DOI: 10.15439/2014F316
ACSIS, Vol. 2

Abstractions on Test Design Techniques

Marc-Florian Wendland
Systems Quality Center
Fraunhofer Institute FOKUS
Berlin, Germany
marc-florian.wendland@fokus.fraunhofer.de

***Abstract*—Automated test design is an approach to test design in which automata are utilized for generating test artifacts such as test cases and test data from a formal test basis, most often called test model. A test generator operates on such a test model to meet a certain test coverage goal. In the plethora of the approaches, tools and standards for model-based test design, the test design techniques to be applied and test coverage goals to be met are not part of the test model, which may easily lead to difficulties regarding comprehensibility and repeatability of the test design process. This paper analyzes current approaches to and languages for automated model-based test design and shows that they are lacking important information about the applied test design techniques. Based on this analysis, we propose to introduce another layer of abstraction for expressing test design techniques in a tool-independent, yet generic way.**

Keywords- Model-based testing (MBT), test generation, automated test design, test design techniques, UML Testing Profile (UTP)

I. Introduction

The degree of automation in industrial software testing was consequently raised within the last two decades. In the 1990s, efforts had been undertaken to increase the degree of automation for test execution, resulting in today's accepted technologies like keyword-driven testing [3]. Standards have been built upon this principle like TTCN-3[1] or the newly developed ISO 29119 [9] standards family. With the wide-spread acceptance of UML in the late 1990s and the advent of UML 2 early 2000s, the idea to automate also parts of the test design activities was pursued in research and industry. The outcome of these efforts is what is known today as model-based testing (MBT) and test generation. Both, automation of test execution and automation of test design rely on abstraction of irrelevant details. Of course, when it comes down to actually test execution, the abstracted details need to be provided, but this is commonly accepted to be pertinent and indispensable. In keyword-driven testing approaches, the so called adaptation layer is in charge of making logical test cases executable [21].

The UML Testing Profile (UTP) [12] is a modeling language for MBT approaches based on the UML. It is the first industry-driven, standardized modeling language for MBT. It was adopted by the Object Management Group (OMG) as far back as 2003 and is currently under major revision. In addition, the European Telecommunications Standardizations Institute (ETSI) has funded efforts to develop its own modeling language for MBT, called Test Description Language (TDL). Thus, two important technical standardization bodies offer languages to build MBT methodologies upon.

Interestingly, none of the above mentioned standards provides concepts to specify the test design techniques that shall be applied for test generation. This seems inconsistent, since one of the most communicated benefits of MBT is automated generation of test artifacts and the increased systematics, comprehensibility and repeatability of the test design process [20]. Until today, there is no generally accepted approach found in the literature how test design techniques for model-based test generation shall be specified the best. In fact, almost every test generator provides its own proprietary configuration for specifying the test coverage goal. This lead to several issues regarding comprehensibility and repeatability the automated test design activities. Moreover, the exchangeability of test generators on models, even of the same modeling language, becomes risky since it bears a great potential for loss of relevant knowledge.

This paper addresses the abstraction from technical, tool-dependent representations of test design techniques by providing an extensible language framework for specifying tool-independent test design techniques that can be shared across multiple test generators. This step is a consequent evolution of the automation through abstraction principle already applied in keyword-driven testing or test generation. The contributions of the work are:

- A thorough analysis of current approaches to model-based test generation.
- The development of a conceptual model of test design based on the ISO 29110 standard. The conceptual model builds the foundation on which the abstractions of test design techniques rely on.

[1] http://www.ttcn-3.org

- The refinement of the conceptual model with an approach to test generation that is motivated by means of directives and strategies.
- Provision of an extensible, yet flexible UML profile-based implementation of the refined conceptual model as an extension to the UTP

The remainder of this paper is structured as follows:

Section 2 describes the problems of today's approaches to automated test design from the viewpoint of comprehensibility and repeatability. Section 3 elaborates the conceptual model of test design and the refinement towards test design directives and test design strategies. Section 4 discusses the extension of the UTP with test design directives and test design strategies. Section 5 demonstrates the feasibility of our approach by applying it to two non-commercial test generators, i.e., the Spec Explorer and Graphwalker. Section 6 presents the work related to ours. Finally, section 7 concludes our work and highlights future work in that context.

II. Problem Statement

Most of the today's model-based test generators are able to work on UML or derivatives. In this paper, we employ the SpecExplorer[2] and the Graphwalker[3] generation engine that do not operate directly on UML, but on closely related concepts. The SpecExplorer input is actually based on a textual representation of an Abstract State Machine (ASM) [7] which is called Spec#. On contrast, the input for Graphwalker is GraphML, a XML format for describing graph structures. Both test generators can operate on graph structures, although the input format is different. These input formats can be derived from UML behaviors, though. In the last years, we have in particular integrated these two test generators with UTP, which allows us to generate the required input format from the very same UTP model for both generators ([23], [22]). Our overall vision is to integrate a wide variety of test generators with UTP to counteract the broadening of proprietary, yet technically incompatible modeling languages.

Hence, the following problem statement was identified in the context of MBT with UTP, so is the technical solution presented in this paper. The conceptual solution, however, is not bound to any particular modeling language.

