From U2TP Models to Executable Tests with TTCN-3 - An Approach to Model Driven Testing -

Justyna Zander[1], Zhen Ru Dai[1], Ina Schieferdecker[1,2], and George Din[1]

[1] Fraunhofer Fokus, TIP,
Kaiserin-Augusta-Allee 31,
10589 Berlin, Germany
{j.zander,dai,schieferdecker,din}@fokus.fraunhofer.de
[2] Technical University Berlin, Faculty IV,
Straße des 17. Juni 135,
10623 Berlin,
Berlin, Germany

Abstract. The approach towards system engineering according to Model-Driven Architectures (MDA) with code generation derived from model implies also an increased need for research on automation of the test generation process. This paper presents an approach to derive executable tests from UML 2.0 Testing Profile diagrams automatically. In particular, an approach to derive executable tests within the Testing and Test Control Notation (TTCN-3) is discussed. The transformation rules between the source U2TP meta-model to the target TTCN-3 meta-model are given.

Keywords: UML 2.0 Testing Profile, UML, Testing, TTCN-3, QVT, Model transformation, MOF, MDA.

1 Introduction

Recently, the attention around automatic derivation of executable code from abstract models has been raised in the context of MDA (Model Driven Architecture [1]). We believe that that this concept can also be used also in testing area. Therefore, it is proposed to enhance MDA with a separate development line for testing artefacts [12]. We believe that derivation of executable tests from their models is possible to some extent. Due to complete test designs we gain the advantage of reduced work on pure tests programming. Several efforts have been undertaken to establish an approach to automate - or at least to provide significant support for an automated - test generation. Algorithms have been defined to derive tests from formal system specification given in various notations. But today, none of the approaches is widely used in the industrial practice for large applications [17]. As UML and MDA have gained much momentum in industry, we focus on using these concepts to show that retrieving executable test instances from system model can be supported via test skeleton generation combined with manual completion of the tests.

MDA prescribes certain model artefacts to be used along system development, how those models may be prepared and their relationship [1]. It is an approach to system development that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform [3]. Main MDA artefacts are platform independent system models (PIMs), platform specific system models (PSMs) and system code [1][14]. There is a clear distinction between PIM, PSM and system code although it depends on the context, the development process and the details of the system and target platform, where the border between PIM, PSM and system code is to be placed. Within these three abstraction levels, transformation techniques are used to translate model parts of one abstraction level into model parts on another abstraction level. These transformations can also be used to specify the relations and invariants between the models on different abstraction levels, which are the base to check the consistency between models and to validate models against each other. These MDA abstraction levels can also be applied to test modelling [15] as according to the philosophy of MDA, the same modelling mechanism can be re-used for multiple targets [16]. Similarly, test models can be specified platform independently and platform specific before generating executable test codes [8].

As shown in Fig. 1, platform independent system design models (PIM) can be transformed into platform specific system design models (PSM). While PIMs focus on describing the pure functioning of a system independently from potential platforms that may be used to realize and execute the system, the relating PSMs contain a lot of information on the underlying platform. In another transformation step, system code may be derived from the PSM. Certainly, the completeness of the code depends on the completeness of the system design model [8].

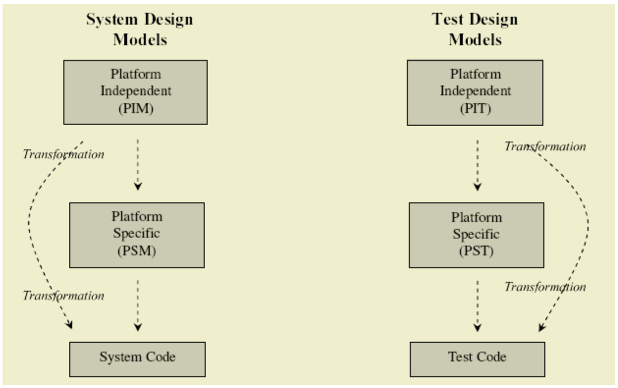


Fig. 1. System and Test Development

According to model driven testing[1] approach, a platform independent test design model (PIT) can be transformed either directly to test code or to a platform specific test design model (PST) [10]. Finally, the test design model can be transformed into executable test code from either PST or PIT.

This paper presents transformations between UML 2.0 Testing Profile (U2TP [4]) specifications used to represent PITs and Testing and Test Control Notation (TTCN-3 [5]). The transformations are specified as transformation rules between the U2TP

[1] We define model driven testing as testing based-MDA.

meta-model [4] and the TTCN-3 meta-model [10]. Afterwards, the generated output is completed and compiled to executable test code in Java [18].

U2TP and TTCN-3 meta-models are both defined as Meta Object Facility (MOF) models [1]. Transformation rules provided in this paper define relation between source and target meta-classes of these meta-models, while the transformations are performed on model (instance) level, i.e. deriving parts of TTCN-3 modules from parts of U2TP specifications. This procedure is shown in Fig. 2.

