A Model Transformation Approach for Translating Conceptual Database Schemas into Executable Database Systems

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 two-phase model transformation that translates a conceptual data schema into executable relational database scripts. It exemplifies a layered approach to model transformation, which applies design constraints at suitable levels of abstraction, offering routes to implementation in any DBMS. The model-to-model transformation step converts a rich conceptual schema containing records, fields, associations, aggregations and generalisations into a normalised logical schema containing only tables and keys. Rules split and merge associations, promoting some to tables, and are sensitive to the semantics of disjoint or overlapping generalisations, and strong or weak aggregations. The model-to-text transformation step converts tables into the target Data Definition Language, here MySQL, in which range constraints are encoded using triggers (c.f. Oracle, in which these are native). The translation framework adopts the direct manipulation approach, exemplifying the Visitor and Composite design patterns, and is readily customised for generating different target DDLs.

Keywords: ReMoDeL, Model-Driven Engineering, Model Transformation, Domain-Specific Modelling Language, Automatic Code Generation, MySQL.

1 Background

Model-Driven Engineering is a development methodology that concentrates on the idea of raising the level of abstraction at which software systems are constructed, to reduce the costs of development and improve programmer productivity. As a result of the rapid proliferation in general-purpose programming languages and multiplication of target platforms, the twin problems of accidental implementation complexity and platform diversity have become critical obstacles. MDE seeks to address this, by using models as the primary artefacts in the development lifecycle. Developers agree that this will help to overcome the earlier drawbacks [1] and [2].

Model-Driven Architecture (MDA) is one widely-recognised strategy, proposed by the Object Management Group (OMG) in 2001, for realising MDE [3]. This is currently being adopted by software tools that seek to use the model-driven approach.

The MDA standard specification explicitly aims to integrate many other OMG standards, Unified Modelling Language (UML) [4], Object Constraint Language (OCL) [5], Meta-Object Facility (MOF) [6], and Query/View/Transformation (QVT) [7] in order to produce a coherent MDE approach that can provide executable systems that are automatically-generated from specifications [8]. MDA, through its three kinds of models: Computational-Independent Model (CIM), Platform-Independent Model (PIM), and Platform-Specific Model (PSM), aims to provide an automatic transformation from PIM to many PSMs based on a target platform technology, and then generate executable code from the low-level platform-specific models [1].

2 Model Transformation Approaches

Model transformations are a common task in all MDE approaches and play a key role in mapping models between different levels of abstraction. The MDA initiative envisages a specific analysis-to-design transformation step from the source Computational Independent Model (CIM) to the target Platform-Independent Model (PIM); and another design-to-implementation transformation step from the source PIM to the target Platform-Specific Model (PSM) [3], from which final code will be generated. These are to be performed using the QVT approach, a pattern-driven, rule-based strategy for performing graph transformations on models [2], [7] and [13].

General issues regarding transformations have been noted, specifically whether these are complete, and whether they are reversible [9]. Transforming the PIM into the PSM may require several transformation steps, via intermediate models, and may require further human input, to add specific detail that cannot be inferred from the higher-level models or local context. In this case, transformations would only be partially automated, and systems could not be regenerated, without overwriting the added details. Eventually, reversible, or bi-directional transformations may be able to offer a round-trip approach to MDE; although the more abstract models would have to be treated merely as views of the detailed models, which retained full information.

A number of pattern-driven graph transformation approaches have emerged in the last few years supporting this approach. Many of them have developed specialised languages for metamodelling and transformation, which can be viewed as Domain Specific Languages for performing MDE. These include the Atlas Transformation Language (ATL) [10], UML-RSDS [11], the Kermeta language [12], and Acceleo for code generation [14]. All these endeavours position themselves within the MDA framework, for example by adhering to some of the OMG standards, such as to OCL for representing constraints, or to the MOF for expressing metamodels. In practice, they adhere to these standards in different degrees. While both ATL and Kermeta define metamodels that conform to EMOF, the essential MOF [6], ATL offers a pattern-driven approach that appears to satisfy the goals of QVT [7], whereas Kermeta is a uniform, imperative language, capable of arbitrary graph transformations, and is capable of applying more sophisticated sequential and transitive operations [12].

3 Reusable Model Design Languages (ReMoDeL) Approach

ReMoDeL is a research project that seeks to realise MDE using more minimalist techniques than the existing approaches, such as MDA. In the long term, the project aims to develop a complete path from business-level systems analysis models to fully functioning software system implementations, by a series of layered transformations. The starting point will be high-level models, close to the business domain. The envisaged processing involves some model-to-model *translation* steps, mapping a source model to a different target models; some model-to-self *transformation* steps, optimising a model in-place; and some model-to-code *generation* steps. How the various high-level models will be combined, possibly "folded" together in the style of aspect-oriented programming, is currently an open research question [16].

