CHAPTER 1

INTRODUCTION

1.1 Introduction:

The Model-driven Architecture (MDA) approach has been recognized as a methodology that can help enhance agility and speed in the implementation of enterprise IT systems.

"Companies that adopt the MDA gain the ultimate in flexibility: the ability to derive

code from a stable model as the underlying Infrastructure shifts over time. ROI flows from the reuse of application and domain models across the software lifespan".[1]

The OMG's Model Driven Architecture initiative is aimed at increasing productivity and re-use through separation of concern and abstraction.
A Platform Independent Model (PIM) is an abstract model which contains enough information to drive one or more Platform Specific Models (PSM). Possible PSM artifacts may include source code, DDL, configuration files, XML and other output specific to the target platform.
MDA has the capability to define transformations that map from PIMs to PSMs. This facilitates the development of a system in abstraction, and simplifies implementation of that system across a variety of target platforms. In the model-driven architecture, QVT(Queries/Views/Transformations) is a standard for model transformation defined by the Object Management Group. Query/View/Transformation Specification The QVT specification has a hybrid declarative/imperative nature ,with the declarative part being split into a two-level architecture.

1*[Dr. Richard Soley, OMG Chairman and CEO]

1.2 Problem:

In this new scheme of writing applications using UML and UML_based language most of the so far effort was focus on how to write a program under this trend but still testing is not addressed in depth. The specific question addressed by this thesis is How to execute test cases using suitable Test Execution Approach?

Although there are a number of attempts we believe this attempt would be unique in terms of ideal interpretation of MDA which concerns with three ultimate goals : re-use, portability, and interoperability.

1.3 Objectives

1- Generate Test cases from a model using Some Coverage criteria.

2- Execute Test cases using suitable Test Execution Approach.

3- Standardize Test cases Generation from a model.

4- Evaluate the proposed testing model using case study.

1.4 Importance of research:

Enrich the MDA community with testing Execution experience regarding tools and coverage criteria.

1.5 Scope:

The approach we follow can be treated as black box testing because we start from requirements like PIM. White box testing is not under this scope. How to generate test cases not the main point in this thesis.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter is about the state-of-the-art in this thesis. It defines the Testing in software Engineering, Test cases, how to Generate Test cases, the coverage Criteria, The MBT, MDA and Related work.

2.2Testing in software Engineering

Testing is one of the most important parts of software life cycle. Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. (Cem Kaner, Florida Institute of Technology November 2006). According to IEEE(Ref IEEE 83a) Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements.

Testing reduces the cost of development by saving rework effort and increased customer satisfaction by insure that system is quality enough or not before delivering process. The art of Testing is about designing Test cases.

The role of software Testing is:

- Verification: Are we doing things right? (this is process).
- Validation: Are we doing the right things? (This is requirement). According to IEEE 1059, Guide for Software Verification and Validation Plans: "Software Verification and Validation (V&V) is a disciplined approach to assessing software products throughout the product life cycle. A V&V effort strives to ensure that quality is built into the software and that the software satisfies user requirements. V&V provides software management with insights into the state of the software project and products, allowing for timely change in the products or in the development and support processes."(Arunkumar Khannur,2011)

To validate any system we need to generate Test cases from system requirements.

2.3 Test cases

A test case is a set of conditions generate from system requirement to verify the expected output or expected results against actual output or actual results.

A good test suite is the one that build with an expert knowledge of the system under test (SUT).

By executing the Test cases in the system environment you can compare the actual output and the expected output. So this kind of testing is called dynamic testing and it's the common one.

2.4 How to Generate Test cases

Test case design has always been fundamental to the testing process. Test case generation has always been fundamental to the testing process. It has been recognized since automation of testing like in keyword-based testing, …etc

Generation Test cases is the process of generating test suites from model developed from system requirements , these are like Platform in-depended Model or Platform Specific Model (PSM Test cases) in the MDA applications .They are high level models.

We need Test case Design process to cover all system component and ensure the system is whole tested. Generation process can be manual or automated by Test generation tools.

When you generate test cases from system requirement you ensure or validate the system requirement and verify it's Implemented correctly.

This validation activity is automated in MBT tools through the concept of traceability. A traceability is about finding a relationship between Test cases and requirements after indentified.

2.5 Coverage criteria

To measure what percentage of code has been exercised by a test suite, one or more coverage criteria are used. Coverage criteria are usually defined as a rule or requirement, which test suite needs to satisfy.

Paul Ammann, Jeff Offutt (2013).

We need Coverage Criteria to:

1. Measuring the adequacy of a test suite:

When statement coverage(or other coverage metrics) of the test run as an indicator of the quality of the test suite, if its only 40 percent that means 60 percent of the SUT statement were never executed by the test suite. Then the test suite is not adequate and more tests must be designed.

2. Deciding to stop testing:

By putting some conditions for example when statement coverage reaches 80 percent decide to stop test.

Mark Utting , (2007)

2.5.1 Association-End Multiplicity(AEM) Coverage criteria

It's example of coverage criteria more suited to UML-based models specifies the instances of a class at the end of association relation. For example in a Student Information system association in Figure 5 shows that a single instance of the Department class can be associated with one or many instances of the Student class. An AEM criterion determines the set of representative multiplicity tuples that must be created during a test. The multiplicity domain is partitioned into equivalence classes, and one value from each class is selected for the set of representative

multiplicities. The default partitioning approach is illustrated using the Student Information system association shown in Figure 2.1.

- Create a multiplicity set by arbitrarily selecting a value from each partition.

A possible multiplicity set for class Student is {0, u, n} and one for Course is {0, v, m}, where m and n are upper limits determined by the tester, u is a value from the set {1,..., n-1} and v is a value from the set {1,…m-1}.

- Create a set of configurations {(r,s)1, (r,s)2,...} from the Cartesian product of the multiple sets of the Student and Course instances, where r is the number of Student instances linked with s instances of Course. For the association, this configurations set is:

{(0,0),(0, v),(0, m),(u,0),(u,v),(u, m),(n,0),(n,v),(n, m)}.