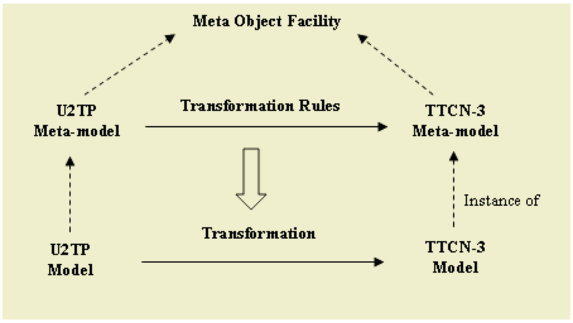


Fig. 2. Transformation of U2TP to TTCN-3

The goal is to get executable tests from U2TP models automatically, however, in general the generation will only be semi-automatic as U2TP specifications can be very abstract so that further details are needed to make the tests executable. Examples include the addition of concrete data, timing or default behaviours.

The environment, which is used to demonstrate the feasibility of our approach is Eclipse with its UML2.0 plug-in [19]. U2TP is realized as an extension of the UML 2.0 plug-in via its Java API. The transformation rules are also realized in Java. The transformations generate objects within a TTCN-3 meta-model instance, which enables the compilation and execution of the tests designed previously in U2TP.

The paper is divided into six sections. After the introduction, Section 2 is devoted to the U2TP and TTCN-3 meta-models which are used as source and target for the transformations. Additionally, we discuss Eclipse and its UML 2.0 plug-in as a tool which is used to implement and demonstrate our approach. In Section 3, the transformation theory in the context of model driven testing is discussed. Section 4 provides the methodology of retrieving the executable test code, which is possible by applying presented transformation rules and appropriate compilation. The transformation rules could be formalized in Query/View/Transformation (QVT) rules defined by CBOP/IBM/DSTC [3]. However due to lack of vendors providing appropriate tools and because of the limitations of the UML 2.0 profiling support in Eclipse, we had to realize the transformation rules directly in Java. Thus, we define our own mapping language and rules based on meta-model classes. In Section 5, an example of U2TP diagram is presented and the transformation rules for this example are described. Furthermore, the same example analysis, but resulting from application of the transformer implementation is continued. In Section 6, the results are discussed and conclusions are taken. Finally, future work challenges are outlined.

2 Related Work

Research as well as industrial work related to generation of executable tests from UML models according to MDA concepts is being continuously developed. LEIRIOS Test Generator™ tool (LTG) [23] implements the Smart Testing concept. It supports Model Based Testing - an approach in which one defines the behaviour of a system in terms of actions that change the state of the system (state machine). UML 2.0 models are used for automatic generation of test sequences. LEIRIOS core technologies implement smart heuristics to compute the test cases.

Objecteering Software [24] on the other side provides the opportunity of working with pragmatic design and coding tools, which combine UML modelling, code production, debugging and Java application testing in a single environment. Objecteering/UML tool is integrated into the Eclipse 2.0 platform. This integration allows the Java developer to take advantage of a strongly model-oriented tool, which, when integrated with a dedicated Java environment, associates the support of UML modelling with the support of Java development. Objecteering/UML tool bases however on UML 1.4 meta-model.

Finally, Telelogic TAU Generation2 [25] represents generation of advanced software development and testing tools, supporting the latest industry-standards for visual systems and software development (UML 2.0 Testing Profile) and systems and integration testing (TTCN-3). Telelogic team provides an approach that automates testing activities covering test specification, development of testing software and execution of test campaigns. U2TP is selected as modelling language for test case specification. The models are then transformed to TTCN-3 language, which is used for describing executable test cases.

Our approach is to use similar methodology as LEIRIOS deriving executable tests from UML 2.0 models, however we extend the models with U2TP concepts and integrate our tool with Eclipse platform as Objecteering team does. We develop also transformation rules from U2TP to TTCN-3 as offered by Telelogic, but we define the rules on the meta-model level using methods available in Eclipse to implement our approach.

3 Theoretical Background

The transformation between U2TP and TTCN-3 is obtained by use of the Eclipse framework for meta-modelling, repository generation and read/write access to model data in repositories. We store model information in Eclipse meta-modelling framework (EMF [21]) based repositories. The transformation rules are defined between source and target meta-models (see Fig. 3) and applied to concrete meta-model instances, i.e. source and target models in U2TP and TTCN-3 respectively. We design and develop test specifications in U2TP and perform the transformations on model level so as to get TTCN-3 test model instances.

In the following section we describe the main concepts of U2TP and TTCN-3, as well as introduce Eclipse being the tool used for the transformation.

