Fokus!MBT – A Multi-Paradigmatic Test Modeling Environment

Marc-Florian Wendland Andreas Hoffmann Ina Schieferdecker
Fraunhofer Institut FOKUS
Kaiserin-Augusta-Alle 31
10589 Berlin
{marc-florian.wendland,
andreas.hoffmann,
ina.schieferdecker}@
fokus.fraunhofer.de

ABSTRACT
UML modeling environments for doing model-based testing are often not very comfortable to use and burden some knowledge about the internals of UML to the users, respectively test engineers. Test engineers, however, are seldom experts in UML, thus, the gain of efficiency model-based testing approaches entail, is reduced by a too generic tooling. The tool Fokus!MBT, developed by the competence center MOTION of Fraunhofer FOKUS, is a multi-paradigmatic test modeling environment based on the UML Testing Profile, an OMG-adopted industry-driven notation for model-based testing. Fokus!MBT simplifies the creation and authoring of test models with methodology-specific support. It is built on top of Eclipse Papyrus, a powerful open source UML modeling environment, which, in turn, relies on the Eclipse Modeling Framework and the Graphical Modeling Framework. This paper provides deep insights into the basic concepts and technical realization of Fokus!MBT as well as into the lessons we have learned during development and application.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Software/Programm Verification – *validation, tracing, symbolic execution, testing tools.*

General Terms
Design, Standardization, Languages, Theory, Verification.

Keywords
Fokus!MBT, UML Testing Profile (UTP), Unified Modeling Language (UML), Model-Based Testing (MBT), test modeling environment.

1. INTRODUCTION
Following most recent literature and surveys about model-based testing (MBT) and its adoption by the industry ([1], [28]), one has to conclude that MBT is still not widely spread used. One reason might be the lack of commonly agreed standards and methodologies, but also the lacking support of dedicated modeling environments for actually doing MBT. This paper discusses the development and concepts of Fokus!MBT, a test modeling environment based on OMG's Unified Modeling Language (UML) [16] and UML Testing Profile (UTP) [17].

The paper is strictly dedicated to three aspects of Fokus!MBT, i.e., a) its general idea, b) its technical and service-oriented architecture, and c) its methodology-specific authoring support. We are not going to discuss process-related or organizational challenges MBT approaches have to overcome and supporting tools need to address. We are aware of that tooling is just a single component in an enterprise infrastructure, however, the existence of supportive tooling is inevitable for any technical language, standard and methodology.

The remainder of this article is structured as follows: Section II distinguishes methodology-specific UML modeling environments (such as Fokus!MBT) from general purpose UML modeling environments in terms of complexity. Section III provides a rough overview of Fokus!MBT, its key characteristics and its history. Section IV, V, VI and VII discuss in greater detail the technical and logical architecture of Fokus!MBT. These parts constitute the main part of this paper. Section VIII describes the lessons we have learned while both developing and applying Fokus!MBT. Section IX discusses work related to ours. We concentrated on Eclipse-based tools solely. Finally, section X concludes the paper and discusses future work.

Throughout this paper, we write UML metaclasses with a starting upper case, such as Message, MessageEnd etc.

2. DIMENSIONS OF COMPLEXITY
UML is by definition a general purpose modeling language (GPML), thus, UML modeling environments can be seen as general purpose modeling environments (GPME). Prominent representatives of UML GPMEs are MagicDraw, Enterprise Architect (EA), Rational Software Architect (RSA) or Eclipse Papyrus. GPMEs, regardless which language they support, do not impose a certain methodology on the user. They are considered as methodology-independent, whereas they ensure technical compliance to the specification of the GPML.

Fokus!MBT is based on UTP (thus, on UML), but in contrast to the above mentioned GPMEs, it realizes a certain methodology and offers dedicated services and tailored user interfaces (UI) to the test engineers to adhere to that methodology. We call this kind of tooling methodology-specific modeling environment (MSME).

ACME'13, July 01 2013, Montpellier, France

http://dx.doi.org/10.1145/2491279.2491282

In the context of modeling, the term *methodology* needs to be further differentiated. We do see two dimensions for modeling methodologies, i.e., syntactical and semantic methodology. The first one deals with restricting the use of syntactical constructs of a modeling language. In UML, an integer value might be expressed as LiteralInteger, as LiteralString typed by or as InstanceSpecification classified by Integer. Neither of these ways is wrong or right per se. Correctness depends on approach-specific guidelines the user has to stick with. In contrast, semantic methodology specifies how domain-related concepts are expressed with the language. In case of UTP the domain is testing, without being more precise about the kind of systems that are going to be tested. The semantic methodology of Fokus!MBT prescribes how certain test-related targets have to be achieved by using the language. As an example, Fokus!MBT prescribes that each argument of a Message in a test case needs to be an InstanceSpecification of a data partition. This restriction is simply based on a specific testing practice, namely that each stimulus to the system under test (SUT) can be classified into at least one logical partition (i.e., equivalence class).

Both kinds of methodologies entail complexity, we refer to as *technical complexity* (entailed by syntactical methodology) and *methodological complexity* (entailed by semantic methodology). Whereas the complexity of GPMEs is more or less determined by the specification of their GPML (we call *compliance complexity*), MSMEs have to master both technical and methodological complexity, we refer to as *combined complexity*.