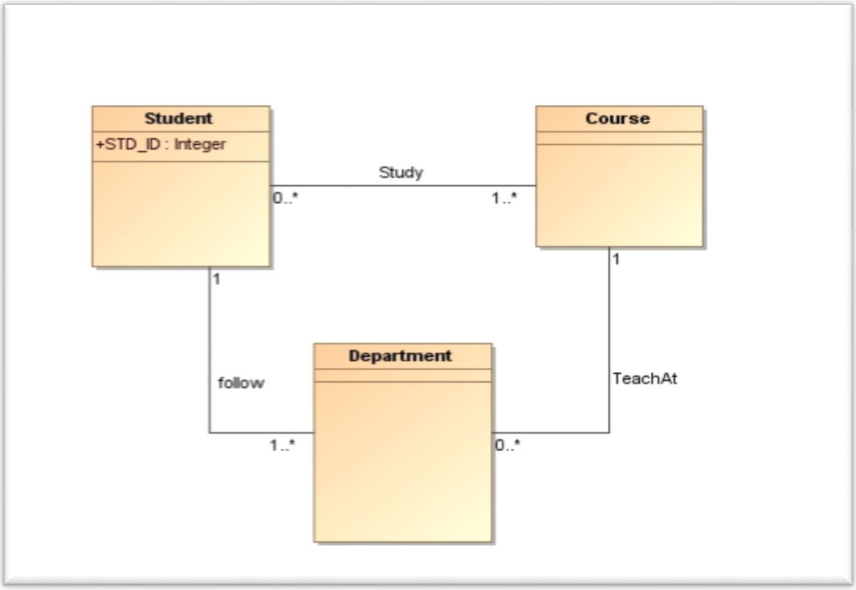


Figure [2.1] Student Information system

Figure 2.1 show the sample Student Information system which content Student class, Course class and department class and relationship between them.

2.6 Model Based Testing(MBT)

Tests are generated from a model, which is derived from the requirements documentation. It is a Black-box testing. The following are the four main approaches known as model-based testing:

1. Generation of test input data from a domain model.

The model is the information about the domains of input values. The automatic generation of test inputs is obviously of great practical importance, but is does not solve the complete test design problems because it does not help us to know whether attest has pass or failed.

2. Generation of test cases from an environment model.

Form these environment models it is possible to generate sequences of calls to the SUT, however like the previous approach, the generated sequences do not specify the expected outputs of the SUT.

3. Generation of test cases with oracles from a behavior model.

This is obviously a more challenging task then generating test input data or test sequences that call the SUT but do not check the results. To generate tests with oracles, the test generator must know enough about the expected behavior of the SUT to be able to predict or check the SUT output values. The advantage of this approach is that it is the only one of the four that addresses the whole test design problems from choosing input values and generating sequences of operation calls to generating executable test cases that include verdict information.

4. Generation of test scripts from abstract tests.

Focus on transformation the abstract test case into a low-level test script that is executable. With this approach the model is the information about the structure and API of the SUT and the details of how to transform a high-level call into executable

test scripts. “Mark Utting ,(2007),Practical Model-Based Testing: A Tools Approach, San Francisco: Morgan Kaufmann,Elsevier Inc”.

Here in this thesis we used the third one Generation of test cases with oracles from a behavior model.

