ReMoDeL: **Re**usable **Mo**del **De**sign **L**anguages

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

Ahmad F Subahi and Anthony J H Simons
(A.Subahi , A.Simons}@dcs.shef.ac.uk

Outline

- Introduction
- Brief Description of ReMoDeL
- Case Study: Database Generator for MySQL
- Transformation Composition in ReMoDeL
- Conclusion and Q/A

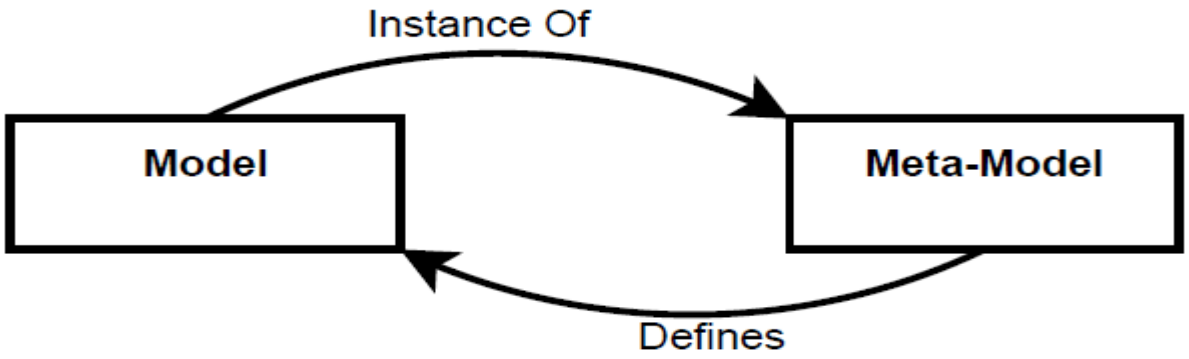
http://staffwww.dcs.shef.ac.uk/people/A.Simons/remodel/

Outline

- *Introduction*
  - *Concept Definitions*
  - *Styles of composition*
  - *Units of modularity*
- Brief Description of ReMoDeL
- Case Study: Database Generator for MySQL
- Transformation Composition in ReMoDeL
- Conclusion and Q/A

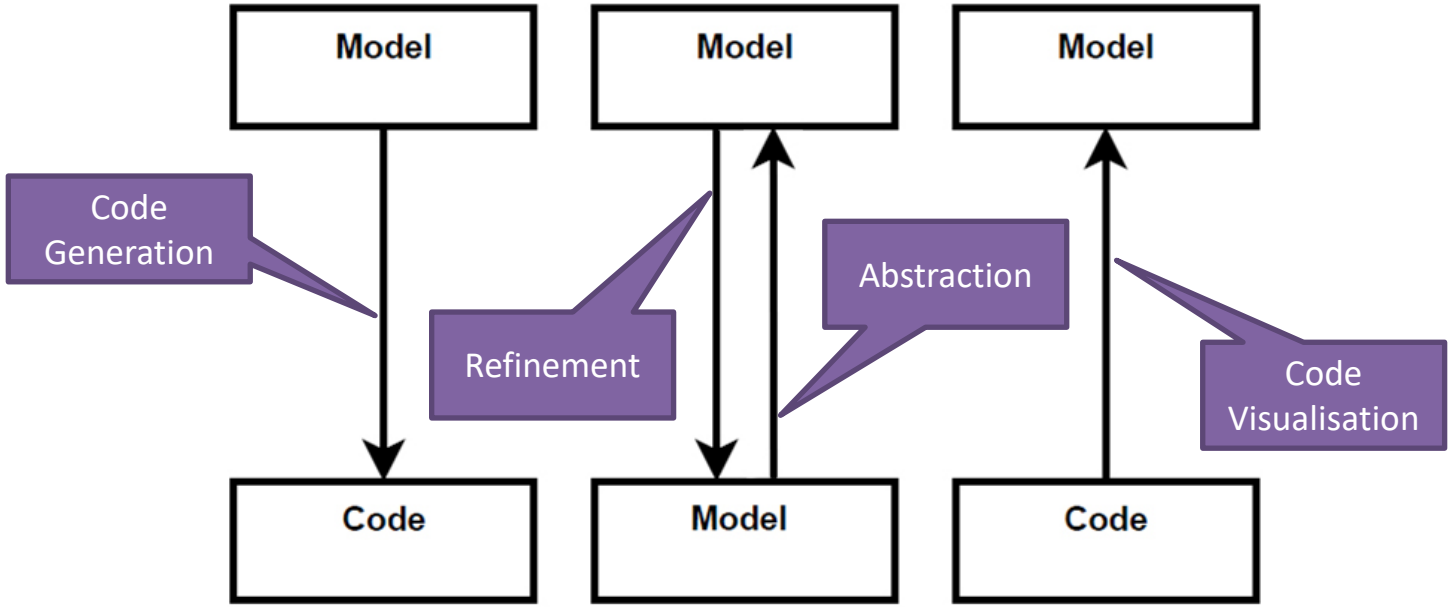
Concept Definitions I

- Model-Driven Engineering (MDE):
  - A software development methodology that uses models as the first class entities in the development process (lifecycle).
  - A model conforms to another model (metamodel).