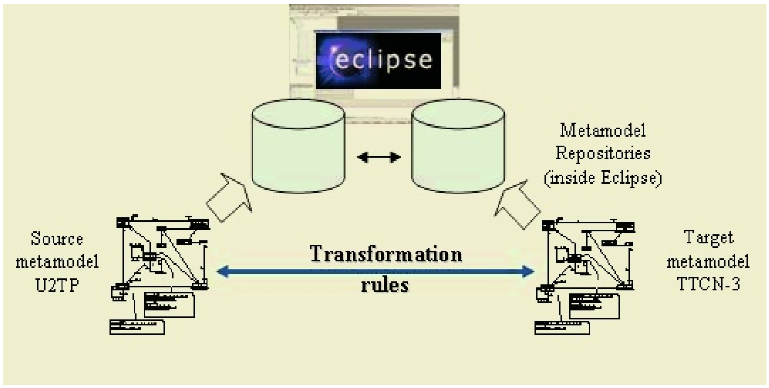


Fig. 3. Transformation Architecture

3.1 The UML 2.0 Testing Profile

The UML 2.0 Testing Profile (U2TP) defines a language for designing, visualizing, specifying, analyzing, constructing and documenting the artefacts of test systems. It is a test modelling language that can be used with all major object and component technologies and be applied to test systems in various application domains. U2TP can be used stand alone for the handling of test artefacts or in an integrated manner with UML for a handling of system and test artefacts together [4]. The UML 2.0 Testing Profile extends UML 2.0 with test specific concepts like test components, verdicts, defaults, etc. These concepts are grouped into concepts for test architecture, test data, test behaviour and time. Being a profile, the U2TP seamlessly integrates into UML. It is based on the UML 2.0 meta-model [2] and reuses UML 2.0 syntax. The U2TP concepts are structured into:

- Test architecture concepts defining concepts related to test structure and test configuration, i.e. the elements and their relationships involved in a test,
- Test behaviours concepts defining concepts related to the dynamic aspects of test procedures and addressing observations and activities during a test,
- Test data concepts defining concepts for test data used in test procedures, i.e. the structures and meaning of values to be processed in a test, and
- Time concepts defining concepts for a time quantified definition of test procedures, i.e. the time constraints and time observation for test execution [9].

A detailed structure of U2TP concepts is given in Table 1.

Table 1. Overview of the Testing Profile concepts [8]

Architecture concepts	**Behaviour concepts**	**Data concepts**	**Time Concepts**
SUT	Test objective	Wildcards	Timer
Test components	Test case	Data pools	Time zone
Test context	Defaults	Data partitions	
Test configuration	Verdicts	Data selectors	
Arbiter	Test control	Coding rules	
Scheduler			

In [4], the meta-model of U2TP is also introduced and explained. It is the source meta-model for the transformation and hence a basis for defining the mapping rules as well as to develop source test models being transformed. It is the input for our transformation work.

Although Eclipse provides EMF, the UML2 plug-in of Eclipse [19] and the profiling mechanism of this plug-in for extensions of UML require that the U2TP meta-model is written in Java from scratch. The UML2 plug-in is based on the UML 2.0 meta-model [2] but provides a specific realization of this in the context of EMF. It allows us to develop a U2TP plug-in for Eclipse and to integrate it with the TTCN-3 plug-in for Eclipse [18].

3.2 TTCN-3 and Its Meta-model

The Testing and Test Control Notation version 3 (TTCN-3 [5]) has been developed at the European Telecommunication Standardization Institute (ETSI) and has been also standardized at the International Telecommunication Union (ITU-T). TTCN-3 is a test specification and implementation language to define test procedures for black-box testing of distributed systems. It enables tests execution, if appropriate tools and system under test (SUT) are available. In [10] a meta-model for TTCN-3 is provided, which represents the concept space of TTCN-3 and enables the use of TTCN-3 in the context of meta-modelling, repositories and model transformations.

The main objectives for the development of the TTCN-3 meta-model were:

- The separation of concerns by separating the TTCN-3 concept space and semantics (represented in the TTCN-3 meta-model) from TTCN-3 syntactic aspects (defined in the core language and the presentation formats).
- The ability to define the semantics on concept space level without being affected by syntactic considerations e.g. in case of syntax changes.
- To ease the exchange of TTCN-3 specifications of any presentation format and not of textual TTCN-3 specifications only.
- To ease the definition of external language mappings to TTCN-3 as such definitions can reuse parts of the conceptual mapping from other languages.
- To integrate TTCN-3 tools into MDA based processes and infrastructures[1].

The TTCN-3 test meta-model defines the TTCN-3 concept space with additional support for the different presentation formats. It does not directly reflect the structure of a TTCN-3 modules but rather the semantics structure of the TTCN-3 language definition. It is defined as a single package with concept structures for types and expressions, modules and scopes, declarations, and statements and operations.

The TTCN-3 meta-model is the target used in our transformation and another base for the definition of the mapping rules. Each meta-class of the target meta-model is named applying the same convention: the logical name for the TTCN-3 concept represented by the meta-class being prefixed with "TT" to make the meta-classes easily identifiable as meta-classes from TTCN-3. The meta-model for TTCN-3 language is technically defined in UML by using the Rational Rose tool [22]. The EMF [21] generator provided by Eclipse was used to generate the TTCN-3 repository by the creation of a corresponding set of Java implementation classes from this Rose model.