A key research goal is to identify the natural constraints that may be applied to each level of representation, such that these may be automatically applied in the transformations to the next lower levels. The lowest-level models created by the translation steps will contain fully adequate, common and generic implementation details required for automatic code generation on different platforms [16].

While ReMoDeL is inspired by, and shares some of the goals of, the OMG MDA initiative, it is not necessarily constrained by any requirement to replicate all the richness detail of OMG standards (UML [2], OCL [4], MOF [6], QVT [7]). Instead, the aim of ReMoDeL is to realise a simpler, more minimalist proof-of-concept using available technologies, Java and XML, and to develop a reference implementation. Further objectives of the wider project include systems evolution and re-engineering live systems, which themes are not relevant to the current paper [16].

The system specifications are expressed so far through two types of ReMoDeL languages, namely, the Object-Oriented Programming (OOP) model as a lowest-level common programming language with full details for implementation in a variety of different OOP languages [16], and the Database and Query (DBQ) [19].

The work reported in this paper focuses on the complete generation of executable database scripts from a high-level conceptual schema in DBQ. The overall architecture exemplifying the Visitor and Composite design patterns in which it consists of two separate translation steps (sketched in Fig. 1).

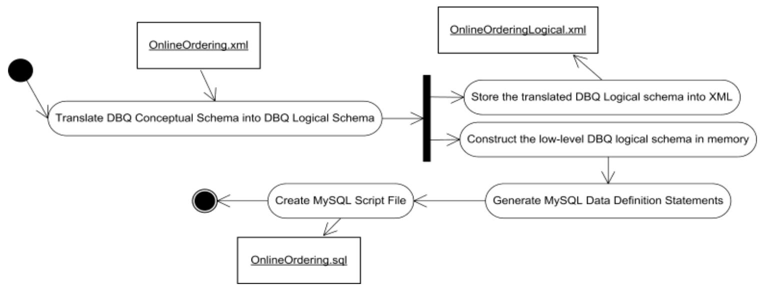
OnlineOrdering.xml
OnlineOrderingLogical.xml
Translate DBQ Conceptual Schema into DBQ Logical Schema
Store the translated DBQ Logical schema into XML
Construct the low-level DBQ logical schema in memory
Generate MySQL Data Definition Statements
Create MySQL Script File
OnlineOrdering.sql

Fig. 1. ReMoDeL Database Generator Framework

The first step (the Schema Translator) performs a model-to-model translation from high-level to low-level DBQ concepts. This step is common to all schema translations and is independent of the target language. Figure 2 illustrates the Translator hierarchy. These translators parse a high-level DBQ specification, build an in-memory model of the conceptual schema, and then construct a second memory model of the low-level schema and passed to the second step.

The internal architecture of the model-to-model transformation approach is based upon the concept distribution strategy. This means that each DBQ concept in the high-level schema is handled separately by a particular sub-translator, represented via a java class in the framework, which is responsible for applying relevant mapping rules to construct the corresponding low-level DBQ concepts. For example, the *RecordTranslator* class is responsible for translating the high-level DBQ *Record* elements into *Table* representations, including in DBQ Logical Schema.

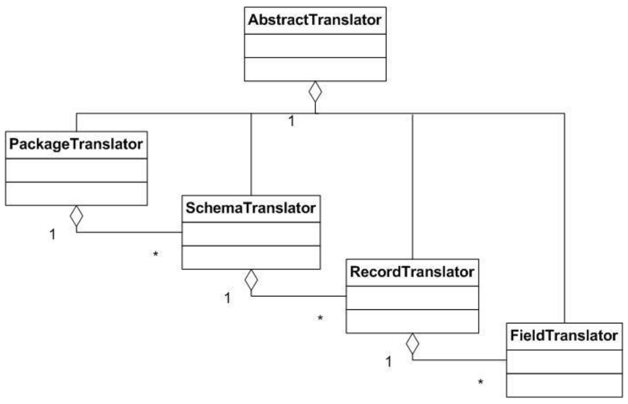

Fig. 2. The Internal Architecture of DBQ Schema Translator

The fundamental aspect of the design is constructing a DBQ Logical Schema on the fly that is accessible by all sub-translators. A new instance (a low-level concept) is built and holds required attributes that specify the internal logical schema. Elements are attached as children or siblings to form the actual table structure [17].

