CHAPTER 1

INTRODUCTION

1.0 Introduction

This chapter is an introductory chapter to the research thesis. It presents background of software testing, Model based testing. Also it states the problem and explores the objective of the research.

1.1 Background

Software testing is an important technique for assessing the quality of software product. Software testing is the process of analyzing software item to detect the difference between existing and required condition (that is, bugs) and to evaluate the features of the software item.

Traditionally the testing process is based on manual work. Manual testing is an expensive, time consuming. Moreover, testing should be repeated each time a system is modified. Hence testing would be an ideal candidate for automation. Consequently there are many test tools available nowadays. Most of these tools support the test execution process. (Jan Tretmans, 2002)

Automating software testing can save significant amount of money, and save as high as 80% of manual testing effort have been achieved and produce better quality software more quickly than would have been possible by manual testing. (Mark Fewster, 1994)

Model Base Testing (MBT) is the automatic generation of software test procedures, using models of system requirements and behavior.
Model base testing process involves many steps: model the system under test (SUT), generate abstract tests from the model and then execute the tests on the SUT. The steps of modeling and generation tests are distinguish model based testing from other kinds of testing, in online model-based testing tools, generate abstract tests and execute them are usually merged into one step, whereas in offline model based testing, they are usually separate. (Mark Utting) this research focused on offline model based testing.
Model based testing helps to reduce testing effort while increasing test quality.

An automate test execution requires the generation of test scripts.
An automated testing helps in shorten the development cycles, avoid cumbersome repetitive tasks and help improve software quality. Once the test suit is automated, no human intervention is required. (guru99.com)

The test script may be written in some standards programming or scripting language or in special testing language. In execution test, doing the same for number of test cases will result in one script for each test case, this requires more cost, time and effort. Test scripts are a necessary part of test automation. (Mark Fewster, 1994)

Model Driven Architecture (MDA) is defined and supported by the Object Management Group (OMG). This process uses Unified Modeling Language (UML) as the main development language. MDA is aimed at increasing productivity and re-use through separation of concern and raising abstraction. A Platform Independent Model (PIM) is an abstract model which describes the application concepts while Platform Specific Model (PSM) is an implementation level. MDA has the capability to define transformations that map from PIMs to PSMs. MDA is aimed to automate software development. To

automate this process Object Management Group (OMG) has developed a Query/ View/Transformation (QVT) tool (OMG, 2007).

QVT rules define standard way to transform source models into target models

1.2 Problem Statement:

Testing is expensive. To use a test execution tool to automate tests, you will be writing scripts. An automated test script is more expensive to write and requires more effort and time.

This research focuses on how to automate test scripts using MDA.

Using MDA aimed to reduce cost through the application life cycle, reduce development time and to improve software quality.

1.3 Objectives:

There are some objectives of this research

- Developing PIM metamodel tests
- Developing PSM metamodel for implementing of test
- Develop mapping Rules for PIM to PSM
- Evaluating the proposed solution

1.4 Thesis Organization

This layout of this thesis is organized as following:

The second Chapter discusses the concepts of Software testing and its principles, MBT and its goals and benefits, test script and its principles, metamodel, MDA and its Models, QVT language. Finally discusses many related works to this research. So this is about the state of the art in this field.

The third Chapter discusses the approach which we followed to automate generating Test script from Test case. It shows how the principles of MDA are applied developing PIM metamodel to represent Test case, and developing PSM metamodel to represent Test script, then shows the mapping rules to automate transform of PIM to PSM.

Finally, the fourth Chapter presents discussion and conclusion of this research.

CHAPTER 2

LITERATURE REVIEW

2.0 Introduction

This chapter is about the state of the art. This research draws on lot, it talks about software testing, the MBT goals and benefits, test script and its principles, metamodel, MDA, QVT, and related work.

2.1 Software testing

In any software module there are almost always software bugs and design defects, that means complete testing is infeasible.

Software testing is the process of executing a program or system to finding errors, it performs to evaluate the software item to find the difference between the input and expected output, also to evaluate the software feature. The process of the testing should be done during the development process. (Mark Fewster, 1999)

In other words, software testing is verification and validation process, verification is the process that makes sure that the product performed the required conditions at the beginning of the development phase, where the validation is the process that makes sure that the product performed the specific requirement at the end of the development phase. (Mark Utting et al, 2007)

2.1.1 Principles of Sofware Testing

There are two principles of software testing: blackbox testing and whitebox testing.

Blackbox testing(functional testing) is a testing technique whereby the design of tests is based on just the requirements or specification of the system under test, not on knowledge about the implementation of the system (Mark Utting , 2007). And Whitebox testing (structural testing) is a testing technique wherein the design of tests uses knowledge about the implementation of the system (Utting et al ,2007).

Black box testing is often used for validation, and white box testing is often used for verification.

In current software engineering cycle the design and testing activities are separated. The testing has a long history but basically faces two common problems, the maintenance problem which means changes on SUT interfaces, for example, or the requirements. And the automation (i.e. testcases) problem. Model-based testing is a trend to solve the two problems. It focus on model as first class as stated by Robert (1999) testing is about models.

2.2 Model Based Testing

Model based testing is a set of techniques and tools to automate the generating of test cases relies on a model of a system (Requirements, behavior). Also it is executing artifacts to perform software testing or system testing.

The model is usually created manually from information specifications or requirements, and then automatically test suite is generated that contain test sequences and the test oracle.

The test sequences is used to control the system under test, using it in different conditions to test it for conformance with the model, and the test oracle watch the progress of the implementation and issues a pass or fail verdict.

2.2.1 Model Based Testing Goals

- To bring the benefits of automation to an additional portion of the test cycle.
- To provide testers with more effective tools to create test cases
- Trace to requirements
- Justify risk based decisions
- Reduce cost and cycle time. (Mark Utting)

After write an abstract model of the system under test, the model based testing tool generates a set of test cases from the model.
The model based testing process divides into the following five main steps:

1. Model the system under test and/or its environment. In this step of MBT write an abstract model of th system which want to test, then use tools (automate tools) to check if the model is consistent with the desire behavior.

2. Generate abstract tests from the model. In this step use some test selection criteria to generate abstract test from the model, which are sequnces of operations. This abstract test is the main outputs of this step.

3. Concretize the abstract tests to make them executable. This step use to transform abstract tests into executable concrete tests using transformation tool or writing some adapter code that wraps around the SUT, this step aimed to remove the gab between the abstract test and the concrete SUT.

4. Execute the test on the SUT and assign verdicts. In this step execute the concrete test. There are two ways of execution , online Model-based testing and offline Model-based testing. In online MBT the test will execute during produce, that means the tool which uses will manage the process of execution and record the result. In offline MBT first generate tests then execute them and record the result.

5. Analyze the test result. This is the final step, after execute the test must analyze the result of execution and reports the failure for each test.

2.2.1 Concretization phase

This step is an important step of MBT processes it to transform the test case to test script, it involves three main approaches: **1.The Adaptation Approach** in this approach, a wrapper is adding around the SUT to lift up the SUT interface to the abstract level so that the abstract tests can be interpreted at that level.

