Meta-modelling and Model Transformation–the two pivots of Model Driven Architecture

Amaranth Wei He supervised by Hans Vangheluwe
School of Computer Science,
McGill University,
Montreal, Canada
whe6@cs.mcgill.ca

December 21, 2004

Abstract

Initiated by Object Management Group (OMG), Model Driven Architecture (MDA) recently stirs enormous interests of both the research organizations and the industry corporations. This new approach is intended to play a key role in the fields of information system and software engineering. It specifies an automated process of developing information systems from high-level analysis to code generation. Two prerequisites for the success of MDA application are - an extensible meta-modeling framework and an automated model transformation mechanism.

1 Introduction

Model Driven Architecture (MDA) is the OMG initiative that attempts to separate business functionality specification from the platform-specific implementation. It provides a solution to develop the software system at the business model level. Although perfect and practicable in principle, two issues need to be addressed satisfactorily for the successful application of MDA- an extensible meta-modeling framework and an automated model transformation mechanism.

This report introduces the basic concepts of MDA in section 2. Then, in section 3 and 4, it looks into some of the technology details of the two most important issues in MDA- meta-modeling and model transformation. After the discussions, an implementation of a MDA application is given in section 5. The tool supporting the implementation is ATOM3-a meta-modeling and model transformation tool developed by Modeling, Designing and Simulation Laboratory at School of Computer Science in McGill University.

2 MDA Introduction

2.1 The OMG Vision

In 2001, Object Management Group launched a new movement in developing software systems by proposing Model Driven Architecture. Firmly based on OMG's many well-established modeling standards like UML, MOF, XMI etc., MDA provides a revolutionary way of developing software systems. It separates the technology dependent concepts from independent concepts and proposes solutions to automate the software development process. The ambition of OMG is to raise the abstraction level of developing systems to a new height and to settle the problems that has long exist in the traditional software industry. As stated in [7], the promise of MDA is —" to allow definition of machine-readable application and data models which allow long-term flexibility of implementation, integration, maintenance, testing and simulation."

2.2 MDA development process

MDA defines a full life cycle based on the use of various models automating a seamless process from analysis to code generation. Figure 1 and Figure 2, given in [8] illustrate the traditional software development process and the MDA process.

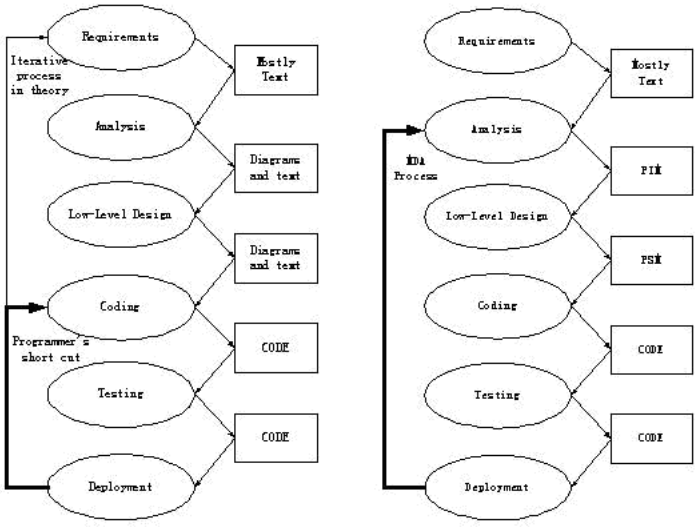


Figure1. Traditional software development process Figure2. MDA development process

Although they both consist of the same six stages in their life-cycles, there're several distinguished features in a MDA process

- In the analysis stage, we build a model with a high level of abstraction that is independent of any implementation technology. This model is called Platform

Independent Model (PIM). Unlike the UML diagrams and text in a traditional development process, which were only used to convey meanings between people, this PIM is an executable model that can be calculated by computers.

- In the low-level design stage, the PIM is transformed automatically by certain tools into one or more Platform Specific Models (PSMs). A PSM combines the specifications in the PIM with the details that specify how the system uses a particular type of platform. PSMs are also executable models
- In the coding stage, PSMs are transformed to source codes. In MDA, coding is no longer a manual work. It is performed by transformation tools.

In conclusion, a MDA development process consists of the following three steps:

- Step1 Using specific tools to build Platform Independent Model which includes all the business logics of the system but no technology information.
- Step2 After deciding on the implementing platforms, using transformation tools to transform PIM to corresponding Platform Specific Models.
- Step3 Using transformation tools to transform PSMs to corresponding source codes.

2.3 MDA Framework

The MDA framework consists of the following building blocks.

2.3.1 Models

A model is a simplification of a system, which captures the essence of the system. There're three types of models in the MDA process, each at a different abstraction level.

- Platform Independent Model (PIM)
  These models describe the structure and behaviour of the system independent of any technology implementation platform.
- Platform Specific Model (PSM)
  These models are expressed in terms of particular implementation platforms.
- Code
  In MDA, source codes can also be viewed as a special kind of models.

2.3.2 Transformations

In MDA, transformations means the automatic generation of a target model from a source model, according to a transformation definition[8].
In essence, MDA is a series of transformations from high-lever models with no technology information to elaborated platform-specific models. Best practices and accepted defaults allow for these transformations.
The two most important transformations in MDA are:

- Transformations from PIM to PSMs
- Transformations from PSMs to cods

2.3.3 Languages for describing the source and target models

Models are depicted in certain languages. Since the models in MDA are executable models, the languages depicting them should be well-defined, which means they should have strict syntax and semantics. In MDA, we use a special language definition mechanism, which is formally named meta-modelling. We'll elaborate this mechanism in the next section.

2.3.4 Language for describing the transformations

Transformations specify how the constructs in the source models are mapped to target models. This is depicted as a set of transformation rules. A well-defined language is needed to describe the transformation definition in order that it can be understood by computer programs.

2.3.5 Transformation tools

In a MDA process, transformations are performed by transformation tools, which are special programs that support certain transformation rules from the source models to target models.

2.4 MDA specification

A complete MDA specification consists of a definitive platform-independent UML model, plus one or more platform-specific models (PSM) and interface definition sets, each describing how the base model is implemented on a different middleware platform. An example of a MDA specification is given in Figure3

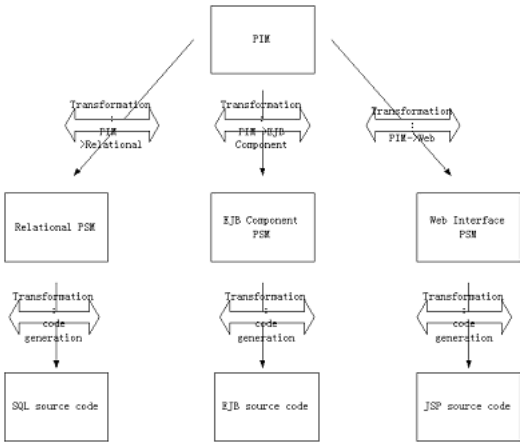


Figure3 The structure of the MDA specification of a three-tier web-based information system

In Figure 3, we depict a typical three-tier web-based information system. The MDA specification of the system consists of the following building blocks

- a PIM
  This PIM is written in UML. It comprises the information about all the business logics of the system.
- Three PSMs
  a Relational Database PSM depicted in an Entity-Relationship diagram
  a EJB PSM written in UML profile.
  a Web Interface PSM written in UML profile
- Three PIM to PSMs transformations
  a PIM to Relational Database PSM transformation
  a PIM to EJB PSM transformation
  a PIM to Web interface PSM transformation
- Three PSMs to codes transformations
  a relational database PSM to SQL transformation
  a EJB PSM to Java code transformation
  a Web interface PSM to JSP and Html transformation

2.5 Benefits of adopting MDA

[8] explains why MDA is a solution to many of the long-existed problems in software development.