Concept Definitions II

- Model Transformations (MT):
  - A program that takes models (inputs) and produces other models (output).



Some Existing Approaches

- Styles of composition
  - Heterogeneous composition, with glue code, or lifting and grounding, e.g. UniTI (Vanhoof et al, 2007)
  - Homogeneous composition, e.g. graph transformations in extended UnQL/JSON (Hidaka et al, 2009)
    - ReMoDeL has homogeneous XML graphs, exogenous models
- Units of modularity
  - Standard scale, e.g. whole rules in (Kurtev et al, 2006)
  - Large scale – e.g. module superimposition in (Wagelaar et al, 2008, 2010)
  - Fine scale – e.g. composed CRUD operations (Goknil et al, 2008)
    - ReMoDeL has fine-scale surgery of source, target graphs

Outline

- Introduction
- *Brief Description of ReMoDeL*
  - General aims and goals
  - Example models: DBQ language
  - Example framework: for database generation
- Case Study: Database Generator for MySQL
- Transformation Composition in ReMoDeL
- Conclusion and Q/A

ReMoDeL Overview

- A multi-view, multi-level MDE approach
  - Multiple process, time, data, code models
  - Intermediate representations, to support folding
- Shares some goals of OMG's MDA
  - Forwards transformation, traceability
  - But liberal attitude to OMG standards
- Aims to develop a simpler proof-of-concept
  - Use available technologies (Java rules, XML models)
  - Use direct manipulation, imperative framework
  - Develop a reference implementation (practical!)

Database and Query Language

- DBQ: Database and Query Language
  - One of several XML dialects used in ReMoDeL
  - Defines conceptual and logical data schemas
  - Supports logical and functional queries
- Supported concepts
  - High level, e.g. record, field, association, generalisation, aggregation
  - Low level, e.g. table, field, primary/foreign key
- Model transformation goals
  - Data normalisation (endogeneous/exogeneous)
  - Database DDL script generation (exogeneous)

DBQ Conceptual Model - I

Record with fields (High-level)

```
<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="{0-120}" default="0"/>
</Record>
```

Conceptual Entity

Unique Name

Field Type

Candidate Key

Conceptual Attribute

Range Constraints

Default Value

DBQ Conceptual Model - II

Association with end-roles (High-level)

Association

Multiplicity

```
<Association name="Lives">
  <Role name="owner" type="Person" multiple="mandatory" />
  <Role name="home" type="Address" multiple="mandatory" />
</Association>
```

End Role

DBQ Logical Model

Table with PK, FK fields (Low-level)

Logical Table

Generated Foreign Key

```
<Table name="Item">
  <Field name="specialOfferCode" type="Natural" size="12"
   refer="SpecialOffer" />
  <Field name="id" type="Natural" size="12" key="total" />
  <Field name="unitPrice" type="Decimal" />
</Table>
```

Logical Field

Primary Key

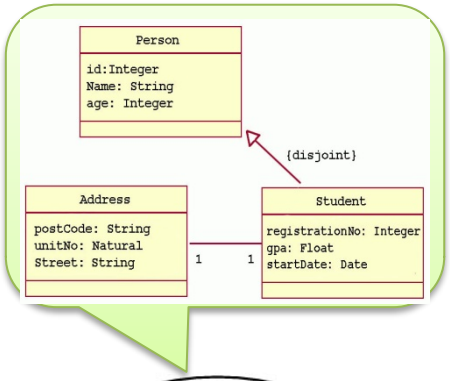
ReMoDeL Database Generator

- "Database Generator" Project
  - First implementation of the ReMoDeL idea
  - Possible component in the larger framework
- What the "Database Generator" does:
  - Two-phase composition of translation, generation activity
  - Selectively normalises a conceptual data model
  - Generates executable DDL scripts, for different RDBMS
- Model transformation approach
  - Java agents responsible for different levels of model detail
  - Delegate to sub-agents; request context from super-agents
  - Transformation rules are methods, suitably ordered and named
  - Visitor pattern traverses, modifies nodes of XML graphs

