TEST DERIVATION AND REUSE USING HORIZONTAL TRANSFORMATION OF SYSTEM MODELS

by

Jenis Kavadiya

A THESIS

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SOFTWARE ENGINEERING

School of Innovation, Design and Engineering

Mälardalen University
Västerås, SWEDEN

June 2009

Supervisor

Antonio Cicchetti,
Ph.D in Computer Science,
PostDoc Researcher at Mälardalen University
The Thesis committee
Certifies that this is the approved version of the following thesis:

TEST DERIVATION AND REUSE USING HORIZONTAL TRANSFORMATION OF SYSTEM MODELS

APPROVED BY
SUPERVISING COMMITTEE

_________________________
Supervisor: Antonio Cicchetti

_________________________
Examiner: Mikael Sjödin
Abstract

Software Testing and Test generation are carried out manually in many organizations and is a time consuming and inefficient method. Model based testing (MBT) offer lot of promises for effective and efficient way of automatic generation of tests and offer better support for automated testing. Despite of the benefits of MBT, its adoption in real industrial practice is very limited. The thesis work is aimed to make MBT process and its adoption more efficient and effective. In simple words, Model Based Development (MBD) includes generation of code from system models. Model Based Testing includes automatic generation of test infrastructure from test models. Both MBD and MBT involves activities such as creation of models (system/test), model checking and generation of platform-specific models using model compilers. In this thesis we purpose approaches for making MBT more efficient and easily adoptable by reusing the artifacts of the MBD like system models, model compilers and model checkers for the purpose of MBT.

An important step in adoption of MBT is selection of appropriate MBT tool. These tools vary on different criteria like test modeling languages, algorithms, domain of testing and so on. Our work also describes a comparative study of various MBT tools that can guide in selection of suitable tool.
Preface

This report is final thesis of my Master Degree in Software Engineering at Mälardalen University, Sweden. I started to work on Master thesis for Ericsson from 15-Jan-2009 and presented the results on 01-Jun-2009.

I would like to thank my supervisor Mr. Antonio Cicchetti, PostDoc researcher at Mälardalen University for his comments, suggestions, and having the time to review thesis work. The thesis contents and structure is highly benefited from his invaluable experience for which I am very grateful.

Furthermore I would like to express my sincere gratitude to my academic mentor Daniel Flemström and my examiner Mikael Sjödin for their guidance. I would like to thank my mentor from Tata Consultancy Services Mr. R Venky and Mr. Arun bahulkar. Finally, I would like to acknowledge the important role my parents as they encouraged and supported me with my studies and work.

Thank You!

Jenis Kavadiya
# Table of Contents

Abstract ........................................................................................................................................... 3  
Preface .............................................................................................................................................. 4  
Table of Contents .......................................................................................................................... 5  
Acronyms and Definitions* .............................................................................................................. 6  
  1. Introduction ............................................................................................................................... 8  
  2. Background ............................................................................................................................... 9  
    2.1. Model Driven Engineering ............................................................................................... 9  
    2.2. Model Based Testing ........................................................................................................ 11  
      2.2.1. Model Based Testing Process ...................................................................................... 11  
      2.2.1.1. Creating test models and test specification .............................................................. 11  
      2.2.1.2. Automatic generation of abstract test cases ........................................................... 12  
      2.2.1.3. Generation of test scripts ....................................................................................... 12  
      2.2.1.4. Test execution ......................................................................................................... 12  
      2.2.1.5. Conformance testing ............................................................................................... 12  
    2.2.2. Advantages of Model Based Testing ........................................................................... 13  
    2.2.3. Drawbacks of Model Based Testing ........................................................................... 14  
  2.3. Model Reuse Approaches ....................................................................................................... 16  
    2.3.1. Shared Model Approach: ............................................................................................. 16  
    2.3.2. Separate Model Approach: ......................................................................................... 16  
    2.3.3. Hybrid or Interleaving Model Approach: .................................................................... 17  
  3. Tools Survey for Model Based Testing Tools ......................................................................... 19  
  4. Tools and Technology ................................................................................................................ 21  
    4.1. Executable UML and Qtronics Modeling Language ....................................................... 21  
    4.2. BridgePoint and Conformiq Qtronic .............................................................................. 21  
  5. Test Derivation using Model Transformation ........................................................................... 27  
  6. Lessons Learned ....................................................................................................................... 31  
  7. Related Work ............................................................................................................................ 33  
  8. Conclusion and Future Work .................................................................................................... 34  
  9. Reference ................................................................................................................................. 35
Acronyms and Definitions*