A. Test Design Techniques in MBT

If we consider the commonly understood advantages of MBT – such as efficient solutions for test design, increasing the degree of automation, prevention of loss of knowledge by using (semi-)formal models, more systematic and, even most important, repeatability of test case derivation and self-explanatory of test specifications ([20],[6]) – MBT comes along with an indispensable change of paradigms for testing activities. The most central artifact in an MBT approach should be the model itself, so test processes have to move from a document-centric to a model-centric paradigm. A model that describes test-relevant information from a tester's point of view is called *test model.* A test model is a "… model that specifies various testing aspects, such as test objectives, test plans, test architecture, test cases, test data etc." [12]. The UTP, in combination with UML, provides a test engineer with numerous possibilities for building test models, since it offers the expressiveness of UML and amends it with test-specific artifacts. Thus, UTP is deemed suitable to support the change to the model-centric paradigm where test models are single source of truth. Even though not as a dedicated concept, UTP allows for modeling the inputs for test generation just by relying on the underlying UML concepts. Inputs to test generation are referred to as test model in the ISO 29119 terminology as well. To avoid confusion with the much broader understanding of a test model given by the UTP specification, we henceforth refer to inputs to test generation as test design models. ISO 29119 defines test design as all activities in a test process that actually derive test cases, test data and test configuration from test conditions. This derivation may be carried out automatically or manually. The term automated test design is commonly known as test generation.

Even though UTP is an expressive language, it does not offer concepts to specify the test design techniques that shall be applied for deriving test artifact. If we consider the before mentioned benefits of MBT, first and foremost test generation, it is most surprising that one of the most important information for automating the test design activities is missing in the test model: The information about which test design techniques shall be carried out on the test design model by the test generator. In other words, the information why a certain test artifact has been generated is not part of the test model. As a matter of fact, today's test generators define their own proprietary presentation of test design techniques that resides within the tool. It complicates, however, the application of an entire model-centric approach to testing, for it does not allow integrating all the required information into the test model. This can have severe implications, since it may easily happen that the applied test generator shall be replaced, for whatever reason, while the defined test design techniques shall be retained. If this happened and access to the previous test generator is not given any longer, the information where a certain test artifact originated from in the first place is lost.

Figure 1 (a) illustrates this problem in a three-layered-approach. The domain layer encodes the test model (more specific, the test design model), which it is completely decoupled from a certain test generator and simply focus on the specification of a system under test. The engine layer, in

[2] http://research.microsoft.com/en-us/projects/specexplorer/
[3] http://www.graphwalker.org

TABLE I.
INVESTIGATION OF TEST GENERATORS

Generator	Input	Output	Configuration	Paradigm	License
Spec Explorer	Spec# (C#)	NUnit Test Cases	Coord Language	1-way (not centric)	Free
MBTsuite	UML Activity / StateMachines	Proprietary	Settings in the tool	1-way (not centric)	Commercial
Conformiq	UML-approximated StateMachine	Proprietary	Settings in the tool	1-way (not centric)	Commercial
Graphwalker	GraphML	Sequence of Strings	Command line parameter	1-way (not centric)	Open Source
Tedeso	UML approximated-Activity	Proprietary	Settings in the tool	1-way (not centric)	Commercial
CertifyIT	UML and OCL	Proprietary	Settings in the tool	1-way (not centric)	Commercial

contrast, is the most fundamental component of a test generator. It is, conversely, completely decoupled from the domain layer and simply operates on its inputs, without taking into account where that input comes from. Both capabilities, complexity and underlying principle of the engine layer vary among test generators from powerful symbolic execution (like the SpecExplorer) to a simple traversal engine that simply operates on already explored graph structures such as finite state machines (like the Graphwalker). Regardless how powerful or sophisticated the generation engine actually is, it is necessary to transform the information encoded in the domain layer into a format understood by the generation layer. A mediator, an adaptation layer, is required. An adaptation layer is a tool-specific component that serves two purposes. First, it transforms (***γ(i)***) the input ***i*** (i.e., a test design model contained in the test model) into a format understood by the test generation engine. Secondly, it offers some kind of interface to the test analyst in order to configure the generation engine. We call this the configuration (***c***) of a test generator. The configuration contains the information of which test design technique shall be applied to the input i. For example, the SpecExplorer is configured with a proprietary language called coord. If the user wants to ensure a certain traversal order of events, he/she has to specify a regular expression over events. This is called a scenario in coord terminology. The semantics of the coord scenario matches with the standardized specification-based test design technique scenario testing in ISO 29119. This information ought to be part of the test model for it contains important test-relevant information to comprehend the automated test design activities. This holds true for other test design techniques and further test generators as well. TABLE I lists a few of the commercially relevant or prominent open source test generators that fit into our view of MBT. Interestingly, all of the investigated generators do offer proprietary means to configure the generation engine. Furthermore, none of these generators really follow the model-centric paradigm, since they simply employ the test design models for the purpose of test generation. There is commonly no feedback of the generated test cases into the test model in order to abide by the single source of truth principle as proposed and described by Wendland ([23], [24]). This is a situation we strive to improve with our work with our work.

B. Abstractions of Test Design Techniques

We propose to abstract from a tool-specific representation to tool-independent representation of test design techniques. Fig. 1 b) illustrates this abstraction. The configuration (shaded grey) are extracted from the tool-specific layer and abstracted (***α(i)***) to test design techniques that are part of the test model itself. The adaptation layer is still required to transform the input ***i*** (i.e., the test design model plus tool-independent test design techniques) into the input format ***γ(i)*** for the generation engine. As such, it is possible to share test design techniques across multiple test generators that provide an adapter for the utilized test design technique. With test design techniques removed from the realm of a specific test case generator and becoming part of the test model a more holistic approach to test design is being provided and the process gains transparency, repeatability and comprehensibility. Such an approach is in-line with the idea of abstraction for test generation as it is done for MBT [18], but for the specification of test design techniques instead of the test design model. This abstraction of test design techniques has not yet been discussed in the literature.