The second step of transformation (the Schema Generator) performs model-to-code generation in a particular database engine. The overall structure of the DBQ Schema Generator is quite similar to the other generators in the ReMoDeL Code Generation Framework, which is designed to support code generation for different database engines. Thus, generating databases in MySQL, Oracle, and even more are possible when a relevant version of Database Generator exists (here, we illustrate with MySQL). Sub-generators for each database system are implemented separately and grouped as concrete classes in java [17] (see Fig. 3).

Due to the similarity of the SQL syntax of several database systems, a layer of abstract classes is constructed on the top of all groups of sub-generators to hold and share the common behaviours of the widely-known relational database vendors, which might be duplicated within different kinds of generators (see Fig. 3).

Consequently, concepts might be generated differently, based on the most accurate SQL syntax for each database system [17].

These generators require as input the in-memory model of the low-level DBQ specification and generate, as output, script files in the chosen database input language. Figure 3 shows the specialisation of the Generator framework for MySQL.

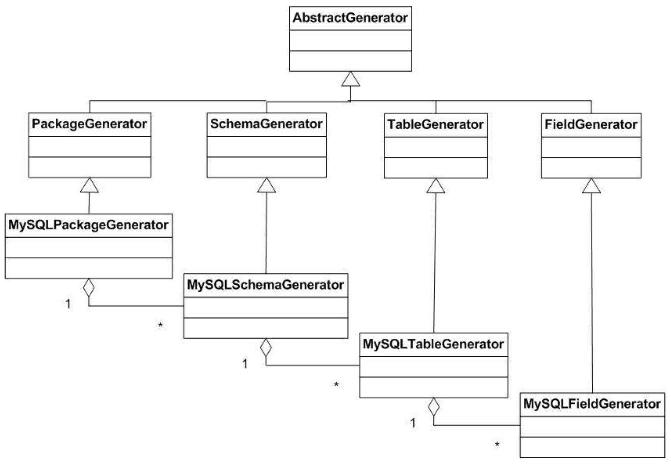

Fig. 3. The Internal Architecture of DBQ Schema Generator

In overall, he translation approach adopted in the current work differs from approaches taken by others, such as ATL [10], Kermeta, ETL and UML-RSDS [11], in that all transformation algorithms are encoded as methods of the transformation framework. The algorithms are imperative, using an ordered set of transformations on XML trees. The separation of concerns into translation and generation steps is deliberate, to support generation of optimal code in different target database input languages. Rather than focus on a pattern-driven, declarative approach, we wanted to push the limits of what kinds of sophistication could be included in a transformation, using simple, direct manipulation strategy in Java. All models are expressed in the DBQ language, an XML dialect that supports data and query design in ReMoDeL.

4 The Database and Query Language (DBQ) Representation

Unlike contrasting approaches [10], [11], and [12], which develop their own novel languages to encode metamodels and model transformations, in the current work, we seek to use simple and available technologies, XML and Java. According to this, the Database and Query model (DBQ) is provided as an XML dialect, similar to other models used within the ReMoDeL family. Rather than design DBQ as the direct casting of SQL into XML syntax (similar to the way in which JavaML encodes the syntax of Java [18]), the concrete form of the language is more like an annotated parse tree. The focus is on ensuring that DBQ captures all the necessary information to enable the desired transformations.

The DBQ language is one of a family of related XML-based languages used in the wider ReMoDeL project. It can be considered a kind of Domain Specific Language for representing the concepts from data schemas and database systems. It is possible to represent database designs at different levels of abstraction, for example at the conceptual schema level, or the logical database schema level. A DBQ contains the structure of a database design for a particular application. A model may include the high-level data schema definitions, high-level query definitions, normalised data tables, and low-level expressions representing optimised queries. In the work reported in this paper, we consider mostly the process of transforming high-level conceptual schemas into low-level logical data models [17].

Like every other model in the ReMoDeL family, DBQ conforms to a standard metamodel. DBQ contains expressions that model both conceptual schema and logical schemas. The metamodel concepts in DBQ are illustrated in Fig. 4. This shows the main generalisation relationships (only) between the concepts in both high-level and low-level DBQ and the base metamodel.

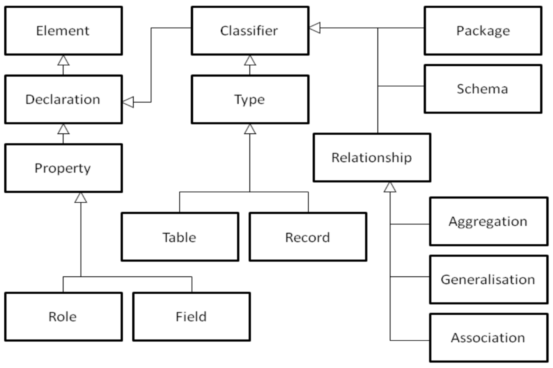

Fig. 4. Metamodel for the DBQ language specification

Each terminal node corresponds to a DBQ XML element. Nodes in the metamodel are grouped by their common features like similar attributes or child elements [19]. Similar to the purpose of database schema is the Relational Database Systems [20], the DBQ *schema* concept is regarded as a space that holds a definition of entities, relationships, and queries for a target group of users. It appears in both conceptual and logical DBQ schemas. Listing 1 and 2 demonstrate respectively a portion of the Document Type Definitions (DTD) for the high-level DBQ and the low-level one.