Bridgepoint - A MDE tool from Mentor Graphics typically used for embedded system

Conformiq - A company founded in 1998 with it’s headquarter located in Finland that develops tools for automated test design

ETSI - European Telecommunications Standards Institute, an independent, non-profit, standardization organization in the telecommunications industry in Europe

HTML - Hypertext Mark-up Language used for developing web pages

MBD - Model Based Development, emphasis on use of models for development activities and products [1]

MBT - Model Based Testing, a testing methodology in which test cases are fully or partially derived from models [2]

MDA - Model Driven Architecture, a software design approach and guidelines by OMG that supports domain engineering and MDE

MDE - Model Driven Engineering, a software development methodology that focuses on using models more closely to real domain rather than computing.

MOF - Meta Object Facility, a meta-meta language by OMG

MT - Model Transformation, for deriving target model from source models

OMG - Object Management Group, an International not-for-profit computer industry consortium that develops standards

PIM - Platform-Independent Models, they are independent of the specific technological platform used to implement them [3]

PSM - Platform-Specific Models, they are linked or dependent on a
specific technological platform [4]

QML - Qtronic Modeling Language, modeling language used in MBT tool
Conformiq Qtronic

QVT - Query/View/Transformation, standard for Model Transformation in
MDA defined by OMG [5]

Qtronics - MBT tool build by an organization named Conformiq

SUT - System Under Test

SVN - Subversion, is a version control system

TTCN-3 - Testing and Test Control Notation Version 3, standardized testing
language from ETSI

UML - Unified Modeling Language, standardized general-purpose
language for modeling software intensive system

UTP - UML Testing Profile, a modeling language by OMG for supporting
testing related activities

XMI - XML Metadata Interchange, an OMG standard for exchanging
metadata information via XML

XML - Extensible Markup Language, a general-purpose specification for
creating custom markup languages

xtUML - Executable UML, a UML profile based on Shlaer-Mellor method
that supports MDA

* The definitions are not complete in the global context rather relevant to thesis work
1. Introduction

In many organization, especially embedded and product line, Model-Based Development is practiced and code is generated from models known as System models. These models are usually written in UML, its profiles like Executable UML (xtUML) or similar languages. They can be so detailed that they are capable of generating the entire software replacing manual handwritten coding. Model based testing is pretty new to many organizations and many of them are still evaluating the suitability of MBT for automating the test generation process.

In this work, we propose to exploit system modeling activity efforts by partially (or fully) deriving test models from system models. In our experiment, system models written in Executable and Translatable UML (xtUML) are transformed into test models in Qtronics Modeling Language (QML) using horizontal model transformations. These test models serve as an input for an MBT tool like Conformiq Qtronic and generates test infrastructure. Using the plug-in adaptor developed for the MBT tool, the test output is customized to be generated in the same modeling in which the system models are written.
2. Background

2.1. Model Driven Engineering

In general, a model is a simplified placeholder of a reality and its purpose is to represent some aspect exactly the same way a real thing should have represented. A model helps in visualizing the system, simplifies system understanding, analyzing characteristics of the system before it is build and guides in construction of the system. In software engineering a model like UML state-chart describes possible states and its transition (aspect) an object (reality) can have. A model is written in a language specified by its meta-model. A meta-model is written in a language specified by its meta-meta model. A meta-model specifies the properties of the model itself.

Model Driven Engineering (MDE) is an emerging fields of software engineering that emphasis on the use of models as an integral part of software development and using them in various phases of software lifecycle. Model Driven Architecture (MDA), an OMG initiative for MDE uses modeling standards (UML, MOF and CWM) to address the problem of changing infrastructure/platform (technology, programming languages, and operating system) through the reuse of platform independent models [6].