III. A CONCEPTUAL MODEL OF TEST DESIGN

The conceptual field of test design techniques is actually well known. Several academic and industrial literature deals with the application and formalization of test design techniques for different test design models. A good overview is given by Utting [21] and ISO 29119-4 [9]. Based on the concepts and terminology provided in ISO 29119, a conceptual model of test design can be deduced (see Fig. 2) which will be explained in great detail in the subsequent sections.

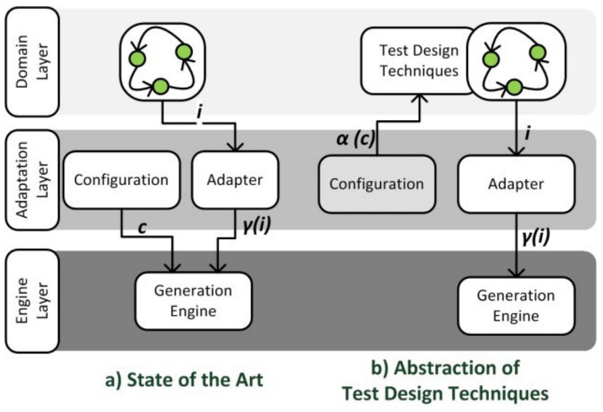

Fig. 1 Abstraction of test design techniques

A. *The Principles of Test Design*

The derivation of test artifacts is usually done by applying a test design technique, or ad-hoc if no systematic approach is applied. A test design technique is a method or a process, often supported by dedicated tooling that derives a set of test coverage items from an appropriate test design model. The test design model is obtained from the identified test conditions. According to ISO 29119, a test condition is a "testable aspect of a component or system, such as a function, transaction, feature, quality attribute, or structural element identified as a basis for testing." A test analyst utilizes the information accompanied with the test conditions to construct the test design model in whatever representation. This gave rise to Robert Binder's famous quote that testing is always model-based [1]. A test design model refers to a specification of the expected behavior of the system under test that is represented either as mental model, informal model, semi-formal model, or formal model.

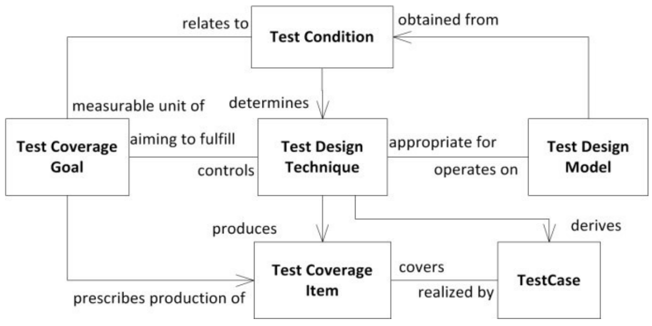

Fig. 2 A conceptual model of test design

As always with models [19] the test design model must be appropriate for the test design technique to be applied. An inappropriate model might not be able to produce an optimal result. The details of a test design model can usually be derived from the test conditions of the test basis.

There is a correlation between the test design technique and the test design model, however, since both are determined or influenced by the test conditions. For example, if the test condition indicates that the system under test might assume different states while operating, the test design model may result in a finite state machine (FSM) or similar. Consequently, a test design technique (like state-based test design) is most likely to be applied on this test design model.

A test design technique tries to fulfill a certain test coverage goal (the term used by ISO 29119 is *Suspension Criteria*, which is actually not that commonly understood). A test coverage goal determines the number and kind of test coverage items that have to be derived from a test design model and represented as test cases. The actual derivation activity might be carried out manually or in an automated manner.

A test coverage item is an "attribute or combination of attributes to be exercised by a test case that is derived from one or more test conditions by using a test design technique" [9]. The term test coverage item has been newly introduced by ISO 29119, thus, it is expected not to be fully understood at first sight. A test coverage item is usually been obtained from the test condition, and made been explicit (in a sense of that it can be used for coverage analysis etc.) through a test design technique. The following example discusses the subtle differences between test condition, test design model and test coverage item:

Let us assume there is a functional requirement that says the following: "If the On-Button is pushed and the **system** is *off*, the **system** shall be *energized*."

The bold words indicate the system under test, the italic words potential states the system under test shall assume und the underlined word an action that triggers a state change. According to ISO 29119, all the identifiable states (and the transitions and the events) encoded in the functional requirement represent the test conditions for that system under test. A state machine according to the test conditions would represent the test design model. As test design technique could be decided to be structural coverage criterion like transition coverage or similar. The test coverage goal would represent a measurable statement about what shall be covered after the test design technique has operated on the test design model. This might be one of Chow's N-Switch-Coverage 0 like full 1-Switch-Coverage (or transition-pair coverage). The test coverage items would eventually be represented by all transition pairs that have been derived by the test generator, and which are finally covered by test cases.

However, there are certain inaccuracies in the ISO 29119's test design concepts which are subsequently classified into three issues.

1) *Test Coverage Calculation*

At first, the term *test coverage*, defined by ISO 29119 as the "degree, expressed as a percentage, to which specified coverage items have been exercised by a test case or test cases", does not take the actual number of potentially available test coverage items into account. According to the given definition of test coverage, the coverage would always be 100% since it is calculated on the actual derived test coverage items. What is missing is a calculation of all test coverage items that actually should be derived. Otherwise, it would be possible to state that 100% test coverage has been achieved, even though just 10% of all to be covered test conditions were actually covered. This is in particular relevant for model-based approaches to test design, for the test coverage items are usually not explicitly stored for further test case derivation, but rather automatically transformed into test cases by the test generator on the fly. This means that in today's model-based test generators the test cases always cover 100% of the derived test coverage items. This is just consequent, since the test coverage items were derived according to a specific test coverage goal, thus, the test design technique only selected those test coverage items (out of all potentially reachable test coverage items) that are required to fulfill the test coverage goal. Ending up in a situation where the eventually derived test cases would not cover 100% of the produced test coverage items would violate the whole idea of specifying test coverage goals.