2. The transformation approach this approach involves transformation all abstract tests into an executable test scripts by adding the necessary details and translate them into some executable languages.

3. The Mixed Approach is a combination of two previous approaches, in this approach add some adapter code around the SUT to raise its abstraction level part of the way toward the model and make testing easier, then transform the

abstract test into more concrete forms that match the adapter interface. There are some benefits of this approach that the transformation can be easier, since the levels of abstraction are closer, and the adapter can be more model-specific, which may allow it to be reused for many different models.

Generally, online testing requires the use of the adaptation approach, and offline testing may use either approach or a combination of two approaches.

2.2.2 Benefits of MBT

MBT have various benefits

1. SUT Fault Detection. Testing is aimed to finding errors in the SUT. Model-based testing find greater than or equal to the number of errors that is finding by manually design test suits, but this depends on the experience and skills of the tester, deposite this the Model-based testing is as good as or better at fault detection than manually design test.

2. Reduce Testing Cost and Time. Model-based testing takes less time and effort to write and maintain the model and to generating tests.

3. Improved Test Quality. Design process of test manually is depending on ability and skills of engineers, this makes this process not qualify to generating test. By using MBT can handle this problem, that because MBT generates test cases automate, that makes the design Process is systematic and repeatable. MBT can uses to measure quality of the test suit by using the model. Because MBT takes less time and cost, it can generate more tests than it possible to generate by manual test.

4. Requirement Defect Detection. In MBT after build abstract model of SUT can exposes the model issues in informal requirement. During generating tests if that an error or missing in requirements, the modeling phase will exposes that. Requirements problems are the major source of the system problems. Any defect detects in requirement phase that better, easy to fix and cheaper than detect later.

5. Traceability. MBT has ability to bind test case with the model and with informal requirement in process that called traceability. By using traceability can explain why test case is generated and when optimize test execution. It helps to execute just the tests that effects by any change of model. Traceability makes important Relation between informal requirement and test case which consist of three aspects, Reqs-Model traceability, Model-Test traceability, and Reqs-Test traceability which it combine the Requirement and test case. The Requirement Traceability can use as a measure of test suit quality.

6. Requirement Evolution. In manual test if the Requirements are changes, these changes requires a large amount of time and effort to update the test suit. But with MBT just update the model then generates the test, this requires less time and effort. When the Requirements or the model evolves that requires tools to analyze the different between the old Requirements and the new one. (Mark).

2.3 Test script

A test script is the most important concept in this research so it will be dealt with first in this section. A test script is a test case transformed into executable language on the SUT

"A test script is an executable version of a test case, which is usually written in a programming language, a scripting *language, or a tool-specific executable notation." (Mark Utting, 2005).* Test script is an essential part of automation testing.

"A test script is the data and/or instructions with a formal syntax use by test execution automation tool, typically held in a file. A test script can implement one or more test cases." (Mark Fewster, 1999).

Scripting can created manually. And can written in a formal language so the tool can understand, written and editing script, that makes using tool is best than people with programming knowledge.

2.3.1 Good script

Script is very flexible. And to perform a task there will usually be many ways of coding a script.

Since script form is an important part of most test automation, we should insure it is good. A good script must be easy to use, easy to maintain. Writing a good script requires more effort, there are many principles to reduce the effort in writing a good script and to achieve the reusability and increase productivity and decrease the maintenance cost.

- Annotated, to guide both the user and the maintainer.
- Functional, performing a single task, encouraging reuse.
- Structured, for ease of reading, understanding, and maintenance.
- Understandable, for ease of maintenance.
- Documented, to aid reuse and maintenance. (Mark Fewster, 1999).

2.3.2 Script techniques

There are different scripting techniques. These techniques will be used together. Each one of them has some advantages and disadvantages that affect the time and effort, it takes in implementing test cases supported by the scripts.

The scripting techniques described are:

- Linear scripts
- Structured scripts
- Shared scripts
- Data-driven scripts
- Keyword-driven scripts (Mark Fewster, 1999).

2.3.2.1 Linear Scripts

Linear script is scripting technique uses when record the whole of each test case performed manually. In this technique uses a single script to replay a test case in its entirety. Thus with more complex application and test, this process is likely to take long time. Linear scripts can records manual task and starts automating without planning. Any user can uses it not just programmer. Linear scripts good for demonstrations. These are some advantages makes the linear scripts ideal for some tasks. Linear script can use to automate any repetitive action, to automate edit to update automated tests. Linear scripts can be useful for conversation, and for demonstrations or training.

Linear scripts do have a number of disadvantages:

- To automate test needs for too long time than running it manually. And needs some maintenance effort when the SUT changes.
- There is no reuse of scripts

- Linear scripts are vulnerable to software changes
- They are expensive to change. (High maintenance cost).

2.3.2.2 Structured scripts

Structured script is the same with structured programming; it uses some control structures to control the execution of the script. These control structures are: sequence, selection and iteration, they are gives a script the ability to make a decision by using 'if statement', and ability to repeat a sequence of instructions when it requires by using 'loops'.

A good using of these control structures leads to maintainable and adaptable script that will support an effective and efficient automated testing regime. So these using requires for programming skills.

In structured scripting the script can be made more robust to test and to check the reasons of the test fail, however the script is more complex.

2.3.2.3 Shared scripts

Shared scripts are shared by more than one test case, this help to writing or recording the actions required in less time. By using this technique can start automate test with rapidly changing software, this reduce the maintenance effort and cost, and take less effort to implement the same tests. This technique is suitable for small system.

2.3.2.4 Data-driven scripts

A data-driven scripting technique uses a separate data file to store the test inputs and read this inputs from it. In this technique the same script enables to run different tests with different inputs and different outcomes, so can implement more test cases with little effort. Testers can add a new tests even has no knowledge about scripting tools, this adding can be done very quickly. This

technique requires little maintenance effort. The disadvantages of this technique are that the writing of control scripts needs programming skills, and the initial set-up needs more time more effort.

2.3.2.5 Keyword- driven scripts

A keyword- driven script is the extension of the data-driven technique, this technique uses a single control script to support a wider variation associated test cases. The implementation of automated test cases is more complex.

Specify of any action in details makes scripts very complex. The keyword-driven technique uses data-driven technique to specify automated test cases without details by using a set of keywords, this keyword interpreted by the control script.

Data-driven testing uses a set of scripts, these scripts are more generic and reusable, which reduce the maintenance problem.

2.4 Automated software testing

Automated software testing becomes very important control mechanism to ensure accuracy and stability of the software through each build.

To reduce a hard human effort in testing, can attitude automated software testing by using some existing frameworks or tools to automate some activities in software testing, such as the JUnit testing framework to write unit test inputs and their expected outputs.

There are several reasons to use automated tests:

- Speed up testing to accelerate release
- Allow testing to happen more frequently, and be done with less skill
- Reduce cost of testing by reducing manual labor
- Improve testing coverage and reliability
- Ensure consistency

- Make testing more interesting
- Develop programmer skills.