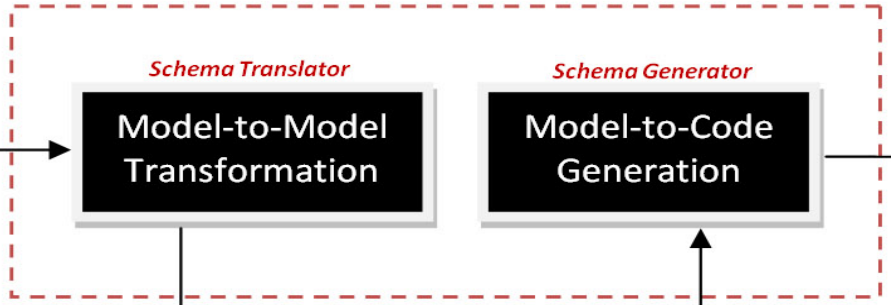
The Architecture I

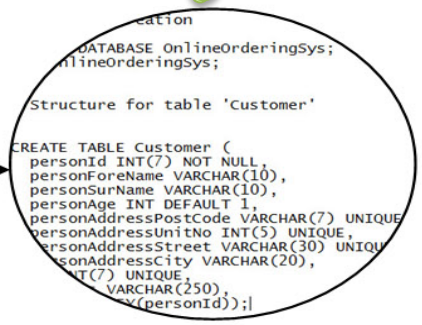
ReMoDeL Database Generator

Composition of two transformations

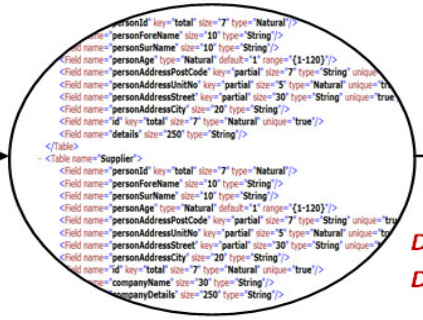


DBQ Conceptual Data Model