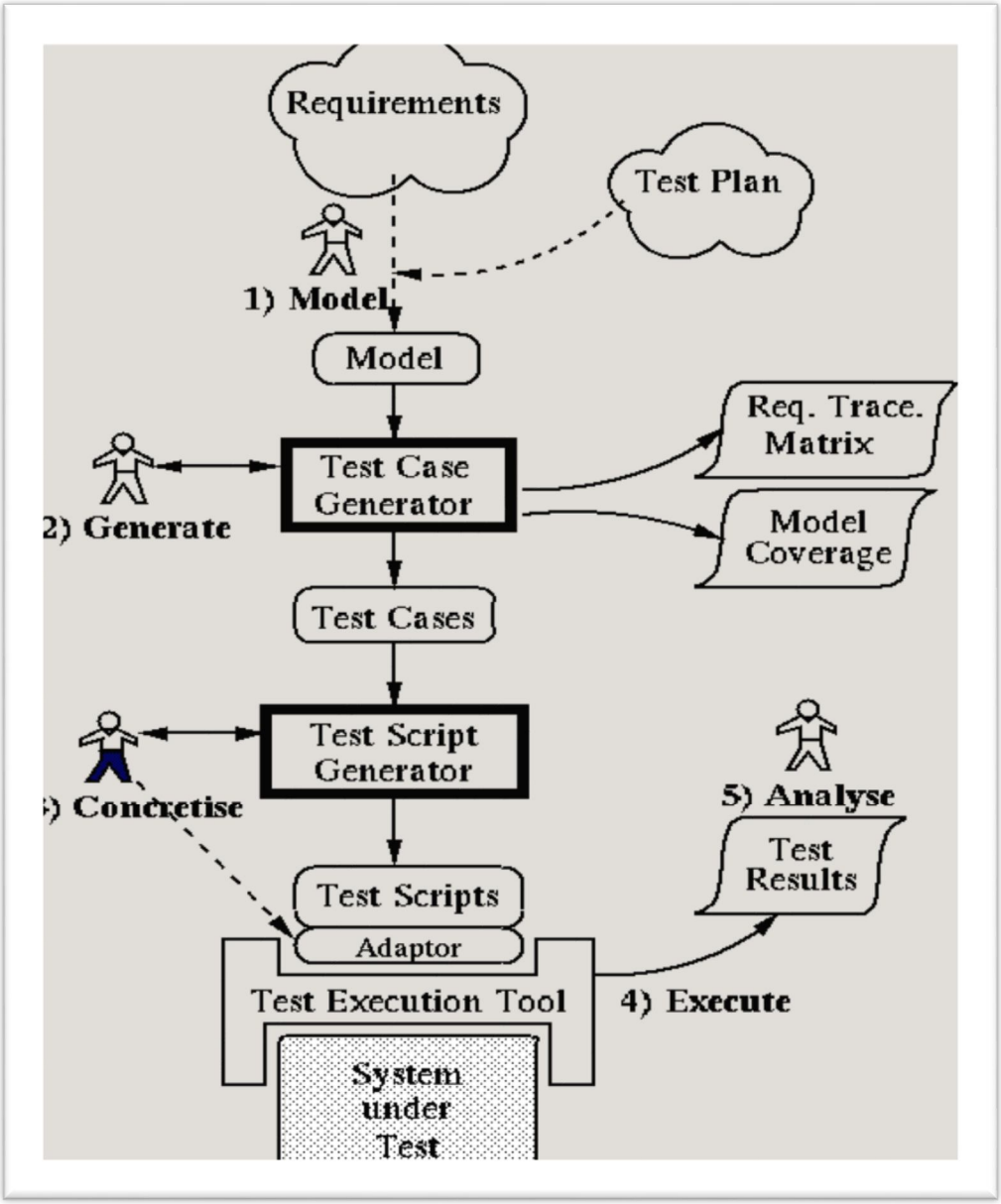


Figure [2.2] Model Based Testing steps.Mark Utting ,(2007)

Figure 2.2 explains an abstract model of SUT written by the test designer from system requirements, Test plan. And he adds the results from the model to test case

generator to generate a set of test cases from the model. Transform the test cases to test scripts that is the concretize step do by the programmer. In the execution step the executions of the test scripts doing by test execution tool after adapter code written by the programmer too. Then get results to analyze the whole system test doing by test designer.

2.7 Model Driven Architecture (MDA)

MDA is an OMG standard development method. This initiative is intended to develop applications in established domains without entirely writing any new program code (AHMED, 2010). The main idea of MDA is to raise the level of abstractions in software engineering to develop complex applications in simpler ways (Vallecillo-Moreno, 2004).

2.8 Platform Independent Model (PIM)

PIM is a model that describes the requirements of the system. PIM captures a viewpoint of the system (Chirs Raistrick, 2004). PIM is a description of a software system at a high level of abstraction (Anne Klepper, 2003). We use UML modeling language such as class model for structure part. It is the concept of business layer in this model such as banking transactions application where one has account and can do many transactions.

See Figure 2.3.

2.9 Platform Specific Model (PSM)

PSM is a model that describes specific platform specification. The PSM contains what is expressed in the PIM with the added concern about the platform details it is intended for (Chirs Raistrick, 2004). It is expressed using UML also.

The software components of java beans is an example of PSM than can implement the banking PIM as we can see in Figure 2.3

2.10 Mapping PIM to PSM

Mapping is set of rules or information for transform the PIM to specific platform (PSM).

An MDA mapping provides specification for transformation of PIM into a PSM regarding a particular platform. the platform model determines the nature of the mapping.

Transformation is a process of converting PIM to PSM or modifying PIM to generate new model. OMG has adopted standard mapping rules called QVT.

QVT (Query/View/Transformation) is a standard set of languages for model transformation defined by the Object Management Group.(Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) (pdf). Object Management Group. Retrieved 9 May 2011.).