3.3 Eclipse

The Eclipse Project [19] is an open source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools. It is composed of three subprojects: the Eclipse Platform, the Java Development Tools (JDT), and the Plug-in Development Environment (PDE). The success of the Eclipse Platform depends on how well it enables a wide range of tool builders to build advanced integrated tools. The Eclipse Platform provides building blocks and a foundation for constructing and running integrated software-development tools [20].

We use PDE to create the U2TP plug-in for Eclipse based on the UML2 Project [19]. Additionally, the Eclipse Modelling Framework (EMF) [21] being a modelling framework and code generation facility enables us to build partly the tools based on the structured meta-models. EMF is a Java framework and code generation facility for building tools and other applications based on meta-models defined in EMF. The EMF Ecore defines the meta-model for all the models handled by the EMF.

4 Transformation Approach

We define transformation from U2TP models to TTCN-3 models. Since TTCN-3 provides a direct generation of executable tests we provide by this translation also a direct way towards test code. Based on concrete U2TP specifications the user is enabled to generate TTCN-3 code, to complete the TTCN-3 definitions if needed afterwards and to execute his/her tests finally. The idea is to provide transformation rules which enable to map the concepts on meta-model level. However, the transformation itself is performed on the model level.

The UML 2.0 Testing Profile is targeted at UML 2.0 providing selected extensions to the features of TTCN-3 as well as restricting/omitting other TTCN-3 features. In general, a mapping from TTCN-3 to U2TP is possible but not the other way around. For the U2TP to TTCN-3 mapping, restrictions on U2TP level are necessary that restrict the U2TP definitions to executable models. In the following, we assume U2TP models which can be mapped to TTCN-3. The principal approach towards the mapping to TTCN-3 consists of two major steps. U2TP stereotypes and associations are selected and assigned to TTCN-3 concepts. Afterwards, procedures to collect required information for the generated TTCN-3 modules are defined [5].

In Fig. 4, the specific application of U2TP to TTCN-3 transformation is considered in the general framework of MDA-based testing [11], where platform-independent tests (PIT) relate to platform-independent system models (PIM) and platform-specific tests (PST) relate to platform-specific system models (PSM). We provide in this paper mapping from a more abstract test design in U2TP down to a detailed technical level in TTCN-3.

Afterwards, the generated test code is completed in TTCN-3 and changed into executable test code. The translation from PITs to PSTs for specific target system platforms is not considered in this work. Also, we do not explicitly model the target test platform (and hence the specifics of the test code dealing with technical test platform characteristics) but rely here on the capabilities of TTCN-3 to generate and adapt executable tests by use of the TTCN-3 runtime interfaces (TRI [6]) and the TTCN-3 control interfaces (TCI [7]).

The way of getting the test code from TTCN-3 repository is performed by using a TTCN-3 compiler (e.g. TTthree [18]). After provision of a test adaptor, the tests originally being designed in U2TP can be performed.

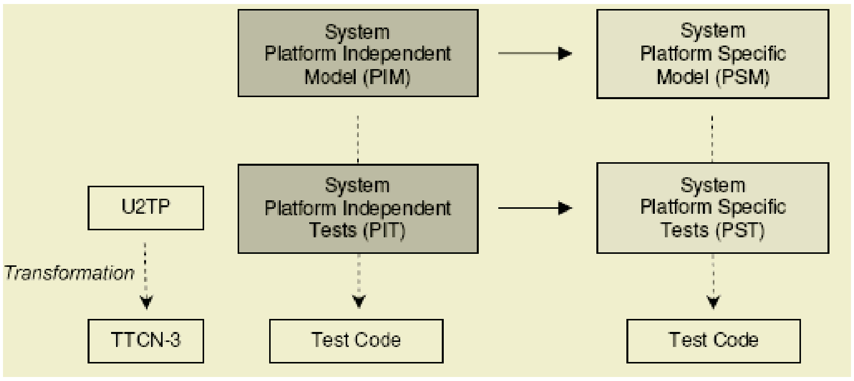

System Platform Independent Model (PIM)
System Platform Specific Model (PSM)
U2TP
System Platform Independent Tests (PIT)
System Platform Specific Tests (PST)
Transformation
TTCN-3
Test Code
Test Code

Fig. 4. Transformation of PIT/PST in U2TP to Test Code in TTCN-3

Mapping rules (provided in the next section) define the connection between appropriate nodes of source and target meta-models. These nodes are stereotypes (and extensions of UML 2.0 meta-classes), primitive types or interfaces in case of U2TP and meta-classes in case of TTCN-3 meta-model.

4.1 Mapping Rules Between U2TP and TTCN-3 on Meta-model Level

Mapping is a mechanism for transforming the elements of a model conforming to a particular meta-model into elements of another model that conforms to another meta-model [12]. Mapping is specified using some languages. The description may be in natural language, an algorithm in an action language, or in a model mapping language. A desirable quality of a mapping language is portability. This enables use of a mapping with different tools [13].