OMG addresses the problem of rapid platform evolution using MDA principles as [7]:
- Decouple neutral business models from variable platforms
- From code-centric to model-centric approaches
- Transformations as assets

It is about separation and transformation of platform-independent models (PIM) to platform-specific models (PSM) using standard mappings. The main idea behind MDA is the abstraction of “What” from “How” and let the users focus on “What” and “How” will be taken care by model driven tools. Although MDE is in its infant stage, it is seen as a promising solution to address the software productivity problem both in academia and in the industry. Model Driven Development can bridge the gap between business and IT, by capturing business objectives in the form of models (process flow, business rule, information, organization, and integration models) and converting it to code (IT) [8].

The Object oriented systems are based on the principle that “Everything is a class” and an object has a relationship “instance of” with class, which further “inherits
from” super class. The basic driving principle behind MDE is “Everything is a model”, and a system has a relationship “represented by” model which in turn “conforms to” meta-model [9]. An important point here is that in object oriented world a meta-class is a class of a class whereas in MDE a meta-model is not the same as model of a model.

Model transformation is central to MDA. In model transformation targets models are derived from source models with both source and targets confirming to their respective meta-models. When source and target meta-models are same the transformation is called endogenous otherwise it is called as exogenous transformation [10]. When source and target models are at different levels of abstraction it is called as Vertical transformation for ex: from system models to code or test models to test scripts. When both are at same level of abstraction the transformation is known as Horizontal transformation [11] for ex: test models derived from system models. A transformation itself can be a model that conforms to a given meta-model. Thus their can be Higher Order transformation which take transformation as input and produces other transformations as output. Software lifecycle can be viewed as a chain of mode transformation [9].QVT is a standard from OMG in MDA and defines a way for transforming source to target models. The source and target model may conform to the MOF meta-model. Also the transformation program can be a model that conforms to MOF meta-model. This means that abstract syntax of QVT should confirm to MOF meta-model. As of now, QVT specifies model to model transformation only. Model to Text and vice versa transformation are addressed by the use of Domain Specific Languages (DSL). [5]
2.2. **Model Based Testing**

Software testing is a dominant activity as a part software quality assurance to verify the behavior of the software with its specification. There are different strategies for performing testing. One way is to do manual testing which is oldest and widely used solution. This is an expensive and time consuming method when testing is to be repeated whenever changes to the software are made. Capture and Replay tools are used to record the manually conducted tests and rerun them repeatedly. However they suffered from the drawback of recoding whole tests again even when there are small changes to the System Under Test (SUT).

Another solution is to use Script based testing which has shown the potential of fully automating the testing process. It has been observed that the size of test script typically grows comparable to the size of software itself and is difficult and costlier to maintain. Model based testing is among the recent solution for automatic generation of test scripts as well as test cases. [12]

MDE is not only about MBD but also about MBT; both involve generation of Platform-specific models (system code/test code) from Platform-independent models. The models used by MBD are known as System models whereas those used in MBT are called Test models. System models depict the actual behavior of the software. Test model represents the intended behavior of the software, the system is supposed to possess. Software testing is set of activities to demonstrate that intended and actual behaviors of the system differ or does not differ. Model based testing verifies the behavior specified by test models against the actual system or system models.

2.2.1. **Model Based Testing Process**

Model Based Testing process usually consists up of five steps from creation of the model to finding the results of execution of the test cases. Following is the brief description of the MBT process [13] also shown in Figure below.

2.2.1.1. **Creating test models and test specification**

A model of the system which is to be tested is build on the basis of requirement specification know or available for the system. The model represents the intended behavior of the system and can be very abstract that it might omit various other aspects, functions or characteristics.
The test selection criterion when defined, describe or responsible for the test suite. The criteria may be related to the functionality of the system (requirement based) or to the structure of the model (transition coverage, state coverage)

2.2.1.2. **Automatic generation of abstract test cases**
Test models and test case specification created in the previous step are given the inputs to the test generator and abstract test cases are generated as output. These abstract test cases more closely resembles to platform independent assets and in some MBT tools these are generated on the fly without accessibility to the users.

2.2.1.3. **Generation of test scripts**
The platform independent abstract test cases suite is translated to test script which is platform specific usually specified in some programming or testing language like C, C ++ or TTCN. The translation is carried out by software module called adaptor or scripter. Some MBT tools provider delivers number of scripter along with their tool. For each specific platform we need a separate scripter.