The definition of complexity is important to understand the needs for dedicated MSMEs like Fokus!MBT. The main target of any MSMEs is to hide combined complexity from the user while authoring the model, by offering (syntactical and semantic) methodology-specific support. How this is actually achieved varies from tool to tool, of course.

3. Fokus!MBT AT A GLANCE

Fokus!MBT[1] is an integrated test modeling environment that supports test model authoring by guiding the user through methodology-specific support. It utilizes UTP as language for expressing test models. Fokus!MBT is built upon three premises:

1. Adherence to the single source of truth paradigm[2] which manifests in a single test model that encodes and integrates all test-relevant information.
2. Hiding of combined complexity that UTP, UML and the Fokus!MBT methodology impose on the test engineers, so that they can concentrate on their mission-relevant knowledge instead of wasting creativeness and resources by taming test models.
3. Provision of a testing service-oriented architecture for integrating different test service implementations such test case generators etc.

Fokus!MBT's main goal is to provide domain and testing experts with an integrated modeling environment that helps them to perform their work quickly, easily and free of errors. MBT without adequate and decent authoring tool support bears the danger of being rejected by the test engineers [9]. GPMEs like MagicDraw, RSA, EA or Eclipse Papyrus are by intention independent to any methodology. The granted degree of freedom to the user may easily lead to situations where non-UML, but domain experts rapidly get frustrated by the fact that they have to know UML by heart in order to reach their goals in reasonable time. Fokus!MBT strives to offer the same convenience user are used to by today's development environments for Java.

Technically, Fokus!MBT is built on top of Eclipse, the Eclipse Modeling Framework and the Graphical Modeling Framework (GMF). From the very first beginning, it was incepted as an Eclipse-based tool that could be integrated with several Eclipse-based UML GPMEs like RSA or Papyrus. The current version of Fokus!MBT is seamlessly integrated with Eclipse Papyrus, though, due to its openness and accessibility.

3.1 Supported MBT Approach

In the literature, several approaches to MBT can be found. Fokus!MBT supports the so called second generation of MBT approaches [1], which manifests in using a separate test model that is created and authored independently from the system specification, no matter whether it is model-based or document-based (see Figure 1). The advantages and disadvantages of the different MBT approaches are well-described in greater detail by Pretschner [20] and Schieferdecker [26].

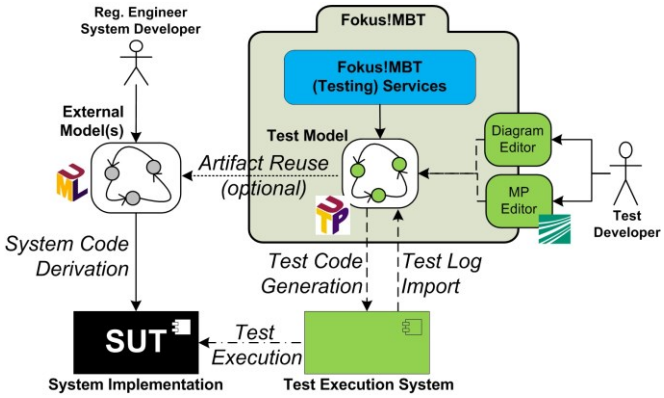

Figure 1. Fokus!MBT overall approach

The separate test model approach allows test engineers doing MBT, even though the development process is not performed in a model-driven manner. If a system model is available and accessible (some companies' process policies rigorously prohibit artifact reuse at testing side), however, Fokus!MBT allows optionally reusing certain aspects of system, or other models. In addition, the single source of truth principle, Fokus!MBT relies on, is shown. The central test model is shared among and enriched by testing services such as test case or test report generators.

3.2 Test Requirements-Driven Methodology

The semantic methodology of Fokus!MBT is highly based on the idea of early testing, i.e., let the testing activities begin as early as possible. Therefore, we employ the concept of test requirements which are derived by the test engineers immediately after the requirements specification has been released. A test requirement is "an item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element."[11] A test requirement represents a very early, mostly textual specification of how an aspect of a system requirement shall be verified. The relationship