Listing. 1. The fragment of the Document Type Definition of the DBQ Conceptual data model

```
<!ELEMENT Schema (Record+, Generalisation*, Aggregation*,
Association*)>
<!ELEMENT Record (Field+)>
<!ELEMENT Generalisation (Role, Role+)>
<!ELEMENT Aggregation (Role, Role+)>
<!ELEMENT Association (Role, Role+, Field*)>
```

Listing 2. The fragement of the Document Type Definition of the DBQ Logical Schema

```
<!ELEMENT Schema (Table+)>
<!ELEMENT Table (Field+)>
<!ATTLIST Table name CDATA #REQUIRED>
```

4.1 The DBQ Conceptual Schema

This high-level fragment of DBQ is used for modelling various concepts of conceptual schemas such as Entities, Associations, Attributes, Generalisations and Aggregations. Here we present the "Online Ordering System", a fairly complex conceptual data model, represented as UML Class Diagram (see Fig. 5), to be used throughout this paper to demonstrate consistently the DBQ representations of the domain concepts and relationships in conceptual and logical level of schemas, as well as the transformation process within the ReMoDeL Database Generation Framework to achieve the final executable MySQL script file of the given system.

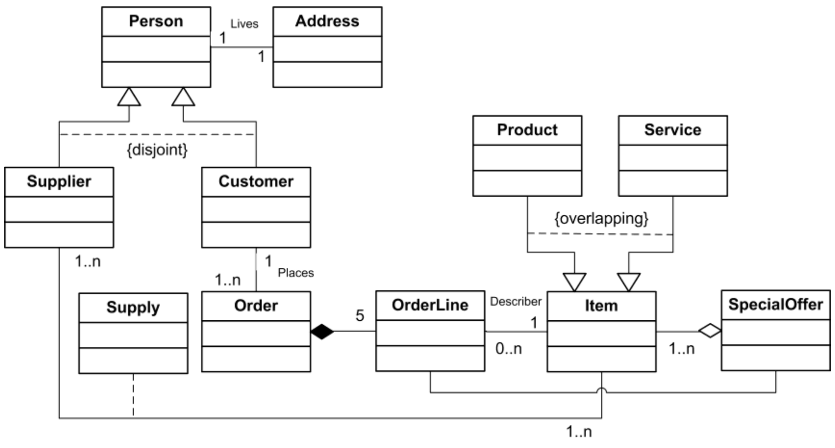

Fig. 5. The Online Ordering System Conceptual Data Model

The underlying representation of entities at the conceptual level is expressed as DBQ *Record* elements, which are consists of a number of *Fields* to specify the entity features and specifications. *Record* and *Field* have a crucial attribute called *name*, which is must appear with all elements. A number of specification attributes are used in *Fields* to specify their *types*, maximum *size*, and *Range* and *Set* constraints [19].

Primary keys are expressed via an attribute *key* that appears in DBQ *Field* element. There are three kinds of *keys* can be handled within the language: *total* for representing a single field primary key, *partial* for indicating a composite primary key, and *auto* for auto-increment primary key [19]. Listing 3 illustrates the declaration of the Person entity using the high-level DBQ concrete syntax.

Listing 3. The fragment of the high-level DBQ model demonstrating the Person entity**.**

```
<Record name="Person">
  <Field name="id" type="Natural" size="7" key="total"/>
  <Field name="foreName" type="String" size="10" />
  <Field name="surName" type="String" size="10" />
  <Field name="age" type="Natural" range="{1-99}" default="1"/>
</Record>
```

The DBQ conceptual representations of the relationships between entities of the Online Ordering System are provided through the concepts: *Association*, *Generalisation*, and *Aggregation*. Each notion involves at least two DBQ entities, which represents the actual end-role records connected by a particular relationship. The direction that is required in special relationships, *Generalisation* and *Aggregation*, can be specified using *head* attribute to indication that the arrow is toward the record in the *head* value (Listing. 3) [19]. The language is able to distinguish between two generalisation types: disjoint between *Person* and its subtypes *Customer* and *Supplier*, and overlapping between *Item* and its subtypes *Product* and *Service*. In addition to this, aggregations and composite aggregations are also expressed using high-level DBQ language. An Aggregation between *SpecialOffer* and its part: *Item*. A composition between *Order* and its part: *OrderLine*.