2.5 Metamodel

This concept is central to the methodology (MDA) we used for the solution of the problem so it will be explained as a second important component. A metamodel is a model of a model. It's a model that defines the language for expressing a model. Metamodel is needed to store the modeling data in form of the metadata and helps to model the system. The MDA metamodel is a data hub in the development of the system with any modeling languages. (Prabhu Shankar Kaliappan). The metamodelling technologies often use an abstract syntax. The UML metamodel is viewed as defining the language for creating a model, and the MOF as defining the language for creating metamodels (Colin Atkinson, 2002).

2.6 Model Driven Architecture (MDA)

The Object Management Group (OMG) is founded in 1989 as standards organization to help reduce complexity, lower costs and to present a new software applications, some of it's accomplishments are the Unifide Modelling language (UML), Meta Object Faciliy (MOF) and XML Metadata Interchange (XMI). These standerds helps in model driven development. Later OMG adopted a new framework called Model Driven architecture (MDA) using by OMG as approach for using model (Frank Truyan, 2006).

MDA provided a new way to use models than use traditional source code.

Model is an abstraction of a system, it can provides a simple view of the system, and can uses for planning. The most commonly used models are the UML models which uses as a programming language.

MDA has three advantages against other methodologies of software development: transferability that is connected with platform independency, interoperability that is closely related to standard development and reusability that is the result of the previous two advantages. (Martin Kardos, 2010).

MDA aimed to increase the application reuse, reduce the cost and complexity of application development, reduce the time, and improve application quality (Igor Sacevski).

2.6.1 MDA Models

The basic concepts of the MDA are the following models: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specification Model (PSM), and the transformation techniques and mapping (Frank Truyan, 2006).

2.6.1.1 Platform Independet Model(PIM)

PIM is a model with high level abstraction independent on the implementation technology, developed using many notations like UML. MDA usually has multiple levels of PIMs, these levels may different from basic to advances structural and behavioral modling.

PIM is stored in Meta Object Facility and it considered as input to the mapping step which will produce a Platform Specific Model (Prabhu Shankar Kaliappan, 2007).

2.6.1.2 Platform Specific Model (PSM)

The PSM is a technology metamodel where native APIs of the platform is modeled in abstract way. It can be also produced by the transformation from PIM. PSM is different from PIM in abstraction level where implementation

concepts appear while in PIM only application concepts like in student registration system only the concepts related to the academic business. PSM contains enough information to allow code generation. The platform model provides concepts for use in the PSM. Because we need to map PIM to PSM often using automated tools the next section is about that.

2.6.1.3 Transformation Techniques

A central aspect of MDA is the concept of model transformation, in which one model is converted into another model of the same system. A mapping is a set of rules and techniques used for this modification, a mapping tells how elements of a certain type should be transformed into elements of another type. In MDA the most typical case is transformation from PIM to PSM, using standard mappings like XMI (XML Metadata Interchange). However transformations may be used between PIMs, between PSMs, from PSM to PIM as well as from PIM to PSM. The output model of transformation may be simple code (Prabhu Shankar Kaliappan, 2007).

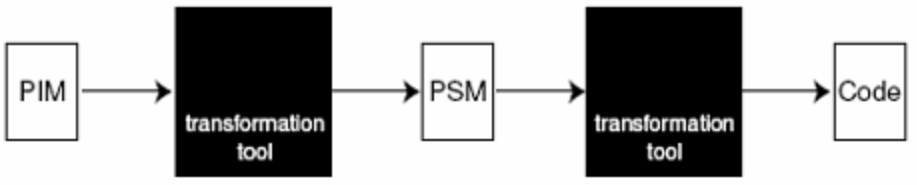


Figure 2.1: The MDA Transformation process
(Prabhu Shankar Kaliappan, 2007).

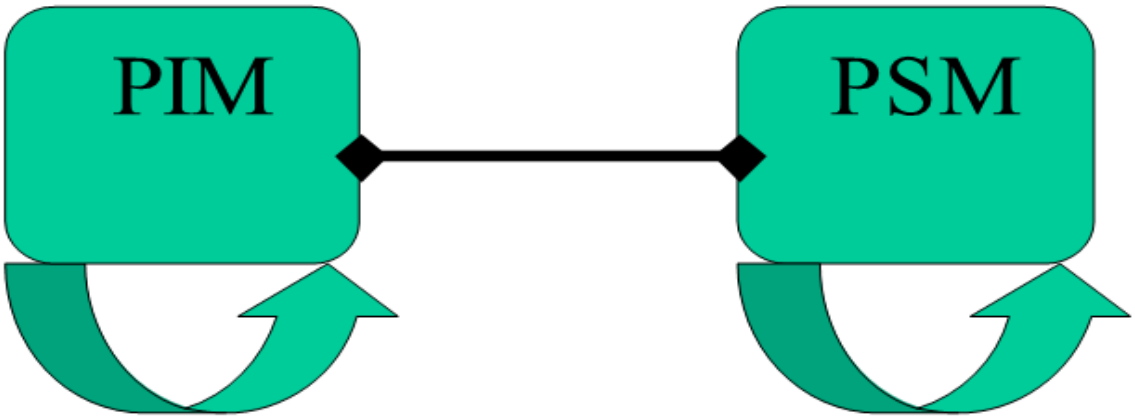


Figure 2.2: Dimensions of Transformation between PIM and PSM
(Xiuhua Zhang, 2002)

2.7 Query View Transform (QVT)

A model transformation mapping must be specified using some languages, it can be a natural language, an action language, or a dedicated mapping language. QVT is a standard for model transformations language in the MDA architecture developed by the OMG (Object Management Group). It is central to any proposed MDA. It provides a way to transform source models to target models. These source and target models must adapt to the MOF meta-model. Specifically, this means that the abstract syntax of QVT must conform to the MOF 2.0 meta-model. QVT defines three specific languages named: Relations, core and operational/mapping. These languages are organized in the layered architecture.

There is an open source tool set allow to develop projects using MDA methodology, these are like MediniQVT and EMF [www.eclipse.org/emf].

In recent years, several works on testing and test scripts have been proposed. We presented a brief overview of some best known works.

2.8 Related Works

(A.Z. Javed and et al, , 2007) proposed a method that generates test cases from the Platform-independent model of an application using MDA tools. This method is based on sequence diagrams. They devised two sets of transformations: horizontal transformations using Tefkat(PIM to PIM), and vertical transformation using MOFScript (PIM to PSM). They used MDA approach for generating unit test cases in two steps. In the first step, they modeled a sequence diagram as sequence of models calls (SMC) which is then automatically transform into a general unit test case model by applying model-to-model transformations. In the second step, model -to-text transformation are applied on the xUnit model to generate platform specific test cases that are concrete and executable. They have implemented prototype tool for generating test cases (PSM) from sequences of method (PIM). During execution of test cases, the return values of method are checked and the method invocation chain is monitored using a tracing tool.

