. It employs two composition strategies. The first (internal) strategy deploys a hierarchy of Java agents, each concentrating on a particular level of detail in the source model and delegating to collaborators for the next level of detail. The second (external) strategy involves a linear composition of transformations, whereby we perform a multi-level translation, with intermediate representations.

3 ReMoDeL Database Generation Framework

The ReMoDeL project [14] aims to develop a proof-of-concept for MDE using simple, available technologies such as Java and XML. The approach is based on a layered, forwards-transformation methodology leading from multiple abstract views, via intermediate representations, to concrete models and generated code. XML-based modelling languages express different aspects of the system, such as the high-level Work Flow Graphs (WFG), and Database and Query (DBQ) models [15, 16], or the low-level Structured Programming Language (SPL) or Object-Oriented Programming (OOP) models [14, 16].

We believe that forcing model transformation via intermediate representations improves the quality of a translation, by making each step explicit and controllable. It reduces the number of different source-to-target mappings that will eventually be

required (c.f. the EU language translation problem). It reveals intermediate levels at which new constraints may usefully be applied (c.f. the work of Marr in computer vision [18]). In general, a multi-layered approach provides better support for folding together different abstract views of a system, as translations progress towards the concrete. The lowest-level models contain complete, generic implementation details, ready for automatic code generation on different platforms [14, 16].

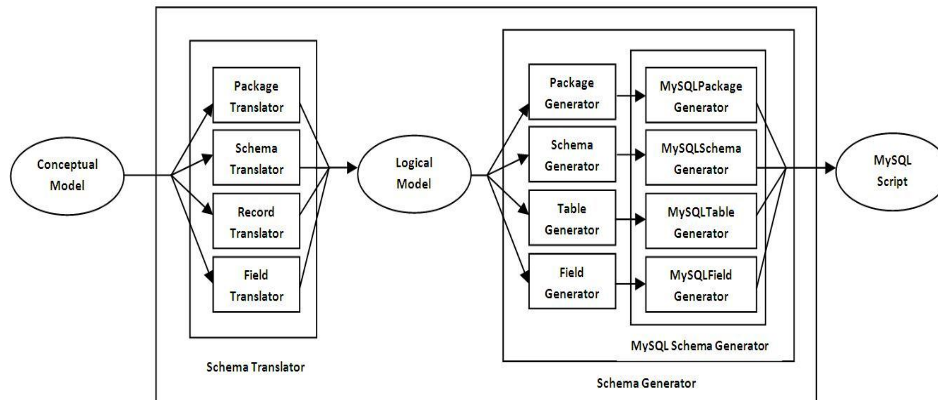

Fig. 1. Composition of schema translation and database generation

The database generation framework reported in this paper is one example of this approach, which transforms a high-level conceptual data schema into executable database scripts. The framework outwardly consists of a linear composition of two transformations (fig. 1). The first step is a model-to-model translation from a conceptual data model, consisting of records and semantic relationships, into a low-level logical model, consisting of tables, both expressed in the DBQ language. This stage performs data normalisation and is common to all schema translations, being totally independent of any target language. The second step generates code in the particular idioms of any database scripting language, using MySQL for illustration, but we also remark on differences where the target for generation is the Oracle database.

Composition also plays a role internally. The initial translation step is performed by a hierarchical composition of Java agents, each concerned with a different level of detail in the source models (fig. 2). Each class obeys a *translate (source, target)* protocol, acting on a pair of input and output models, and delegates to sub-translators that follow the same protocol, when the appropriate level of model detail is reached. The compositional hierarchy of the translators follows the structure of the high-level DBQ concepts, namely, *PackageTranslator*, *SchemaTranslator*, *RecordTranslator*, and *FieldTranslator*. This decomposition improves the maintainability and modularity of the translation code. Translation rules are Java methods that explore the structure of the source model and perform surgery upon the target model (both expressed as graphs, encoded as XML trees). The orchestration is controlled by determining a suitable order for handling the high-level DBQ model concepts.