- Productivity
  In MDA, the focus of development shifts to developing PIM. Much of the coding is done by the transformation tools. Developers will concentrate on the business logics instead of the technical details.
- Portability and Interoperability
  Tools supporting transformations to different platforms will be available. The same PIM can be automatically transformed into multiple PSMs by these tools. These tools can also generate the bridges between different PSMs.
- Maintenance and Documentation problem
  As a description of the system in high-level abstraction, PIM is also directly related to the final implementation. Changes made to the system will eventually be reflected in PIM, In this way, high-level documentation (PIM) will remain consistent with the actual code while the system evolves.

3 Meta-modeling in MDA

The application of MDA requires that the models, whether PIMs or PSMs, be described in well-defined languages which is suitable for automated interpretation by

a computer. Since modeling languages like UML often take the forms of graphics, the traditional methods like Backus Naur Form(BNF), which has been successful in defining text-form languages, are not appropriate here. A different mechanism is needed to define the modelling languages in MDA. On the one hand, this mechanism should be able to describe the syntax and semantics strictly. On the other hand, it should support the extensibility of the modelling languages. The solution is a meta-modelling framework based on the traditional four-layer metadata architecture. .

3.1 The concept of meta-modeling

In [4], the concept of meta-modeling is proposed as a solution to the following questions:

- Systems in different domains or different parts in a complex system are often modeled by different formalisms that are most appropriate for describing their behaviors.
- The need to investigate the overall behavior of the system with different parts described by different formalisms

The essence of meta-modeling is to use an appropriate modeling language to model the different formalisms. Each formalism is described as the possible structures that can be expressed in the language. Based on this common meta-model, it's possible to generate a tool supporting all these formalisms. Also, transformations between models described in different formalisms can be described explicitly. Entity-Relation formalism and UML Class Diagram are often used for meta-modeling.

3.2 The classical framework for meta-modeling

3.2.1 The four layers of the classical framework of meta-modeling

The four layers of the classical framework with a concrete example, are illustrated in Figure 4. [6]

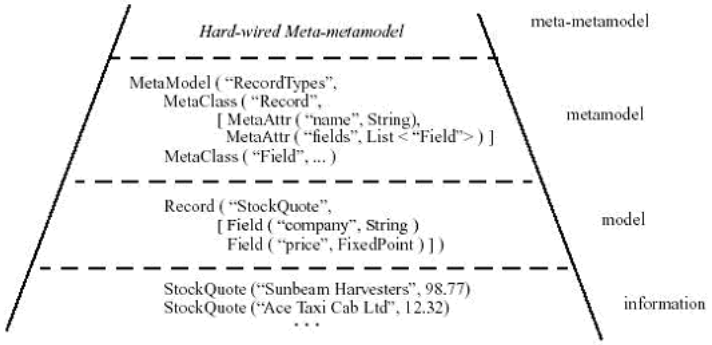


Figure4. Classical Four Layer meta-model architecture

- It can support any kind of models and meta-models.
- It can support interchange of arbitrary metadata (models) and meta-metadata (meta-models) between parties that use the same meta-meta-model.

3.3 MOF meta-modeling framework

Meta Object Facility (MOF), the metadata architecture standard proposed by OMG, is based on the traditional four layer metadata architecture.
Figure 5 illustrates the four layers of MOF framework with an example given in [8]. The meaning of each layer is described below.

- Layer M0: The instances
  The M0 layer is where the actual instances of the systems exist. In our example, the customers ”Dr. Joe Nobody” and ”Mr. Mark Everyman” are both items in the real systems.
- Layer M1: The Model of the System
  The M1 layer contains models of the system. In our example, it is where the concept ”Customer” is defined: a UML Class named ”Customer” with two UML Attributes named ”title” and ”name” respectively. Each element at the M0 layer is an instance of an element at the M1 layer. So, both ”Dr. Joe Nobody” and ”Mr. Mark Everyman” are instances of ”Customer”
- Layer M2: The Model of the Model
  An element at the M2 layer specifies the elements at M1 layer. The Elements at the M1 layer are instances of the elements at M2 layer. In our example, it is in the M2 layer that the concept ”UML Class” and ”UML Attribute” are defined. The UML Class ”Customer” and ”Order” are instances of ”UML Class” at M2 layer
- Layer M3: The Model of M2
  Similarly, M2 layer elements are defined at M3 layer. In MOF meta-modeling framework, MOF is the standard M3 language, all the modeling languages are instances of MOF.

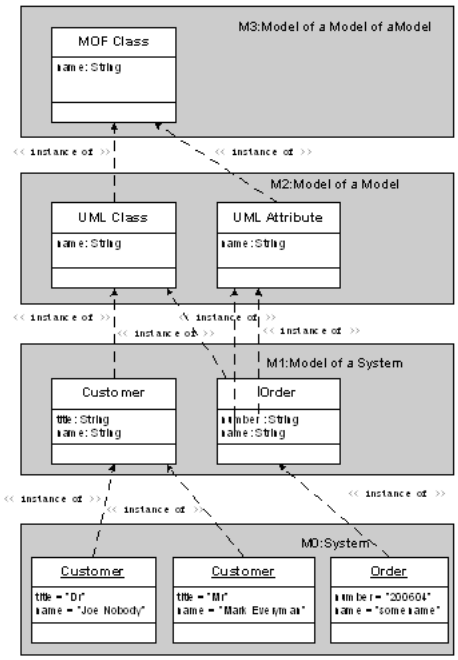


Figure5. An example in MOF meta-model framework

3.4 The importance of meta-modelling in MDA

Meta-modeling provides the basis for describing and transforming models in MDA. This is demonstrated in two ways

- It is used to describe the source and target models.
  In MDA, both PIM and PSM are described in M2 modeling languages (meta-models).
- It is used to describe the model transformations
  In MDA , the model transformations are specified by sets of rules defined in terms of the corresponding meta-models.

3.5 Some issues in the current meta-model framework

Although it is generally recognized that an extensible meta-modeling framework is the prerequisite of MDA application, there's still no consensus on the form of this framework. Recently there are a lot of discussions about the problems existed in the MOF framework. Many researches have been carried out in this area and some incomplete solutions have been proposed.

3.5.1 Strict Meta-modeling

The characteristics of MOF meta-model framework that has been discussed most is strict meta-modeling. The precise definition of this concept is given in [2] as follows:
In an n-level modeling architecture, M0,M1,Mn-1, every element of an Mm-level model must be an instance-of exactly one element of an Mm+1-level model. For all 0¡=m¡n-1, and any relationship other than the instance-of relationship between two elements X and Y implies that level(X)=level(Y).
This restriction means that in MOF meta-model framework :[2]

- the meta-modeling framework takes the form of a linear hierarchy
- Levels are formed purely by instance-of relationships
- Levels have strict boundaries that may not be crossed by relationships other than instance-of relationships
- IInstance-of relationship only exists in immediately adjacent two levels

3.5.2 Problems caused by strict meta-modeling restriction

Since strict meta-modeling restricts the instance-of relationship in two immediately adjacent levels, a model can only define the semantics of its direct instances, and can have no effect on entities created by further instantiation steps. This instantiation mechanism, which is called ”shallow instantiation”, causes some fundamental problems when the meta-levels scale up to more than two levels.[1]
Two problems that are discussed most are: [1]

- Ambiguous Classification This problems occurs when an instance need to be assigned both the physical classifier and the logical classifier.
- Replication of Concepts Because shallow instantiation fails to carry information across more than one level, it’s often necessary to duplicate information at multiple levels

3.5.3 Some proposals for solving the problems

A lot of research work has been devoted to solve the problems in the current meta-modeling framework. Some solutions have been proposed to cure the symptoms. Below are two of them.