Listing 4. A part of the high-level DBQ model (overlapping generalisaiton and Aggregation)

```
<Generalisation head="Item" disjoint="false">
  <Role name="item" type="Item" multiple="mandatory" />
  <Role name="product" type="Product" multiple="optional" />
  <Role name="service" type="Service" multiple="optional" />
</Generalisation>
<Aggregation head="SpecialOffer" composite="false">
  <Role name="offer" type="SpecialOffer" multiple="mandatory"/>
  <Role name="item" type="Item" multiple="onemany"/>
</Aggregation>
```

The *Association* types, such as 1-to-1, M-to-1, M-to-N are realised by recognising the multiplicity of their end-roles via the value of the attribute *multiple*, which indicates the number of occurrence in the association, namely, *mandatory*, *optional*, *zeromany*, *onemany*. These values are also used in specifying the number of occurrence of the subtype records in *Generalisation* and *Aggregation*, whereas the attribute *quantity* is used to determine the actual number of parts in *Composition* [19]. In regard to the proposed Online Ordering System Data Model, M-to-1 associations between *Customer-Order*, and *Item-OrderLine, and* M-to-N association between *Supplier* and *Item* and a 1-to-1 association between *Person* and *Address* are distinguished and expressed using high-level DBQ language (Listing. 5).

Listing 5. A part of the high-level DBQ of the 1-to-1 and M-to-1 Associations

```
<Association name="Lives">
  <Role name="person" type="Person" multiple="mandatory" />
  <Role name="address" type="Address" multiple="mandatory" />
</Association>
```

```
<Association name="Places">
  <Role name="seller" type="Customer" multiple="mandatory"/>
  <Role name="order" type="Order" multiple="onemany"/>
</Association>
```

4.2 The DBQ Logical Schema

The translated DBQ logical schema differs from the conceptual one, in that it consists only of a common logical representation of Tables and Fields (viz. all relationships from the conceptual model have been converted into foreign keys). This fragment is used to describe a platform independent model at the lowest-level of abstraction that is closest to the physical database representation. The assumption is that a comprehensive level of abstraction that represents all generic specifications required for any database generation is achieved and expressed via the DBQ logical model. This low-level model is used as a source model in the generation approach. A snapshot of this common representation of this model is illustrated in Listing 6.

According to the presented Online Ordering System conceptual schema (Fig. 5), the normalised schema can be demonstrated using the following UML Class Diagram (Fig. 6) including the actual structure of tables, and their relationships. This low-level DBQ model is constructed as a result of applying the precise transformation rules by the framework sub-translators, such as *RecordTranslator*, and *FieldTranslator*.

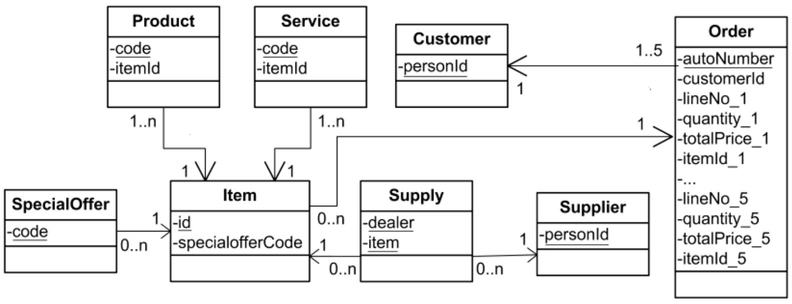

Fig. 6. The Normalised Online Ordering System Logical Data Model

Lisitng 6. A part of the low-level DBQ model that demonstrates the representation of translated Online Ordering System Logical Schema after applying merging and referencing rules**.**

```
<Table name="Customer">
  <Field name="personId" type="Natural" size="7" key="total" />
  <Field name="personForeName" type="String" size="10" />
  <Field name="personSurName" type="String" size="10" />
  <Field name="personAge" type="Natural" range="{1-120}"
default="1" />
  <Field name="personAddressPostCode" type="String" size="7"
   key="partial" unique="true" />
  <Field name="personAddressUnitNo" type="Natural" size="5"
```

```
   key="partial" unique="true" />
  <Field name="personAddressStreet" type="String" size="30"
   key="partial" unique="true" />
  <Field name="personAddressCity" type="String" size="20" />
  <Field name="id" type="Natural" size="7" key="total"
   unique="true" />
  <Field name="details" type="String" size="250" />
</Table>
<Table name="Order">
  <Field name="id" type="Natural" size="7" key="total" />
  <Field name="date" type="Date" />
  <Field name="details" type="String" size="250" />
  <Field name="cusId" type="Natural" size="7" refer="Customer"/>
  <Field name="lineNo_1" type="Natural" size="10" />
  <Field name="quantity_1" type="Natural" default="1" />
  <Field name="totalPrice_1" type="Decimal" />
  <Field name="itemId_1" type="Natural" size="12" refer="Item"/>
  <Field name="lineNo_2" type="Natural" size="10" />
                    ...
  <Field name="lineNo_5" type="Natural" size="10" />
  <Field name="quantity_5" type="Natural" default="1" />
  <Field name="totalPrice_5" type="Decimal" />
  <Field name="itemId_5" type="Natural" size="12" refer="Item"/>
</Table>
```