(Fuqing Wang and etal, 2009) proposed an efficient way to transform test cases in word documents to executable programs using MDA. by using MDA they presented a more efficient way to software development by giving a higher-level abstraction with standarized model and implementing the automatic transformation among different levels of model or code. There are three phases in this proposed method according to MDA: transformation from CIM to PIM, transformation from PIM to PSM, transformation from PSM to code. By using this method they could reduced the cost of testing because in this way lots of duplicated work is diminished, and also they could inproved the testing efficiency, and reused the artifact easily. Results show that the development

time of executable test cases (Test Scripts) are considerably redused and test maintenance is simplified.

(Yang Liu and et al, 2010) proposed a methodology of automatic generation of test cases based on MDA. the process of generating test cases is that a platform- independent model is converted into a platform-independent test model through level conversation, and the platform-independent test model is converted into the corresponding test cases through vertical conversation. They have PIM model represented by UML and PIT test model represented by U2TP in the conversion from the PIM to the PIT, They make the PIM as a source model, the PIT as a target model. The conversation rules from the system model to the test model are designed using the ATL model to model conversation method . The conversation rules test model to the test cases are designed using the MOFScript model to code conversation method . then the revelant test cases are generated.

All papers mentioned in this research based on MBT and transformation concepts from model-to-model, and used the MDA as a solution approach as this research. They used the MDA approach for the easy transformation and its support for automated based on different model transformation languages. (A. Z. Javed and etal, , 2007) , (Yang Liu and et al, 2010) are focused on test cases. While (Fuqing Wang and etal, 2009) focused on test cases and test scripts like this research. This research based on automated generation of test script using QVT language. They have used different transformation tools like ATL/MOFScript, and Telfkat.

This research proposed methodology of automatic generating of test script from test case using MDA (PIM-to-PSM) to reduce cost, time and improve software quality. This generating is done by QVT transformation rules.

CHAPTER 3

AUTOMATING TEST SCRIPTS GENERATION PROCESS

3.0 Introduction

This research proposed a way to automate generating test script from test case. Test case is a sequence of SUT interactions. Test script is executable version of a test case.

This can be done by automated transform test case to test script using MDA methodology for automation by using QVT transformation rules as standard for model transformations.

3.1 Methodology Steps

The ways we followed in this research can be summarized in these steps:

- Developing PIM metamodel tests
- Developing PSM metamodel for implementing of test
- Develop mapping Rules for PIM to PSM
- Automatic transformation from PIM to PSM

3.1.1Developing PIM metamodel Tests

In this section we develop a PIM metamodel for test case [see the Figure 3.1] PIM is an abstract model which contains enough information to drive one or more Platform Specific Model (PSM).

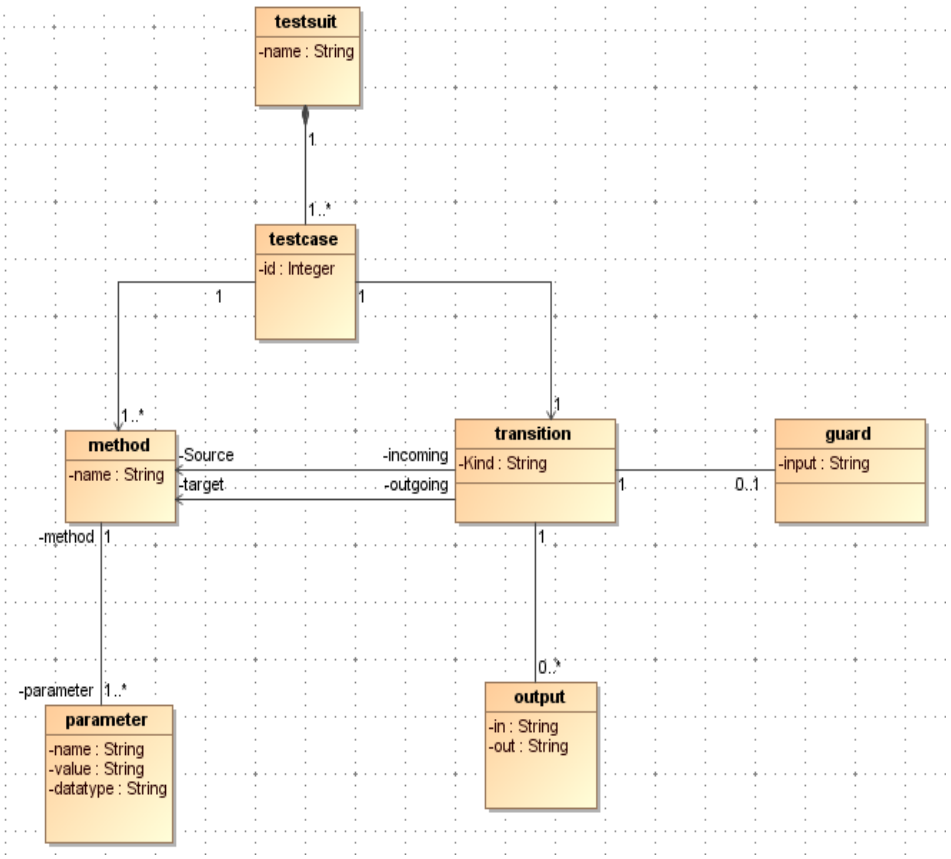


Figure: 3.1Test case Metamodel (PIM)

The PIM metamodel is expressing the test case which modeling by using UML. This figure describes the elements of the test case in this research. These elements are: *test suit*, *test case*, *method*, *transition*, *parameter*, *guard*, and *output*. The instance of *test suit* has attribute called *name* which is string type, the attribute of *test case* is *id* integer type, the instance of *test suit* is a set of test cases, the *method* instance has attribute *name* which is string type, this attribute

represent the *method* name, the *transition* instance has attribute *kind* which is string type, which represent the transition kind (*source* and *target*) and has association with *method*, their association is *source* and *target*. The instance of the *test case* has association with the instance *method* and with the instance *transition*. The instance *parameter* has three attributes *name, value* and *data type* which is string type. The name represents the parameter name and the value represents the parameter value. The instance *guard* has attribute *input* which is string type. The instance *output* has two attributes *in* and *out* which is string type. The *method* instance has association with the *parameter*. And the *transition* has association with the *guard* and *output*.

The case study is a developed an Automatic Teller Machine (ATM) specification written by [Mark] for its software which represents our SUT. The ATM will service one customer at a time. A session starts with the insertion of a customer ATM card into the card reader slot of the machine. Then the ATM reads the card. (If the reader cannot read the card to any insertion problem, the card is ejected, and displayed an error screen and the session is aborted). Then ATM asked the customer to enter a personal identification number (PIN), and then allowed to perform one or more transactions, choosing from a menu of possible types of transactions (withdrawal, deposit, transfer, inquiry) in each case. After each transaction, the ATM asked the customer if would like to perform another. The ATM must be able to provide all above services to the customer.

In withdrawal transaction a customer must be able to do a cash withdrawal from any suitable account linked to the card, in multiples of 10.00 SDG. Customer must be get approval from the bank before cash is disbursed.