2) *Output of Test Design Techniques*

Test design techniques do not only derive test cases, but also test data or test configurations. The test design process deals with the derivation of all aspects that are relevant for finally executing test cases. The test configuration or test interface (i.e., the identification of the system under tests, its interfaces and the communication channels among the test environment and the system under test) is a crucial part of each test case, when it comes down to execution. Same, of course, holds true for test data. In this paper we deal not with the generation of test configuration, however, test data generation is covered.

3) *Test Design Techniques Are Not Monolithic*

The concept of a test design technique, as defined and described by ISO 29119, needs to be further differentiated. In relevant, yet established standards for industrial software testing (such as ISO 29119, IEEE:829 and ISTQB) a test design technique is described as a monolithic concept. This is not the case, because the actual test derivation process consists of a number of techniques that represent distinguished course of actions to achieve test coverage. These courses of actions operate in combination with each other to derive the desired test coverage items. Those techniques contribute their semantics to the overall test design activity for a given test design model. Examples for well-known strategies are structural coverage criteria or equivalence partitioning, but also less obvious and rather implicit strategies like the naming of test cases or the physical structure or representation format of test cases. For example, the so called State-Transition test design technique might be based on an extended Finite State Machine (EFSM). Hence, the sole application of structural coverage criteria (like all-transition-coverage etc.) might not suffice to produce executable test cases, for EFSM may also deal with data inputs, guard conditions etc. By adding also data-related techniques (such as equivalence partitioning) to structural coverage criteria, it is possible to explore and unfold the EFSM into an FSM that ultimately represents the available test coverage items for finally deriving test cases. So, the discussion gives rise to the fact that the conceptual model of ISO 29119 regarding test design techniques shall be further differentiated to allow combining several test design techniques with each other in a systematic manner.

B. *Towards Strategies and Directives*

When further differentiating the concept of test design technique, a wider search beyond the field of testing of software systems seems to be appropriate. Based on what was discussed earlier, test design techniques are required to be grouped by different test design process, thus, they are reusable. Test design techniques are consequently decomposed into a thing that groups different techniques and the techniques themselves. This conceptual structure is similar to the Business Motivation Model (BMM) [14] concepts for directives and strategies (see Fig. 3). We adapt the terms directives and strategies for the scope test design.

The BMM provides a fine-grained conceptual model to analyze the visions, reasons and influencers of a business (or endeavor) in order to deduce its overall motivation. The BMM is enunciated in the Semantics of Business Vocabulary and Business Rules (SBVR) [15], a standard which is by the OMG, a standard which is adopted by the OMG to formalize a vocabulary for semantically documentation of an organization's business facts, plans and rules.

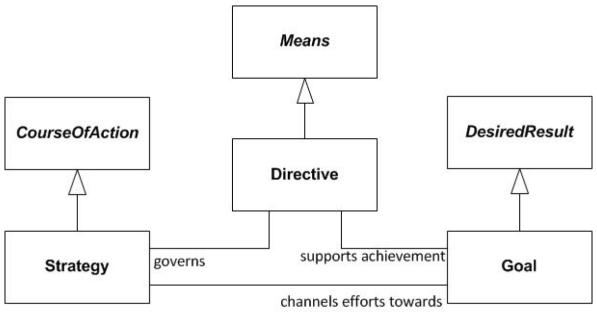

Fig. 3 Relations of Strategy, Directive and Goal

Notwithstanding the motivation for BMM to apply directives is outside of MBT in the first place, the BMM contains concepts and notations that can be beneficial to the realization of test design directives and test design strategies. According to BMM a *directive* is a means to achieve a certain goal. A *goal* is a statement about a state or condition of the endeavor to be brought about or sustained through appropriate *means*. Therefore, a directive (as specialized means) utilizes (a set of) strategies that are governed by the directive to achieve the goal. A *strategy* channels efforts towards the achievement of that goal. This means that the same strategy can be utilized by different directives in order to achieve different goals, hence, strategies are reusable across directives. The notions of strategy, directive and goal can be mapped to the *business* test design. The BMM goal would map almost inherently to the ISO 29119 concept test coverage goal. As with a goal, a test coverage goal imposes a condition on the test design activity that need to be achieved in order to deem the test design activity completed. Since BMM strategies are the actual actions that need to be carried out in a controlled manner, the notation of BMM strategy stands for a single test design technique, such as equivalence partitioning, all-transition-coverage or similar. The directive, however, does not have a direct counterpart in the ISO 29119 conceptual model on test design. From a logical point of view, it is part of the test design technique concept even though not explicitly enunciated. We are going to leverage the notion of strategies and directives for the area of model-based test generation in order to refine the ISO 29119 conceptual model with test design strategies and directives that replaces the monolithic test design technique.

C. Refined Conceptual Model of Test Design

This section mitigates the conceptual imprecisions of the ISO 29119 conceptual model of test design by further differentiating the test design technique into test design directives and test design strategies. These notions are adopted from the BMM. Fig. 4 shows the redefined test design conceptual model in which the monolithic test design technique is split up into test design strategy and test design directive.