Concepts in the Logical Schema are distributed to be handled by the relevant sub-generator for a specific database. In our case, the MySQL Database Generator, via its precise rules of transformations, promises that the generated database systems satisfy the requirements of the 3NF. It extracts critical information from the model and takes proper transformation decisions. The mapping technique between the concepts in DBQ Logical Schema and those in the targeted database is regarded a main process in order to construct valid SQL syntax for a particular database [17].

5 Model-to-Model Translation

As a general guideline, records in the DBQ conceptual schema are translated into actual *Table* elements in the logical schema. In-place model modifications might be done before the final translation into the target model, for instance, combining *Records* before being translated into a single table when applying a proper merging rule for handling 1-to-1 associations first, and then converting *Composition*, and *Disjoint Generalisation* relationships using a suitable naming convention [19]. This can be noticed from the combined structure of *Customer* table in Listing 6 above.

Fields and primary keys at the high-level DBQ model are normally translated into the actual Filed elements in the low-level model, unless some primary keys are reassigned as UNIQUE in the merged tables to maintain its semantics, as in *Person-Address* relationship. Records without an explicit primary key are supplied by an automatically-generated Auto-increment primary key, as in Order table (Fig. 6). Furthermore, *Roles* are translated into foreign keys, which are used to provide a physical representation of the relationships, such as Overlapping Generalisation, Aggregation, and M-to-1 Associations, between the actual tables within the generated logical Schema [17] (see foreign keys added to *Order* and *Item* table in Fig 6).

While the ability of recognizing compound primary keys in referenced tables raises the demand of generating compound foreign keys in the other side of association; the DBQ Schema Translator performs intelligent splitting rules on Roles to generate a number of reference fields that reflects a composite primary key. This demonstrated through the translation of a compound primary key of the *Address* record in the 1-to-1 Association with *Person* (see the new DBQ declaration of *Customer* table Listing 6).

As it seen, in order to generate the most ideal database design that satisfies the requirements of the 3NF, sensible minimisation techniques of data dependency are applied by the DBQ Schema Translator in transforming relationships defined between entities. As the high-level DBQ schema contains adequate specification attributes for all kinds of associations, it is able to distinct and provides an explicit expression for both *Generalisation* types: *Disjoint* and *overlapping*, as well as *Aggregation* and *Composition*. This plays a key role in the automated transformation decision to achieve a reasonable balance between the query complexity, system performance and disk space [20]. The de-normalisation approach that is used by the DBQ Translators to flatten the composite aggregations (Order table in Fig. 6) can be regarded as a remarkable modelling example to illustrate one of the intelligent decision points within our approach for managing the database performance and disk space [20].

6 Model-to-Code Translation (Code Generation)

In the DBQ logical schema, *Table* and *Field* must be converted into equivalent database tables and fields for a targeted system (MySQL) using the suitable MySQL sub-generator. There is no demand to take large number of smart actions at this level as the required designing and normalising processes are accomplished and tables reconstructed at the model-to-model translation stage. The generators applies a sequence of element traversing to Table and Fields elements in the low-level DBQ XML parse tree and streaming the output into a target script file. The following listing (Listing. 7) illustrates the generated DDL of MySQL Customer and Product tables. The referential integrity concept is applied to maintain the overall consistency in which the generator produces an ON DELETE CASCADE statement when required [17], as in the declaration of the *Product* table (Listing 7).

Lisitng 7. The fragemnt of the generated MySQL DDL representing two generalisation types and composite aggregation

```
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, ... lineNo_5 INT(10), quantity_5 INT
  DEFAULT 1, totalPrice_5 DOUBLE, itemId_5 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);
```

According to MySQL limitation in which it does not support Check Constraints as Oracle and other database systems do, using the new Stored Procedure feature in MySQL version 5, as an alternative way to perform data validation, is considered critical decision [24]. ReMoDeL MySQL Database Generator introduces a rule for generate Before Insert Triggers for tables that have fields with a specified range constraint. The value of the DBQ default attribute is used as a save alternative value of that field in the case of invalid user input [17] (See Listing 8).