A withdrawal transaction asks the customer to choose a type of account to withdrawal from (checking) a menu of possible accounts, and to choose a dollar amount from a menu of possible amounts. The system verifies that it has sufficient money on hand to satisfy the request before sending the transaction to

the bank. (If not, the customer is informed and asked to enter a different amount.) If the transaction is approved by the bank, the appropriate amount of cash is dispensed by the machine before it issues a receipt. A withdrawal transaction can be cancelled by the customer pressing the cancel key any time prior to choosing the dollar amount. (Mark Utting, 2007).

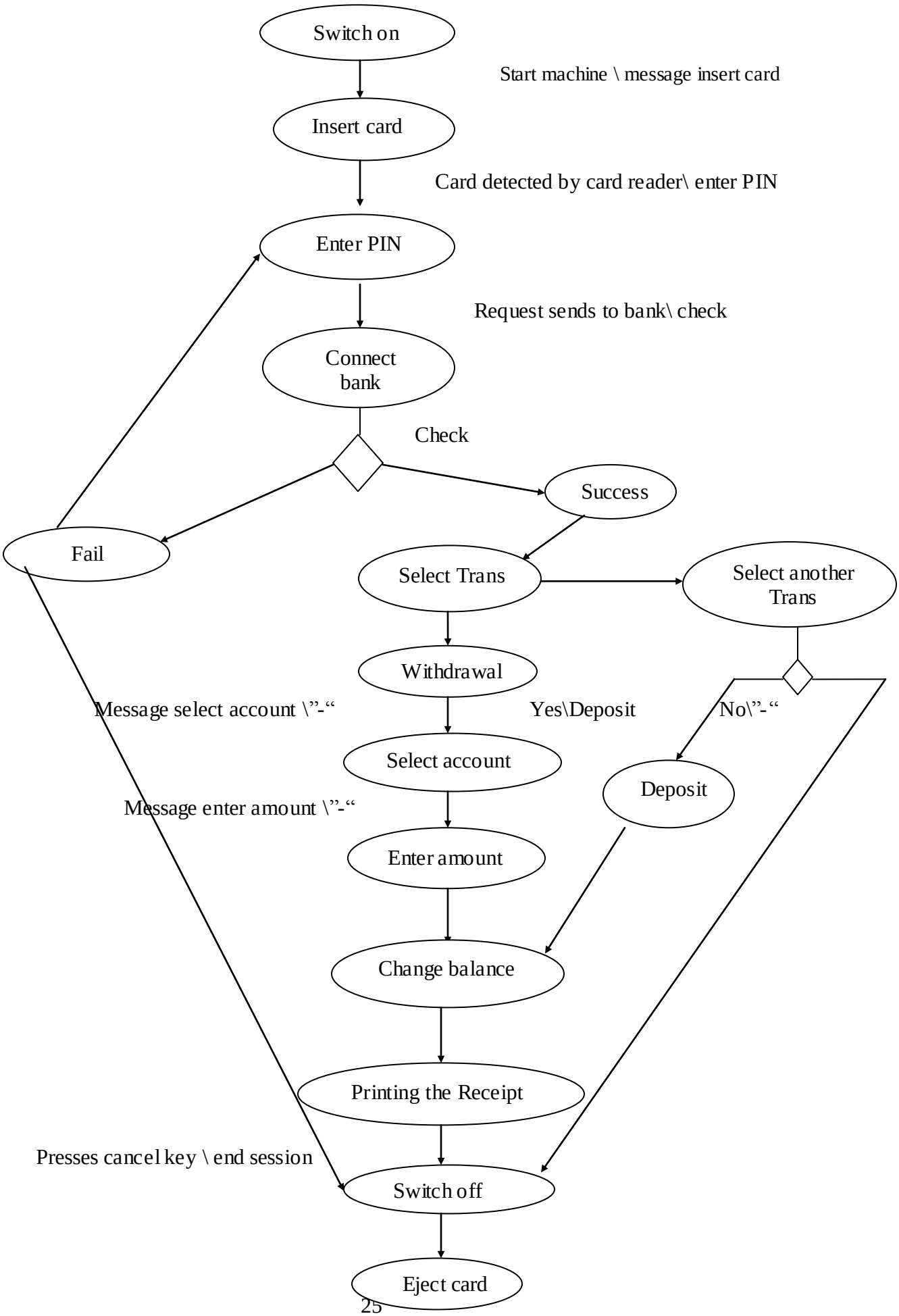
ATM Test case sample

Testing Withdrawal data gathering

switchOn (); setCash(100); custInsertCard(card1); custEnterPin(PIN_OK); custSelectTrans(WITHDRAWAL); custSelectAcct(CHECKING); custEnterAmount(20); custAnotherTrans(false); switchOff();

Withdrawal is the name of *test case*, the *switchOn* is the value of the id attribute, the methods names of *test* case are *switchOn, setCash, custInsertCard, custEnterPin, custSelectTrans, custSelectAcct, custEnterAmount, custAnotherTrans, switchOff*. The parameters for each method, switchOn and switchOff methods has no parameters, the rest parameters values and names are a *value 100*,name *card1*, *name PIN_OK*, name *CHECKING*, a value 20, name *false*. The transitions (in\out) of the *test case* are, *Start machine\ message insert card, Card detected by card reader\enter card, Request sends to bank\ check, Right PIN\ PIN_OK, Message select account\"_", Message enter amount\"_", Yes\select deposit, NO\"_", Presses cancel key\end session*. These transitions appear in the figure 3.2 of state machine diagram which contains events and transitions.

Figure 3.2 Behavioral Model of the SUT diagram


Switch on
Start machine \ message insert card
Insert card
Card detected by card reader\ enter PIN
Enter PIN
Request sends to bank\ check
Connect bank
Check
Success
Fail
Select Trans
Select another Trans
Withdrawal
Message select account \"-"
Yes\Deposit
No\"-"
Select account
Deposit
Message enter amount \"-"
Enter amount
Change balance
Printing the Receipt
Presses cancel key \ end session
Switch off
Eject card

3.1.2 Developing PSM Metamodel Tests

In this section we develop a PSM metamodel which represent the test scripts platform concepts (see the Figure 3.3).