A *test design strategy* describes a single, yet combinable (thus, not isolated) technique to derive test coverage items from a certain test design model either in an automated manner (i.e., by using a test generator) or manually (i.e., performed by a test analyst). A test design strategy represents the logic of a certain test design technique (such as structural coverage criteria or equivalence partitioning) in a tool- and methodology-independent way and is understood as logical instructions for the entity that finally carries out the test derivation activity. Test design strategies are decoupled from the test design model, since the semantics of a test design strategy can be applied to various test design models. This gives rise to the fact that test design strategies can be reused across different test design models. This fits with the more general notation of a strategy that can be utilized by several means. The intrinsic semantics of a test design strategy, however, needs always be interpreted and applied within the context of a test design model. According to and slightly adapted from the BMM, this context is identified by a test design directive.

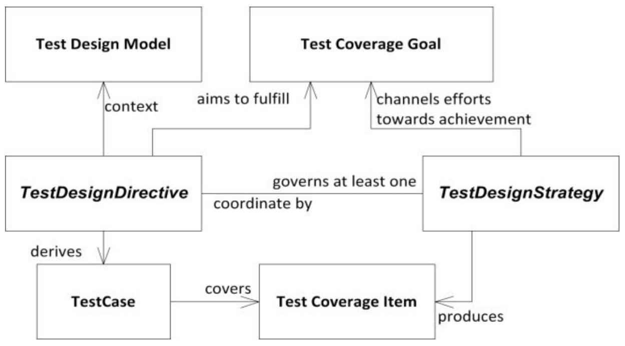

Fig. 4 Redefined conceptual model on test design

A *test design directive* governs an arbitrary number of test design strategies that a certain test derivation entity has to obey to within the context of a test design model. A test design directive is in charge of achieving the test coverage goal. Therefore, it assembles appropriately deemed test design strategies to eventual fulfill the test coverage goal. The assembled test design strategies, however, channel the efforts of their intrinsic semantics towards the achievement of the test coverage goal. The test coverage items that are produced by test design strategies are always fully covered, thus, they are reduced to a pure transient concept. According to Fig. 1 b), the test design directive will be passed to the tool-specific adaptation layer, since it is the test design directive that has access to all required information. At first, it specifies the test design models out of which test artifacts shall be generated. Next, it governs the test design strategies that shall operate on the test design models. In the next sections, we show an implementation of the conceptual model as UML profile.

IV. A LANGUAGE FRAMEWORK FOR TEST DESIGN

The implementation of the refined ISO 29119 conceptual model on test design was from the very first idea incepted as an extension of the UTP. This mitigates one of the most obvious deficiencies of the UTP and allows the creation of fully comprehensible test models. The extension is kept most flexible, so that new test design directives and test design strategies can be easily incorporated.

A. Realization as UML Profile

Since the test design framework has to be kept minimalistic and left open for multiple modeling and testing methodologies, it is important to find a means to not being too intrusive while defining the framework. Fortunately, a UML profile grants all capabilities of MOF-based metamodels, so it is possible to utilize the concepts of derived unions and subsetting properties. Fig. 5 shows the elements of the language framework.

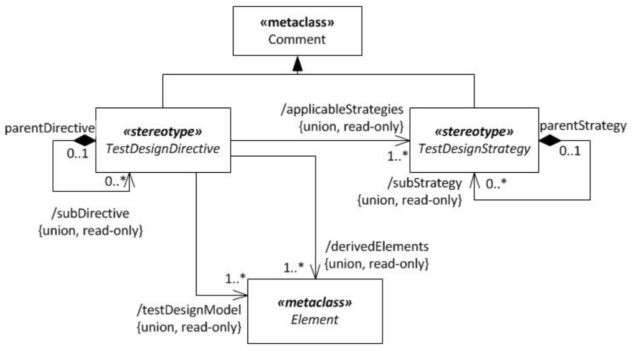

Fig. 5 A UML profile of test design

Both stereotypes test design strategy and test design directive extend the UML metaclass *Comment.* Comment is a semantic-free element of UML, usually used to add notes or documentations to other elements. Applying a test design strategy stereotype onto a Comment turns the Comment into a test design strategy. Same, of course, holds true for test design directive. According to the BMM and the refined ISO 29119 conceptual model of test design, a test design directives establishes associations to one or multiple test design strategies. This means that a test design directive might be composed of several test design strategies. Both test design strategy and test design directive are abstract concepts which means they need to be further specialized in order to be applicable. This is, of course, on purpose, since it is infeasible to foresee each and every test design directive or test design strategy of every existing or new methodology in the future. Thus, the language framework must under all circumstances not restrict a test analyst if he/she wants to define new test design strategies or directives.

A test design directive establishes two further associations to the most fundamental metaclass in the UML metamodel, i.e., *Element*. One association captures the semantics of identifying the allowed test design models for a specialized test design directive, the other represents a trace link to the actually derived elements. The language is again left open as much as possible, because there is no reason why a certain elements of UML should be spared out as test design model or generated element. As with the associations to test design strategies, the specializations of test design directives are responsible to fill these derived unions with reasonable information.

The main benefit and most powerful characteristics of the language framework is the fact that an appropriate tooling can access all available information of any existing specialized test design strategy or test design directive by using the MOF reflection capability at runtime (i.e., modeling time). This enables tool vendors to build a complete and sophisticated tooling on basis of this minimalistic framework without taking care of future extensions. As soon as new specializations of test design strategies and directives are made accessible to the modeling tool, they can be utilized at once.