2.2.1.4. **Test execution**
This phase is not much specific to Model based testing and is same as execution of any script based testing

2.2.1.5. **Conformance testing**
In this step the results of the execution of the test cases of the above step are compared with the expected output or behavior specified by the test models.
2.2.2. Advantages of Model Based Testing

- **Automatic Test Case Generation**: The test cases need not to be designed and written manually and can be automatically generated from the test models.

- **Faster Test Design**: Once the test models are ready the test cases can be generated in virtually no time as compared to manual testing. But the creation is the test model still takes time which is not required for manual testing in which the model is mostly mental (in the mind of the test designer). As the organizations become more matures in model based testing, the time and effort required for creation of the models significantly reduces. The approach presented by us in this thesis work for model reuse significantly reduces the time for creation of test models.

- **Higher Test Quality/ Better Test Coverage**: With manual testing it is possible that many tests cases may be forgotten or omitted because of the human error. But with the use of tool the possibility of human error is eliminated. Of course there are chances of errors and omission in test models itself. But the probability of omission is less because of the fact that models are at more abstract level and are easy to find for error than the system itself. As in our thesis work the test models are partially (or fully) derived the probability of omission/error is still reduced.
• **Finding requirement defects:** During creation of test models certain behaviors of the system that is to be tested is represented. In practice when representing such behavior highlights or reminds many defects, omissions or contradictions which were present in the requirements specified in the system models or the system itself.

• **Traceability:** Once the test cases are generated the user might want to ensure that requirement or behavior that he/she wants to test is covered by the generated test cases. Apart from this other traceability like coverage can be used to shown the portion of the model or test criteria covered by the test cases.

• **Easier Test Maintenance:** Whenever there are changes to the system or the test strategy the test suite need not to be completely redesigned. Instead only the corresponding changes to the test models need to be done and entire test suite can be regenerated.

• **Automatic Generation of Test Data:** Some MBT tools automatically generate test data for example for conditional statements, boundary values and so on. Few tools also generate the combinations of the test data from the data presented in the test models.

• **Determination of Verdicts and Test Result Evaluation:** Based upon the test models MBT tools can automatically determines the verdict. And once the test execution is completed the verdict can be automatically evaluated against the actual results of the test execution.

• **Online MBT:** In online testing test cases generation and are test execution and test case generation takes place in parallel. Thus in general when a test case is generated it is tested before the generation of next test case. Thus Non deterministic testing and infinite test suite possible using online approach. Online MBT can also react to continuous changes made to the test models.

### 2.2.3. Drawbacks of Model Based Testing

• **Skills required for model based testing:** Testers are considered as less technical persons. Model based testing requires testers to be equipped with the knowledge of MBT tools, modeling mathematics, formal languages and some graph theory
• **Splitting and Merging of models:** In practice for medium to bigger size projects, test modeling requires more than one person to be involved. This involves splitting the modeling tasks and merging them again to yield a completed test system. However this merging and splitting is a difficult task and may involves a lot of inconsistency problems.

• **State space explosion:** Most MBT tools use state diagrams in creation of test models. Gradually as the size of model grows the number of states grows exponentially. This makes the model checking, model maintenance, test case generation and coverage evaluation more difficult and time consuming. [16]
2.3. **Model Reuse Approaches**

Approaches are classified into Shared, Separate or Hybrid on the basis of extent of reuse of development models for deriving test models.

**2.3.1. Shared Model Approach:**

In Shared Model approach same models were used for the purpose of both MBD and MBT. The system is generated from the models from which tests are also derived. Hence the system is tested against itself. The expected output part of the tests or verdicts can not be derived using MBT tool [17]. If system is tested using this approach, without manual derivation of verdict, it is more or less testing of code generator or model compiler.

A benefit of this approach is that models have to be created only once and are verified twice both from MBD and MBT perspective. Model maintenance is low, for instance if changes have to be done to behavior of system, changing to the parent model will not only update the system models but also the test models and therefore the generated software as well as test suite. However if there is bug in the parent model, MBT won’t detect such bugs as test cases are also derived from the same defective models.