- Two Fundamental Meta-dimensions [2]
  This addresses the problem of ambiguous classification by proposing a two-dimensional modeling framework, one physical dimension and one logical dimension.
- Deep Instantiation [1]
  This proposition cures the shallow instantiation symptoms by introducing the concepts of ”Potency” and ”Single and Dual Fields” into the model class, thus allowing the modeling element’s class feature be acquired automatically by the instantiation steps.

3.5.4 UML2.0 and MDA meta-modeling solution

While still under revision, UML2.0 is supposed to provide the solution to MDA meta-modeling framework. In UML2.0, the definition of the UML is organized into two parts:

- The infrastructure which describes the overall framework within which UML modeling is performed
- The superstructure which populates this framework with modeling concepts that constitute the UML modeling language.

It is generally accepted that unless a sound meta-modeling framework is provided, the success of MDA is not possible

4 Model Transformation in MDA

As we have discussed in Section 2, the MDA approach specifies mappings from Platform Independent Model(PIM) to one or more Platform Specific Models(PSMs). While meta-modeling provides the foundation for describing the PIM and PSMs, it is the model transformation that act as the engine of the whole MDA process.

4.1 Basic concepts in Model transformation in MDA

4.1.1 Some important definitions

Before looking into the details of model transformation in MDA, there're some important definitions need to be clarified. In [8], the definitions are given as below:

- Model transformation
  A model transformation is the automatic generation of a target model from a source model, according to a transformation definition.
- Transformation definition
  A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target Language.
- Transformation rule
  A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.
- Transformation definition language (transformation model)
  Transformation definition language is the language in which to write the transformation definitions. In MDA, it is also considered as a model as the languages depicting PIM and PSM.

4.1.2 Relations between meta-modeling and model transformation in MDA

The relations among PIM, PSM, meta-models, meta-meta-models, model transformation definition language, transformation definition and transformation tools is illustrated in Figure6 [8] . It provides the view of the complete MDA framework we have discussed so far.

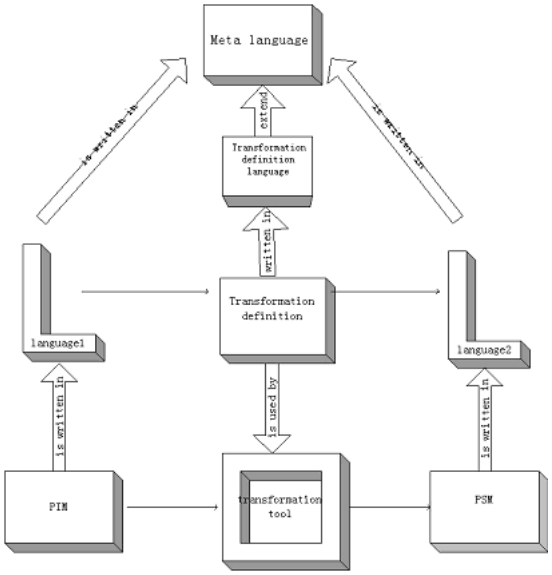


Figure6. The complete MDA framework

4.1.3 Model Transformation procedure

A model transformation consists of the following three successive steps:

- Step1. Selecting components from source models
- Step2. Constructing and populating new models to form the target of the transformation
- Step3. Modifying the source model so that it is used to form part of the target model

4.1.4 MDA model transformation language requirements

Transformation definitions in MDA should be depicted in well-formed languages in order that it can be executed in computers. Based on experiments of several different methods to implement the model transformation, [5] proposes the set of requirements for a transformation language suitable for describing model to model mapping rules required to realise the MDA vision.

- Match sets of source model elements

- Match elements by type and precise-type
- Filter the set of matched tuples based on associations attribute values, and other criteria
- Establish associations between source and target model elements
- Handle recursive structure with arbitrary levels of nesting.

4.1.5 The current situation of model transformation standardization in MDA

Although the importance of model transformation in MDA is generally recognized, there's no well-established foundation to rely in describing how we take an instance of the source model and transform it to produce an instance of the target model. A lot of experiments have been carried out and some solutions have been proposed as the approaches to define the model transformation definitions. These can be grouped into two categories:

- Procedural definition of the transformation, with explicit source model traversal and target object creation and update. Graph Transformation is an example in this category. In next section, we will provide a MDA implementation powered by Graph Transformation.
- Declarative definition of the transformation, with implicit source model traversal and implicit target object creation. Generated XSLT is an example in this category.

MOF2.0 QVT, which is still in revision, is supposed to provide a standard framework and language for defining model transformations in MDA.

4.2 Graph Transformation and its application in MDA

Models in MDA are described by visual modeling languages like Class Diagram. Since most established techniques for language definition are based on abstract syntax tree, yet visual languages usually have a graph-like structure, we need a special mechanism to deal with them. With its established mathematics foundation and its competency in processing visual models, graph transformation recently draw much attention of the research society for its potential as a sound approach for defining model transformations in MDA.

4.2.1 Concepts of Graph Transformation

Some basic concepts of Graph transformation is given in [3] as below:

- Graphs
  A graph consists of a set of vertices V and a set of edges E such that each edge e in E has a source and a target vertex s(e) and t(e) in V, respectively.
  In object-oriented modeling, we often consider graphs at two levels: the type level (a class diagram) and the instance level(all the object diagrams), where

classes and objects are described as the vertices while associations and links are described as the edges. Typed and attributed graphs are used in order that the attributes of the classes and values of the objects can be properly specified Figure 7 shows an example of a class diagram and its object diagram [3]

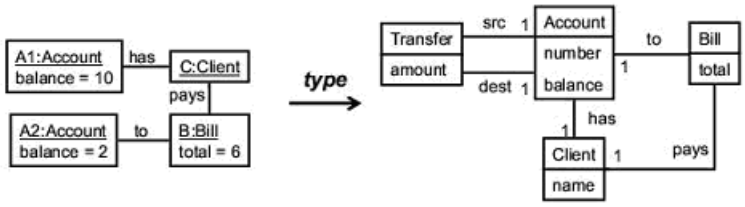


Figure7. A Class Diagram and its object diagram

- Graph transformation rule
  A graph transformation rule p: L-¿R consists of a pair of instances of the type graph (class diagram), where left-hand side L represents the pre-conditions of the rule while right-hand side R describes the post-conditions

- Graph transformation rule application
  The application of a graph transformation rule consists of three consecutive steps:
  Step1. Find an occurrence o—L of the left-hand side L in the current object graph G
  Step2. Remove all the vertices and edges from G which are matched by L/R
  Step3. Glue the remaining structure of G with R / L An example of the transformation application is given in [3] by Figure 8