B. *Libraries of Test Design Strategies*

As already stated in section *Towards Strategies and Directives*, test design strategies bear the potential to be leveraged by different test design directives. It is not possible to navigate from a test design strategy to a test design library explicitly (it has to be said that the MOF capabilities allows the navigation of so called non-navigable association ends – this is an essential precondition for the language frameworks adaptability). A test design strategy actually does not have to be aware with which test design directives it is associated with, since the evaluation and realization of the semantics of a test design strategy in the context of a test design model is done by the adaptation layer of a test generator. On domain level, it is sufficient to limit the combination of test design strategies in the context of test design directives. Since test design strategies are more or less autarkic concepts, it is possible to develop libraries of well-known and accepted test design strategies and to make them accessible to the developer of a new test design directive.

Predefined libraries for test design strategies make in particular sense when it is prescribed (by a standard or company-wide test strategy or policy) which test design strategies are permitted within the test process. By building libraries (and appropriate tool support), test managers or test analyst are able to define a canonical list of test design strategies that shall be exclusively used. This counteracts, for example, violations of such test strategies and fosters, at the same time, consistency among different test design activities of the same test process.

C. *Integration of Test Generator Capabilities*

In order to integrate on the language framework, it is required that each integrated test generator need to propagate its meta-information into the language framework. The most important meta-information [5], of course, is the information about which test design strategies the test generator can realize via its adaptation layer. The language framework provides all required information to enunciate these meta-information. Fig. 6 illustrates the integration of the SpecExplorer and Graphwalker with the language framework.

D. *Provision of Test Generation Services*

As a proof-of-concept, we have combined the language framework with the existing UTP and integrated it into our academic test modeling environment Fokus!MBT [24]. As test generators we employed the previously mentioned Graphwalker and SpecExplorer. As part of the tool integration, we created two test design directives (one for each test generator) with appropriate test design strategies. The result can be seen in Fig. 6.

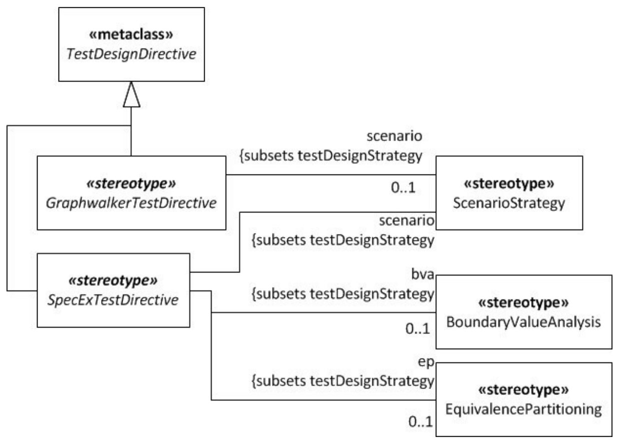

Fig. 6 Definition of test design directives

In Fokus!MBT, a test design directive is considered to identify an accessible test generation service. A test generation service is in its simplest essence a single test generator that is integrated with the language framework. However, a test generation service might also consist of a set of test generators realizing the associated test design strategies. The term test generation service abstracts from the physical representation of the test generation approach.

A test design directive is tightly bound to a test generation service, which in turn comes along with an appropriate adaptation layer. Since Fokus!MBT is based on Eclipse, we utilize the extension point mechanism of Eclipse to identify a test generation service by means of the test design directive. The input validation process is crucial, since only test design models the adapter can map into the input format of its respective engine are allowed to be passed to the engine. In Fokus!MBT, a specialized test design directive can be associated with a set of validation constraints. What is not shown in this picture is the case, if the input validation detects constraint violations. If this happens, the tester is notified about the details of the input validation process.

V. EVALUATION

Let us assume we have an EFSM as test design model that consists of five states and five transitions and a global state variable *x* of type Integer. The according state machine is defined by the following state-transition-table (TABLE II):

TABLE II.
EXAMPLE EFSM

Source	Input (i)	Action	Output (o)	Target
initial	-	x := 0	-	*s1*
s1	i ≥ 0 and i ≤ 5	x:= x+i	x	*s2*
s1	i = 1000	x:= x+i	x	*s3*
s2	i = 3	x:= x-i	x	*accepting*
s3	-	x:= x-5	x	*accepting*

This implies that there are two paths available, which are *sc1* = {*initial→s1, s1→s2, s2→accepting*} and *sc2* = {*initial→s1, s1→s3, s3→accepting*}, where → denote the transition. There are no further guard conditions defined on the transitions for the sake of simplicity. Let us further assume that we want to apply the scenario test design strategy with both the Graphwalker and SpecExplorer engine, where the desired scenario is *sc1*. In addition, we want to apply boundary value analysis, if data generation is possible. So, we define the following two test design directives:

```
SpecExTestDirective 'specex' :=
      testDesignModel = EFSM
      testDesignStrategies :={
       scenario {events = sc1} and
       boundary value analysis {values = 1}
      }

GraphwalkerTestDirective 'gw' :=
      testDesignModel := EFSM
      testDesignStrategies :={
       scenario{events=sc1}
      }
```

The SpecExplorer is capable of generating inputs based on given constraints. The transition *S1→S2* has a constraint defined that the input value must be in a range [0..5]. In Fokus!MBT, range constraints are modeled as UML::Interval. As said before, the configuration of the SpecExplorer engine I s done via *coord*. The respective adaptation layer transforms the test design directive and the referenced test design model into the coord representation. The scenario test design directive is transformed into a SpecExplorer scenario, which is then further used for the exploration. The transformation of the EFSM into the ASM is not part of the paper. Please refer to the DOME case study of the REMICS case study.