Executable MySQL Script

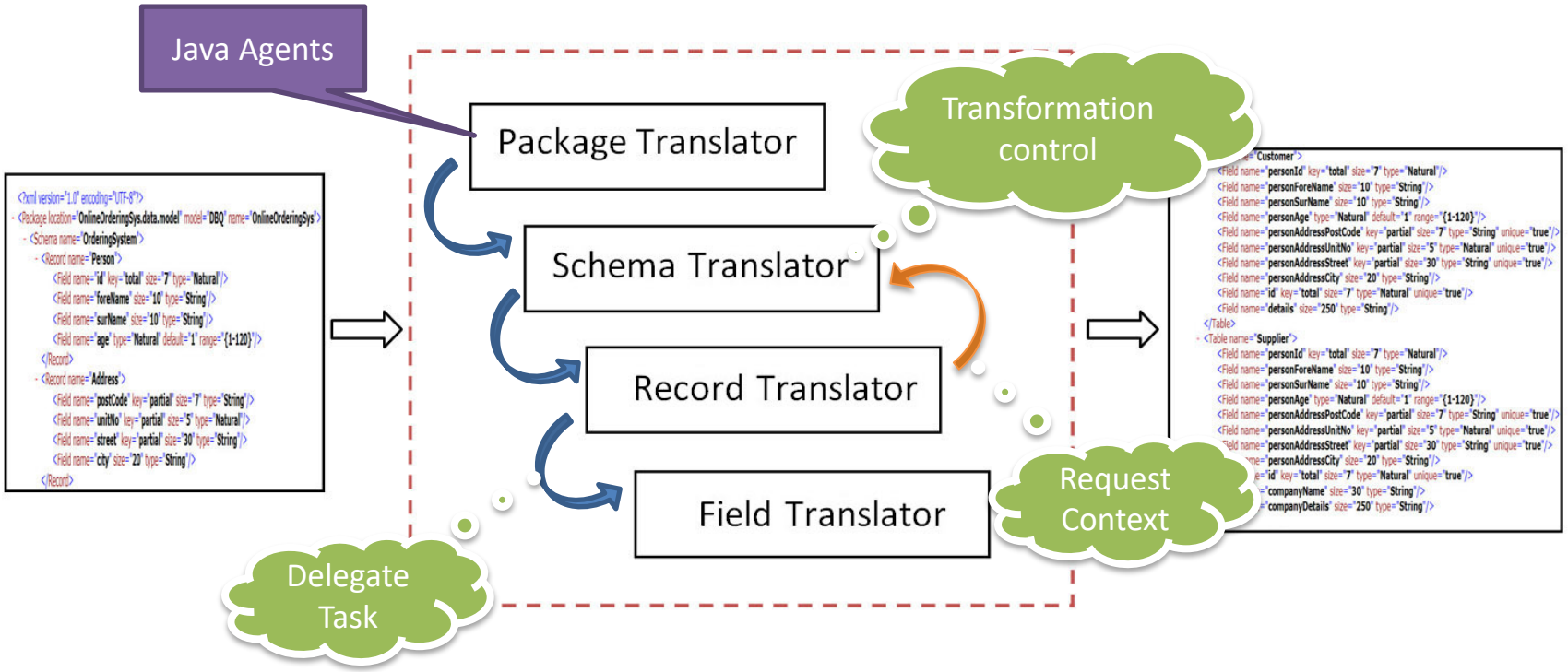


DBQ Logical Data Model

The Architecture II

Model-to-Model Transformation Phase

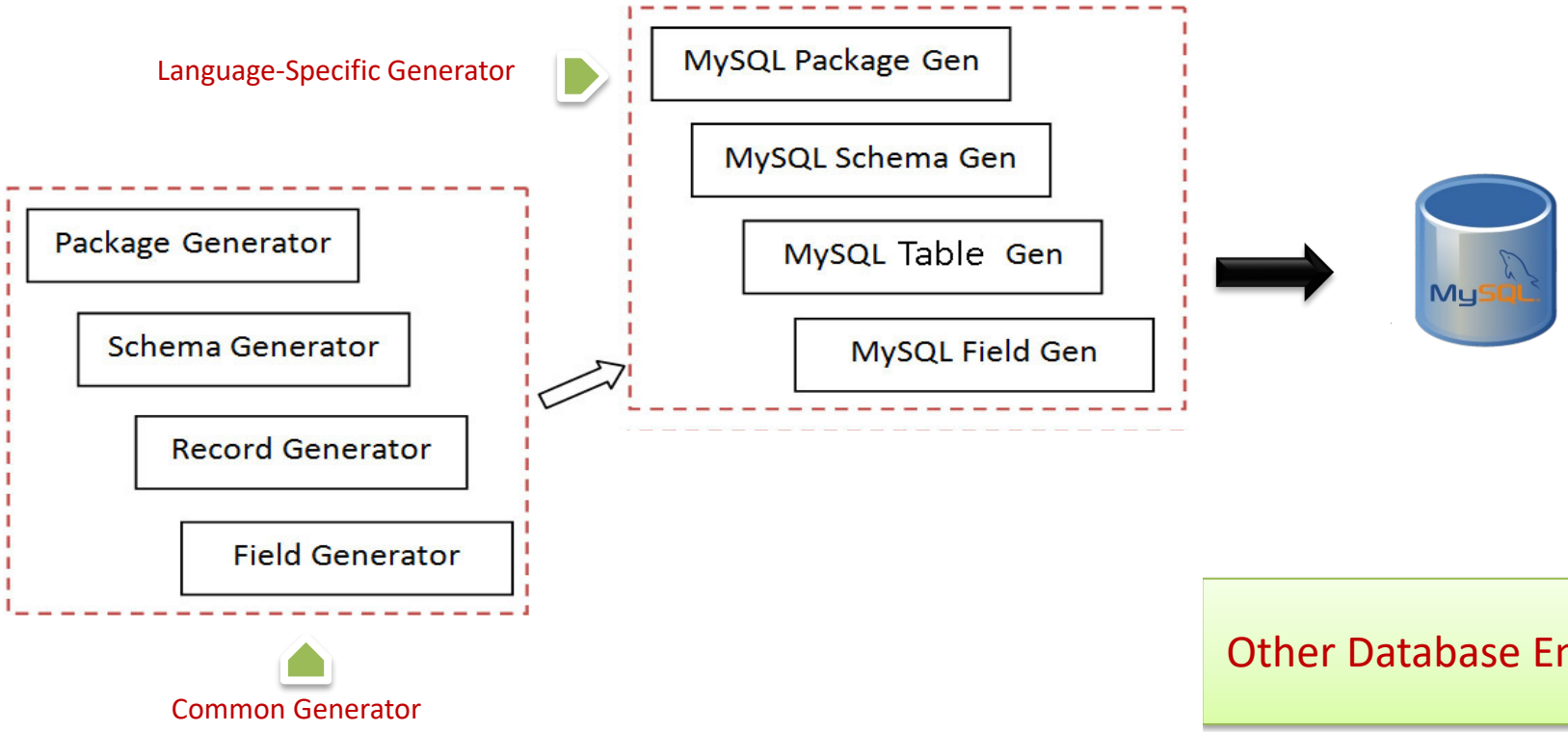
Conceptual data model into Logical data model