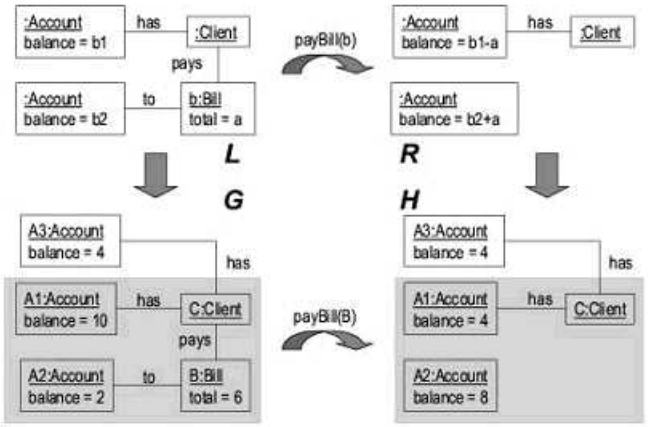


Figure8. A transformation rule application

4.2.2 Graph Transformation applications in processing visual modeling language

[3] discusses the application of Graph Transformation in processing visual modeling languages, especially how it can be used to specify mappings between concrete syntax and abstract syntax and between abstract syntax and semantics. Figure9, given in [8], illustrates the three layer of a visual modeling language. In Figure 9, each mapping indicated by the arrows can be depicted by a set of graph transformation rules

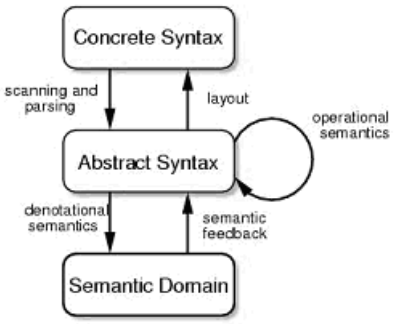


Figure 9. A layered view of a visual modeling language

4.2.3 Graph Transformation applications in MDA model transformation

If we consider the environment of target model as the denotational semantic domain of the source model, the model transformation process is just the application we've already mentioned in 4.2.3. In section 5, we give an example of implementing MDA. The model transformation is powered by a set of graph transformation rules.

4.3 MOF2.0 QVT proposed solution

The MOF2.0 QVT, which is still under revision, is supposed to provide a comprehensive solution to transformations in MDA.

4.3.1 The layers of transformation definitions

The proposal provides a two-layer framework to define transformations

- Infrastructure
  This is the low-level specification which is useful for tool vendors. It contains the core language to precisely define transformations.
- Superstructure
  This is the high-level language for end users. The semantics of the superstructure are given by its translation into the infrastructure.

4.3.2 Implementation language

A standard language MTL(Model Transformation Language) , which comes in both graphical and textual form, is provided in the proposal. The language uses pattern matching as its mechanism to create transformations.

5 Implementing MDA in ATOM3

In this section, we implement a web-based information system as an example to show how to develop software systems in MDA applications. The tool we used for the implementation is ATOM3(A Tool for Multi-formalism Meta-Modelling), which is developed in Modelling, Simulation and Design Lab at McGill University.

5.1 The necessary tools for supporting MDA development process

As a new approach to build software systems, MDA requires a set of developing tools totally different from those used in a traditional development process. Below is the list of tools that are indispensable in the MDA development process.

5.1.1 The environments to build the meta-models and models

As we have discussed in Section 3, both PIMs and PSMs are defined by certain meta-models. These meta-models are models in their own rights from the view of the meta-meta models. In the MDA development process, we need the tools that support the meta-model of PIM, so that we can create and edit the models in the environment. Sometimes, instead of using separate tools supporting the PIM or PSM meta-model, we can use the tools that support their common meta-meta-model. In that case, we can build the meta-models of both PIM and PSMs in this environment, and use these meta-models to generate tools for building corresponding models.

5.1.2 The environments to specify the transformation rules

Another very important issue in developing a MDA application is prescribing the transformation specifications. To fulfill this task, tools are needed which provides a user-friendly interface to accept the input and modification of the transformation rules.

5.1.3 The environments to execute the model transformations

For the processing of the models in the development, we will need the tools that can execute the model transformations. These tools take the source models and the transformation rules as inputs and generate the target models as outputs.
The above mentioned environments are necessary for developing MDA applications. They can be either implemented as separately tools, or as an integrated tool, which is exemplified by ATOM3. We will discuss it in more details in the coming sub-section.

5.2 Introduction to ATOM3

ATOM3(A Tool for Multi-formalism Meta-Modelling),which is developed in Modelling, Simulation and Design Lab at McGill, can be used to describe different formalisms, as well as to generate custom tools to process models expressed in the corresponding formalism[4]. It also includes graph rewriting techniques as the mechanism to perform the transformations between formalisms. So, we can use this tool to specify the PIM and PSM meta-models, build the PIM, and prescribe the transformation rules from PIM to PSM.

5.2.1 The layers of meta-modeling framework in ATOM3

AtoM3 has a meta-modelling layer in which different formalisms (PetriNet, Class Diagram, State Chart,etc.) are modelled graphically. There're three layers of the meta-modeling framework in ATOM3.

- Meta-meta model layer
  The ER(Entity-Relation) formalism extended with constraints is available at this layer. In the concrete syntax of ER formalism in ATOM3, entities are represented by rectangulars while relations are depicted as diamonds. Constraints can be specified by the cardinalities or as pre- or post- conditions of certain events (creating, modifying or deleting an entity or a relation, for example).

- Meta-model layer
  This layer includes the formalisms depicted by ER formalism. The formalisms that are available in ATOM3 now includes Petri Net, Finite State Automata, Sequence Diagram, State Chart, GPSS, etc. ATOM3 also provide an environment for users to build their own meta-models. Figure 10 shows the State Chart formalism depicted as a model in ER formalism.

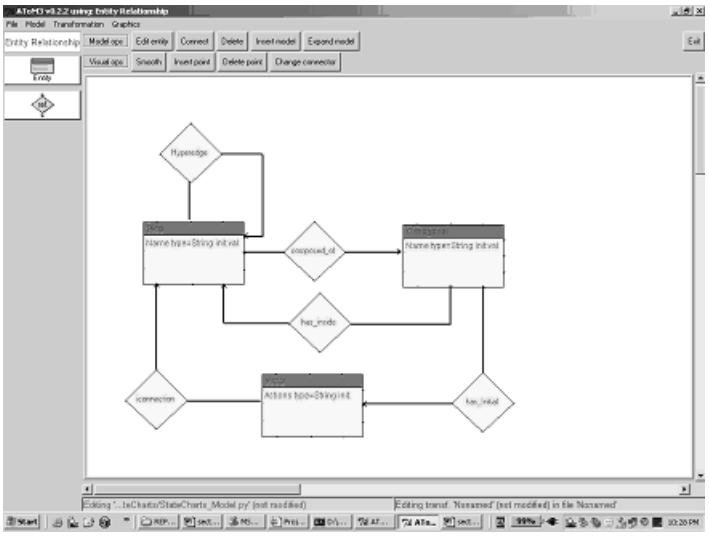


Figure10. State Chart meta-model depicted by ER meta-model

- Model layer
  The models built in a certain meta-model exist in this layer.

5.2.2 Graph Grammar in ATOM3

In ATOM3, Graph Grammar is used both for concrete syntax definition and model transformation definition.

- For the concrete syntax definition, Graph Grammar specifies the appearance of each entity in the formalism and the mapping to Abstract Syntax
- For the model transformation definition, Graph Grammar specifies how the constructs in the source model (LHS) should be mapped to constructs in the target model (RHS)

5.3 Background of the information system in the project

5.3.1 Business background

To illustrate the process of developing a MDA application, we will build an ordering system for a breakfast shop. The business background is depicted in [8].