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

[2] http://en.wikipedia.org/wiki/Single_Source_of_Truth

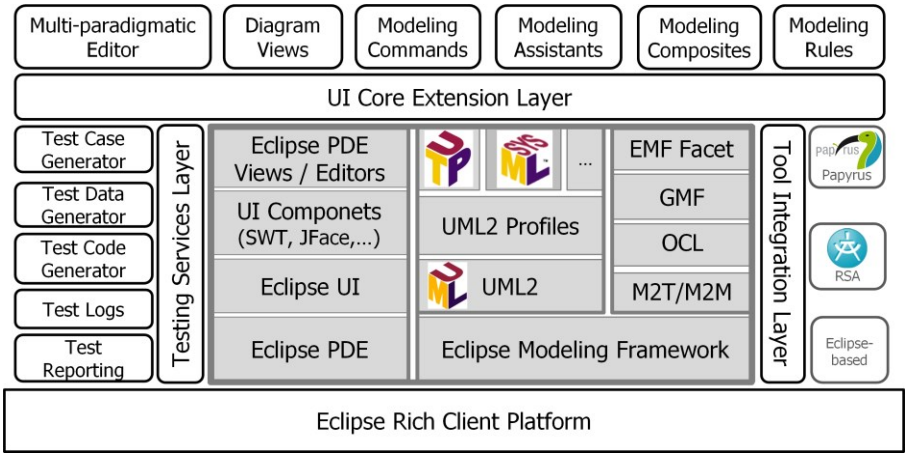

Figure 2. Architecture and technology stack of Fokus!MBT

between system and test requirements is one-to-many, meaning, that a single system requirement is verified by at least one, but usually multiple test requirements. Each of those test requirements specifies a certain condition that must hold true for the corresponding system requirement and therefore serves as the basis for later to be realized test cases.

The test cases, either generated or manually derived, establish traces to the test requirements they realize, which in turn keep traces to the system requirements they verify. After execution of the test cases, which is out of scope of Fokus!MBT, the test execution logs are re-imported into the test model as test logs and eventually linked with the test cases. By doing so, a complete and seamless traceability network is established that grants test engineers and test managers a quick overview of the coverage status of the system's requirements (see Figure 3), the main goal of validation techniques such as testing.

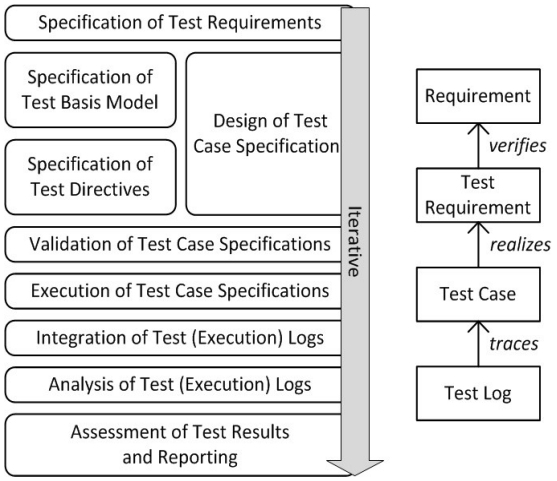

Figure 3. Fokus!MBT methodology

3.3 History

As far back as 2007, Fokus!MBT was originally incepted as part of the work package Advanced Validation & Verification Techniques within the EU FP6 research project Modeling solution for complex software systems (MODELPLEX). It was based on a metamodel for testing purposes (TestingMM) that was a conceptual merge of well-known and established standards like UML, UTP, MARTE [18], SysML [19] and TTCN-3 [4], augmented with some proprietary concepts. A supportive UI was missing, though. Due to the limited support for model authoring, we used MagicDraw to design the test models, which were exported to EMF and finally converted to TestingMM by using a dedicated model transformation. The first generation of Fokus!MBT and TestingMM were successfully applied to industrial research case studies ([30], [23], [31], [24]).

The second generation of Fokus!MBT ([1], [34], [2], [36]) was developed with the beginning of 2010. It was significantly improved by features like GMF diagrams, form-based editors and test log analysis capabilities. With that version, Fokus!MBT started to feel like a real tool that could be used by external domain experts and test engineers [35]. A still restricting shortcoming of TestingMM was its lack of expressivity regarding graph-based behavioral descriptions such as state machines in the first place. As a consequence, the intended single source of truth principle was not attainable, since TestingMM could only be used for expressing the output of a test case generator. At this point in time, we had to decide where to go with TestingMM: either to incorporate graph-based concepts for describing test behavior, or to replace TestingMM with UML and UTP and associated profiles like SysML. The first option would have ended up in a nearly re-implementation of UML with the significant shortcoming of still being technically incompatible with UML tools. The latter one would have had the impact of re-implementing each existing service due to the data model change. Neither of these options was appealing in those days.

At the same time Eclipse Papyrus became more and more stable and accepted by the modeling community. After spending some time on investigating the capabilities, architecture and features of Papyrus, we eventually decided to abandon TestingMM for UTP. This was the starting point for the third generation of Fokus!MBT.

Fokus!MBT has been contributed to and further developed in various research and industrial projects. In the ITEA2 project VERDE [33], Fokus!MBT was applied to two different case studies provided by Alstom and THALES Alenia Space. In the EU FP7 project ReMICS [22], Fokus!MBT is used for safeguarding the modernization of a legacy system that is going to be brought into the Cloud. In the EU FP7 projects MIDAS [14] and RASEN [21], Fokus!MBT is employed for both classical functional testing but also in the area of security testing, in particular model-based fuzz testing.

4. Fokus!MBT ARCHITECTURE

Fokus!MBT is designed to be flexible enough for being integrated into various testing tool and process landscapes, because in most companies tool and process landscapes are already present and not created from scratch [7]. It is, therefore, inevitably that new tools offer generic interfaces for interoperating with other tools employed in the overall process.