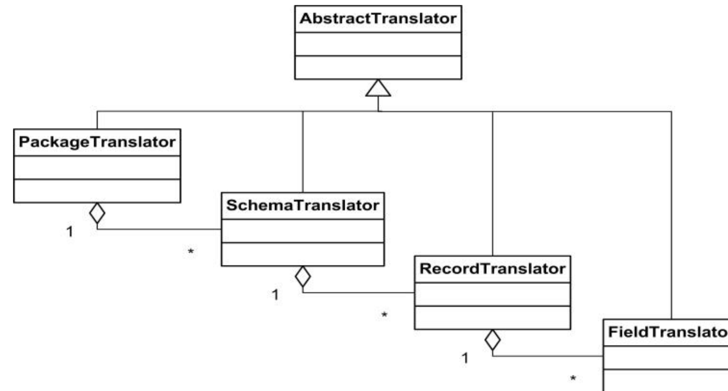

Fig. 2. The internal composition of agents for the schema translation step

The architecture for the second database generation phase (not illustrated) is similar, consisting of a compositional hierarchy of *PackageGenerator, SchemaGenerator, TableGenerator* and *FieldGenerator* agents. These classes are an abstract layer in a framework that is specialised for the different flavours of SQL required for different database engines. In this example, four specialised generators *MySQLPackageGenerator, MySQLSchemaGenerator, MySQLTableGenerator* and *MySQLFieldGenerator* are used to produce specific code for the MySQL database engine. These classes follow the protocol *generate (source),* acting on a single input model and generating database scripts in the MySQL language. The layer of abstract classes captures what is common in the synthesis of generic SQL, common to all database vendors. A different specialisation layer may be used to generate Oracle SQL. Further specialised layers may be provided for each new target database vendor [16].

4 Case Study: Online Ordering System

This section presents a case study for model transformation, the "Online Ordering System". The initial model is a fairly complex conceptual data schema, encoded in DBQ and visualised as the UML class diagram in (fig. 3). This model contains records, associations, generalisations and aggregation relationships, with the illustrated semantic refinements. The first model translation step maps from this to the logical data model illustrated in (fig. 4). This contains selectively normalised tables and fields, some of which are marked as primary keys.

In general terms, the translation rules map records in the source to tables in the target having similar sets of fields. The mapping is not strictly one-to-one, since tables may be merged; and extra linker tables may be created. Semantic relationships in the source are rendered implicitly in the target as linked pairs of primary key and foreign key fields. Specific patterns are handled as follows.

Associations: Where records are in 1:1 association (*Person, Address* in fig. 3), these are merged to satisfy 3NF; the retained concept acquires the renamed fields of the deleted concept. Where records are in 1:M association (*Customer, Order* in fig. 3),

this is translated into a foreign key (FK) on the many-side (*Order.customerID* in fig. 4) relating to the primary key (PK) on the one-side (*Customer.personID* in fig. 4). Where records are in M:M association (*Supplier, Item* in fig. 3), a linker table is created for the whole association (*Supply* in fig. 4), storing FKs for each related table (*Supply.dealer, Supply.item*). The linker table is named after the association, which is named in the source model. Where associations have their own fields (c.f. a UML association class), these are also promoted to tables in the target model.

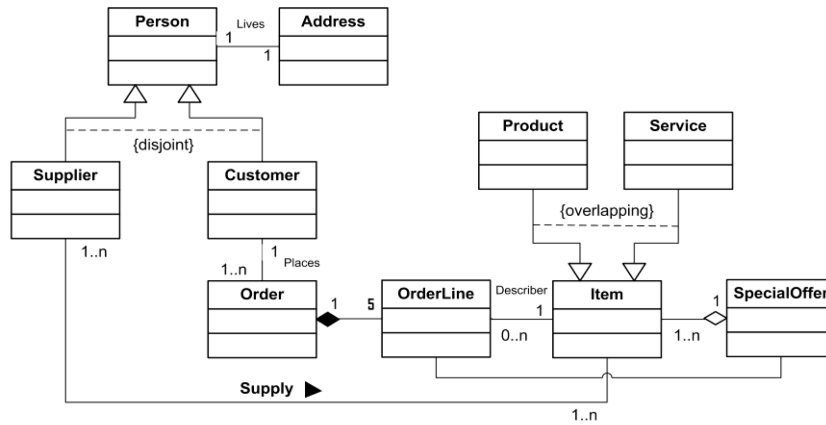

Fig. 3. The conceptual schema for the *Online Ordering System*

Generalisations: Where subclasses are *disjoint* (*Supplier, Customer* in fig. 3), tables are created only for the concrete subclasses; the fields of the abstract parent (*Person*) are replicated in each table and the parent is deleted. Where subclasses are *overlapping* (*Product, Service* in fig. 3), tables are created for all records, preserving 3NF. Note how the rules make an intelligent decision about selective de-normalisation, for increased speed of retrieval (eliminating the need to join tables). So far, we have not yet explored a third "fat superclass" strategy for the overlapping case. The translation does not currently handle multiple inheritance.

Aggregations: A weak aggregation relationship (*SpecialOffer, Item* in fig. 3) is treated like a 1:M association, inserting a FK in the part (*Item.specialOfferCode* in fig. 4) relating to a PK in the whole (*SpecialOffer.code*). Where a stronger *composition* relationship is indicated (*Order, OrderLine* in fig. 3), the repeated fields of the part are indexed and merged with the whole (*Order*, in fig. 4). This is another optimisation, to speed the retrieval of whole orders. Selective de-normalisation plays a key role in the automated transformation decision to achieve a reasonable balance between query complexity, system performance and disk space [16].