Lisitng 8. A part of the generated MySQL DDL representing a trigger for enforcing constraint

```
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;
```

7 Comparison with other Approaches and Tools

We contrast the algorithmic approach taken in this paper with the ATL pattern-driven transformation rules for converting a UML class diagram to a relational data model in [10]. In the published ATL example, database tables were constructed primarily from a subset of classes that were marked as persistent (using a UML stereotype). The attributes of other volatile classes, if related by association, were aggregated in the persistent classes according to rules that "flattened" the data model. This included a treatment of generalisation, which created "fat superclass" tables by transferring the attributes of subclasses up to the root persistent class. This translation will later reduce the cost of joining database tables, but at the cost of wasting space (subsets of the fields will be irrelevant for some instances and will take on null values). Overall, the emphasis above was in describing the "flattening rules".

Therefore, it can be argued that the transformation performed by the DBQ Schema Translator makes better use of the different kinds of semantic relationships present in the DBQ conceptual model. In particular, it can distinguish between different kinds of association (with different kinds of multiplicity), and different kinds of aggregation (including the stronger composite aggregation).

In regard to the UML-RSDS approach for translating UML to RDBMS [15], a clever way of translating OCL-like specifications and constraints into imperative Java routines is introduced through sequential transformation rules that are converted into Java methods with pre and post conditions. Unlike the Transformation algorithm of UML-RSDS tool, the DBQ Schema Translator can treat two kinds of generalisation (overlapping and disjoint) differently [17]. The transformation implements all the rules of entity-relationship modelling, such that records with 1-to-1 associations are merged and linkers are introduced for M-to-N associations [20]. The transformation is therefore more idiomatic and may claim to be more standard than the special-purpose transformation described in [10].

The Schema Translator's treatment of generalisation yields two different translations. Where the generalisation set is marked as disjoint, the translator creates separate tables for each specific subclass, each incorporating the inherited fields, since no logical object will ever be an instance of more than one class. This provides a low-cost access plan for queries, as does the ATL "fat superclass" approach, since both translations remove the need to join tables to access fragmented objects; however our translation does not waste any space [21]. Where the generalisation set is marked as overlapping, all records in the generalisation set must be converted into tables, since logical objects might be split across all three tables. This has the disadvantage that more joins would be required to reconstruct logical objects, but the advantage of a finer granularity for updating tables. The "fat superclass" approach of ATL would provide a more efficient access plan, but still waste space. Eventually, the cost depends on the numbers of associated records [21].

The Schema Translator's treatment of aggregation likewise yields two different translations. Where an aggregation is marked as composite, this is a hint to the translator that the contents of the associated parts may be incorporated directly inside the whole (de-normalisation). The result of this is analogous to the result of applying ATL's "flattening" rules, where the head of the composite aggregation is analogous to ATL's persistent class, and the incorporated fields are analogous to the attributes of ATL's associated volatile classes. Where an aggregation is not composite, tables are created for each record and connected to the whole by foreign key.

Therefore, it can be said that ReMoDeL Database Generation framework, with an adequate DBQ physical database specifications, offers the most accurate standard transformation solution for database generation applications. In the two levels of transformation, the approach is able to generate an executable database schema for a specific database system. In contrast, constructing a transformation program (model) with both source and target models and metamodels are essential to perform model-to-model transformations within ATL approach. The output of ATL approach is not an executable model or code. Further approach is needed to perform the model-to-code transformation [10]. Transformation rules, indeed, can be distinct in each ATL application for the same scope of transformation, which depends on the user experience in constructing the mapping definition (model).

The code generation approach within ReMoDeL follows an alternative code generation technique than the commonly-used template-based approach adopted in Acceleo [14]. Acceleo is regarded as the implementation of the OMG MOF standards, which is known as the prototype-based approach. In that approach, a valid template of the target code that determines its content must be defined and developed before establishing the process of code generation [14]. In ReMoDeL, there is no demand to users to concentrate on building template for various platform technologies.

8 Future Work

The long-term ReMoDeL project eventually aims to integrate a number of different frameworks that perform various types of model transformations. In parallel with the current work reporting on the DBQ model, other work has been carried out on developing a common OOP [16]. Eventually, these two models may be linked, to provide a more powerful model transformation to generate software systems coupled in different ways to different back-end databases.