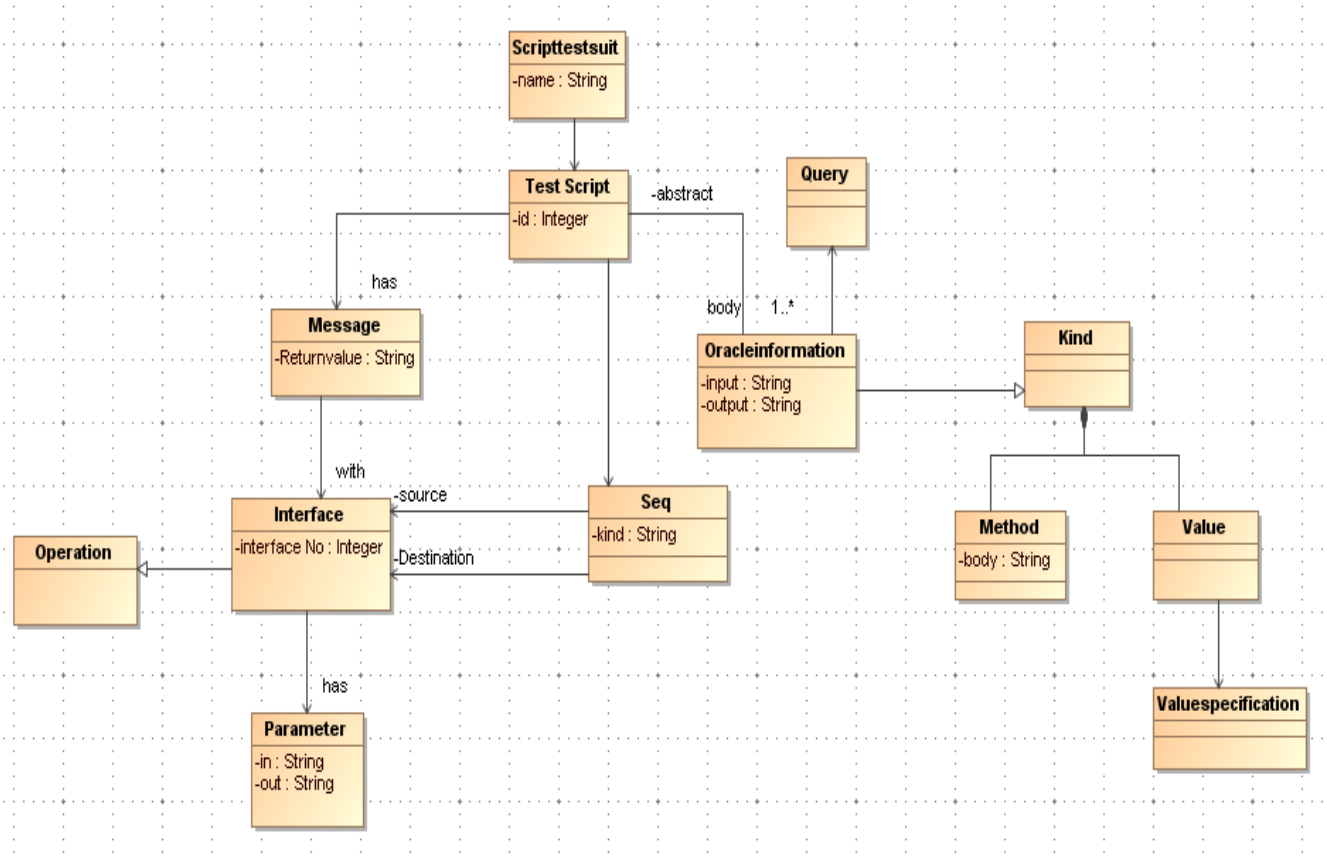


Figure: 3.3Test script Metamodel (PSM)

To write test script (manually) for test case must use some software technology or platform like scripting languages. The test script for test case withdrawal is written in JUnit in the example in Figure 3.4. The test script is generated in concretization step by transform all test cases into test scripts.

```
public class ATM Test case
{
      Private Card card;
      Private Account account;
       Private Trans Trans;
      Private balance balance;

public setUp()
{
}
public void main()
{
Public enum  PIN_TYPE {PIN_OK,PIN_KO};
Public enum Trans_TYPE{Transfer, Withdrawal};
Public ATM atm:new ATM;

Messageresult= atm.insertcard();
Messageresult=atm.enterPIN();
Messageresult=atm.enteraccount();
assertEqual(result,MESSAGE.SUCCESS);
assertTrue(user pin PIN[1]:PIN_OK);
messageresult=atm.selectTrans();
result=atm.withdrawal();
result = atm.balance(100);
Message result=atm.enteramount (20);
assertEqual(account.balance,accuont.balance - amount(20));
assertEqual(account.balance,account.balance - amount(20) == 80);
assertTrue (account. Balance (80));
}
}
```

Figure 3.4Test script for ATM using JUnit

From test case withdrawal

Scripttestsuit is a collection of many *testscripts. Scripttestsuit* has attribute *name* which string type, this name is a *name* of *scripttestsuit, testscript* has attribute *id* which is integer type, any *testscript* has *message* with *interface*, the *message* has attribute *returnvalue* which is string type, *interface* has attribute *interfaceNO* which is integer type, and any *interface* has *parameter* with attribute *in* and *out* which represent *input value* and *output value,* the kind of the *interface* is *operation*, The *oracleinformation* instance has two attributes *input* and *output* which is represent the *input value* of the *testscript* and the expected *output*, it has two kinds *method* and *value*, the *value* has *value specification*, *seq* has attribute *kind* which is string type, and has association with *interface* in association of *source* and *destination*. *Testscript* has association with *seq*.

3.1.3 Developing Mapping Rules

The foundation of MDA architecture is creation of models. Models are representing the APIs of the platform and application specification. However, there is an important issue – transformation among these models.
Transformation of a model is a process when one model is a source, converted into another model – destination with the use of certain transformation rules (Kardos et al, 2010).
For any transformation, first we should map every metamodel element(s) in the source to their corresponding target element(s) (model-to-model transformation). In this research mapping PIM to PSM done specify informally table [3.1] which shows how the test case metamodel elements mapped to test script metamodel elements:

Table 3.1 Mapping rules table

Test case Metamodel Elements (PIM)	Test Script Metamodel Elements (PSM)
Test suit-name-string	**Script testsuit**-name-string
Test case-id-integer	**Test script**-id-integer
Transition-kind-string	**Seq**- kind – string
Transition – source-string	**Seq**- source – string
Transition – target-string	**Seq**- destination – string
output-in-string	**Oracle Infomation**-input-string
Parameter-name-string	**Interface** - name-string

3.1.4Automatic transformation from PIM to PSM

To automate generating test script from test case we have used MDA automation machinery which baisclly centered on QVT engine.

The following are steps practically followed to do this automation which depends on EMF [www.eclipse.org/emf] which is rich model manipulation case tool developed as open source for MDA programming, MediniQVT for written executable rules of mapping and XMI [www.omg.org/index.html]which enables migration of models from tool to another tool without much effort.

These steps are:

1. After drawing the PIM and PSM models in the Magic draw tool, export them to XMI files.
2. Using Eclipse tool create new project as EMF project (Eclipse Model Framework).
3. Import the XMI files to Eclipse as EMF project to create Ecore files Metamodel and .genmodel files based on UML.
4. Write java file to create PIMInstance.
5. In the QVTmedini import The PIM Ecore and PSM Ecorse (metamodels files).
6. Write QVT rules to mapping source to target.
7. After determine source, target, and qvt mapping file and create trace folder does the configuration of the run.
8. Do run to generate the result of mapping (PSM instance).

To automate test scripts, firstly create the PIM instance, which is a result of the first four steps. This PIM instance created in Eclipse tool.