Key fields: Records may suggest candidate key fields using a tagged value from: *key = {total, partial, auto}*. A *total* field is used as the PK. The set of all *partial* fields is used as a compound PK. If no key field is marked, an *auto* PK is generated. PKs may be demoted to dependent unique fields by other rules; for example, when merging 1:1 associations, one PK is retained, the other demoted. Likewise, when flattening a disjoint generalisation, the parent's PK is retained, and the subclasses'

keys are demoted. Again, in a composite aggregation, the key for the whole is retained and the keys of the enclosed parts are demoted. If an *auto* PK is ever demoted, it may be safely deleted. FKs are synthesised from the names of the association end-roles (elided in fig. 3), or the associated records, or both. FK fields refer to their corresponding PKs in the target model (*Order* and *Item* have FKs in fig. 4). A size threshold rule controls a further intelligent decision about when to replace a large compound FK by a simple FK, for the sake of optimising table join operations. In the related table, the compound PK is demoted, replaced by an auto PK.

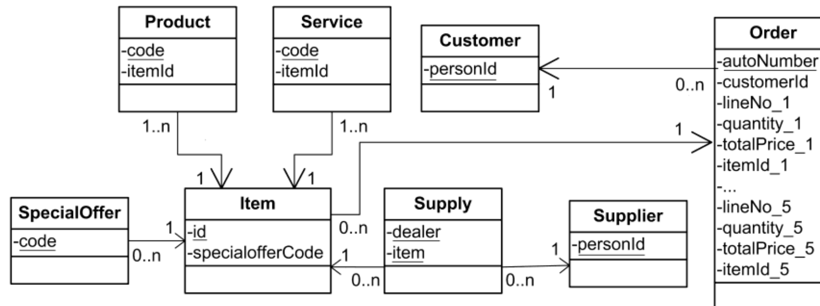

Fig. 4. The selectively normalised logical schema for the *Online Ordering System*

The database generation step translates the logical model (fig. 4) into database scripts, here in the MySQL data definition language (DDL). Part of the output file defining the *Customer* and *Order* tables is shown in (listing 1). Much of the target code follows directly from the logical model. The generic rules applied during generation seek to preserve semantic constraints, for example producing an ON DELETE CASCADE statement to maintain referential integrity (*Order*, in listing 1).

Listing 1. A fragment of the generated MySQL DDL showing flattened tables for the subclass *Customer* and the aggregate class *Order* (partly elided)

```
CREATE TABLE Customer (
  personId INT(7) NOT NULL, personForeName VARCHAR(10),
  personSurName VARCHAR(10), personAge INT DEFAULT 1,
  personAddressPostCode VARCHAR(7) UNIQUE,
  personAddressUnitNo INT(5) UNIQUE,
  personAddressStreet VARCHAR(30) UNIQUE,
  personAddressCity VARCHAR(20), id INT(7) UNIQUE,
  details VARCHAR(250), PRIMARY KEY(personId));
CREATE TABLE Order (
  autoNumber INT NOT NULL AUTO_INCREMENT, date Date,
  details VARCHAR(250), cusId INT(7) NOT NULL,
  lineNo_1 INT(10), quantity_1 INT DEFAULT 1,
  totalPrice_1 DOUBLE, itemId_1 INT(12) NOT NULL, ...
  PRIMARY KEY(autoNumber), FOREIGN KEY(custId) REFERENCES
  Customer(personId) ON DELETE CASCADE, FOREIGN KEY(itemId_1)
  REFERENCES Item(id) ON DELETE CASCADE);
```

However, certain aspects must be handled specially by MySQL-specific generators. ReMoDeL DBQ supports fields with *range constraints* on their values. Whereas these may be translated directly into CHECK constraints in some flavours of SQL (such as the Oracle DDL), the target MySQL DDL did not support these. Instead, the generator emits BEFORE INSERT trigger procedures (from MySQL version 5) as an alternative way to perform critical data validation [17]. A range check for a *Customer's* age is shown in (listing 2). If validation fails, this field takes on a default value, defined in the DBQ model.

Lisitng 2. A fragment of the generated MySQL DDL showing a trigger procedure for enforcing a range constraint on the age of each customer in the database

```
CREATE TRIGGER customerCheck BEFORE INSERT ON Customer
FOR EACH ROW
  IF (NEW.personAge < 1 OR NEW.personAge > 99) THEN
    SET NEW.personAge = DEFAULT;
  END IF;
```

5 Discussion and Conclusion

Unlike the declarative approaches of e.g. [2, 10, 13], all transformation rules in ReMoDeL are encoded as imperative methods of the agents in the transformation framework. This avoids problems of rule-ordering and non-deterministic firing [9], by performing an ordered set of transformations on XML trees. The private methods of each agent have access to a local portion of the source and target models, and, via their dominating parent agents, to more widely-scattered information. This solves some of the tangling/scattering issues [9] by providing agent protocols to access non-local information, c.f. the Law of Demeter, where this is required.