The Architecture III

Model-to-Code Transformation Phase

Logical Data Model to MySQL DLL Script



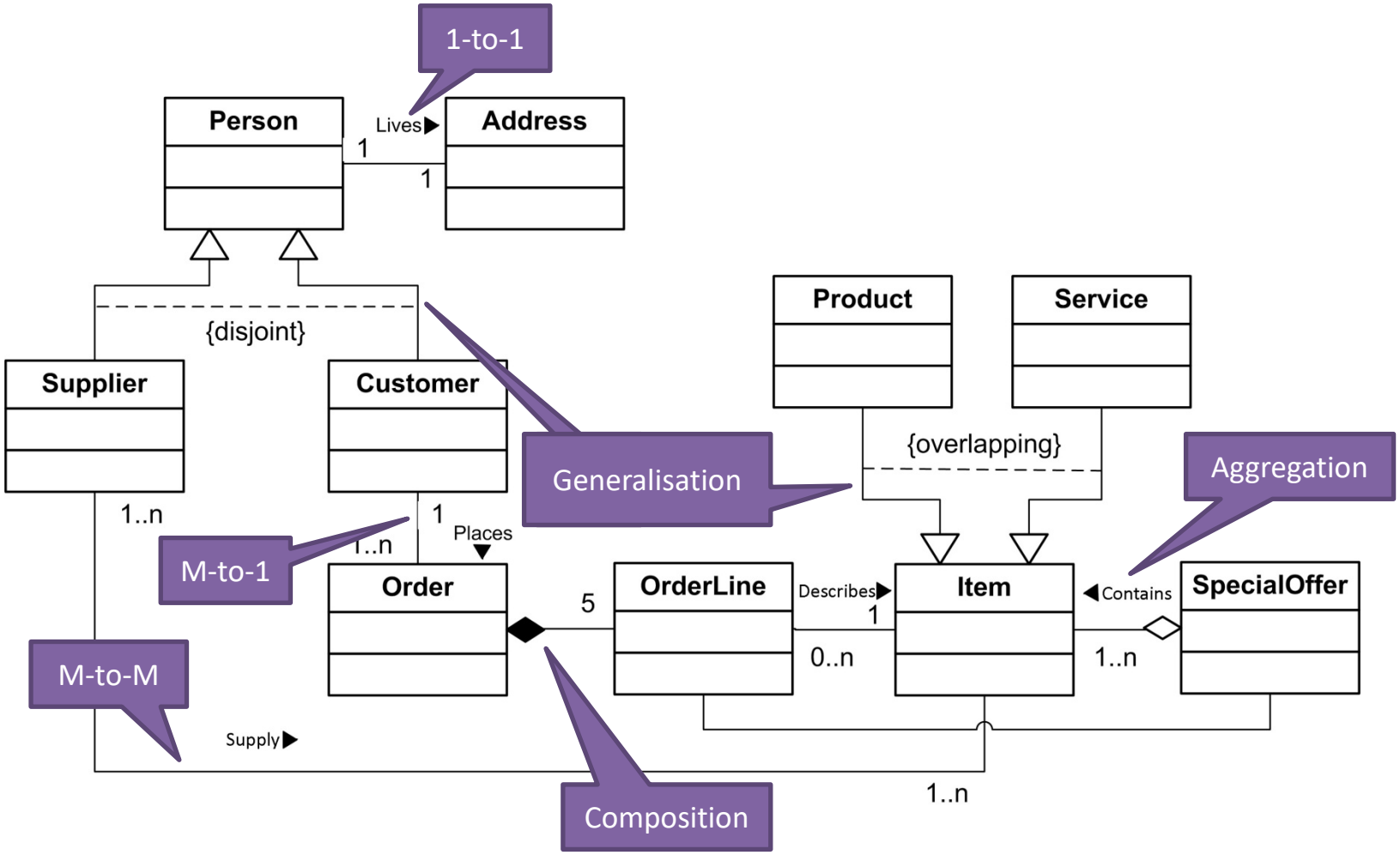
Rationale

- Two stage transformation is appropriate, because
  - Data normalisation strategies independent of DDL script generation
  - Different SQL constructs supported in target DDLs
- Model translation step
  - Full normalisation for traditional RDBMS
  - Selective denormalisation (esp. generalisation, strong aggregation) for improved performance
- Code generation step
  - Oracle supports field range constraints
  - MySQL only supports if-added triggers

Outline

- Introduction
- Brief Description of ReMoDeL
- *Case Study: Database Generator for MySQL*
  - *Kinds of mapping rules supported*
  - *Examples of model-to-model transformation*
  - *Examples of model-to-code generation*
- Transformation Composition in ReMoDeL
- Conclusion and Q/A

Online Ordering System



General Mapping Rules

Conceptual DBQ Concepts	Logical DBQ Concepts
Record	Table
Field	Field
One-to-One Association	Merged Table (*Merging rule*)
Many-to-Many Association	Linker Table (*Splitting rule*)
Many-to-One Association	Foreign key
Generalisation (*disjoint*)	Flattened Table (*Flat subclass rule)*
Generalisation (*overlapping*)	Tables (*fully normalised; fat superclass?*)
Aggregation (*Weak Aggregation*)	Foreign key
Composition (*Strong Aggregation*)	De-normalised Table (*Aggregating rule*)

Transformation Order

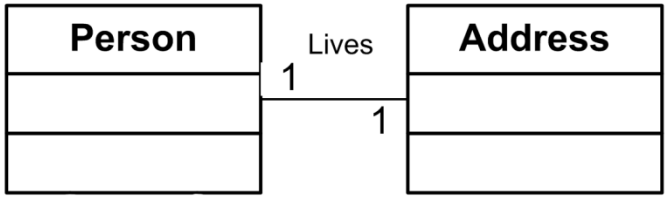
```
public class SchemaTranslator extends AbstractTranslator {

 private Element target;

 public SchemaTranslator(Element source, PackageTranslator parent) {
                // Install source model
       super(source, parent);
       target = new Element(source.getName());
 }

 public Element translate() throws TreeException {
       target.setAttribute("normal","true");
                // The order of translation
       translateOneToOneAssoc();
       translateOneToManyAssoc();     // private methods of
       translateGeneralAggreg();      // this SchemaTranslator
       translateManyToManyAssoc();
       return target;
 }
```

Step 1: Merging Records




```
<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="{0-120}"
  default="0"/>
</Record>
```

Schema Translator

Record Translator

Re-naming and Merging

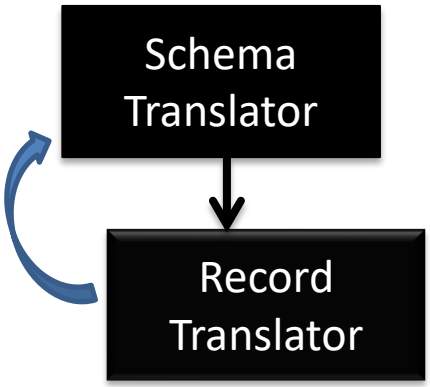
```
<Record name="Address">
 <Field name="postCode"
   type="String" size="7"
   key="partial"/>
 <Field name="unitNo"
  type="Natural" size="5"
  key="partial"/>
 <Field name="street"
  type="String" size="30"
  key="partial"/>
 <Field name="city"
  type="String" size="20"/>
</Record>
```