The mapping language used in this paper is developed by us. A transformation rule represents the basic unit of mapping between an arrangement of source elements and an arrangement of target elements [13]. Transformation rules are used in our case to express mappings from concepts of the U2TP meta-model to concepts of the TTCN-3 meta-model. For example, we map each U2TP TestComponent stereotype to the TTComponentType meta-class of TTCN-3. Such a procedure is needed for each element of the source meta-model. Thus, we map each stereotype, interface, primitive type, as well as properties, operations and parameters to appropriate meta-classes and associations of the target meta-model. We used the comparison provided in [4], Chapter 6.6.2 as the base for developing the transformation rules and concretized and completed these rules (Table 2), which defines the semantic relation between U2TP elements and TTCN-3 meta-model elements. Mapping rules provided at this part present selected, the most important, however relatively simple issues. The meta-classes of source meta-model have a correspondence in target meta-model. Hence, concrete mapping rules between elements of U2TP and TTCN-3 meta-models are provided.

Table 2. Relation between U2TP and TTCN-3 meta-model elements (excerpt)

U2TP Element	**TTCN-3 Meta-model Element**
SUT	*system association of TTTestcase* *TTVariable*
TestComponent	*TTComponentType*
TestCase	*TTTestcase*
TestContext	*TTModule* *TTComponentType for the MTC type*
TestConfiguration	*TTFunction* *TTPortLinkKind* *TTCreateTC* *TTStartTC*
TestObjective	*TTComment*
Arbiter	*TTComponentType* *TTExternalFunction or TTFunction*
Verdict	*TTVerdict*
ValidationAction	*TTExternalFunction or TTFunction*
Default	*TTDefaultType* *TTAltstep*
DefaultApplication	*TTDefaultKind*
Stimuli	*TTOutputKind*
Observation	*TTInputKind*
Coordination	*TTOutputKind* *TTInputKind*
LogAction	*TTLog*
InteractionOperator(alt,determAlt)	*TTAlternative*
InteractionOperator(loop)	*TTLoopKind*
DataSelector	*TTExternalFunction or TTFunction*
DataPartition	*TTExternalFunction or TTFunction,* *TTTemplate*
CodingRule	*TTWithKind*
LiteralAny	*matching/expression*
Timer	*TTTimer,* *TTTimerType*
StartTimerAction	*TTTimerStatementKind*
StopTimerAction	*TTTimerStatementKind*
ReadTimerAction, TimerRunningAction	*TTTimerOp*
TimeOutAction, TimeOut, TimeOutMessage,	*TTTimeOut*
Duration	*TTFloatType*
Port	*TTPort , TTPortKind, TTPortType*
Parameters	*TTModuleParameter*

All the mapping rules presented above are connected mostly with single concepts. However, there are such elements like time zone or scheduler that cannot be transformed one by one. The time zone concept cannot be directly expressed in TTCN-3 so that it has been not yet considered in the mapping. Furthermore, it is assumed that the scheduler is implicitly present in the TTCN-3 semantics and therefore realized by every TTCN-3 run time environment, so that there is no need to transform it.

Further investigations in the context of U2TP diagrams are done. The attention is focused especially on Class Diagram, Sequence and Interaction Diagrams. Prototypical implementation of the transformations serves as reliable proof of described concepts. Here, appropriate algorithms to order mapping of various elements are investigated. Different approaches for each type of UML 2.0 diagrams are elaborated.

5 An Example

Hence we would like to introduce an example of diagram mapping so as to show how the mentioned U2TP meta-model concepts can be mapped to TTCN-3 meta-model concepts. In the example, we show how the particular elements of U2TP given in Fig. 5 are mapped to TTCN-3 meta-classes on the base of a Sequence Diagram.

The **sequence diagram** in Fig. 5 specifies the behaviour for *InvalidPIN()* Test Case. The test objective of this test case is: Verify that if a valid card is inserted, and an invalid pin-code is entered, the log with the content *"PIN incorrect"* is stored.

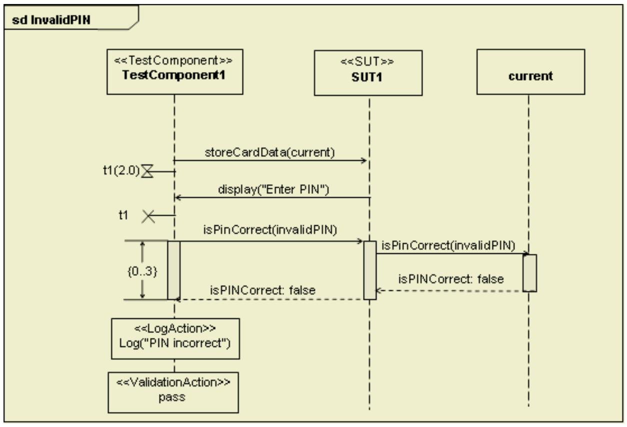


Fig. 5. Sequence Diagram – the behaviour of the *InvalidPIN* test case

The interaction specifies the expected sequence of messages (Stimuli – e.g. *storeCardData(current)*, Observation – e.g. *display("Enter PIN")*) between *Test Component1* and *SUT1*, when used as a test behaviour. During a Test Case, *Log("PIN incorrect")* is used to store log event information. Validation Action sets the verdict to pass. Validation Actions use an arbiter to calculate and maintain a verdict for a Test Case. Test Cases always return verdicts. This is normally done implicitly through the arbiter and doesn't have to be shown in the test case behaviour. In the example, an arbitrated verdict is returned implicitly.