![Diagram of Common model approaches](image)

**Figure 4 Common model approaches [17]**

**2.3.2. Separate Model Approach:**

In this approach different models were used for the purpose of MBD and MBT. Verdicts can be automatically derived from test models using MBT tools [17]. In
many situations testing yields better results when it is independent of implementation. In separate model approach expected behavior (test models) are independent of actual behavior (system models). However the time and effort in creating and maintaining separate models is very high especially when changes made to system models also requires test suite (hence test models) to be updated. Moreover testers and domain experts are non technical persons especially in the field of software modeling; creation of test models from scratch makes adoption of separate model approach more difficult.

![Figure 5 Separate model approach [17]](image)

### 2.3.3 Hybrid or Interleaving Model Approach:

In this approach [18] test models are derived partially or fully from the system models or vice versa. As shown in figure 1 below, partially derived test models are completed by enriching them with details mostly manually. The completion does not mean enriching the entire software test details but only the completeness of aspect under test.
Test models need to be detailed and formal enough for an MBT tool to generate test infrastructure, when using hybrid approach this also imposes limitations on the system models from which the test models are derived. Usually Models for general purpose software are not too detailed and formal, however embedded & real time systems, mission critical software and product based organization are exception.
3. Tools Survey for Model Based Testing Tools

In order to select MBT tool for our thesis we perform tools survey based on certain parameters. Then we have work closely with some of these tools and to test compatibility of these tools with each other. To our surprise we found that none of the tool is compatible with other and has got some or the other problem. Following is the summary of our experiment and validation of the claims made by the tool [13]

a) TorX

<table>
<thead>
<tr>
<th>Type</th>
<th>Academic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Automated test generation and implement the testing theory of conformance relations between models and implementations</td>
</tr>
<tr>
<td>Domain</td>
<td>Telecommunication</td>
</tr>
<tr>
<td>Features</td>
<td>Dedicated testing model. Support LOTOS and SPIN modeling language. Support non determinism through online testing and also support offline mode. It is based on an Input-Output Labeled Transition System model of the SUT.</td>
</tr>
</tbody>
</table>

b) LEIRIOS Test Generator (LTG)

<table>
<thead>
<tr>
<th>Type</th>
<th>Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Test case generation using model coverage criteria (State coverage, structural and data coverage).</td>
</tr>
<tr>
<td>Domain</td>
<td>Reactive systems, embedded software, smart card or e-Transaction applications</td>
</tr>
<tr>
<td>Features</td>
<td>Support both UML models and B Abstract machines (B is a Pre/Post notation). It also provides the oracle for each generated test case. Abstract test cases generated from model are translated into executable test scripts using adaptors specific to target execution environment. Support Dedicated testing model.</td>
</tr>
</tbody>
</table>
c) **MatLab Simulink V&V**

<table>
<thead>
<tr>
<th>Type</th>
<th>Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Traceability and model coverage using simulink/stateflow models</td>
</tr>
<tr>
<td>Domain</td>
<td>General purpose</td>
</tr>
<tr>
<td>Features</td>
<td>Shared model approach. It facilitates automatic documentation by generating reports of model coverage, simulation, and test results. Provide for Traceability of requirements, develop and check for customize modeling standards, model coverage and uses formal methods to verify designs. It runs only on models hence it is an offline approach.</td>
</tr>
</tbody>
</table>

---

d) **Automatic Efficient Test Generator (AETG)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Test Input generator using n-way search algorithm.</td>
</tr>
<tr>
<td>Domain</td>
<td>Lot of Experiments in 1997 with Telecom domain.</td>
</tr>
<tr>
<td>Features</td>
<td>Uses combinatorial testing approach &amp; significantly reduce number of generated test cases but the oracle for each test case has to be provided manually. Environmental &amp; dedicated model and offline testing approach</td>
</tr>
</tbody>
</table>
4. **Tools and Technology**

4.1. **Executable UML and Qtronics Modeling Language**

The industry de-facto modeling standard, the core UML is too generic and lacks features to be used in MBD for specific applications. UML profiles are extension mechanism of the basic UML concepts which are used to fit the specific needs such as code generation, testing or for other domain specific purpose. xtUML is one such UML profile, a modeling language capable of creating platform-independent executable models and for code generation. xtUML has rich set of constructs that allow model compilers to transform them to virtually any programming language/PSM. Apart from adding features it has removed platform specific constructs like aggregation and composition from UML. Models in xtUML usually composed up domain charts, class diagrams, statechart diagrams and the action language.[19]