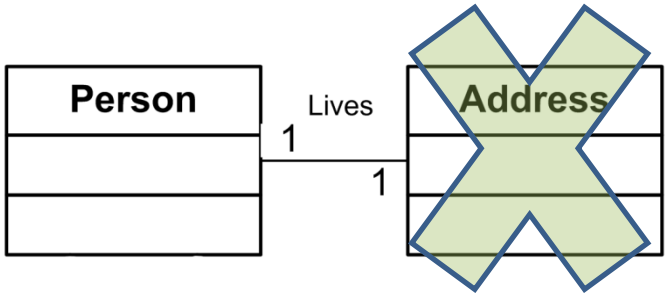
Tree Surgery Example

```
public class RecordTranslator extends AbstractTranslator {

 // source model is an Association, target is a Table
 private void translateOneToOneAssoc() throws TreeException {
        Element major = getRoleType(getMajorRole(model));
        Element minor = getRoleType(getMinorRole(model));
        target = new Table(major.getName());
        for (Field field : major.getChildren("Field"))
                target.addContent(field.clone());
        for (Field field : minor.getChildren("Field")) {
                Field renamed = field.clone();
                renamed.setValue("name", mergeName(
                        minor.getValue("name"),
                        renamed.getValue("name")));
                target.addContent(renamed);
        }
        ...
        getParent().addTable(target);  // API of SchemaTranslator
 }  ...
```

Step 2: Field Renaming

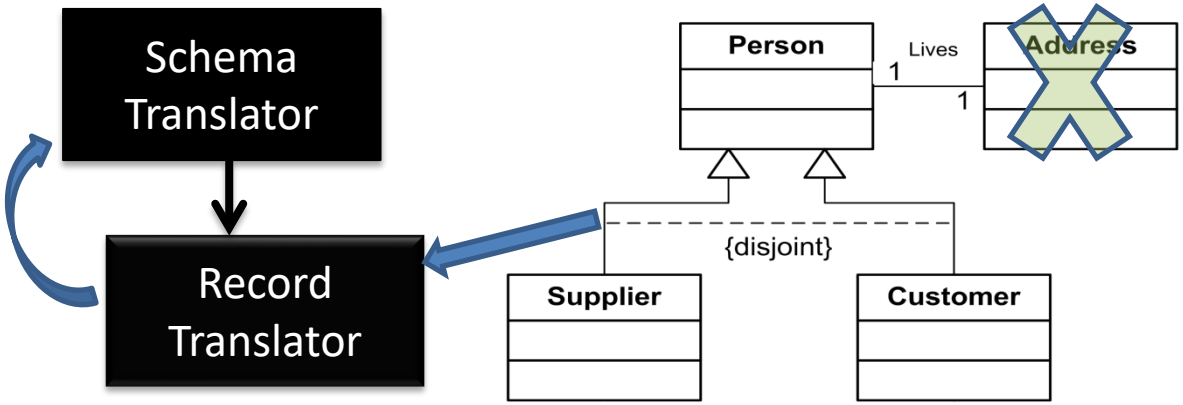




Person table after merging fields from Address

```
<Table 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="{0-120}" default="0"/>
   <Field name="addressPostCode" type="String" size="7"unique="true"/>
   <Field name="addressUnitNo" type="Natural" size="5" unique="true"/>
   <Field name="addressStreet" type="String" size="30" unique="true"/>
   <Field name="addressCity" type="String" size="20"/>
</Table>
```


Step 3: Flatten Inheritance



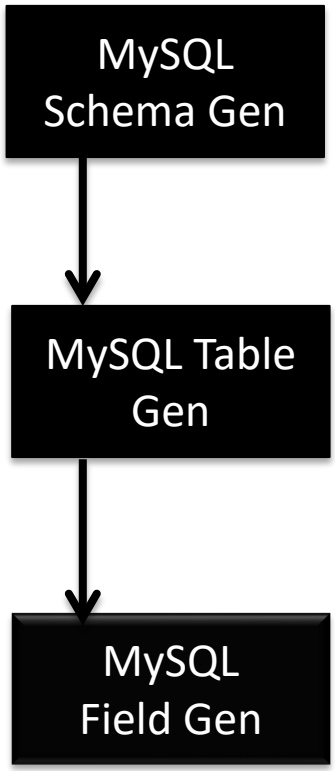
Customer table after flattening fields from inherited Person table

```
<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="{0-120}" default="0"/>
 <Field name="personAddressPostCode" type="String" size="7" unique="true"/>
 <Field name="personAddressUnitNo" type="Natural" size="5" unique="true"/>
 <Field name="personAddressStreet" type="String" size="30" unique="true"/>
 <Field name="personAddressCity" type="String" size="20"/>
 <Field name="id" type="Natural" size="7" unique="true"/>
 <Field name="details" type="String" size="250"/>
</Table>
```