The diagram also illustrates the use of a Timer – *t1* and a duration constraint (*{0..3}*). The Timer is used to specify how long the Test Component1 will wait for the Observation. Thus the Timer – *t1* is started after sending a Stimulus by *Test Component1* to *SUT1*. Once the message (Observation) has been received by the *Test Component1*, the Timer is stopped.

Mapping rules given below are extended in such a way that the whole path of the inheritance of TTCN-3 meta-classes is given. In this way better overview on the meta-models structure is presented. Additional restrictions, like associations are provided to enable the recognition of some important relations in the TTCN-3 meta-model.

Let us consider `TestComponent` stereotype, which is used for the creation of test components and their connection to the SUT and to other test components. It specifies *TestComponent1* in Fig. 6. `TestComponent` is mapped to `TTComponentType`. `TTComponentType` has a `TTScope` which is an abstract meta-class in the TTCN-3 meta-model. It is also a `TTComplexType`, which inherits from `TTType`. `TTType` inherits from `TTDeclaration` and this respectively is associated with `TTModule` which inherits from `TTScope` (see Fig. 6).

```
U2tp::TestComponent →
TTCN3::TTScope::TTComponentType
TTCN3::TTScope::TTModule@TTDeclaration::TTType
::TTComplexType::TTComponentType
```

Fig. 6. Test Component mapping

Symbols used in the creation of mapping rules are given in Table 3:

Table 3. Symbols and their Meaning used in Mapping Language

Symbols	Meaning
`meta-class1::meta-class2`	inheritance of meta-class2 from meta-class1
`meta-class1@meta-class2`	meta-class1 is composed of meta-class2
`meta-class$enumeration`	enumeration is included in the meta-class as an attribute type
`enumerationExample(value)`	represents the value of given enumeration

Applying the transformation rules to all the concepts presented in Fig. 5, we get the following results. `Stimulus` is the element of U2TP meta-model responsible for sending messages, calling operations, and replying to operation invocations. An element corresponding to `Stimulus` on model level is i.e. *storeCardData(current)* in Fig. 5. Stimulus is mapped to `TTOutputKind(OutputKind_call)` in our example. `TTOutputKind` is included in `TTOutput` meta-class as an attribute type. `TTOutput` meta-class inherits from `TTOtherStatements`, while this inherits from `TTFunctionElement`. `TTFunctionElement` is associated with `TTModule`, which inherits from `TTScope`.

```
U2tp::Stimulus →
TTCN3::TTScope::TTModule@TTFunctionElement::TTOtherStatements
::TTOutput$TTOutputKind(OutputKind_call)
```

Fig. 7. Stimulus mapping

`Observation` means according to U2TP specification - receiving messages (receive), operation invocations (getcall), and operation replies (getreply). An element corresponding to `Observation` in the example is e.g. *display("Enter PIN")* in Fig. 5. Here, observation is mapped into `TTInputKind (InputKind_getreply).`

`TTInputKind` is included in `TTInput` meta-class as an attribute type. `TTInput` meta-class inherits from `TTOtherStatements`, while this inherits from `TTFunctionElement`. `TTFunctionElement` is associated with `TTModule`, which inherits from `TTScope`.

```
   U2tp::Observation →
   TTCN3::TTScope::TTModule@TTFunctionElement::TTOtherStatements::TTI
ntput$ TTInputKind(InputKind_getreply)
```

Fig. 8. Observation mapping

For the time-quantified control of the communication between two components, a `Timer` is used. The U2TP stereotypes `StartTimerAction` and `StopTimerAction` are responsible for *t1(2.0)* starting and *t1* stopping (see Fig. 5). They are mapped to `TTTimerStatementKind(start, stop)` respectively. `TTTimerStatementKind` is included in `TTTimerStatement` meta-class as an attribute type. `TTTimerStatement` meta-class inherits from `TTControlStatements`, while this inherits from `TTFunctionElement`. `TTFunctionElement` is associated with `TTModule`, which inherits from `TTScope`.

```
   U2tp::StartTimerAction →
   U2tp::StopTimerAction →
   TTCN3::TTScope::TTModule@TTFunctionElement::TTControlStatements
   ::TTTimerStatement$TTTimerStatementKind(stop, stop)
```