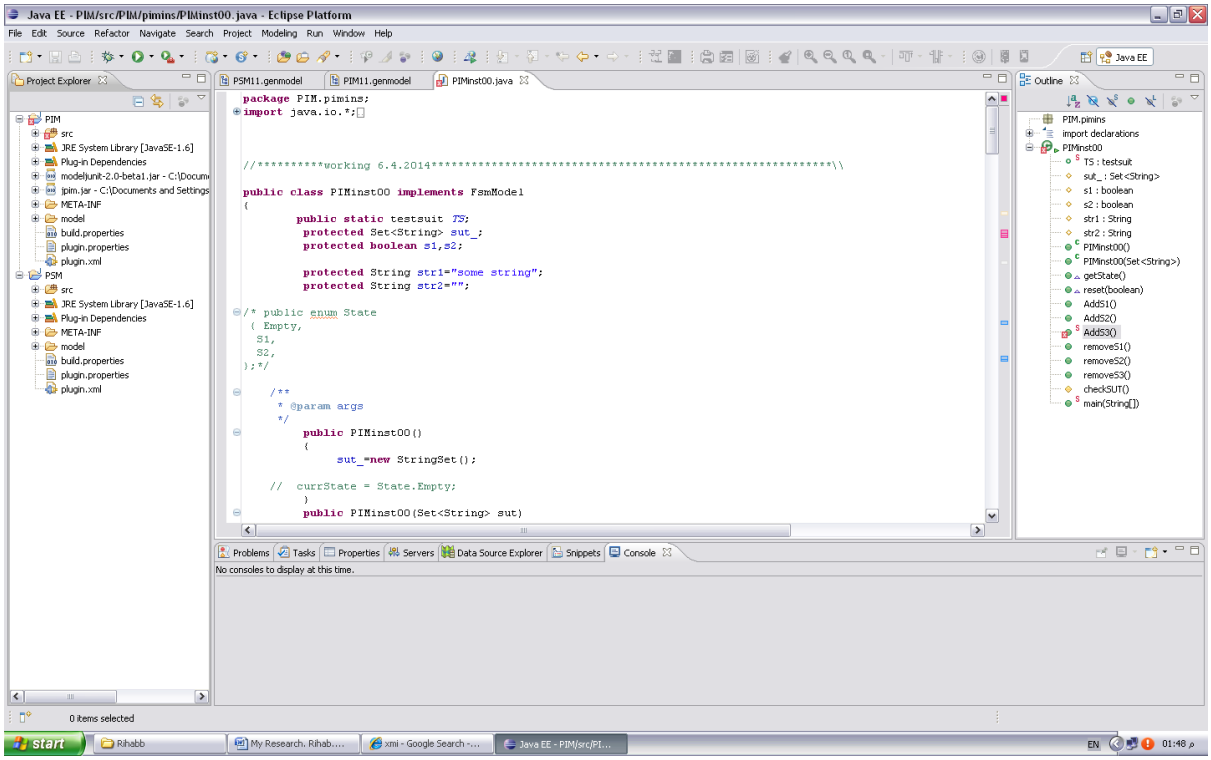


Figure 3.5 Eclipse Tool

PIMInstance.xmi

```
<?"xml version="1.0" encoding="UTF-8?>
xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" >
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:PIM="http:///PIM.ecore"
<"xsi:schemaLocation="http:///PIM.ecore PIM.ecore
</"PIM:testsuit name="TestSuitInstance>
</"PIM:testcase id="1>
</"PIM:transition target="/6" Kind="TRInstance" Source="/5>
</"PIM:method name="Adds1>
</"PIM:method name="Adds2>
</"PIM:method name="removes1>
</"PIM:method name="removes2>
</"PIM:parameter name="PARInstance" value="PARValInstance>
</"PIM:guard input="GRInstance>
</"PIM:output in="InInstance" out="OutInstance>
<xmi:XMI/>
```

Figure 3.6 PIM Instance

```
/* --*This transformation is uni-directional in direction "Test script" and maps test case
.elements to test script elements.
*It is based upon the example in the official QVT specification at http://www.omg.org/cgi-
.bin/apps/doc?ptc/05-11-01.pdf
*/
transformation pim2psm (pim:PIM, psm:Data) {
-- map each Test Case to Test Script
--*****************************************

 top relation TS2scriptTS {
 pn: String;

 checkonly domain pim p : PIM::testsuit{name = pn};
 enforce domain psm s : Data::Scripttestsuit{name = pn};

 enforce domain pim x : PIM::testcase{};
 enforce domain psm m : Data::TestScript{};
}
top relation TC2TS{
no:Integer;

checkonly domain pim x : PIM::testcase{id=no};
enforce domain psm m : Data::TestScript{id=no};
}
top relation method2Oper{
nm:String;

checkonly domain pim mth : PIM::method{name = nm};
enforce domain psm Oper : Data::Operation{name = nm}
     }
top relation Trans2Sequ{
k: String;

checkonly domain pim tra : PIM::transition{kind=k};
enforce domain psm S : Data::Seq{kind=k};
}

 top relation output2Oracinfo{
outp: String;

checkonly domain pim op : PIM::output{out=outp};
enforce domain psm orcinfo :
Data::Oracleinformation{ouput=outp};
 }

 }
```