New PK

Old PK

Step N: Code Generation



Combined Table

```
CREATE TABLE Customer (
  personId INT(7) NOT NULL,
  personForeName VARCHAR(10),
  personSurName VARCHAR(10),
  personAge INT DEFAULT 0,
  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));
```

PK

Trigger for check constraint

```
CREATE TRIGGER customerCheck BEFORE INSERT ON Customer
  FOR EACH ROW
    IF (NEW.personAge < 0 OR NEW.personAge > 120) THEN
      SET NEW.personAge = DEFAULT;
    END IF;
```


Sample Code I

```
public class MySQLSchemaGenerator extends SchemaGenerator {

   public void generate() throws TreeException, IOException
   {
      try {
            openFile(getTypeName() + "DB.sql");
            createDatabase();
            writeTables();
            writeTriggers();
            closeFile();
         }
   }
```

Sample Code II

```
public void writeTable() throws TreeException, IOException
  {
    write("CREATE TABLE "+ getTypeName()+ " (");
    if(!hasPKeyFields)
    {
    write("  autoField INT UNSIGNED NOT NULL AUTO_INCREMENT,");
    }
    writeFields();
    if(hasPKeyFields)
        writePrimaryKeys();
    else
    {  write("  PRIMARY KEY(autoField)"); }
    if(hasFKeyFields)
    {
       write(",");
       writeForeignKeys();
    }
    write(");");
}
```

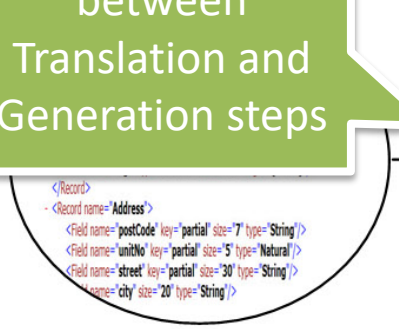
Outline

- Introduction
- Brief Description of ReMoDeL
- Case Study: Database Generator for MySQL
- *Transformation Composition in ReMoDeL*
  - *Linear composition of transformations (external)*
  - *Hierarchical compositions of agents (internal)*
- Conclusion and Q/A

External Composition

Linear Composition (2-Phase)