5.3.2 MDA specification of the system

We will implement the system as a web-based application. The system takes the form of a typical three-tier framework which is already illustrated in sub-section2.4.
To build the complete system, we should implement the transformations from PIM to all the three technology platforms. In this project, we only implemented the transformation to Relational Database, which is enough for the understanding of the MDA developing process.

5.4 The elaborated developing process in ATOM3

5.4.1 Defining the PIM meta-models

The most appropriate formalism to describe the business model is Class Diagram. We can describe the Class Diagram formalism in ER formalism. In ATOM3, this description is implemented as a model built by ER meta-model. What we should do is to build the model according to Class Diagram definition in the ER meta-model environment. Figure11 shows the model of Class Diagram in ER meta-model.

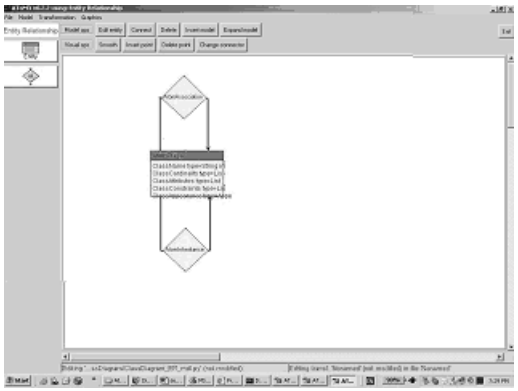


Figure11 Class Diagram meta-model depicted in ER formalism

We can infer from this figure: in the Class Diagram meta-model:

- The only entity in the formalism is AtomClass, which is specified by ClassName, ClassAttribute,etc.
- There’re two kinds of relations among AtomClass entities: association and inheritance.

We can also specify the concrete syntax and semantics of the Class Diagram formalism.

- Concrete syntax is given by specifying the appearances of the entities and relations when editing the model. Figure12 shows the editing window for assigning appearance to the AtomClass entity. Similarly, we assign a line as the appearance of the association relation and a line with a triangle in the end as the appearance of the inheritance relation.

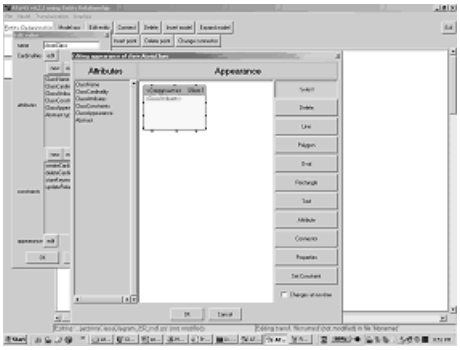


Figure12. The attribute editing window of AtomClass

- We can give the semantic meaning of the entities and relations by specifying the attributes. In this example, we specify that each AtomClass should have a String attribute ClassName, a List attribute ClassAttributes, etc. Constraints are

also used to define the semantics. For the AtomClass entity, we give several constraints that will be triggered when creating, deleting or updating the entity.

After defining the formalism by building the model in ER meta-model, we can save this model for future modification. ATOM3 also generates a tool to support the models being depicted by Class Diagram. This is implemented as including the new formalism as a meta-model which can be launched for building corresponding models.

5.4.2 Defining the PSM meta-models

Since our target technology platform is Relational Database System, the PSM meta-model should be clear and specific enough to depict the structure of the database tables in the system. Although traditionally ER formalism has been used to depict the database tables, in our project we use a meta-model with tailored features of ER formalism. We name this meta-model Table Diagram. Although this meta-model is very simple, it can fulfill the work of elaborating the Relational Database PSM. Figure13 shows the Table Diagram described as a model in ER meta-model

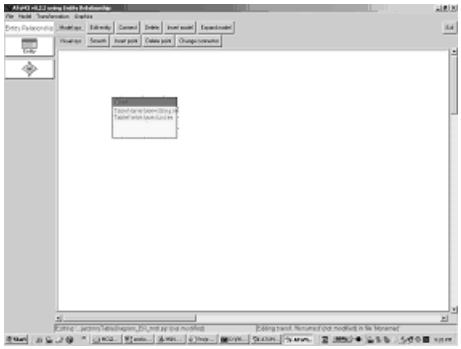


Figure13. Table Diagram meta-model depicted in ER formalism

We can infer from Figure13, in the Table Diagram formalism:

- The only entity in the formalism is Table which is specified by a String attribute TableName and a List attribute TableFields
- No relation is applicable in the formalism.

In the attribute editing window, we choose the appearance for the Table entity

Figure14. The attribute editing window of Table

After saving this model and generating corresponding meta-model, Table Diagram can be used later for building models in this formalism.

5.4.3 Building the PIM of the system

The only model we should build manually in the MDA process is the PIM. This model should include all the business logics in the system. In practice, PIMs are depicted as class diagrams. In our project, to build the PIM, we should first launch the Class Diagram meta-model in ATOM3. In Figure15, the launched meta-model listed in the left column, is MyClassDiagram instead of Entity Relationship in previews figures. Then, we can build the model by dragging the Class icon to the canvass, assigning values to the attributes of the classes and connecting two classes by inheritance or association relationship links.
Figure15 shows the PIM built in ATOM3.

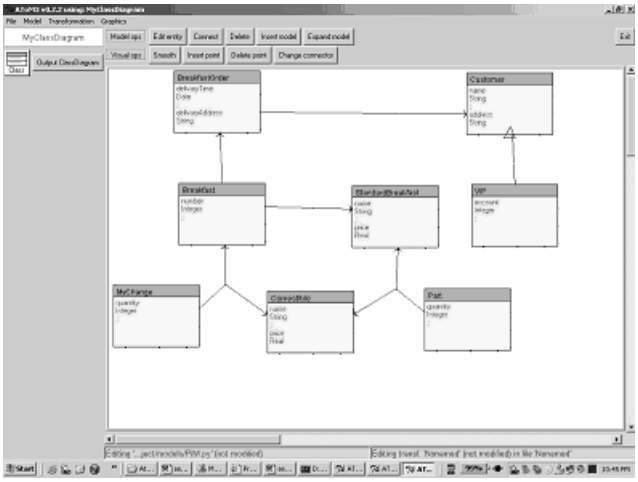


Figure15 PIM of the system

- There are eight classes in the model each is specified by its ClassName and ClassAttributes as defined by Class Diagram meta-model

- There are two association classes in the model: MyChange is the association class of Breakfast and Comestible, Part is the association class of Comestible and StandBreakfast
- VIP is inherited from Customer

5.4.4 Defining the PIM to PSM transformation rules

In our project, the mapping from PIM to PSM is a typical Object Relation Mapping. The transformation rules are quite straightforward. Some of the rules are listed below in plain English:

- Each class should be mapped to a table
- Each simple data type attribute should be mapped to a field

The rules should also includes the data type mapping (for example, a UML string should be mapped to a SQL VARCHAR), the multiplicity of the associations, the navigability, etc.
To automate this transformation process in ATOM3, we can contrive a set of graph grammar. The transformation is then implemented by specifying the source model constructs and the corresponding target model constructs as LHS and RHS of the rules.
Since Graph Transformation is a procedural definition of the transformation, the execution order of the rules is very important. The problem of concurrency should also be taken into account-which occurrence should be chosen if more than one were found to matched the LHS in a certain execution step? In AtoM3, the execution order is implemented as the priority of each rule. As to the concurrency, when more than one occurrence matched the LHS in a step, we can either choose parallel executing all the qualified occurrences, or we can manually select one.
To implement the transformation in our project, several intermediate transformation models are necessary. Our basic idea is to handle the relations between classes first by adding the navigation and inheritance information to each class as class attributes, then mapping each class to a corresponding RDBMS table.
Below are the four sequential graph transformation rules depicted in AtoM3.
Rule1: Handling the association classes
LHS specifies the situation of a class(Class 3 in the figure) being an association class of two other classes(Class 1 and Class 2). In this case, we will add the Key attribute of Class1 and Class 2 to the attribute list of Class 3. This is implemented by editing the attribute of Class 3 in the RHS. Also, in the RHS, the link between the classes disappears. This is just a mechanism to keep track of the associations having been handled