QML is the default language for creating design models in MBT tool conformiq qtronic. The language consists up of textual notation as well as graphical. Design models can be expressed entirely with textual notation in QML and graphical notation is optional. The textual language seems to be superset of java with some concepts borrowed from C#. The graphical notation is state machine with QML textual language as action language. QML design models are internally converted to CQλ before qtronics testing engine execute them. [20]

4.2. **BridgePoint and Conformiq Qtronic**

BridgePoint development suite is a MBD tool used for the development of software (real time, embedded...) and simulation systems by providing the environment for agile MDA and development of xtUML models. The OMG compliant xtUML is used for developing PIM in bridgepoint which are fully testable and executable. These models consist up of class diagram (data), state charts (control), Object Action language (processing, OMG complaint) and Domain packages (for partitioning system scale-up). The xtUML PIM models are translated into code using model compilers. [21]
The BridgePoint development suite consists up of:

a) **BridgePoint Model Builder**: It provides environment for creating xtUML models and contain xtUML metamodel that ensure that model elements are structurally correct and object action language is semantically consistent.

![Figure 7 xtUML models in BridgePoint Model Builder [21]](image)

b) **BridgePoint Model Verifier**: This package provide support for executing, testing and debugging xtUML models before any code or translation takes place.
c) **BridgePoint Model Compilers**: These are used to translate xtUML PIM to optimized source code. The compilers are delivered with source code so that they can be extensible and can be modified to support different operating system, coding standard or platform. Custom model compiler can also be built to support any target language.

![Diagram of BridgePoint Model Compiler](image)

**Figure 9 Working of Bridgepoint Model Compiler [21]**

d) **BridgePoint Model Debugger**: It provides model level debug and test of generated system which provide features like selection of model based breakpoint, execution trace, single step and system inspect capability.
Conformiq Qtronic is a commercial tool for automatic generation of test cases from design models using various coverage criteria (like state coverage, structural, data coverage…). Qtronic Support both online and offline testing and offer testing of non deterministic system and infinite test suite in online mode. Qtronics is available for both windows and Linux. It Uses QML and offer model interchange with other modeling tools using XMI standard. It specially has compatibility and support model interchange with MBD tool Enterprise Architect. Adaptors for TTCN-3 and HTML are provided by default with the tools and scripter for other languages can be created easily using concept of plug-ins. The
Test cases generated by this tool are black box tests as they are generated from external behavior of the model and which also means that they evaluate the system’s external behavior. Our approach in section 5 demonstrates how we can make this black box tests closer to white box approach.

**Figure 11 Architecture of Conformiq Qtronic [20]**

The design model describes the intended behavior of the system and note that these models cannot be created in qtronics. Conformiq Modeler (CM) is separate tool that comes along with qtronics and is used for drawing UML state-machine diagrams and graphical notation of QML. The models created in CM are more compatible with qtronics than other external modeling tools.
Figure 12 QML models in Conformiq Modeler [20]
5. Test Derivation using Model Transformation

The framework below developed, is a creation from the scratch and is work in progress for illustration of some approaches that is used for making adoption and operation of MBT process more efficient. The process of code generation from xtUML system model is represented under cloud in the following diagram. Apart from this vertical transformation of xtUML PIM, it is also possible to transform xtUML system models to any test modeling language. Thus QML test models can be derived partially using model extraction & horizontal transformation of xtUML system models.
Transformation between source and target language is sometimes easier when models of these languages have the same meta-meta model. In this case the
transformation itself can also be based on the same meta-meta model. xtUML is MOF (Meta-Object Facility) based whereas QML is not. As UML is MOF based, the transformation of UML diagrams from xtUML to QML are MOF based [18] for ex: UML state diagrams can be transformed to QML state machines with minor syntactical changes. In conformiq qtronic UML diagrams are stored and represented using XMI format, which can be imported from modeling tools like Enterprise Architect and Bridgepoint. Textual portion of QML models are contained in separate files which can be independently derived from xtUML’s UML part and its action language. xtUML action language part is also platform and language independent. QML textual language is java compatible object oriented language with some non java based extensions (mostly derived from c#). Java also reveals some form of model based development, where the platform independent program (or byte-code) in java language and the java interpreter (similar to model compiler) translates the platform independent code to platform-specific code.