Fokus!MBT can be decomposed into a core component that is framed by three logical layers (see Figure 2). The core component relies on the key technologies mentioned in the grey-shaded rectangles and provides fundamental capabilities for implementing and registering test-related services, UI extension services and integration with specific Eclipse-based modeling environments (such as Papyrus). The core component is in charge of guaranteeing that both the syntactical and semantic methodology is respected, this means, it safeguards the overall model integrity. The logical layers encapsulate technologies and concepts specific to concrete services and modeling environments implementations. They are integrated via Eclipse's extension point mechanism. The purpose of these layers is:

1. Testing Service Layer: Fokus!MBT exhibits semantic service interfaces pertinent for doing testing-related tasks such as test case generation.
2. UI Core Extension Layer: Fokus!MBT claims to be highly configurable and adaptable to a test engineer's needs and skill level to overcome combined complexity and foster acceptance. The core component defines several service extension points which realize on the one hand the idea of a multi-paradigmatic test modeling environment, and on the other hand allow tailoring the UI for different purposes and stakeholders.
3. Tool Integration Layer: The tool integration layer encapsulates any modeling environment-specific implementations from the core component. By doing so, the core of Fokus!MBT might be reused across different EMF-/ GMF-based modeling environments.

The details of the architectural layers are described in greater detail in the following sections.

5. TESTING SERVICE ARCHITECTURE

A priori, Fokus!MBT does provides capabilities to author test models, but it does not provide services for model-based test automation like test case or script generation. This might sound strange, since the most obvious and commercial merit of MBT is test generation. The idea was from the very beginning to offer an extensible yet integrated service architecture for testing services that interoperate through the test model. We distinguish five kinds of services relevant for test automation:

- Test case generation: Services for automatically generating test cases. This may include the generation of test data as well. The completeness of the generated test cases depends, of course, on the capabilities of the adopted test generator.
- Test data generation service: Services for generating test data out of structural specifications and constraints.
- Test code generation: Services for generating (executable) test scripts out of test cases.
- Test report generation: Services for generating test reports and documentation out of the test model.
- Test log importer: Services for reintegrating the test execution results that have been produced by a test execution system (such as TTworkbench or JUnit) externally to Fokus!MBT.

In the projects and case studies Fokus!MBT has been applied to, we have developed a set of service implementations for actually doing MBT. These implementations are integrated via the above mentioned testing service layer. The subsequent sections tersely describe these service implementations except the reporting engines, for which we summarize our experiences in the lessons learned section at the end.

5.1 Test Case Generator: Spec Explorer

Fokus!MBT comes along with a test generation service implementation that utilizes Microsoft's Spec Explorer generation engine [12]. The Spec Explorer leverages Abstract State Machines (ASM) and symbolic execution for generating both test cases and test data. The generation process of Spec Explorer is guided by a set of configurations represented in a proprietary notation called *cord*. Fokus!MBT technically abstracts by providing an extension to UTP for test generation configuration called *test design directives*. Fokus!MBT's Spec Explorer service implementation provides a mapping from restricted UML state machines to ASMs together with a supportive UI (in terms of perspective) for developing Spec Explorer-compliant UML state machines.

The service is in charge of automatically carrying out the mapping from the test model to Spec Explorer input files (see Figure 4). Afterwards, the Spec Explorer engine operates on the input files as defined by the test directives. The generated test cases in Spec Explorer format are then translated into what we call Fokus!MBT canonical test case format, an internal representation which is transformed into Fokus!MBT test case diagrams eventually. The entire process appears completely transparent to the test engineers in a senseb that they do not have to know details of Spec Explorer by heart. The generation process is started via a dedicated wizard. Besides, Fokus!MBT offers diagrammatic support to manually create test cases too, of course.

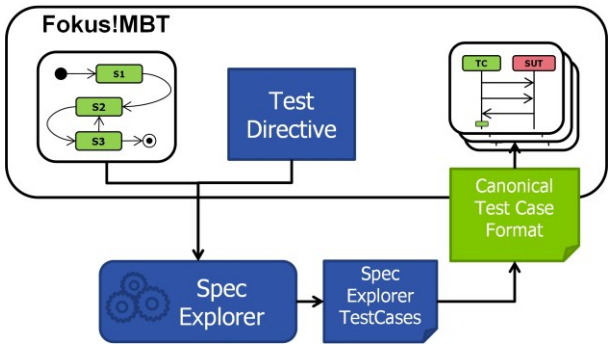

Figure 4. Overview of Spec Explorer test generation process

5.2 Test Code Generator: TTCN3pio

Test cases, either generated or manually derived, can be exported to TTCN-3 test scripts via Fokus!MBT's test code service implementation *TTCN3pio*. It is able to map certain high-level concepts of sequence diagrams deemed necessary for test case specifications like DurationConstraints into their TTCN-3 counterpart, i.e., timer and related actions.

Additionally, TTCN3pio allows emulating the SUT by simply mirroring its expected behavior. In particular when the actual SUT (i.e., the actual implementation under test) is not available, the executability of the test scripts can be proven and demonstrated. We found this capability of TTCN3pio extremely helpful in research projects with any implementation available.

5.3 Test Log Importer: TTCN-3 TCI:TL

After test case execution, which is outside of Fokus!MBT, the test execution logs can be fed back into the test model via Fokus!MBT test log importer service. In UTP, a test log is merely a behavior with additional meta-information, in Fokus!MBT we restrict test logs to be only represented as sequence diagrams.