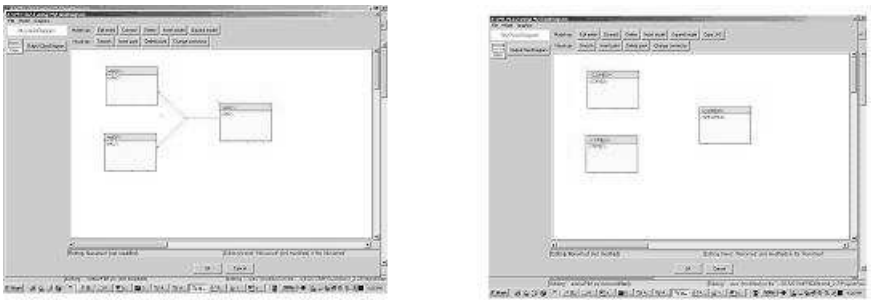


Figure16 The LHS and RHS of Rule1

Rule2: Handling the associated classes

LHS specifies the situation when one class (Class 1) is associated with another class(Class 2). In RHS, we add the Key attribute of Class2 to the attribute list of Class 1 by editing the attribute of Class 1.

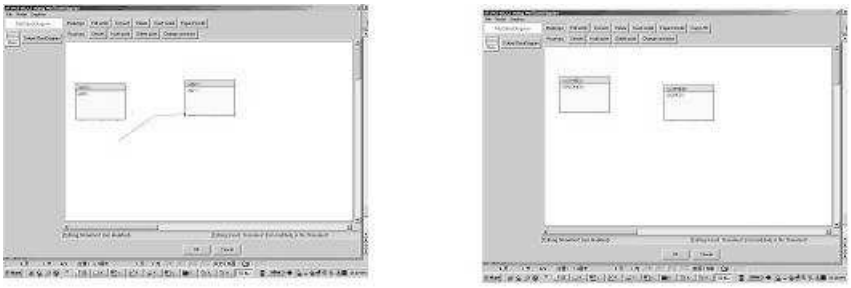


Figure17 The LHS and RHS of Rule2

Rule3. Handling inheritance between classes

LHS specifies the situation when one class (Class 1) is inherited from another class(Class 2). In RHS, we add the whole attribute list of Class2 to that of Class 1 by editing the attribute of Class 1.

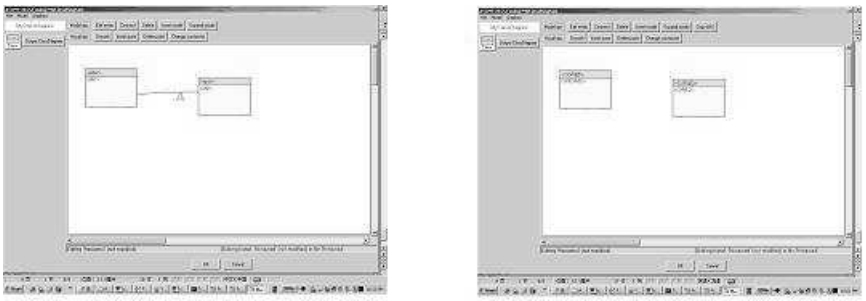


Figure18 The LHS and RHS of Rule3

Rule4. Handling mapping from classes to tables

After finishing executing Rule1-Rule3, we get an intermediate model which includes all the navigation and inheritance information in the class attribute lists. The last step is to transform each class to a table. Type mapping is also handled in this step.

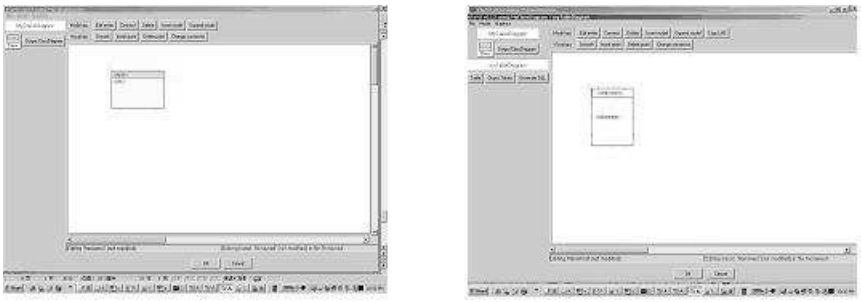


Figure19 The LHS and RHS of Rule4

LHS is a single class, in RHS, we specify the TableName and TableFields attributes of the target table by the following rules

- Map the ClassName of LHS to the TableName of RHS
- Map the attributeName of each LHS attribute to a fieldName of the RHS
- If the attributeType of a LHS attribute is String, map it to Varchar in the RHS, etc.

5.4.5 The execution of the transformation from PIM to PSM

ATOM3 provides an environment both for inputting the graph transformation rules and executing them. After launching the transformation definition illustrated in previous sub-section, ATOM3 also allows for choosing the execution style. To demonstrate the whole procedure of the model transformation from PIM to PSM , we choose to execute the rules step-by-step with sequential manual control. Figure 20 shows the transformation procedure with each intermediate model shown in the model window.

The execution steps are explained as follows:

- Step1: Executing Rule1, the two occurrences which were found matching with the LHS were Highlighted
- Step2: Manually choosing an occurrence to be executed(in this demonstration, we chose MyChange, Breakfast and Comestible)
- Step3: Executing Rule1, the left occurrence matching the LHS were replaced by RHS
- Step4: Executing Rule2, the three occurrences which were found matching with the LHS were highlighted
- Step5: Manually choosing an occurrence to be executed (in this demonstration, we chose Breakfast and StandardBreakfast)
- Step6: Executing Rule2, the two occurrences which were found matching with the LHS were highlighted

- Step7: Manually choosing an occurrence to be executed (in this demonstration, we chose Breakfast and BreakfastOrder)
- Step8: Executing Rule2, the left occurrence matching the LHS were replaced by RHS
- Step9: Executing Rule3, only one occurrence was found matching the LHS, it was executed immediately
- Step10-Step17: Executing Rule4 8 times until all the occurrences were executed.

