Theory and Applications of Model-Driven Engineering

Core Course

Readme of the exam: Theory and Applications of Model-Driven Engineering

Ph.D. Program in Computer Science: XL cycle

Authors:

Alessandro Valerio

Sofiia Prokofieva

Alessio Gabriele

Supervisors:

Ludivico Iovino

Francesco Basciani

July 19, 2025

Readme of the exam: Theory and Applications of Model-Driven Engineering

July 19, 2025

Project: Theory and Applications of Model-Driven Engineering

This project of Theory and Applications of Model-Driven Engineering (TAMDE) presents five case studies where the primary goal is to define source and target metamodels, create model instances, and execute model-to-model (M2M) transformations using the Epsilon Transformation Language (ETL). The project also involves model slicing and visualization using Picto. This document provides a detailed explanation of the case studies and a generalized step-by-step tutorial for setting up the environment and running the transformations in Eclipse.

1 Case studies

1.1 First Case Study: Farmers to Market Transformation

This case study focuses on transforming a detailed model of farmers and their produce into a simplified model representing products available at a market.

The Metamodels

Two distinct metamodels are defined for this transformation, representing the source and target structures.

- **Source Metamodel (A):** `farmers.ecore`. This metamodel captures information about individual farmers and the various fruits they cultivate.
  - `FarmModel`: The root container for all farmers.
  - `Farmer`: Represents a farmer with an `id`, `name`, `age`, and a list of fruits they own.
  - `Fruit`: Represents a type of produce with a `name`, `amount`, `quality`, and `price`.
- **Target Metamodel (B):** `market.ecore`. This metamodel provides a market-centric view. It represents a flattened list of products available for sale, abstracting away the individual farmer's inventory into a single market list.
  - `Market`: The root container for all product selections.
  - `Selection`: Represents a single product line for sale, containing its `name`, `quality`, `price`, `amount`, and the `farmer_id` of the producer.

Transformation Goal

The transformation, defined in `farmer2market.etl`, reads an instance model conforming to `farmers.ecore` (i.e., a list of farmers and their fruits) and produces a new instance model conforming to `market.ecore`. The logic iterates through each farmer and each of their fruits, creating a corresponding `Selection` entry in the market model for every fruit.

1.2 Second Case Study: Customer to Data Warehouse Transformation

This case study focuses on transforming a transactional model of customer purchases into an aggregated, analytical model suitable for a data warehouse.

The Metamodels

Two distinct metamodels are defined, representing the raw transactional data and the summarized analytical data.

- **Source Metamodel (A):** `customer.ecore`. This metamodel captures detailed information about individual customers and their specific purchase histories.
  - `CustomerModel`: The root container for all customers.
  - `Customer`: Represents a customer with a `customer_id`, `name`, `email`, `age`, and a list of their purchases.
  - `Purchase`: Represents a single transaction with a `product` name, `quantity`, `price`, and `date`.
- **Target Metamodel (B):** `warehouse.ecore`. This metamodel provides an aggregated, analytical view. It summarizes each customer's activity into a single "fact" record.
  - `DataWarehouse`: The root container for all customer facts.
  - `CustomerFact`: Represents a summarized view of a customer, containing their `customer_id`, `name`, `email`, along with calculated fields like `total_spent`, `total_orders`, and the `last_purchase_date`.

Transformation Goal

The transformation, defined in `customer2warehouse.etl`, reads an instance model conforming to `customer.ecore` and produces a new instance model conforming to `warehouse.ecore`. The logic iterates through each customer and aggregates their purchase history by calculating the total money spent, counting the number of orders, and identifying the most recent purchase date. This creates a concise, analytical summary for each customer.

2 Step-by-Step Tutorial

Below is the workflow for creating metamodels, instance models, and running transformations using Eclipse and the Epsilon platform.

1. Project and Metamodel Setup

1. **Create Project**: In Eclipse, go to `File > New > Project...`. Select `General > Project` and give it a name (e.g., `CaseStudy1`).
2. **Create Metamodel Folder**: Inside your new project, create a folder named `metamodel`.
3. **Define Metamodels with Emfatic**:
   - Inside the `metamodel` folder, create two new files with the `.emf` extension (e.g., `farmers.emf` and `market.emf`).
   - Write the metamodel definitions in these files. Ensure the `@namespace` URI is unique for each metamodel.
4. **Generate Ecore Metamodels**:
   - Right-click on each `.emf` file.
   - Select `Epsilon > Generate Ecore from Emfatic`. This creates the corresponding `.ecore` files.
5. **Validate Metamodels**: Right-click on each generated `.ecore` file and select `Validate`.

2. Creating an Instance Model

1. **Create Models Folder**: Create a new folder named `models`.
2. **Define Model with Flexmi**:
   - Create a new file with the `.flexmi` extension (e.g., `farmers_large.flexmi`).
   - Write the instance data. Reference the correct namespace URI from your source metamodel (e.g., `<?nsuri farm01?>`).
3. **Generate XMI Model**:
   - Right-click the `.flexmi` file.
   - Select `Generate XMI Model`. This creates an `.xmi` file.

3. ETL Transformation

1. **Create Transformation Folder**: Create a folder named `transformations`.
2. **Create ETL File**: Create a file named `farmer2market.etl`.
3. **Set Up Run Configuration**:
   - Go to `Run > Run Configurations...`.
   - Create a new configuration under `ETL Transformation`.
   - Select your `.etl` file.
4. **Configure Source Model (Input)**:
   - Add a model named (e.g., `farmers`).
   - Select `EMF Model`.
   - Select the `.xmi` file and the corresponding `.ecore` file.
   - Check `Read on load`, uncheck `Store on disposal`.
5. **Configure Target Model (Output)**:
   - Add another model (e.g., `market`).
   - Select `EMF Model`.
   - Specify a non-existent output file.
   - Select the corresponding `.ecore` file.
   - Uncheck `Read on load`, check `Store on disposal`.
6. **Run the Transformation**: Click `Apply` and then `Run`.

4. Visualizing Metamodels with Picto

1. **Setup**:
   - Create a `picto` folder.
   - Copy the `.settings`, `picto` folder, and `.project` file from your Picto installation.
2. **Create Picto Configuration**:
   - Create a file named `farmers.ecore.picto`.
   - Paste the following configuration:

```
<?nsuri picto?>
<picto format="plantuml" transformation="picto/ecore2plantuml/ecore2plantuml.egl
    ">
</picto>
```

   - Note: For `.model` files (e.g., `market.ecore.model`), use the special configuration of the .ecore.model file provided in the tutorial.
3. **Generate Diagram**: Right-click the `.picto` file and select `Picto > Generate diagram`.

5. Slicing / Semantic Importance

References