In accordance with the provided test code generator, we have implemented an importer for test logs in TTCN-3 TCI:TL format [5]. Currently, only the meta-information of a test log, such as starting timestamp, duration and verdict, are fed back. A complete visualization of test logs is still ongoing work. A proof-of-concept was developed in a bachelor thesis [32], though, but has to be further stabilized.

Reintegrating test logs into the test model enables Fokus!MBT to calculate test metrics on and test reports from the test model solely. As a result, the single source of truth principle spans over the entire test process.

6. UI CORE EXTENSION SERVICES

Although the testing services represent the main part for achieving testing-related tasks, users are mainly confronted in their daily work with the UI. Fokus!MBT's UI services are responsible for hiding combined complexity in the first place. This is achieved by introducing a concept we call *methodology-specific context support,* which is highly configurable in order to be aligned with the expectations and skill level of the respective user. Such a UI configuration is persisted in an additional model. The UI configuration model allows different test engineer to create their own UI configuration based on subjective preferences. The UI configuration model contains information about diagrams, editors and editor configuration and further Eclipse workbench-specific things.

As shown in Figure 2, we distinguish several kinds of UI extension services. The subsequent sections describe some of them to a greater extend. Due to page limitations we will not go into details for modeling composites and modeling constraints.

6.1 Multi-Paradigmatic Editor

UML is a graphical modeling notation with just a few textual augmentations. Although diagram-based information visualization is beneficial for some aspects of a test model, especially the behavioral aspects, other aspects like type definitions or test data values are quite cumbersome to be composed in a graphical way. Therefore, Fokus!MBT integrates an multi-paradigmatic editor framework. The editor framework organizes *editor pages* in multiple *editor configurations*. An editor page allows textual or form-based modeling in addition to diagram-based modeling provided by the modeling environment per se.

For example, Fokus!MBT provides editor configurations for test analysis, test design and test result analysis tasks by default. Each page visualizes the information encoded in the test model by using form-based widgets like trees, tables and lists. We call this kind of model authoring *form-based modeling*.

Technically, the multi-paradigmatic editor is based on Eclipse's multi-page editor which is instantiated for a certain editor configuration. Each editor instance dynamically creates the editor pages specified by its editor configuration. Editor page implementations are registered via a dedicated extension point whose identifiers are used within the editor configuration specification in the UI configuration model. This is sketched in Figure 5. When a user changes the UI in terms of brining an editor tab to top, the UI configuration layer gets triggered via Eclipse's internal *IPartListener*. Then, it retrieves which editor configuration was requested to be opened and extracts this configuration from the UI configuration model. The identifiers of the contained editor pages are used for instantiating concrete instances of the editor pages via Eclipse's extension registry. The UI configuration layer creates an instance of the multi-paradigmatic editor and provides it with the retrieved editor pages. Afterwards, the fully configured editor is brought to top as ordinary multi-page editor where each instantiated editor page is represented as section of the multi-page editor.

The multi-paradigmatic editor framework is as generic as possible, since the user can create several editor configurations dynamically by simply composing existing deployed editor pages with each other in different editor configuration. This can be even done while working with Fokus!MBT. The editor configuration dialog allows editor configuration to be modified, created or deleted at runtime.

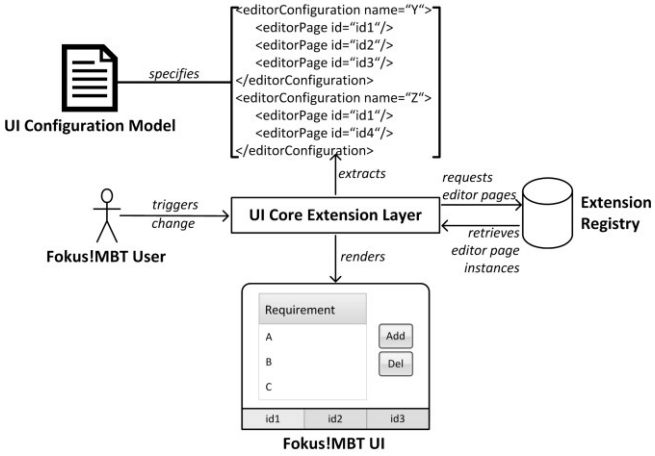

Figure 5. Rendering of editor configurations

6.2 Diagram Views for GMF Diagrams

Fokus!MBT offers an extension point that allows the user to register diagram views based on existing GMF diagram implementations. In a GPML like UML (and, thus, GPME like Papyrus), diagrams commonly offer a multitude of elements to the user. The UML class diagram comprises various structural elements, however, often in specific contexts (e.g., definition of data partitions) only a subset of these elements is actually required. Diagram views are means for reusing the very same diagram (e.g., UML class diagram) in different contexts with different palettes, notational elements and constraints. Fokus!MBT offers certain test-specific diagram views that are based on ordinary UML diagrams:

Test requirement diagram. A class diagram view to create and visualize test requirements and to relate test requirements to system requirements.

Test architecture diagram. A class diagram view dedicated to the specification of parts of the test environment like the test context and test components.