Despite the fact that our proposed Model-to-Model transformation algorithm can work properly with fairly complex data models and has the ability to distinguish between the two kinds of generalisation, there is a demand to improve it to adopt more complicated concepts that might occur in real-world, such as multiple inheritances, which is considered in the transformation approach of UML-RSDS [11].

9 Conclusion

In this paper, we reported on a simple MDE approach, implemented in a two-stage model transformation, for generating relational database implementations from high-level conceptual data schemas. We utilised the ReMoDeL approach and its DBQ language for specifying database entities and relationships in two levels of abstractions: conceptual, and logical schema, as well as applying a series of transformations. The model-to-model transformation step translates DBQ Conceptual Schemas into normalised Logical one taking into account the semantics of disjoint/overlapping generalisations, and aggregations/compositions, whereas the model-to-Code transformation stage generates the MySQL database system from the low-level DBQ schema. This architecture is formulated to sustain the automatic generation of code in various target database systems. A set of rules for database design has been selected to perform a sensible balance between the performance and required disk space, which is used mainly in model-to-model transformation stage.

10 References

1. Kelly, S., Tavenen, J.: Domain-Specific Modelling: Enable full code generation. John Wiley & Sons, Inc (2008)
2. Jezequel, J.: Model-Driven Engineering: Basic Principles and Open Problems, (2003). http://www.irisa.fr/triskell/publis/2003/
3. Object Management Group: MDA Guide, Version 1.0.1, Miller, J., Mukerji, J. (eds.), 12 June (2003). http://www.omg.org/cgi-bin/doc?omg/03-06-01

4. Object Management Group: Unified Modeling Language (OMG UML) Superstructure, Version 2.3, 5 May (2010). http://www.omg.org/spec/UML/2.3/Superstructure/PDF/
5. Object Management Group: Object Constraint Language, Version 2.0, 1 May (2006). http://www.omg.org/spec/OCL/PDF/
6. Object Management Group: Meta Object Facility (MOF) Core Specification, Version 2.0, 1 January (2001). http://www.omg.org/spec/MOF/2.0/PDF/
7. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.0, 3 April (2008). http://www.omg.org/spec/QVT/1.0/PDF/
8. Poole, J.: Model-Driven Architecture: Vision, Standards & Emerging Technologies. In: Workshop on Metamodeling and Adaptive Object Models, ECOOP (2001)
9. Kent, S.: Model Driven Engineering. In: Integrated Formal Methods. LNCS, vol. 2335, pp 286-298. Springer, Heidelberg (2002)
10. Jouault, F., Kurtev I.: Transforming Models with ATL. In: Satellite Events at the MoDELS 2005. LNCS, vol. 3844, pp 128-138. Springer, Heidelberg (2006)
11. Lano, K: Using B to Verify UML Transformations. In: Proceedings of the 3rd Workshop on Model design and Validation (MODEVA 2006), B. Baudry, D. Hearnden, N. Rapin, J. G. Süß (Eds.), pp. 46-61, Genova, Italy, October (2006)
12. Jezequel, J., Barais, O., Fleurey, F.: Model Driven Language Engineering with Kermeta. In: 3rd Summer School on Generative and Transformational Techniques in Software Engineering 2010. LNCS 6491, Springer (Ed.) (2010)
13. Gerber, A., Lawley, M., Raymond, K., Steel, j., Wood, a.: Transformation: The Missing Link of MDA. In: Graph Transformation First International Conference ICGT 2002. LNCS, vol. 2505, pp 90-105. Springer, Heidelberg (2002)
14. Acceleo, http://www.acceleo.org/pages/home/en
15. Lano, K., Kolahdouz-Rahimi, S.: Specification and Verification of Model Transformations using UML-RSDS (2010).
16. ReMoDeL: Reusable Model Design Languages, http://www.dcs.shef.ac.uk/~ajhs/remodel/
17. Subahi, A.F.: ReMoDeL Database Generator. MSc Dissertation, University of Sheffield (2010)
18. Badros, G.J.: JavaML: a markup language for Java source code, In: Proceedings of the 9th international World Wide Web conference on computer networks, pp 159-177. Computer Networks, Amsterdam (2000)
19. Simons, A.J.H., Subahi A.F.: ReMoDeL Database and Query Language Model, Version 1.0 March 2011. Technical Report, Department of Computer Science, University of Sheffield (2011). http://www.dcs.shef.ac.uk/~ajhs/remodel/DBQ/
20. Connolly, T., Beggs, C.: Database Systems - A Practical Approach to Design, Implementation, and Management (4th Edition). Addison-wesely (2005)
21. Eder, J., Kanzian S.: Logical Design of Generalisations in Object-Relational Database. In: 8th East European Conference – Advance in Databases and Information Systems (ADBIS 2004), Budapest, Hungary (2004)