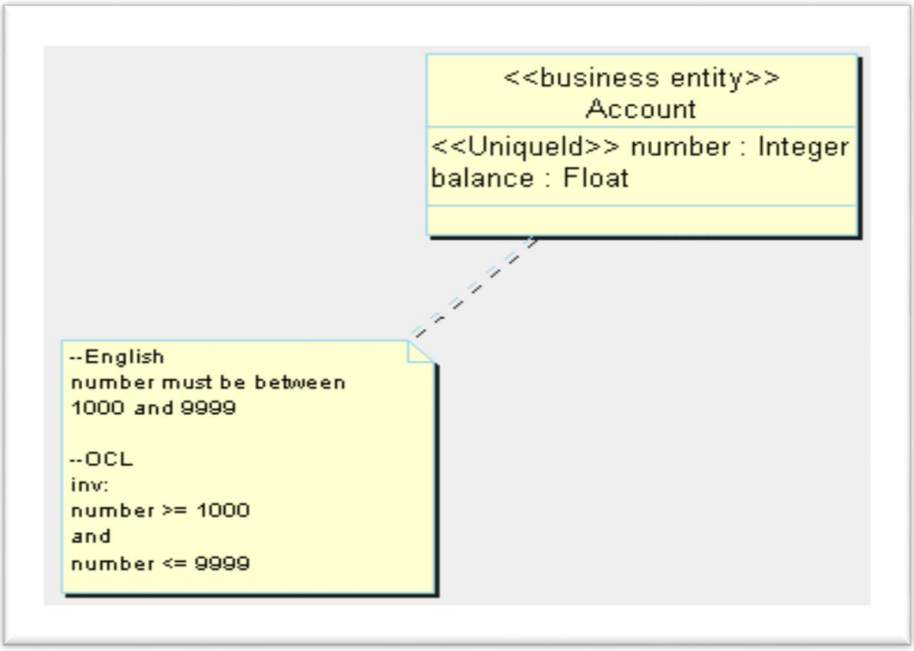


Figure [2.3] PIM Example

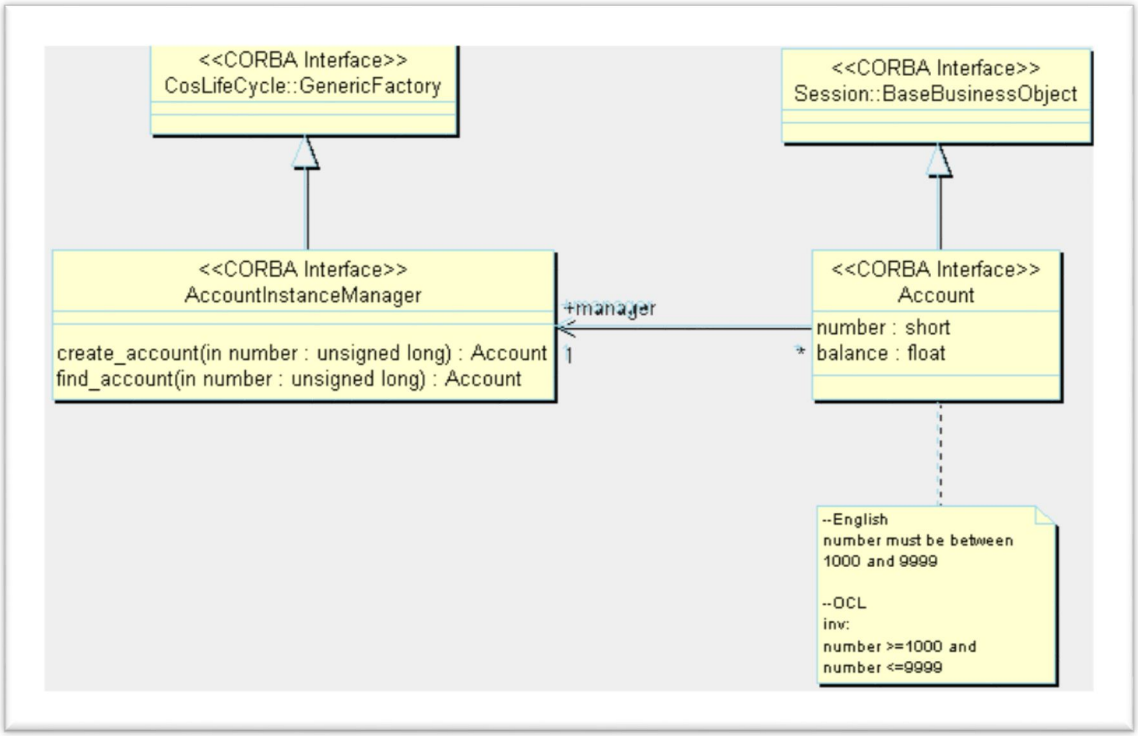


Figure [2.4] PSM Example

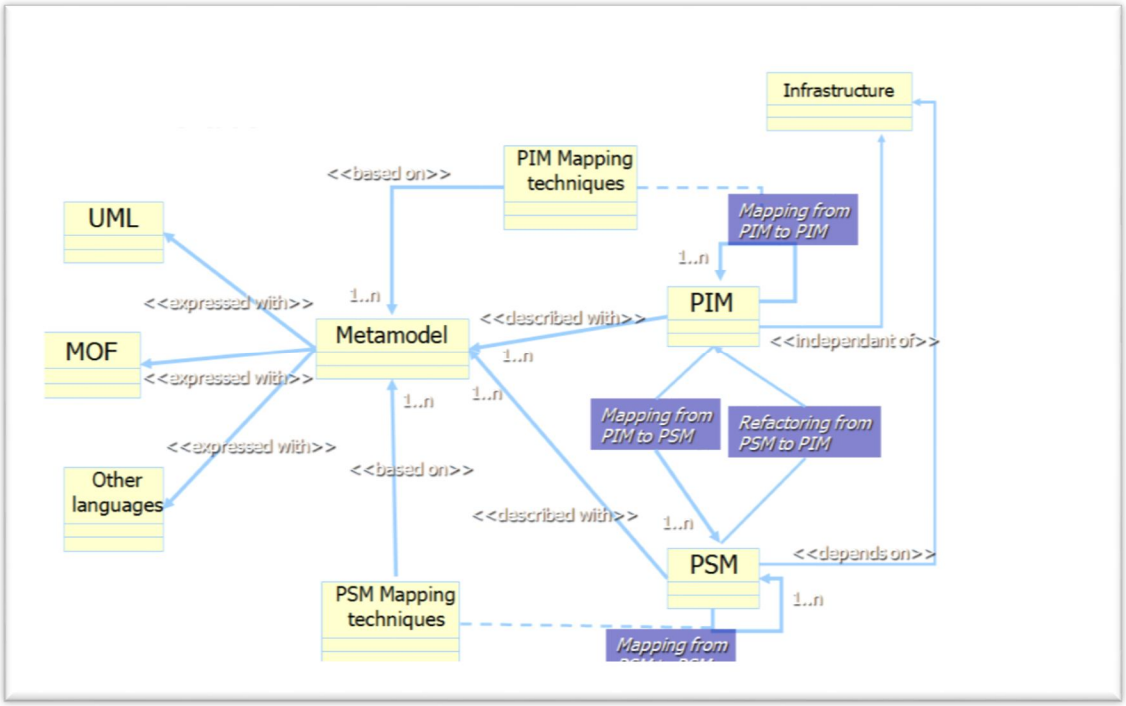


Figure [2.5] MDA Meta-model

2.11 Tests Execution

Having used model-based testing to generate a nice test suite, we almost Always want to automate the execution of that test suite. Automating the execution of the generated tests so the whole Test process is automated has great benefits such as we can execute more tests, reduce our test execution costs, reduce the overall testing time (Mark, 2007).