Figure 3.7 QVT Mapping Rules

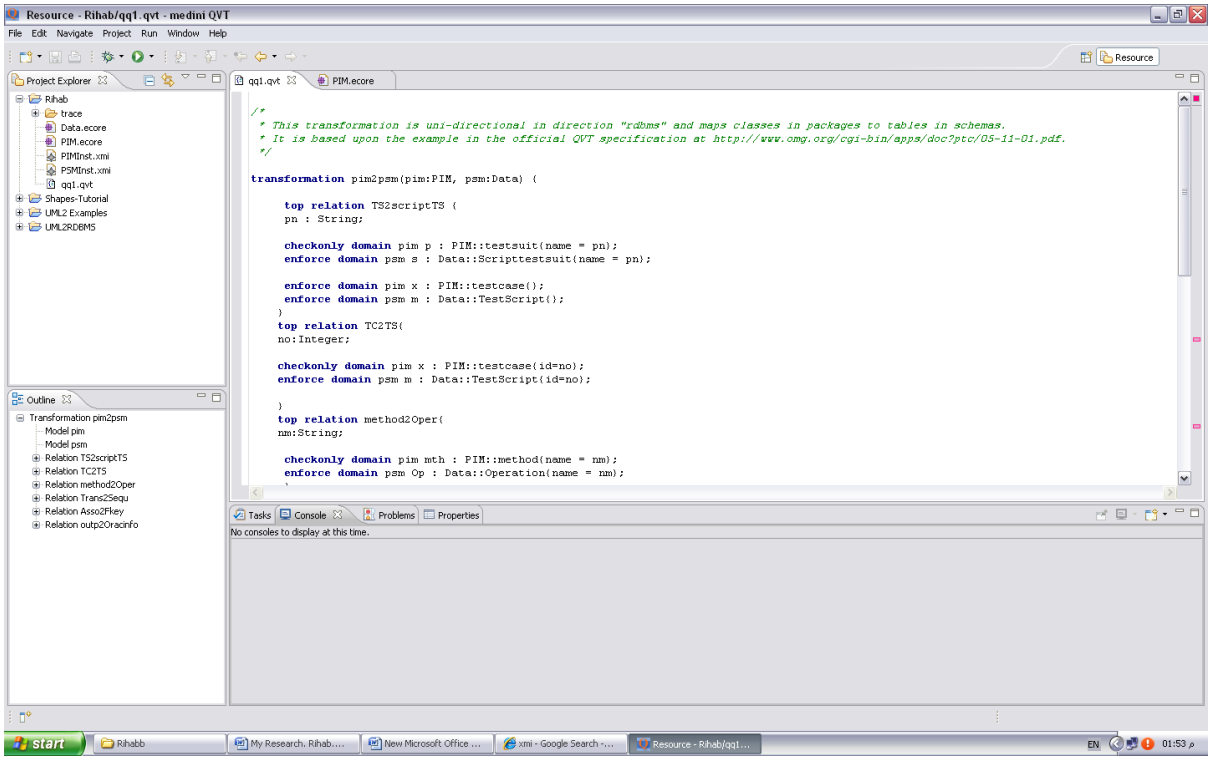


Figure 3.8 MediniQVT Tool

The result of mapping is the generating of PSMinstance

PSMInstance.xmi

```
<?xml version="1.0" encoding="UTF-8"?>

<xmi:XMIxmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:Data="http:///Data.ecore"
xsi:schemaLocation="http:///Data.ecore Data.ecore">

  <Data:Oracleinformation output="OutInstance"/>

  <Data:Seq kind="TRInstance"/>

  <Data:Operation name="removes2"/>

  <Data:Operation name="removes1"/>

  <Data:Operation name="Adds2"/>

  <Data:Operation name="Adds1"/>

  <Data:TestScript id="1"/>

  <Data:Scripttestsuit name="TestSuitInstance"/>

  <Data:TestScript/>

</xmi:XMI>
```

Figure 3.9 PSM Instance

CHAPTER 4

CONCLUSION

4.0 Discussion and Conclusion

This research proposed a method that generates test script from test case using MDA tool without using much programming skills. This trend was adopted by OMG (nonprofit organization) in an innovation called MDA.
In MDA principles transformations from test case to test script using standard mapping tool like QVT focal.

In MDA there are different alternatives to get new information in the transformation from one model to another (e.g. using profile, using metamodels, patterns and markings, etc) for this research a metamodel mapping approach to specify the transformation. They are PIMs and PSMs where the former is used to represent the test case and the later is for test scripts

The objectives of this research have been implemented in followed steps:
Firstly the PIM have been developed for testcase by using UML which described in (Figure 3.1). This figure contains all the elements of test case. This model is build based on real case study (a number of testcases). Secondly the PSM have been developed for testscript (after investigating real testscripts for that testcase written manully) by using UML which described in (figure 3.3). This figure contains all the elements of test script.

After the creation of the PIM and the PSM, we transformed the elements of test case to the elements of testscript in mapping table. This mapping has been mapped formally into QVT rules using MediniQVT to automate transformation from PIM to PSM.

The result of evaluation can be mapped by:
The PIM is capable for representing any testcase in any system with different element for the testcase.
The PSM is more difficult to be developed in representing the testscript, the difficulty comes from the representation of oracle, but we could represent a certain kind like on a Transition of Statemachine which can be automated.

The result of this research can be interpreted as how the generation of test scripts automatically helped in shortens the development cycles, and avoided the repetitive tasks.

The advantages provided by MDA are reduction of costs through reusing PIM and especially PSM for different sets of problems in a domain of testing. This will lead to improve testing quality. It also simplifies the maintenance test script which is a common problem in testing. This is achieved because MDA was based on assuming PIM or PSM or mapping rules are not stable. This facilitates changing PIM which in this case represents the scenario of having new different TestSuits.

The change in PSM which represent different scripting platforms (i.e. instead of Junit a Ruby) although is not studied in this research but its affordable. On other hand this will not hid the complexity of this problem part of that is diversity on testing platforms so more future research is needed.

REFRENCES:

Mark Utting, Bruno Legeard. *practical model-based testing a tools approach.* San Francisco: Morgan Kaufmann,Elsevier Inc, 2007.

MARK FEWSTER, DOROTHY GRAHAM. *Software Test Automation Effective use of test execution tool.* Association for Computing Machinery Inc, 1994.

A. Z. Javed, P. A. Strooper and G. N. Watson. "Automated Generation of Test Cases Using Model-Driven Architecture." *Second International Workshop on Automation of Software Test (AST'07).* Australia: IEEE, 2007.

Tracy Gardner, Catherine Griffin, Jane Koehler, and Rainer Hauser. A review of OMG MOF2.0 Query / View / Transformations Submission and Recommendations towards the final Standard,

Igor Sacevski , Sachead@gmx.net , Jadranka Veseli , jaca@gmx.at .Introduction to Model Driven Architecture(MDA), June 2007.

Prabhu Shankar Kaliappan, psk@informatik.tu-cottbus.de . State of the Art Model Driven Architecture, December 2007.

Frank Truyen. The Fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture (MDA) Cephas Consulting Corp, January 2006.

I. Arrassen, A.Meziane, R. Sbai, M. Erramdami. QVT transformation by modeling from UML Model to MD Model, 2011.

Paul Baker, Zhen Ru Dia, Jens Grabowski, Qystein Haugen, Ina Schieferdecker, Clay Williams. Model-Driven Testing using the UML Testing Profile, 2007.

Mikko Aleksi Makinen. Model Based Appraoch to Software Testing, 2007.

Robert Binder. " Testing Object-Oriented Systems: Models, Patterns, and Tools,1999"

Santiago Melia, Andreas Kraus, and Nora Koch. MDA Transformations Applied to Web Application Development. Santi@dlsi.ua.es , {kochn,krausa}@pst.ifi.lmu.de.

Martin Kardos, Matilda Drozdova. Analytical Method of CIM to PIM Transformation in Model Driven Architecture (MDA), 2010. Martin.cardos@fri.uniza.sk , Matilda.drozdova@fri.uniza.sk

Yang Liu, Yafen Li, Pu Wang. "Design and Implementation of Automatic Generation of Test Cases Based on Model Driven Architecture." *Second International Conference on Information Technology and Computer Science.* Beijing: IEEE, 2010. 1-4.

Fuqing Wang, Shuai Wang, Yindong Ji. "An Automatic Generation Method of Executable Test Case Using Model- Driven Architecture." *Fourth International Conference on Innovative Computing, Information and Control.* Beijing: IEEE, 2009. 1-5.

Jan Tretmans, Marinus J. Plasmeijer: Gast: Generic Automated Software Testing. IFL 2002: 84-100

Xiuhua Zhang "Tools for Mapping Technigue between PIM and PSM" Oslo June 2002

OMG. *MDA Guide Version 1.0.1.* 2003.

—. *MDA: Executive Overview.* 2005. http://www.omg.org/mda/executive_overview.htm (accessed 10 21, 2005).

—. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification version 1.0.* OMG, 2008.

OMG. "OMG Unified Modeling Language (OMG UML),Superstructure, V2.1.2." 2007 b: OMG Document Number: formal/2007-11-02.

ORMSC, Architecture Board. " Model Driven Architecture (MDA)." 2001.

—. "Model Driven Architecture ." 2001.