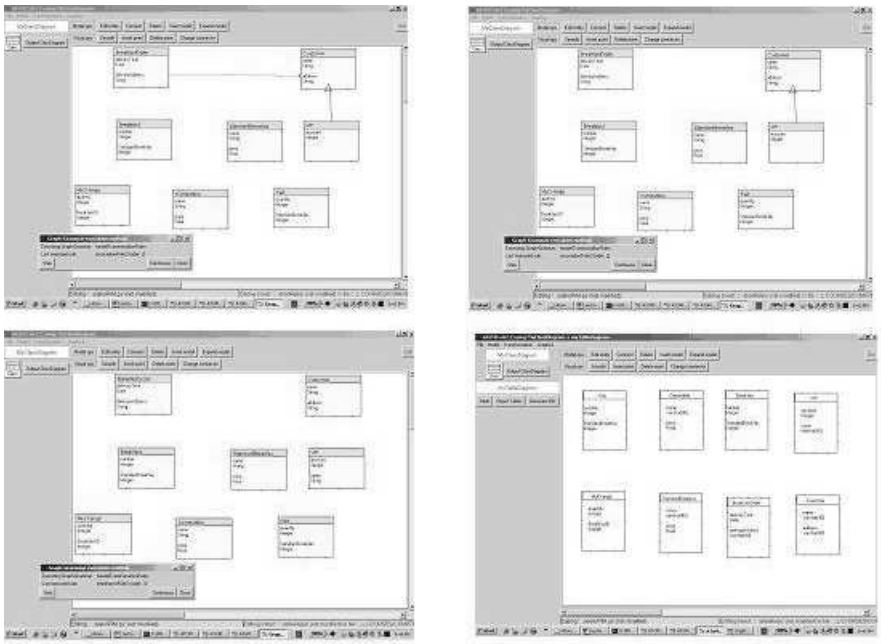


Figure20 The execution procedure of the transformation

5.4.6 Transforming PSM to Source Code

In most cases, the technology information included in PSM is elaborated. So, the transformation from the PSM to the source code is not a difficult task compared with the one from PIM to PSM. In our project, the Relational Database PSM that we get from the transformation is a set of table descriptions with elaborated information of the names and types of all the fields.
In ATOM3, all the models are stored as an Abstract Syntax Graph(ASG) in a certain data structure. Thus, in our Relational Database PSM, each table is a node in the ASG. ATOM3 also provides a mechanism allowing for user-defined manipulation on the model. By adding an action button to the meta-model with combined code, we can get access to the ASG data structure of the current model and manipulate the model data. Figure 21 shows the meta-model of our PSM viewing as a model in ER formulism. We add an entity named ”Generate SQL” to the model and then edit the ”Action” attribute of the button by binding Python code to the button.

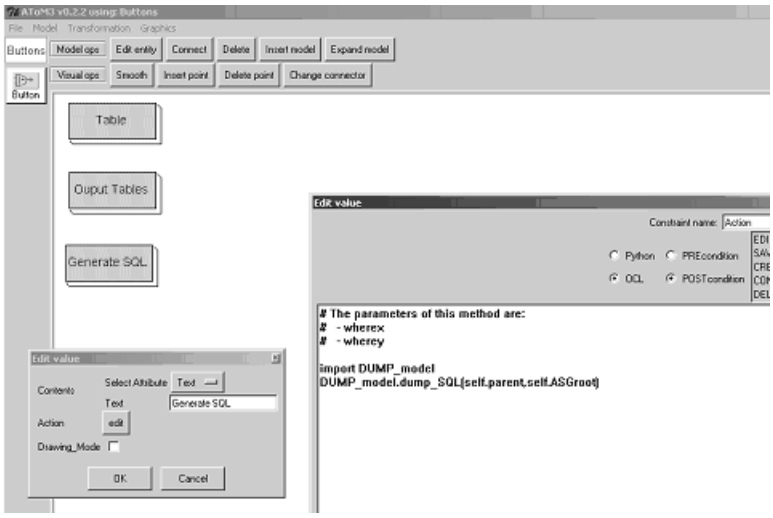


Figure21 meta-model of PSM and the editing window for ”Action” attribute

Since our intention here is to transform the PSM to SQL source code , we can write code biding to iterate over the ASG data structure and applying the following transformation rules(given in [2]).

- For each table generate a ”Drop Table if exists ” text followed by the name of the table, a ”CREATE TABLE” text followed by the name of the table, and a ” ”. Then execute rule 2, followed by rule3, and end with ”;”
- For each column in the table, generate the name of the column, followed by the name of the type and size of the column, then generate ”NOT” if the column may not have the NULL value and end with ”NULL”
- Generate a ”PRIMARY KEY ” text, followed by a comma-separated list of the names of the columns of the primary key, and end with ””

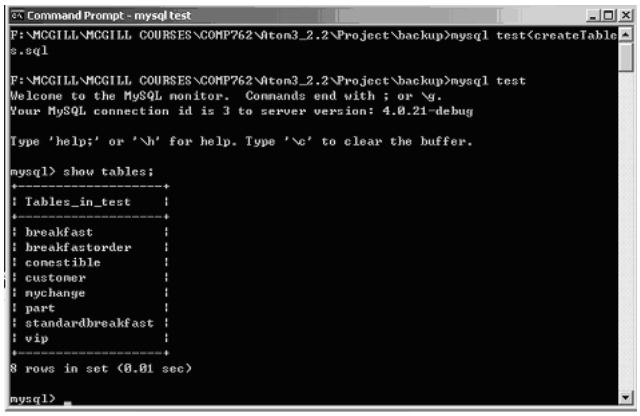
After applying these transformation rules on our PSM by clicking the ”Generating SQL” button, we got output file named createTable.sql with the following source code:

createTables.sql

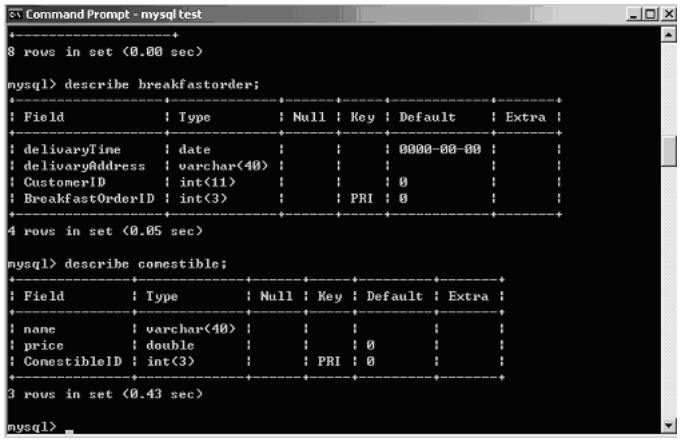
```
Drop Table if exists MyChange ;
Create Table MyChange(
quantity Integer NOT NULL,
BreakfastID Integer NOT NULL,
ComestibleID Integer NOT NULL,
MyChangeID int(3) not null PRIMARY KEY
);
Drop Table if exists Comestible ;
Create Table Comestible(
name varchar(40) NOT NULL,
price Real NOT NULL,
```

```
ComestibleID int(3) not null PRIMARY KEY
);
Drop Table if exists Breakfast ;
Create Table Breakfast(
number Integer NOT NULL,
StandardBreakfastID Integer NOT NULL,
BreakfastOrderID Integer NOT NULL,
BreakfastID int(3) not null PRIMARY KEY
);
Drop Table if exists BreakfastOrder ;
Create Table BreakfastOrder(
delivaryTime Date NOT NULL,
delivaryAddress varchar(40) NOT NULL,
CustomerID Integer NOT NULL,
BreakfastOrderID int(3) not null PRIMARY KEY
);
Drop Table if exists StandardBreakfast ;
Create Table StandardBreakfast(
name varchar(40) NOT NULL,
price Real NOT NULL,
StandardBreakfastID int(3) not null PRIMARY KEY
);
Drop Table if exists Part ;
Create Table Part(
quantity Integer NOT NULL,
StandardBreakfastID Integer NOT NULL,
ComestibleID Integer NOT NULL,
PartID int(3) not null PRIMARY KEY
);
Drop Table if exists VIP ;
Create Table VIP(
account Integer NOT NULL,
name varchar(40) NOT NULL,
address varchar(40) NOT NULL,
VIPID int(3) not null PRIMARY KEY
);
Drop Table if exists Customer ;
Create Table Customer(
name varchar(40) NOT NULL,
address varchar(40) NOT NULL,
CustomerID int(3) not null PRIMARY KEY
);
```