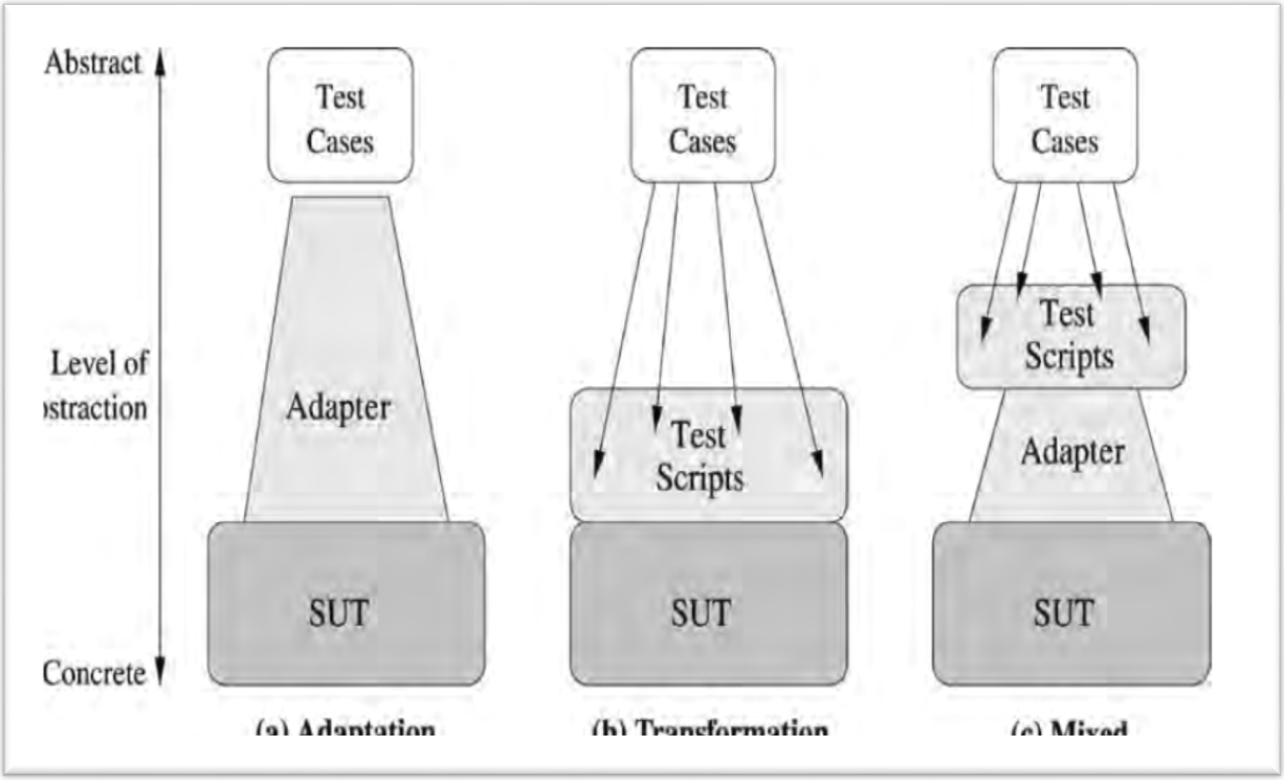
There are two kinds of testing: Online testing means the generation and execution are done in parallel. Offline testing is after generation of tests the execution is done.

There are three approaches to bridging the semantic gap between abstract tests and the concrete SUT that we can check them against the model. If the model is deterministic, we can use either approach; but if it is nondeterministic, then the adapter approach is better.

2.11.1 Approaches to mapping tests

In the adaptation approach write manual code act like bridges between the SUT and test cases.

Transformation approach transforms the tests into concrete test scripts. Mixed approach is combination of the above two approaches .



Figure[2.6] Test Execution Approaches. Mark(2007)

2.11.2 The Adaptation Approach

Writing some adaptation code around SUT and acts as interpreter for the abstract operation calls of model, in the adaptation approach the adapter code has some tasks:

- Setup: preparing SUT to be ready to testing.
- Concretization: translate any call from model-level abstract operation into concrete SUT calls.

- Abstraction: get the SUT results from concrete calls and translate them back into values. Pass them back to the model for comparison process with the expected results.

2.11.3 The Transformation Approach

Transform each abstract test into executable test script.
The transformation process may be performed by a single-purpose translation program that always produces output in a particular language or by amore generic engine that is capable of transforming abstract tests into several languages because it is parameterized by various templates and mappings for each output language.
"mark utting,(2007),Model based Testing practical approach"

2.11.4 The mixed approach

Is a combination of the two approaches. After do the adapter code around the SUT transformation the abstract test into more concrete form. One of the benefits of the mixed approach is the transformation can be easier. And the adapter can be less model specific.

2.11.5 The comparisons among Execution approaches

Adaptation approach is better for online testing because online testing requires a tightly integrated, two-way connection between the model-based testing tool and the SUT.

2.12 Related Work

1. Z. Javed, P. A. (2007) proposed a method that generates test cases from the platform-independent model of an application using MDA tools. We devised two sets of transformations: horizontal transformations using Tefkat and vertical transformations using MOF Script. We have implemented a prototype tool for generating test cases from sequences of method calls to realize the method. During execution of the test cases, the return values of methods are checked and the method invocation chain is monitored using a tracing tool. Currently, we have two versions of its implementation that are for JUnit and SUnit, but the proposed method is general and can be used to generate test cases for any other xUnit testing framework. The method was applied to an example (ATM Simulation).

2. Mohamed M, Samir O, Waseem Al S, A H.(2009) review 15 testing approaches that focus on generated test cases from software models. Use various criteria for classifying these approaches such as the used modeling language, the automated aspect of the approach, the testing target, tool support, etc. built a comparison matrix to provide a clear view of the studied papers.

   The results of this paper can be used by software engineers to select a testing approach that best fit their needs. It can also be used by researchers in the field as a reference work that can help them build upon existing approaches.