Separation between Translation and Generation steps



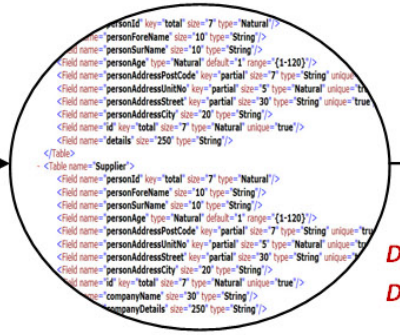
DBQ Conceptual Data Model

Schema Translator

Model-to-Model Transformation

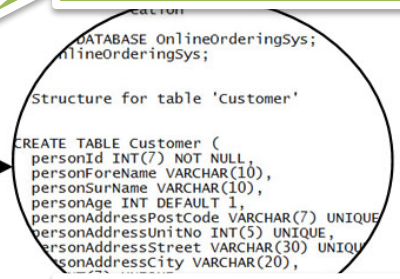
Schema Generator

Model-to-Code Generation



DBQ Logical Data Model

No Glue code

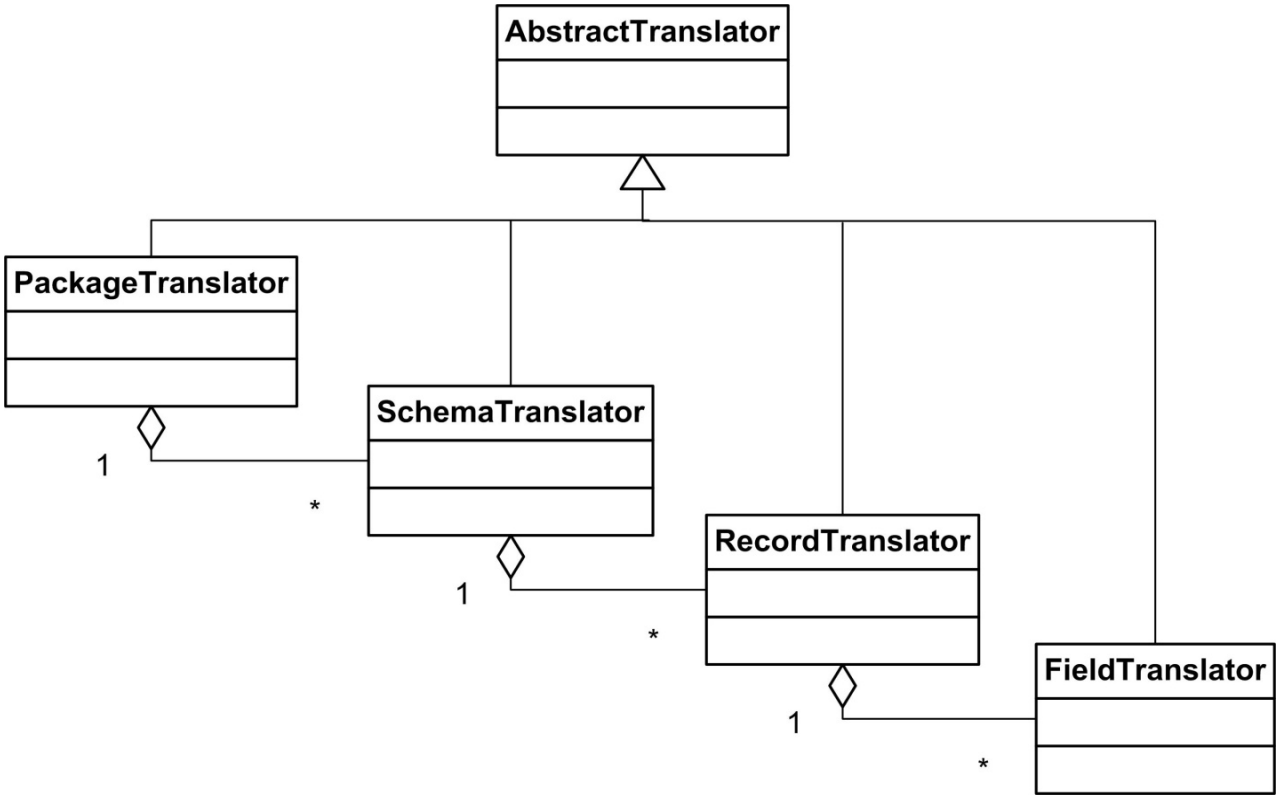


Generate code from a detailed intermediate model

ReMoDeL Database Generator Framework

Internal Composition - I

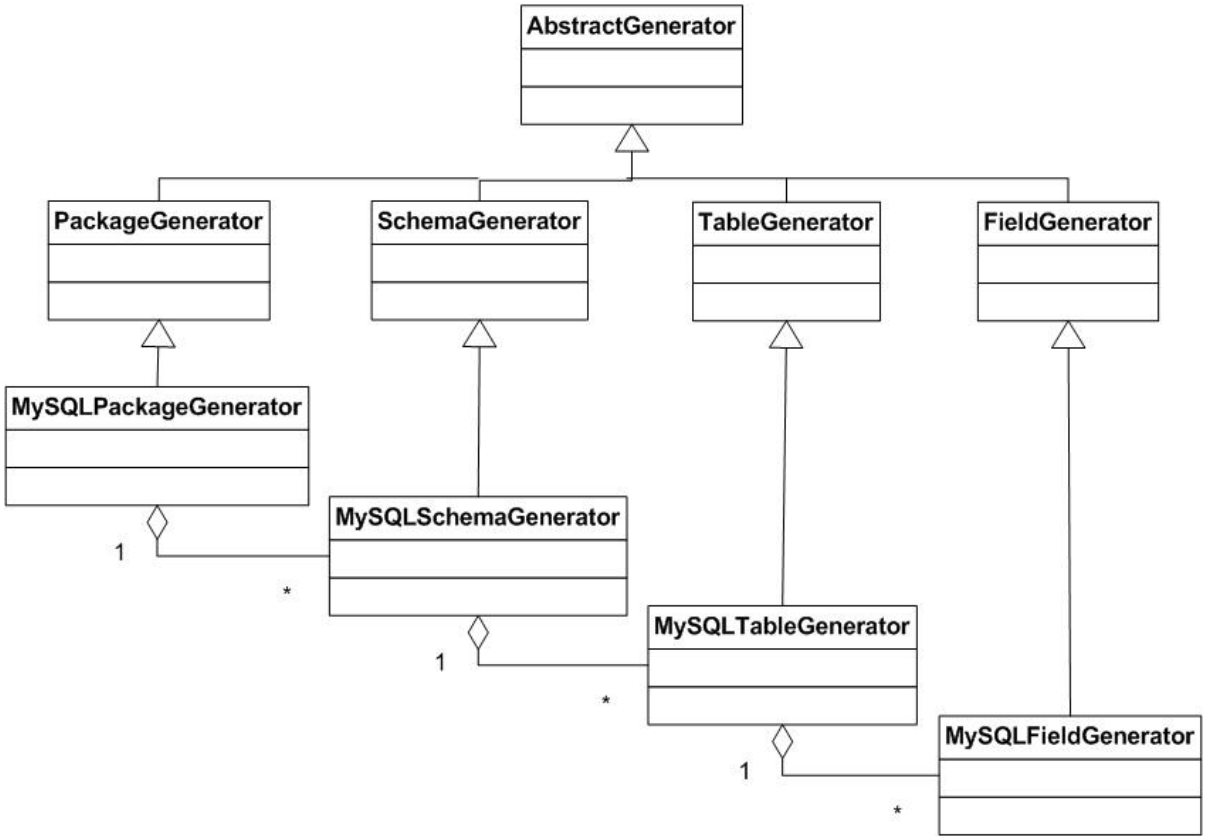
Hierarchical Composition



Model-to-Model Transformation Component

Internal Composition - II

Hierarchical Composition



Model-to-Code Transformation Component

Outline

- Introduction
- Brief Description of ReMoDeL
- Case Study: Database Generator for MySQL
- Transformation Composition in ReMoDeL
- *Conclusion and Q/A*

Thank you … !

Questions