Models expressed in xtUML can be so detailed that the entire software system can be auto generated from them. These models can be virtually considered as the system itself, as the code generators or model compiler usually adds only the language constructs (syntax and semantics) and optimization to these models to yield working software. Thus in one sense, we are making a black box testing tool (MBT tool) to do white box testing by providing it with test models which are derived from, system models that are as good as actual software itself. With this strategy certain testing technique in conformiq qtronic like statement, branch coverage gives better results as they expect the input part of test models to be close as much as possible to the actual system.

Even though a lot of information may be present in xtUML models but it is desirable to extract only relevant information. The information to extract from xtUML models depends on number of factor like testing method, test criteria’s, test configuration, target language support and so on. For example in control flow testing it is required to extract the method call sequence whereas corresponding class diagram and other static details can be ignored. Thus even if qtronic may supports xtUML as its modeling language the entire existing xtUML system models can not be directly inputted as test models to Qtronics.

The output of qtronic is abstract test cases which are platform independent models. These abstract test cases are transformed to platform-specific models i.e. test code/scripts using plugins called scripters. It is possible to write scripters that can convert abstract test cases to platform independent xtUML test scripts. Any xtUML models can be executed by the execution engine of a model based tool before they can be converted to platform-specific models. Thus xtUML test script
can perform testing against xtUML system model way before the test code and system code might not exist.

As shown in the above figure, these xtUML test scripts can be feed back to the model compiler for generation of test code in a platform specific language. The model compiler might earlier exist if during model based development xtUML system models are converted to system code in the same language in which the test code is to be generated. Thus we do not need to write separate model compilers for the MBT when the model compiler for target language already exists for MBD, as source language for both the system model and the test script is xtUML.

Model Maintenance with this approach is low as compared to separate model reuse approach. Whenever there is a change in system model corresponding test models are derived again. Existing manual enrichment of test models should be harnessed and preserved, and should be applied over the newly derived test models. However certain inconsistencies may occur like harnessed manual enrichment exists on an old test model element which is now removed from the system model. As of now this discrepancy is addressed manually.

Bug Tracing (Traversing the entire framework backward): During testing whenever a bug is found it is sometimes possible to automatically trace it up-to the point where the system need to be modified in order to remove that bug. Test code might have caught a bug that can be traced back to the xtUML test script by the MBD tool. Now traces are found in xtUML test script to abstract test case by scripter and further to QML system model by MBT tool. This traced portion in QML models can be part of manual enrichment or of the QML test models derived from xtUML system models. In case it is from the original xtUML system models, a reverse transformation is applied to trace the source of defect in the system model from which the software has become defective.
6. Lessons Learned

a) Knowledge gained during the thesis helped to participate in MDE research

The knowledge gained during the thesis help us to understand and argue about various concepts in MDE. Sometimes we might be wrong but it invoke further thought process. One such example of this is during the thesis I read many times that platform independent models are created using UML / xtUML or other similar OMG standards. These platform independent models are transformed into platform specific models using translators or model compilers. We really doubt that these models can be claimed as “platform independent”. Firstly these models are dependent on the versions of the modeling language itself in which they are created, for example a UML 2.0 model may be an invalid model for UML 1.4. Secondly these models when drawn in different tools are not compatible with each other. The modeling language meta-model only defines some of the rules for construction of models in these language. The modeling tool needs to know various other rules like what is the shape of an element (class), what is the position of the element (coordinated) in the model in order to represent/draw the models. These details are simply omitted by the language specification and without these details concretization of the model is impossible. Thus the platform independent modeling language simply omits the details which makes them platform dependent and without these platform dependent features the creation of models is not possible. Finally, these languages claims that model in these languages can be translated to any platform specific languages and hence are platform independent. For example a UML model can be converted into c or c++ code using different model compilers. Note that these model compilers need meta-model of source and target language. We claim here that we can translate any language for example c++ to any platform specific language; a model compiler can do with platform independent model. A c++ program can be translated to C or Java program. Our translator also just needs a meta-model for source and target language. A c++ language may have an increment operator (i++) but for converting it to a C language our translator will convert it to i = i +1.

b) Don’t trust on the features specified in White papers of the Tools:

We tried number of combination s for compatibility of one tool with others. Some of them in their white paper claims to be fully compatible with another tool
whereas in practice they are not. Some tools claim to do an XMI export so that other tools can consume their output, whereas in reality they do not successfully import the same models that are exported using the same tool.
7. Related Work

Model Based Testing now a days, is a hot topic in the research field and in academia. It is gaining more and more support from the software industry but is practiced only in few organizations and mostly is in their infant stage.