3. APARNA VIJAYA.(2009) This thesis investigates the various approaches that can be used for automatic test case generation from the behavioral model. The advantages with these new approaches are that it gives a better overview of test cases, better coverage of the model and it helps in finding errors or contradictions in minimum time. Two different frameworks/methods were proposed for model based test case generation based on UML representations. These UML models are built in the bridge point modeling tool. The UML diagrams that are being considered here is class diagrams and sequence diagrams. Various transformation rules are applied on these models to produce the state diagrams from the XMI snippets. Further TTCN 3 is used to generate the test cases and oracles from them based on different test and coverage techniques. These frameworks support a systematic testing process, which makes it easier to choose test cases, trace test executions, and analyze test results. Analyses reveal that this is a better way to trace the various deficiencies in the models since there are possibilities that it covers every aspect of the system (both static and dynamic, if class diagrams could include pre and post conditions). Moreover this underlying framework for translation and test generation is very flexible and easily extendable such that it can be customized for further enhancements.

CHAPTER 3

The Proposed Test Execution Model

3.1 Introduction

In this Chapter we present the execution approach for MDA applications. A case study is taken from OMG standard document with a Platform Independent Model (PIM) and Platform Specific Model (PSM) of the system of mapping UML to RDBMS .It explains the main approach steps and concludes with a discussion.

3.2 Methodology

MOF has standard operation with defined semantics (CRUD), the approach will follow these standard operations will interpret each test case using these operations. Because these operations are more general so the call for native SUT API will injected or inserted.

The CRUD operation refers to all major functions that implemented in Relational Database Applications.

The Adapter is the code that bridges the gap between the SUT and test cases. SUT is the system that is to be tested and the model of SUT explains the all parts of system with expected behavior. Test cases are some conditions generated from SUT requirements to verify the expected results against actual results.

3.2.1 How to get Oracles

The oracle are information talks about the excepted output or next state helps decide on Test case pass or fail after executing it. These oracle information should be in the model of testing otherwise its not possible to gain the judgment of pass/fail so not possible to automate execution of test case and satisfy the MBT principle Since PSM has instances(PSM model isnatnces) generated after mapping it will help taking it as a repository of excepted outputs. Taking this information as oracle for test cases execution is one of the main principles of this work.

Figure 3.3 shows the test case sample generating from RDBMS system instance (Student Information system).

```
<?xml version="1.0"?>
  <Test case1 name="Test case_name">
    <Action name="Action_name" >
      <Input >
          <Input1  name=" input_name "></ Input1>
                    (or any Attrbutes)
      </Input>
       <Output name=" Output_name "></Output>
    </Action>
  </Test case1>
```

Figure [3.1] Test case file Example.

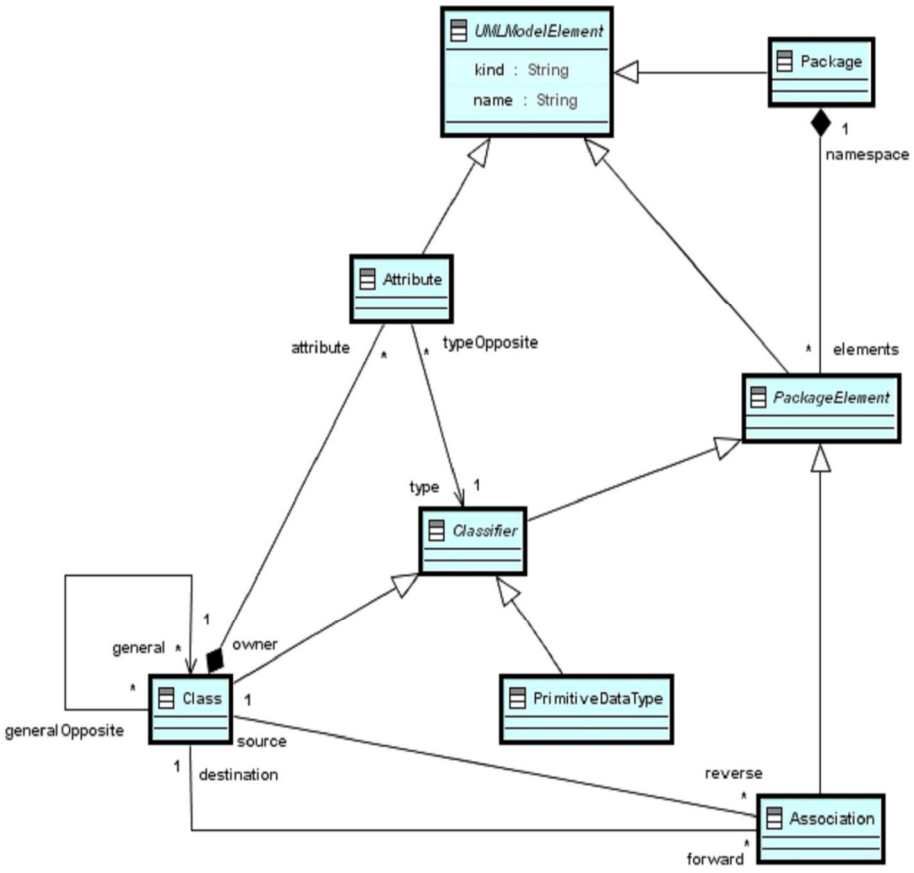
The implementation of this is from PSM after doing step 1 file PSMinstance.xmi can be created, we can read oracle of Test case from it by parsing the elements of XMI file. The alternative way is to get it from the Test case file if augmented with test cases by reading the output element values like in figure 3.2.

```
<?xml version="1.0"?>
<Test case1 name="Test case_name">
<Action name="Action_name" >
<Input >
<Input1  name="input_name "></Input1>
                    (or any Attrbutes)
</Input>
<Output name=" Output_name "></Output>
</Action>
</Test case1>
```