Internally, the methods of each agent dispatch on the names of XML elements in the source, mimicking the type-dispatching in the *Visitor* pattern. The imperative approach supports construction of fine-grained transformations that modify models in-place, where required, with similar benefits to the composed CRUD actions in [10]. Method naming conventions ensure that all translation steps are easily identified for maintenance purposes: this is the main unit of modularity.

While we share the minimalist goals of SiTra [19], which is also Java-based, we do not construct explicit rule-objects to mimic the pattern-driven approach, but use internal dispatching on nodes. While SiTra uses pure Java for its models, we use XML, since this facilitates more rapid modifications to the design of models, when prototyping new transformations. XML is also the *lingua franca* for input and output.

The two-phase linear composition of translation and generation steps illustrates the benefits of intermediate layers. Here, we can support generation of optimal code in different target DDLs from the intermediate model. Similarly, translation from flow diagrams to object-oriented code might need to go via intermediate structured programs [15]. In other respects, our external composition approach is homogeneous, c.f. [2], acting on uniform families of XML-based models, and does not require lifting and grounding to and from an abstract rule layer, as in UniTI [8].

6 References

1. Biehl, M.: Literature Study on Model Transformations. Technical Report, ISSN 1400-1179, Royal Institute of Technology, Stockholm, Sweden (2010)
2. Hidaka, S., Hu, Z., Kato, H., Nakano, K.: Towards a Compositional Approach to Model Transformation for Software Development. Proc. 24th ACM Symp. Applied Computing. pp. 468-475. ACM, Honolulu, Hawaii, USA, (2009)
3. Mens, T.: Model Transformation: A Survey of the State-of-the-Art. In: Gerard, S., Babau, J.P., Champeau, J. (eds.): Model Driven Engineering for Distributed Real-Time Embedded Systems. Wiley (2010)
4. Kaliappan, P.S.: State of the Art - Model Driven Architecture. Technical Report, Brandenburg Technical University, Cottbus, Germany (2007)
5. Mens, T., van Gorp, P.: A Taxonomy of Model Transformation and its Application to Graph Transformation Technology. Proc. Int. Workshop Graph and Model Transformation. pp. 1-17, Tallinn, Estonia (2005)
6. Mens, T., van Gorp, P.: A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science, 152, 125-142 (2006)
7. ATL: A Model Transformation Technology, http://eclipse.org/atl/
8. Vanhooff, B., Ayed, D., Baelen, S.V., Joosen, W., Berbers, Y.: UniTI: A unified transformation infrastructure. In Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., eds.: MoDELS-07. LNCS, vol. 4735, pp. 31–45, Springer, Heidelberg (2007)
9. Kurtev, I., van den Berg, K., Jouault, F.: Evaluation of rule-based modularization in model transformation languages illustrated with ATL. In: Proc. 21st ACM Symp. Applied Computing, pp. 1202–1209. ACM, Dijon, France (2006)
10. Goknil, A., Topaloglu, N.Y., van den Berg, K.G.: Operation Composition in Model Transformations with Complex Source Patterns. Technical Report, University of Twente, Netherlands (2008)
11. Wagelaar, D.: Composition Techniques for Rule-Based Model Transformation Languages. Theory and Practice of Model Transformations (2008), 152-167
12. Wagelaar, D., van der Straeten, R. and Derrider, D.: Module Superimposition: a Composition Technique for Rule-based Transformation Languages, Softw. Sys. Modelling, 9(3), 285-309 (2010)
13. QVT Relations, http://www.alloy.mit.edu/community/node/373
14. ReMoDeL, http://staffwww.dcs.shef.ac.uk/people/A.Simons/remodel/
15. Dojki, U.: ReMoDeL Activity Workflow Models. MSc Dissertation, Department of Computer Science, University of Sheffield (2011)
16. Subahi, A.F.: ReMoDeL Database Generator. MSc Dissertation, Department of Computer Science, University of Sheffield (2010)
17. MySQL, http://www.mysql.com/
18. Marr, D.: Vision. W. H. Freeman (1982)
19. Akehurst, D.H., Bordbar, B., Evans, M.J., Howells, W.J.G., McDonald-Maier, K.D.: SiTra: Simple Transformations in Java. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. eds: MoDELS-06. LNCS, vol. 4199, pp. 351-364, Springer, Heidelberg (2006)