Fig. 9. StartTimerAction, StopTimerAction mapping

`LogAction` is a stereotype of U2TP. *Log("PIN incorrect")* shows its use in the example (see Fig. 5). TTCN-3 provides a log operation for logging test information in the test trace. The `LogAction` is mapped to the `TTLog` meta-class of the TTCN-3 meta-model. `TTLog` meta-class inherits from `TTControlStatements`, while this inherits from `TTFunctionElement`. `TTFunctionElement` is associated with `TTModule`, which inherits from `TTScope`.

```
   U2tp::LogAction →
   TTCN3::TTScope::TTModule@TTFunctionElement
   ::TTControlStatements::TTLog
```

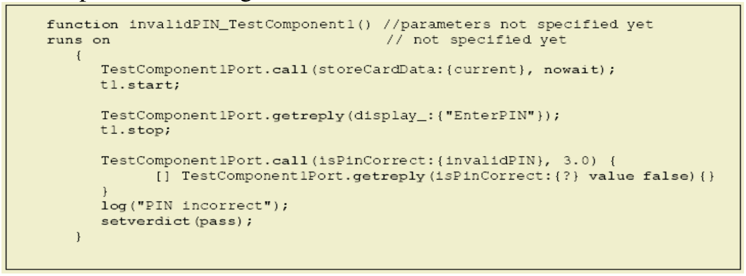
Fig. 10. LogAction mapping

Validation. Action is another stereotype of U2TP. It is an external function resulting in a value of the specific verdict type. It is mapped to `TTExternalFunction` inheriting directly from `TTScope` in TTCN-3 meta-model (see Fig. 11).

```
U2tp::ValidationAction →
TTCN3::TTScope::TTExternalFunction
```

Fig. 11. ValidationAction mapping

The TTCN-3 code created after applying the transformation according to the rules defined above is presented in Fig. 12.



```
function invalidPIN_TestComponent1() //parameters not specified yet
runs on                               // not specified yet
   {
      TestComponent1Port.call(storeCardData:{current}, nowait);
      t1.start;

      TestComponent1Port.getreply(display_:{"EnterPIN"});
      t1.stop;

      TestComponent1Port.call(isPinCorrect:{invalidPIN}, 3.0) {
            [] TestComponent1Port.getreply(isPinCorrect:{?} value false){}
      }
      log("PIN incorrect");
      setverdict(pass);
   }
```

Fig. 12. TTCN-3 code retrieved from the U2TP Diagram

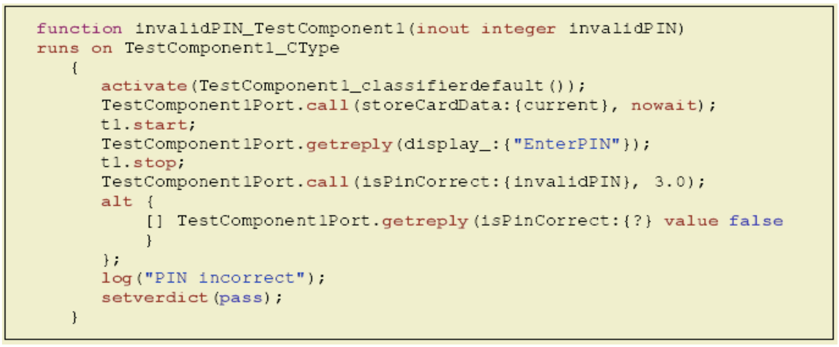
The U2TP test configuration or types definition deserve special attention as they are examples of more complex transformations. Furthermore, for fully specified test cases, all elements of a diagram should be transformed so as to get the whole TTCN-3 code.

In the following, we provide a concrete example of mapping using the whole test specification. Implementation of all the rules mentioned before is the proof of their correctness. We obtained a transformer being able to provide tests in TTCN-3.

Not all the diagrams specified in U2TP are necessary condition to get the full TTCN-3 code. State machine for message flow on one test component presents the same point of view as sequence diagram of the same test behaviour in the context of TTCN-3. Thus, not all available diagrams are used so as to obtain the complete code.

The results of the transformer work for diagram from Fig. 5 are given in Fig. 13.

Behavioural function's name results from test configuration, while body of it is defined alternatively either by sequence diagram or state machine for a test component.



```
function invalidPIN_TestComponent1(inout integer invalidPIN)
runs on TestComponent1_CType
   {
      activate(TestComponent1_classifierdefault());
      TestComponent1Port.call(storeCardData:{current}, nowait);
      t1.start;
      TestComponent1Port.getreply(display_:{"EnterPIN"});
      t1.stop;
      TestComponent1Port.call(isPinCorrect:{invalidPIN}, 3.0);
      alt {
          [] TestComponent1Port.getreply(isPinCorrect:{?} value false
          }
      };
      log("PIN incorrect");
      setverdict(pass);
   }
```

Fig. 13. Transformer Output - TTCN-3 code retrieved from the U2TP Diagram

6 Outlook and Future Work

This paper is devoted to transformation from U2TP test specifications to TTCN-3 code. Transformation rules are defined on meta-model level. Elements of a source U2TP repository (defined by a meta-class of the source U2TP meta-model) are mapped to elements in the target TTCN-3 repository (defined by meta-classes in the target TTCN-3 meta-model). The transformations follow the principles of MDA-based testing, which differentiates between platform-independent tests (PIT), platform-specific tests (PSTs), test code and the relations to the corresponding model artefacts for the system. In particular, a transformation on PIT level is discussed. Selected examples of diagram interactions are provided and the transformation according to the previously defined rules is presented.