Test configuration diagram. A composite structure diagram view that describes the communication channels among instances of the test environment and the system under test.

Test data diagram. A class diagram view to specify data partitions and representatives of those data partitions.

Test case diagram. A sequence diagram view to describe test cases as Interactions between test components and the SUT.

Spec Explorer diagram. A state machine diagram view for specifying input models for the Spec Explorer test generation engine.

Since diagram views that are based on the same GMF diagram implementation are technically indistinguishable from each other, meta-information about every diagram is persisted in the UI configuration model. The meta-information contains the id of the corresponding diagram view for the available diagram as defined in the extension registry and the number of active palette configurations for the diagram view.

6.3 Modeling Command

To the best of our knowledge, every serious UML modeling environment provides the user with a facility to create new model elements through a context menu corresponding to the current user context. We call this kind of support *modeling commands*. In a GPME, modeling commands are independent to a certain methodology, for they have to be applicable to the generality of methodologies. As a consequence, every suitable metaclass of the GPML according to the current selection is offered for creation ensuring at least specification compliance.

In MSMEs both the syntactical and semantic methodology need to be respected in a given user context. As such, Fokus!MBT comes along with tailored, methodology-specific modeling commands. These commands enable and show only those actions that lead to syntactically and semantically correct models. Modeling commands often consist of multiple tasks the user would have to carry out gradually otherwise, like the application of stereotypes or definition of initial values for mandatory properties.

An illustrative example is the definition of the deletion modeling command. Every GPME allows the user to arbitrarily delete model elements from the model. As said earlier, Fokus!MBT prescribes and safeguards a certain model structure. As a result, some semantically important elements must not be enabled for deletion. Fokus!MBT provides a dedicated modeling command for methodologically correct deletion request by the user. If an element is selected which must not be deleted (such as the outermost package for test requirements), the deletion command is not visible at all.

6.4 Modeling Assistants

Modeling assistants offer a comprehensible and condensed view on aspects of the model that require differentiated knowledge about the underlying language concept. In contrast to modeling commands, they are not bound to a single context, but usually integrate a multitude of interrelated metaclasses for a certain goal to be achieved by a set of user actions.

Fokus!MBT currently offers modeling assistants for the creation of InstanceSpecifications and the configuration of Messages in test case diagrams. Both concepts commonly affect a number of interrelated UML metaclasses like InstanceSpecification, Class, Slot and ValueSpecification or Message, MessagEnd, Lifeline, Value Specification and MessageEvent. Figure 6 shows the modeling assistant for Messages.

In manual test case specification, the user has to configure the signature and arguments of a Message. Both are good examples of combined complexity that needs to be hidden. Until UML 2.3[3], the signature of a Message is derived from the corresponding MessageEnd's event. To manifest a certain signature, the user would have to know about the exact UML specification on the one hand and the actually allowed BehavioralFeatures (Operation or Reception) to be invoked due to the methodology on the other hand. Fokus!MBT's methodology regarding invokable BehavioralFeatures is based on Connectors that are established between Ports (only binary Connectors, though) of different parts within a test configuration and over which Messages have to be sent. Taking this into account, it is only possible to establish a Message between (transitively) connected Lifelines. Furthermore, only BehavioralFeatures that are offered by the receiving Port's, identified by the Connector of the Message, provided interfaces can be invoked via that Message. The Message modeling assistant efficiently abstracts from the combined complexity by simply opening a dialog that shows all possibly invokable BehavioralFeatures to the user for selection.

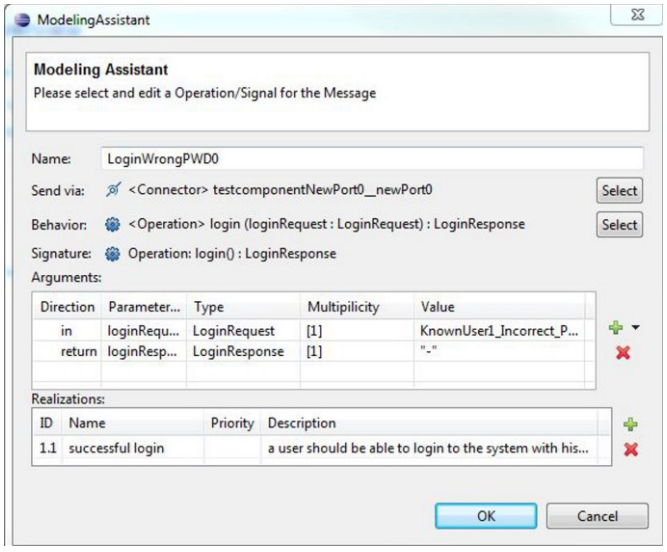

Figure 6. Fokus!MBT Message modeling assistant

The specification of arguments for a Message is another task where the modeling assistant has proven helpful. UML does not prescribe the way arguments have to be specified. Given by the language, any (potentially type compliant) ValueSpecification might be suitable. In contrast, Fokus!MBT requires all arguments to be expressed as InstanceValues referring to an InstanceSpecification of an equivalence class (represented by the UTP stereotype *data partition*). Again, the Message modeling assistant preselects any possible InstanceSpecification and provides them for user-defined selection.