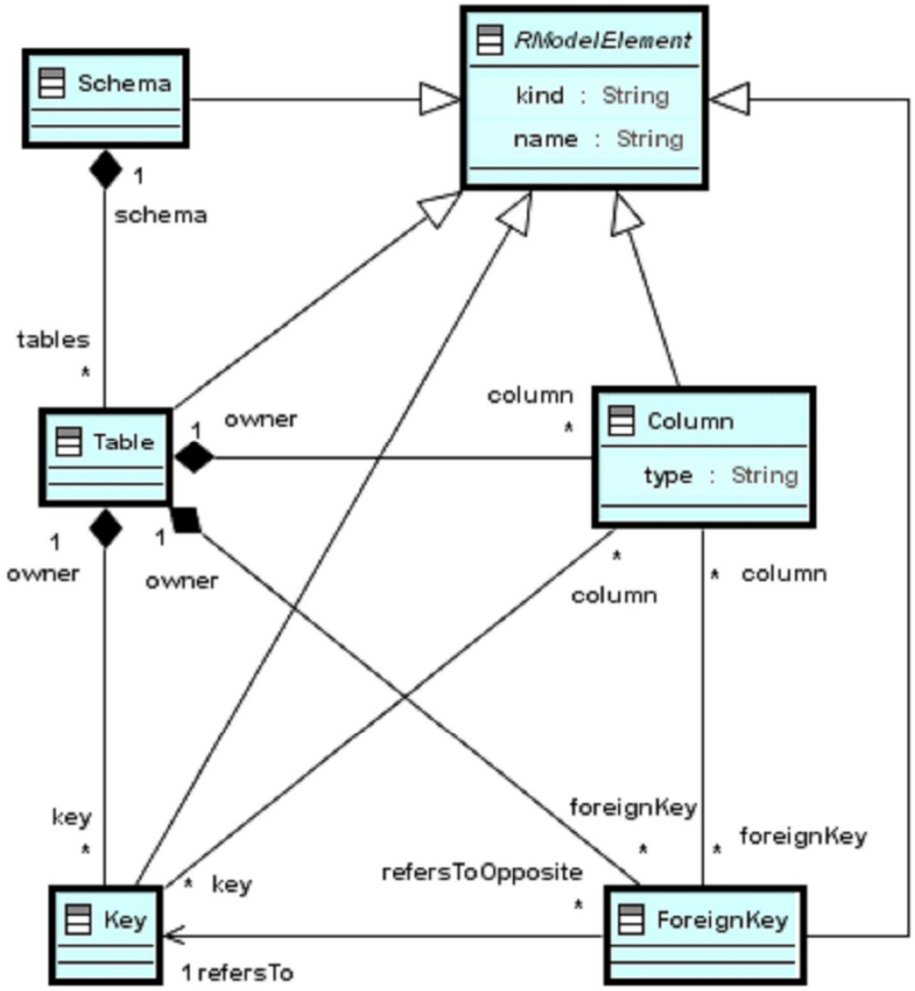
Figure[3.2] Output Element in Test case file.

Case Study: system of mapping UML to RDBMS

This system was an attempt from OMG to show one example of writing programs using this new development methodology .The basic idea is to have two meta-models: PIM for applications concepts and PSM for implementation. In this example the application is bout developing database applications. it maps persistent classes of a simple UML model to tables of simple RDBMS model. The rules are any class maps to tables, Attributes of persistent class maps to column of table, An association between two persistent classes maps to a foreign key as relationship between the tables. a PIM is developed using UML which represents the application concepts a PSM modeled RDBMS concepts of implementation using UML.

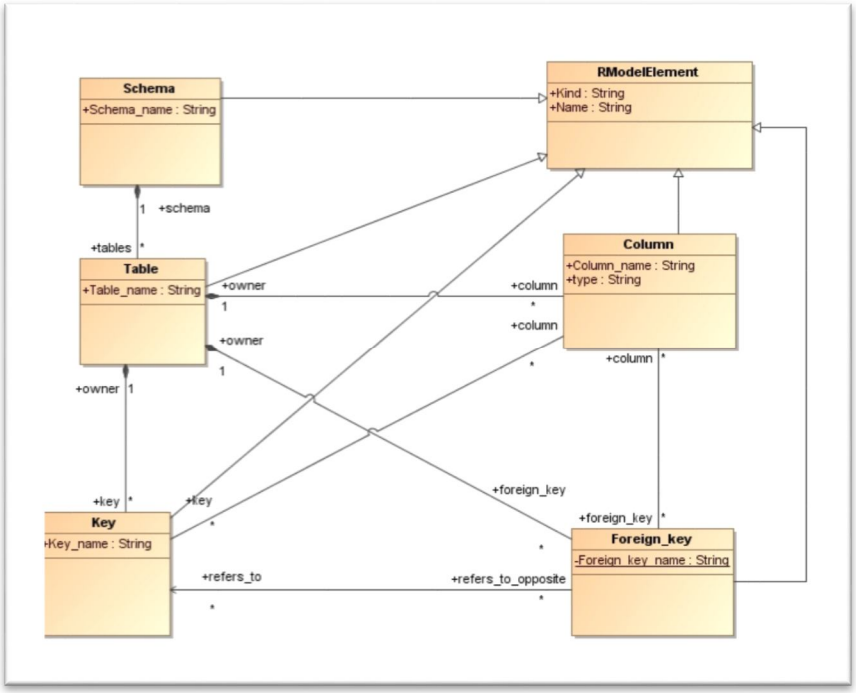


Figure[3.3] UML Meta-Model (PIM)



Figure[3.4] RDBMS Meta-Model (PSM)

3.3 Case study RDBMS system Instance (Student Information System)



Figure[3.5] Simple RDBMS Meta-Model

Figure 3.5 explains the RDBMS System model in general must have schema (name of schema), Table (name of table) ,Column(nameof column,type of column), Key (name of key), Foreign key (name of foreign key), Relations between the classes.

3.3.1 Connect PSM instance to Platform using (CRUD)

Draw the model in figure 3.5 in magic draw tool, Export to EMF UML2 (v.1 x) XMI file ,Import these files to Eclipse to generate java classes, Connect the Eclipse project with MS Access 2007 database file.(CRUD) operations implement after connection Success. For Example table.java we wont to create table in the database have the table's name like Table_name attribute in the model of RDBMS.

3.3.2 Execute Test cases using (CRUD).

These test cases generate using AEM Coverage Criteria from sample system (**College System**).