The works that is closest to ours are by Javed, Strooper and Watson [22] for “Automated Generation of Test Cases Using Model-Driven Architecture” and by Dai [18] for “Model-Driven Testing with UML 2.0”. In [22], model transformation is used to generate test cases from sequence diagrams using one horizontal platform independent transformation (sequence diagram to xUnit) and two vertical platform dependent transformation (xUnit to JUnit and SUnit). They also developed two prototypes Tefkat and MOFScript for model-to-model and model-to-text transformation. Our work is different from their in the way that our approach is not limited to expressing the behavior only to sequence diagram. We apply model-to-model and model-to-text transformation in parallel and not in sequence on one another as in [22].

The work by Dai in [18] applies model transformation to translate UML models to UTP and intelligently purposes the transformation to be MOF based. With UML and UTP both MOF based things are straightforward. But most of the MBT tools don’t use UTP (or MOF based language) as their modeling language and have their own proprietary languages. In this sense our work is more complex which uses a language which is partially MOF based and illustrates the transformation in both MOF based and non MOF based scenario. Dai purposes more general approach whereas we provide more specific mechanism. We also apply certain approaches for making MBT more efficient like reuse of model compilers/adaptors and model checkers.

A number of MBT tools survey are available including at [13], [23]. The scope of these surveys is broader and detailed as compared to ours. Our aim is to find a tool that suite our requirements and once we found a major drawback in the tool we stop further investigation about the tool.
8. Conclusion and Future Work

In this work we researched, evaluate and purpose various approaches and techniques which makes model based testing more efficient and easily adoptable. A major hurdle in adopting MBT is in the skills and efforts required by testers to create test models. One major step we illustrate here is the partial or full reuse of system models for creation of test models using horizontal model transformation. For organizations which have already adopted MBD and willing to reuse the investments made in MBD for the purpose of MBT, we not only suggest to reuse system models but also to have the MBT tool adaptor generate the abstract test cases in the same language as the MBD already uses. This will ensure the reuse of already existing and tested model compilers and model checkers. We also demonstrates that it is sometimes possible to update the test suite automatically when changes are made to the system (i.e. system model to our case), as test models are derived from system models and are responsible for generating test suite. When a defect is found during testing, it is sometimes cumbersome and time consuming to locate which portion of the software is responsible for the defect. We suggest a way to trace back a defect automatically to the point of its origin in code (or system model in our case).

For model reuse through horizontal transformation we purpose mapping that is close to the implementation. We experimented with them with the help of a toy example. A full-fledge matured implementation that actually transforms the entire source modeling language to target is a possible future work. In this work we also provide directions of how to proceed for efficient “model maintenance” from system to test model auto-derivation and “model tracing” from defect to its origin. These two topics need to be further researched and can be implemented to demonstrate that such approaches are possible and are efficient.
9. Reference

[1] Model-Based Development of Embedded Systems, B. Sch¨atz1, A. Pretschner1, F. Huber2, J. Philipps2
[8] Five Principals for Success with MDD, Recording date: Thursday, March 20, 2008 11:00 am, Panelist Information: Richard Soley, Setrag Khoshafian
[9] On the Unification Power of Models, Jean Bézivin University of Nantes
[16] Evaluation of Model-Based Testing on a Base Station Controller by Stefan Trimmel, Final Thesis Department of Computer and Information Science, Linköping University, Sweden
[18] Model-Driven Testing with UML 2.0, Zhen Ru Dai, Fraunhofer FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
[21] BridgePoint® Development Suite, Accelerating high-quality development from Project Technology Inc.
Automated Generation of Test Cases Using Model-Driven Architecture, A. Z. Javed, P. A. Strooper and G. N. Watson, School of ITEE, University of Queensland, Australia

Adapting model-based testing to agile context, Olli-Pekka Puolitaival, Espoo 2008. VTT Publications