In contrast, the *GraphwalkerTestDirective* does not have a test design strategy for boundary value analysis defined, since the generation engine is not capable of generating input values.

After executing bot test generation services, the following test coverage items have been derived:

- SpecExplorer – 2 Test Cases

  #1: *initial→s1(-/-),s1→s2(0/0), s2→s3(3/-2)*
  #2: *initial→s1(-/-),s1→s2(5/5), s2→s3(3/2)*
- Graphwalker – 1 Test Case

  #1:initial→s1,s1→s2, s2→s3

This result is not surprising since the Graphwalker is not able to generate any data at all. Depending on the applied methodology to test generation, this might be not a problem. As a matter of fact, the resulting test cases are mostly of abstract nature, i.e., they are lacking concrete data. These data information need to be applied later on in order to make the test cases executable.

VI. Related Work

The term *test directive* was firstly introduced and defined by [2] in her PhD as "… a collection of test-specific information which, when combined with the system model, derives a test model." The PhD of Dai, however, dealt solely with the derivation of test configurations from existing system models. Therefore, she treated test directives as specifications of single model-to-model transformation rules in a platform-independent manner. The set of combined test directives yielded a complete transformation which generated a test configuration. Our approach, in contrast, deals with the generation of test cases and test data in addition to test configuration.

Friske [5] presented an early idea to specify test coverage goals in generator-independent manner. He leveraged OCL in the context of UML State Machines to define structural coverage criteria like transition coverage or 1-Switch coverage 0. They specified two requirements for the integration of arbitrary test generators into their OCL framework. Firstly, the test generator to be integrated had to follow a two-step generation process. The first step has to generate the test goals (or test coverage items as defined by ISO 29119, the second step to derive test cases from these test goals. The second requirement requests that each generator publishes its metadata regarding its generation capabilities. The idea is quite similar to what we proposed in our work, however, there is no proof-of-concept described that actually realizes the pure theoretically OCL-based approach. Furthermore, it is not clear how strategies for automated test design can be integrated that are not pertinent to the generation of test coverage items (or test goals) but to the derivation of test cases from these test coverage items. For example, the strategies how generated test cases shall be named or finally realized in a compositional sense (decompose test case behavior into several separated pieces of behavior or build monolithic test case behaviors) are essential decision in each test design activity. In our approach, such adjacent strategies can be integrated in the same way as any other strategy. Finally, our experiences with the application of model-based testing techniques in the industry have shown that most testers are not familiar with formal languages like OCL.

Fourneret et al. [4] have presented an approach to model-based security verification and testing of smart-cards based on a dedicated language for security properties. The language is integrated with the UMLsec approach [10] and allows test engineers specifying what a security test generation shall generate. Thus, it enables domain experts to express the strategies for the test generation approach in a non-technical way. A proof-of-concept was done with the commercial CertifyIt [4] test generation tool. The specific test design strategies of that language could be integrated with the test design directives framework in order to make them applicable outside of UMLsec.

Wendland has used of test directives in the context of MBT based on a proprietary metamodel for testing purposes [22]. This work can be seen as the very first approach for an abstracted view on test design techniques based on test directives. Back in those days, the notion of test strategies was not used, though. In contrast, to the work presented in this paper, the former approach tried to develop a modeling framework that was intended to condense the indivisible parts of test design techniques in order to construct a concrete test design technique dynamically. This was much too complex, and yet not applicable.

VII. Conclusion and Future Work

In this paper we had dealt with abstractions of test design techniques in the realm of automated test design based in the context of MBT. We had clearly identified the problem statement that today's approaches to MBT are not that comprehensible, repeatable and flexible as the existing academic and industrial literature in the last decade promised to be. In fact, a full understanding of the automated test design activities within an MBT approach requires access to the applied test generator(s), since they keep the knowledge of the applied test design techniques. This is the main challenge we addressed in this paper. The overall aim is to finally capture all test-relevant information independent of any implementation within the test model itself. This is the only way to ensure a seamless change of paradigms from document-centric to model-centric testing activities. Therefore, we analyzed the conceptual domain of test design compliant with the ISO 29119 standard. We described three issues of the ISO 29119 conceptual model on test design and mitigated them by introducing the notions of test design strategies and test design directives. These pure conceptual notions were subsequently mapped onto a concrete language framework realized as a UML profile. This profile is a most minimalistic realization of the conceptual model and integrates well with the UTP. The reason to go for a UML profile was a natural decision in our work in order to fill the conceptual gap of UTP regarding the specification of test design techniques. Finally, as a proof-of-concept we have described and illustrated how the language framework can be integrated into modeling environments. We therefore used our academic test modeling environment Fokus!MBT[5]. The illustrated proof-of-concept was kept minimal since it was not the scope of the paper to describe a full case study.

The experiences we made and results we obtained from the integration of SpecExplorer and Graphwalker on UTP

[4] http://www.smartesting.com

[5] http://www.fokusmbt.com

and the suggested language framework gave confidence that every of the listed test generators in this paper can be integrated with this approach. Another side-effect of the language framework is that the concatenation of different test generations engines can be achieved rather easily [11].

Future work in the realm of standardization will in particular be channeled towards the new major revision of the UML Testing Profile, which is currently undergoing. As said before, the absence of a facility to specify test design techniques on model level was already complained about. The requirements document of the new UTP version [16] requests precisely such a facility. Our contribution in this area will be minimal language framework presented in section *A Language Framework for Test Design*.