- Samples of Test cases:

✓ Table& Schema

Table Equivalence classes (0,1,2) schema equivalence classes(1).

Cartesian product

(0tab les,1 schema Reg),(1table(student),1schema Reg),(2tables(student, Course) , 1 schema Reg).

✓ Table& column

Table Equivalence classes (1) column Equivalence classes (0,1, 2).

Cartesian product

(Student,0Column),(Student,1columnSTD_ID),(Student,2column

(STD_ID,STD_NAME)).

✓Table& Key

Table Equivalence classes (1)key Equivalence classes (0,1, 2).

Cartesian product

(Student,0Key),(Student,1keySTD_ID),(Student,2key(STD_D,STD_NAME)).

✓ Key& Column

Key Equivalence classes (0,1, 2) column Equivalence classes (0,1, 2).

Cartesian product

(0key,0column),(0key,1columnSTD_ID),(0key,2column(S_ID,STD_NAME)),(1key STD_ID,0Column),(1keySTD_ID,1ColumnSTD_ID),(1key,2column(STD_ID,STD_NAME)),(2key(STD_ID,STD_NAME)0Column),(2key(STD_ID,STD_NAME),1 ColumnSTD_ID),(2key(STD_ID,STD_NAME),2Column (STD_ID,STD_NAME)).

Put this test cases in XML file format see **Figure 3.6.**

```
<?xml version="1.0"?>
  <TestCase2 name="Update Table">
    <Action  name="UpdateStudent" >
       <Input >
        <Set  name="STD_ID"  type="INTEGER" ></Set>
       </Input>
        <Output name="table Updated"></Output>
    </Action>
  </TestCase2>
```

Figure[3.6] Test case Example

Figure 3.3 Shows the update table test case which have test case name(Update Table),Action name(UpdateStudent),input Element Set as input number 1input name (STD_ID) any attribute like input type(INTEGER),Output Element name(table Updated).

Read this file by using java language Parsing XML Libraries collect the Element values and sent it to MS access database by using SQL queries.

3.3.3 How to get Oracles

The output of Test cases written in Test case file as output element. To compare the output of test case with the excepted output try to parsing the PSMinstance.xmi file that have been generated and search the output into it if is found that means test case is valid else it’s not. See Figure 3.7 .

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:Data="http:///Data.ecore">
<Data:Schema Kind="Schema" Name="Schema" Schema_name="Reg"/>
<Data:Table Kind="Table" Name="Student" Table_name="Student"/>
<Data:Table Kind="Table" Name="Course" Table_name="Course"/>
<Data:Column Kind="Column" Name="STD_ID" Column_name="STD_ID"
type="Integer"/>
<Data:Column Kind="Column" Name="STD_NAME" Column_name="STD_NAME"
type="Text(32)"/>
<Data:Column Kind="Column" Name="COURSE_ID" Column_name="COURSE_ID"
type="Integer"/>
<Data:Column Kind="Column" Name="COURSE_NAME"
Column_name="COURSE_NAME" type="Text(32)"/>
<Data:Key Key_name="STD_ID"/>
<Data:Key Key_name="COURSE_ID"/>
<Data:Foreign_key Kind="Column" Name="COURSE_ID"
Foreign_key_name="COURSE_ID"/>
</xmi:XMI>
```

Figure[3.7] PSMinstance.xmi file

3.4 Discussion

Using PSM instance as an oracle is a good methodology because we can execute tests with reusing of this artifact that is already generated during the development of PSM model instance.

On other hand, representing test cases in a standard format has an advantage of reducing the testing effort specifically for similar applications in the PSM domain. In this case our proposed execution approach does not involve change if there is a new PIM for example of student registration system or banking transaction application, Also changing in SUT with new version will not be a big problem because the effort is only one updating the PSM . Then the adapter will change according to PSM by just add the new implementation.

CHAPTER 4

CONCULUSIONS

4.1 Conclusions

This Thesis is about the Test execution Approach for MDA application using MOF operation applying on RDBMS model instance.

The results of this work are test executing approach for test cases that generated from PSM model instance using MOF operation (CRUD).This results make reading, parsing test case file easy because write it in XML format .

The future work is to build the standard queries to insure the actual output is the expected one, these standard queries may be as a services connecting with SUT or different systems under test.

4.2 References

4.2.1 Books

- ***Mark Utting, 2007. Practical Model-Based Testing: A Tools Approach. 1 Edition. Morgan Kaufmann.***
- ***Tarek Sobh,2008 Advances in Computer and Information Sciences and Engineering. 2008 Edition. Springer.***
- ***David S. Frankel, 2008. Model Driven Architecture: Applying MDA to Enterprise Computing (OMG). 1 Edition. Wiley.***
- ***Arunkumar Khannur, 2014. STRUCTURED SOFTWARE TESTING: The Discipline of Discovering. Edition. PartridgeIndia***

4.2.2 Websites

- ***idt.mdh. 2009. TR0964.8.pdf. [ONLINE] Available at: https://www.idt.mdh.se. [Accessed 21 July 2016].***
- ***methodsandtools. 2003. the Model Driven Architecture (MDA). [ONLINE] Available at:http://www.methodsandtools.com/. [Accessed 22 June 2016].***
- ***erts2016. 2016. paper_2. [ONLINE] Available at: https://www.erts2016.org. [Accessed 18 July 2016].***
- ***hhu. 2016. Test_Case_Generation. [ONLINE]Availableat: https://www3.hhu.de.***

[Accessed 31 March 2016].

- ***ieee. 2006. Coverage Metrics to Measure Adequacy of Black-Box Test Suites. [ONLINE] Available at:http://ieeexplore.ieee.org. [Accessed 9 June 2016].***
- ***ida. 2010. TDDD04-2010-LC10.pdf. [ONLINE]Availableat: http://www.ida.liu.se. [Accessed 29 June 2016].***