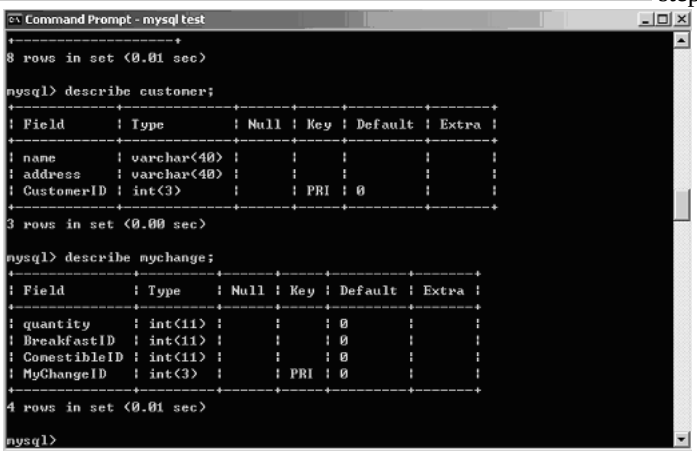
To test the effectivity of the generated source code , we can feed the createTable.sql into MySQL. Figure22 shows the interactions with the MySQL DBMS.


Command Prompt - mysql test
F:\MCGILL\MCGILL COURSES\COMP762\Atom3_2.2\Project\backup>mysql test<createTable
s.sql
F:\MCGILL\MCGILL COURSES\COMP762\Atom3_2.2\Project\backup>mysql test
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.21-debug
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> show tables;
+-----------------+
| Tables_in_test |
+-----------------+
| breakfast |
| breakfastorder |
| comestible |
| customer |
| mychange |
| part |
| standardbreakfast |
| vip |
+-----------------+
8 rows in set (0.01 sec)
mysql>

step1


Command Prompt - mysql test
8 rows in set (0.00 sec)
mysql> describe breakfastorder;
+-----------------+-------------+------+-----+-------------+-------+
| Field | Type | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+-------------+-------+
delivaryTime	date			0000-00-00	
delivaryAddress	varchar(40)				
CustomerID	int(11)			0	
BreakfastOrderID	int(3)		PRI	0	
+-----------------+-------------+------+-----+-------------+-------+
4 rows in set (0.05 sec)
mysql> describe comestible;
+---------------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
name	varchar(40)				
price	double			0	
ComestibleID	int(3)		PRI	0	
+---------------+-------------+------+-----+---------+-------+
3 rows in set (0.43 sec)
mysql>

step2


Command Prompt - mysql test
8 rows in set (0.01 sec)
mysql> describe customer;
+------------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
name	varchar(40)				
address	varchar(40)				
CustomerID	int(3)		PRI	0	
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
mysql> describe mychange;
+--------------+----------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+--------------+----------+------+-----+---------+-------+
quantity	int(11)			0	
BreakfastID	int(11)			0	
ComestibleID	int(11)			0	
MyChangeID	int(3)		PRI	0	
+--------------+----------+------+-----+---------+-------+
4 rows in set (0.01 sec)
mysql>

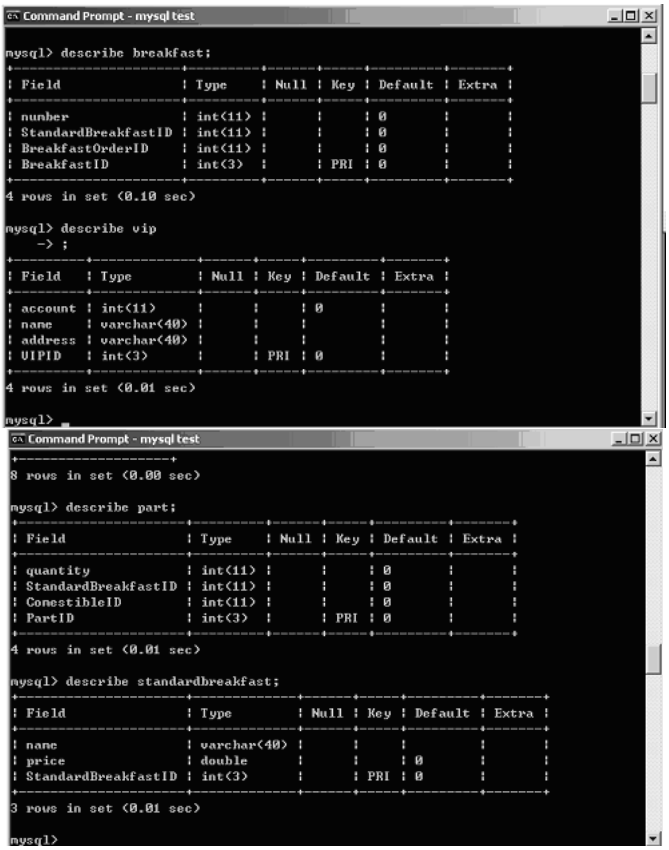


Figure22. Interactions with MySQL

6 Conclution

As the initial prospect of OMG, MDA is supposed to provide a basic technical framework for information integration and tools interoperation based on the separation of Platform Specific Models from Platform Independent Models. In this report, we looked through the basic concepts of MDA. Although theoretically perfect, some technical issues are still open for sound solutions. The fulfillment of MDA requires the mechanisms for strictly describing the models (the data in the system) and the transformations between models(the behavior of the system), so that the tools can be provided to support the automated process of model transformation. This paper looked into some technical details of these two most important issues in MDA. Meta-modeling is an extensible framework to define models. In this framework ,different formalisms can be modeled by a common meta-model, thus facilitates the generation of domain-specific formalisms and the definition of transformation between different formalisms. There have been some researches carried out to solve the problems in the current MOF meta-modeling framework. Some partly solution

have been proposed. The UML2.0 is supposed to provide an appropriate framework as the MDA's new meta-modeling standard. An equally important issue as meta-modeling in MDA is model transformation. Currently, there're no standards yet either as to how to specify the transformation or how to execute it. Two methods are mentioned in this paper-one based on the Graph transformation theory and the other based on OCL and MOF. To illustrate the process of MDA development, we give an implementation in ATOM3. This tool provides the environment for describing formalisms based on ER meta-model and using graph rewriting technology for processing model transformations. MOF2.0QVT, which is still in revision, will probably provide a standard for model transformation in MDA.

References

[1] Colin Atkinson and Thomas Kühne. The essence of multilevel metamodeling. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 2001, Proceedings*, volume 2185 of *LNCS*, pages 19–33. Springer, 2001.

[2] Colin Atkinson and Thomas Kühne. Rearchitecting the uml infrastructure. *ACM Trans. Model. Comput. Simul.*, 12(4):290–321, 2002.

[3] Luciano Baresi and Reiko Heckel. Tutorial introduction to graph transformation: A software engineering perspective. In *ICGT*, pages 402–429, 2002.

[4] Juan de Lara and Hans Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. In *FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, pages 174–188. Springer-Verlag, 2002.

[5] A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood. Transformation: The missing link of mda, 2002.

[6] Object Management Group. Meta object facility(mof) specification version 1.4, 2002.

[7] Object Management Group. Mda guide version 1.0.1, 2003.

[8] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.