While writing this paper, we are already working on the integration of a usage-based test generator [8] and behavioral and data fuzzing generator [18] with the proposed language framework in the context of the EU project MIDAS[6] w

To conclude the achievements of our work, we have shown that different test generators are able to be integrated on an abstracted representation of commonly accepted (or proprietary) test design techniques. In addition, we think this language framework is flexible to allow for a fine-grained adjustment of applied test design techniques. Our work fills the conceptual gap of MBT approaches in this regards and, thus, allows for a more holistic and comprehensible approach for automated test design based on (semi-)formal models.

ACKNOWLEDGMENT

Most parts of the work presented in this paper were funded by the EU projects REMICS (no. 257793) and MIDAS (no. 318786).

REFERENCES

[1] Binder, R., Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison Wesley, 1999.

Chow, Tsun S.: Testing Software Design Modeled by Finite-State Machines. IEEE Transactions on Software Engineering, Vol SE-4, No. 3, 1978. http://dx.doi.org/ 10.1109/TSE.1978.231496

[2] Dai, Zhen Ru: An Approach to Model-Driven Testing – Functional and Real-Time Testing with UML 2.0, U2TP and TTCN-3. Dissertation at the TU Berlin, 2006.

[3] Foster, M. and Graham, D., Software Test Automation. Addison-Wesley Professionals, 1999. ISBN: 978-0201331400.

[4] Fourneret, E. et al., "Model-Based Security Verification and Testing for Smart-cards," ares, pp.272-279, 2011 Sixth International Conference on Availability, Reliability and Security, 2011. http://dx.doi.org/10.1109/ARES.2011.46

[5] Friske, Mario; Schlingloff, Bernd-Holger; Weißleder, Stephan, Composition of Model-based Test Coverage Criteria, MBEES, 2008. 87-94.

[6] Grieskamp, W., Model-Based Testing in the Field: Lessons Learned, *Lecture Notes in Informatics*, Vol P-94 (2006), Pages 189- 196.

[7] Gurevich, Y., Evolving Algebras, 1993: Lipari Guide, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995. http://dx.doi.org/10.1007/978-3-540-74284-5

[8] Herbold, S., Grabowski, J., Waack, S. (2011). A Model for Usage-based Testing of Event-driven Software. 3rd International Workshop on Model-based Verification & Validation: From Research to Practice (MVV). http://dx.doi.org/10.1109/TSE.2010.12

[9] International Organisation for Standardisation (ISO): ISO/IEC 29119, Software Testing Standard, http://www.softwaretestingstandard.org

[10] Jürjens, Jan, Secure Systems Development with UML. Springer-Verlag, 2005. http://dx.doi.org/10.1007/b137706

[11] Lackner, Hartmut; Schlingloff , Holger, Modeling for automated test generation - a comparison. MBEES 2012, p 57-70, 2012.

[12] Object Management Group (OMG): UML Testing Profile. URL: http://www.omg.org/spec/UTP

[13] Object Management Group (OMG): Unified Modeling Language. URL: http://www.omg.org/spec/UML

[14] Object Management Group (OMG): Business Motivation Model (BMM). http://www.omg.org/spec/BMM

[15] Object Management Group (OMG): Semantics of Business Vocabulary and Business Rules (SBVR). http://www.omg.org/spec/SBVR

[16] Object Management Group (OMG): UML Testing Profile 2, Request for Proposal (RFP), document number: ad/2013-12-08.

[17] Pretschner, A. and Philipps, J., Methodological Issues in Model-Based Testing. In: *Model-Based Testing of Reactive Systems*. Springer, 2004, Pages 281-29. http://dx.doi.org/10.1007/ 11498490_13

[18] Schneider, M., Großmann, J., Tcholtchev, N., Schieferdecker, I., & Pietschker, A. (2013). Behavioral fuzzing operators for UML sequence diagrams. In Ø. Haugen, R. Reed, & R. Gotzhein (Eds.) System Analysis and Modeling: Theory and Practice, vol. 7744 of Lecture Notes in Computer Science, (pp. 88–104). Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-36757-1_6

[19] Stachowiak, H.: Allgemeine Modelltheorie, Springer, Wien, 1973, ISBN-10: 3-211-81106-0. http://dx.doi.org/10.1007/978-3-7091-8327-4

[20] Utting, M.; Pretschner, A., Legeard, B.: A Taxonomy of Model-Based Testing. ISSN 1170-487X, 2006. http://www.cs.waikato.ac.nz/pubs/wp/2006/uow-cs-wp-2006-04.pdf. http://dx.doi.org/10.1002/stvr.456

[21] Utting, Mark; Legeard, Bruno, Practical Model-based testing – A Tools Approach, Elsevier, 2007.

[22] Wendland, M.-F., Großmann, J, and Hoffmann, A., "Establishing a Service-Oriented Tool Chain for the Development of Domain-Independent MBT Scenarios," 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems ECBS 2010, Oxford, England, IEEE, 2010, pp. 329-334. http://dx.doi.org/10.1109/ECBS.2010.47

[23] Wendland, Marc-Florian et al., Model-based testing in legacy software modernization: an experience report, in Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA 2013). JAMAICA'13, July 15, 2013, Lugano, Switzerland. ACM 978-1-4503-2161-7/13/07. http://dx.doi.org/10.1145/2489280.2489291

[24] Wendland, M.-F.; Hoffmann, A., and Schieferdecker, I., Fokus!MBT - a multi-paradigmatic test modeling environment, in Proceedings of the workshop on ACadeMics Tooling with Eclipse (ACME 2013), ACME'13, Montpellier, France, ACM 978-1-4503-2036-8/13/07. http://dx.doi.org/10.1145/2491279.2491282

[6] http://www.midas-project.eu