Technically, the modeling assistants are realized as single-page wizards. The difference between a modeling command and a modeling assistant is that the former one commonly affects single

[3] The current version of Fokus!MBT still relies on UML 2.2, but will be migrated to 2.4.1 with Eclipse Kepler.

elements, whereas the latter one is rather a shell for multiple modeling command which are executed in a single transaction.

6.5 Modeling Rules

We consider quality assurance measures like adherence to naming conventions, mutual synchronization of interrelated elements, automated preliminary configuration of elements or model structure clean-ups and so forth as important features for a model-driven engineering environment. We summarize these aspects into the term *automated modeling rules* (*modeling rules* in short). Modeling rules are instructions that are triggered once a model element has been altered. They declare constraints for when they need to be executed and only if these constraints are met after a change has been noticed the corresponding rules will be executed.

Technically, the modeling rules are registered as listeners to the EMF Notifications that are fired when the model was changed. The modeling rules extension point also enables developer to formulate logical expression similar to the Eclipse core expression, but tailored to the conditions of EMF Notifications. Thus, it is possible to specify a condition that evaluates to true if the feature *name* of the notifier *Lifeline* (or of any metaclass that has a meta-attribute *name*) was changed.

Fokus!MBT provides several of these modeling rules to ensure methodological correctness. A simple naming convention of Fokus!MBT that is guarded by such a modeling rule is that the behavioral specification of a test case (which is an Operation with the UTP stereotype *test case* applied) must strictly follow a certain naming convention, which is the name of the test case followed by the suffix *_impl*. This modeling rule will be activated when either the name of the test case or the name of the behavioral specification is changed. In the first case, the name of the behavioral specification is altered accordingly, in the latter case the entire manipulation is rolled back and the old name is re-assigned to the behavioral specification, since the test case is considered to be the master element in our syntactical methodology.

7. TOOL INTEGRATION LAYER

The tool integration layer is in charge of abstracting from any modeling environment-specific technical detail. Fokus!MBT was from the very beginning designed as being able to be potentially integrated in any EMF-based UML modeling environment such as RSA or Papyrus. This gives rise to that any service provided by the core component that operate not only on common parts (such as the UTP test model), but on parts of a specific modeling environment, would have to be kept independent from its technical details. Otherwise Fokus!MBT would maintain hard dependencies to the underlying modeling environment, hence, reuse of the core component would no longer be possible.

Most of these specific technical details come along with the technical architecture of the underlying modeling environment. For one, Papyrus provides a proprietary multi-tabbed editor implementation, whose state (i.e., number of open and available tabs) is maintained in a dedicated model. Any change of the editor (e.g., opening new or switching among tabs) is established via a specific command that operates on the corresponding model. RSA, in contrast, relies on the ordinary editor implementations of Eclipse, thus handling of editors is technically different in both modeling environments. The following example shows how the very same core component functionality, i.e., opening a test case diagram, varies between RSA and Papyrus.

Fokus!MBT allows navigating from an editor page that summarizes all test cases directly to the corresponding test case diagram. In case, the test case diagram has not been created yet, the creation should be triggered automatically, hence transparent to the user. This functionality is modeling environment-specific in bifocal perspective: On the one hand, the GMF implementations of sequence diagrams (as basis of test case diagram) are different in Papyrus and RSA. On the other hand, actually opening a diagram differs technically as well as already explained. However, the editor page that offers the navigation facility is a functionality of the core component and intended to be reused across modeling environments without being recompiled. As a consequence, the modeling environment-specific parts are implemented separately from the core component that actually offers the functionality. To overcome the technical gap, Fokus!MBT splits the (EMF transactional) opening commands into two parts, an abstract part that is implemented in the core component and a tool-specific part that is implemented in the respective tool integration layer. The concrete implementation is then registered against the abstract part and stored declaratively in the OSGI service registry. When the user executes a modeling environment-specific command, the core parts accesses the service registry and requests the specific implementation for the current modeling environment, which is subsequently executed.

8. LESSONS LEARNED

In this chapter we are going to briefly summarize the lessons we have learned while developing and using Fokus!MBT. Due to page limitations, we can only recap the most significant lessons we have learned. We distinguish between lessons we learned while developing and applying Fokus!MBT.

8.1 Lessons Learned from Development

8.1.1 Managing combined complexity

In order to hide combined complexity from the users, both dimensions of complexity need to be precisely identified, analyzed and mitigated by Fokus!MBT. This means that the development team must have a solid knowledge about the underlying technology and methodology. In frequently changing development teams or with a number of part time employees like students (both is common in research institutes), it is a great challenge to keep a solid knowledge about the combined complexity. This holds also true for Fokus!MBT. Most recently, a paper was written [15] that describes a new approach of coping with technical complexity UML entails by masking UML-specific details with a simpler metamodel facade. This is a move in the right direction and we are enthusiastic to adopt this technique for Fokus!MBT in future.