The definition of the transformation rules is almost completed. However, special cases of diagrams set are to be considered. This applies especially to test designs specifying Sequence Diagrams, Activity Diagrams or Interaction Overview Diagrams for the same test behaviours at the same time. Additional algorithms should be developed to let the transformation recognise the same behaviour so as not to repeat the same specification in the final TTCN-3 code. Also huge effort must be undertaken so as to map all possible concepts of U2TP, especially such like timezone or scheduler. We have created Eclipse U2TP Plug-in based on the UML2 Project. Furthermore, we developed a tool enabling the transformation and aim to provide the full transformation with graphical front end in future work.

The transformation results provide skeletons of TTCN-3 code only, which means that additional effort must be taken by the user so as to produce complete test definitions. We believe that a fully automated, complete test generation into TTCN-3 will not be feasible in general as test specifications on detailed concrete level contain additional definitions, which are not available in abstract test models. Still, the details of this deserve further investigation. Last but not least, a further aim will be to consider also the generation of PSTs from PITs and/or from platform-specific system models (PSMs). Which of these two ways of transformations towards PSTs should be taken is not clear yet. Research on this will allow us to investigate the relation between platform specifics on system model and test model side.

References

[1] OMG: Model-Driven Architecture (MDA) http://www.omg.org/docs/omg/03-06-01.pdf, http://www.omg.org/docs/formal/02-04-03.pdf

[2] OMG: UML 2.0 Superstructure Final Adopted Specification, www.omg.org/cgi-bin/doc?ptc/2003-08-02

[3] OMG: MOF Query/Views/Transformations, 2nd Revised Submission, ad/04-01-06, 2004.

[4] OMG: UML 2.0 Testing Profile. Final Adopted Specification, ptc/04-04-02, 2004

[5] ETSI ES 201 873-1 V2.2.1: The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, 2003.

[6] ETSI ES 201 873-5 V2.2.1: The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interfaces, 2003.

[7] ETSI ES 201 873-6 V2.2.1: The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interfaces, 2003.

[8] Z. R. Dai: Model-Driven Testing with UML 2.0, Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA'04), Canterbury, England, September 2004.

[9] Z. R. Dai, I. Schieferdecker: Time Concepts for UML 2.0 Based Testing. Workshop on the usage of the UML profile for Scheduling, Performance and Time (SIVOES 2004), hold in conjunction with the 10TH IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004), Toronto, Canada, May 2004

[10] D. Thomas: MDA Revenge of the Modellers or UML Utopia ? IEEESoftware, May/June 2004

[11] I. Schieferdecker, G. Din: A meta-model for TTCN-3. 1st International Workshop on Integration of Testing Methodologies, ITM 2004, Toledo, Spain, Oct. 2004.

[12] M. Born, I. Schieferdecker, O. Kath and C. Hirai: Combining System Development and System Test in a Model-centric Approach, RISE 2004, Luxembourg.

[13] G. Caplat, J.L. Sourouille: Considerations about Model Mapping, Workshop in Software Model Engineering Oct. 2003, San Francisco, USA, http://www.metamodel.com/wisme-2003/18.pdf

[14] A. Kleppe, J. Warmer, W. Bast: MDA Explained: The Model Driven, Architecture–Practice and Promise. Addison-Wesley Pub Co, 2003.

[15] Gross, H.: Testing and the UML – a perfect fit. Fraunhofer IESE, Technical Report 110.03E, 2003.

[16] J. Siegel, OMG Staff Strategy Group: Developing in omg’s model-driven architecture., 2001.

[17] I. Schieferdecker, Z. R. Dai, J. Grabowski, A. Rennoch: The UML 2.0 Testing Profile and its Relation to TTCN-3, IFIP 15th Intern. Conf. on Testing Communicating Systems - TestCom 2003, Cannes, France, May 2003.Eclipse UML2 Project, http://www.eclipse.org/uml2/

[18] Testing Technologies: TTworkbench - TTCN-3 IDE in Eclipse, www.testingtech.de

[19] Eclipse UML2: http://www.eclipse.org/uml2/

[20] Eclipse Platform: http://www.eclipse.org/platform/

[21] Eclipse Modelling Framework: http://www.eclipse.org/emf/

[22] Rational Rose Tool, http://www-306.ibm.com/software/awdtools/developer/datamodeler/

[23] LEIRIOS Test Generator™ tool, http://www.leirios.com/products.php

[24] Objecteering/UML tool, http://www.objecteering.com/news_events_news_oct2002_eclipse.php

[25] P. Leblanc, White Paper, Implementation of the UML Testing Profile and Production of Executable Test Cases, Telelogic France, 2003