8.1.2 UI core extension layer

During the development of the UI core extension services, we found out that most of these frameworks and concepts are not restricted to Fokus!MBT. The UI core configuration model and the services are actually not directly bound to UML or concrete GMF implementations, but rather operates on EObjects regardless to the surrounding methodology. This gave rise to the idea of extracting the UI core extension services into an independent component that can be reused for tailoring different Ecore-based modeling environments and GMF diagram implementations for being used as MSMEs. Modeling environments we are going to develop in other projects (e.g., for a certain requirements formalization methodology) do not have to start from scratch, but are built upon the common parts of the UI core extension services.

This extraction is a currently ongoing task and might be offered the Eclipse Papyrus project as contribution when it has been finished and stabilized. The tentative name of the UI core extension layer is *Unicorn*.

8.1.3 Complexity of the integration layer

The ability of remaining modeling environment-independent with Fokus!MBT contributes his share to the already present combined complexity of Fokus!MBT. Although the idea seemed wise at the beginning of the development of Fokus!MBT when we were not sure about the future direction of Papyrus. The more Papyrus becomes stable the more we are wondering whether the intended flexibility regarding the underlying modeling environment would ever pay off. In fact, we never had a need so far to build second version of Fokus!MBT for a different Eclipse-based GPML. Doubtlessly, the integration layer implementation makes the overall architecture more complex and we are not yet certain that the complexity it entails will pay off in future.

8.1.4 Processes in a frequently changing team

Maintaining the knowledge about the syntactical and semantic methodology is only one part of the story frequently changing development teams. Clearly structured and well-documented source code and process structures are another vital part in research-oriented development (in commercial development project as well). As often in research prototypes, Fokus!MBT started being developed by as a single person. As the tool constantly matured and was reused by across several testing projects, the development team increased with both full-time employees, part-time employees and students. It holds also true for research projects (thus, also for Fokus!MBT) that the team members are often reassigned other, temporarily more important projects, students leave etc. To not lose control over, respectively precious knowledge about the development, we set up and documented clear process structures and a technical infrastructure for internal team collaboration, including a ticketing system (trac), a continuous integration server (Hudson and Maven), shared team project sets and shared Eclipse run configurations together with shared Eclipse development environments. The aim for doing so was to minimize incidents resulting from varying developing environments and to provide new members a ready-to-use development environment. Especially the ticketing system helped a lot in tracking tasks throughout long periods and the different team members. The team project set was organized in working sets, so that each developer worked on the same working set structure which facilitated team communication. Another important step was to prepare and contribute editor templates to maintain the same coding guidelines within the team.

As helpful as these technical parts have been, as useless was the creation of process documents such as a configuration management documentation. When the team was increased during the development of Fokus!MBT's second generation, we wrote such a document. It contained a variety of aspects such as responsibilities within the team, naming conventions, a detailed description about the plugins and their versions of the development environment and so forth. This document was, however, only valid for a short period, because two important members had left the team, turning most of the documentation obsolete. We decided to not maintain these process documents any longer, though, but rather concentrate on concise technical documentations.

8.1.1 Finding an adequate reporting engine

Generating reports out of models appears as a trivial task. Depending on the completeness, respectively the look and feel of the report, this statement might hold, however, certain requirements on reports are rather challenging to realize. Our first idea was to include the EMF-Adapter for the BIRT engine. This, however, is actually not feasible for Fokus!MBT, since the adapter operates solely on OCL statements for extracting information from the model. These statements quickly became too complex due to the combined complexity of Fokus!MBT models. In addition, we are not aware of how to include GMF diagrams into a BIRT report with the EMF adapter. After a few weeks of investigation we abandoned the BIRT engine for GenDoc2 [29].

GenDoc2 seemed very appropriate, although working inside ordinary Word documents with Acceleo snippets is not very convenient and makes debugging almost impossible. Another severe shortcoming of GenDoc2 is that it simply allows accessing one metamodel per document. Access to any other metamodel (e.g., the GMF Notation model) is established via decoupled services (external functions). This, in turn, makes the implementation of reports more complex. Finally, GenDoc2 was not compatible with Eclipse's UML 2.4.1 implementation. It became so lately, but when we started considering about reporting engines, we skipped GenDoc2 since we were not sure about its further development. So we finally ended up (not yet finished) though in the last possible solution, i.e., we are currently implementing a report directly as M2T transformation for generating HTML files. We had not expected that the generation of decent and sophisticated technical reports would be so challenging.

Once Fokus!MBT will be migrated to Eclipse Kepler and UML 2.4.1, we are confident to re-include GenDoc2 again, since some progress has happened in the meantime around GenDoc2.

8.2 Lessons Learned from Application

8.2.1 UTP bridges the gap

In the VERDE project, we had applied Fokus!MBT to a case study from the transportation domain. Based upon the existing SysML-conform system specification, we built a test model with Fokus!MBT and UTP. A subsequent review session with the use case provider Alstom was very successful due to the fact that the developers were able to easily and quickly comprehend what we actually had specified in the test model. This is a good example of bridging the gap between the development and testing side and of the benefits of UML and its profiling mechanism in general.

8.2.2 Benefits of form-based modeling

The challenge of combined complexity holds true for the user side, too. A still present phenomenon in the industry to our experiences is that test engineers are afraid and skeptic of model-based approaches for doing testing, especially of those that are based on UML. Even within the MBT community, UML or UTP are not widely accepted or trusted for several reasons. Within discussions with the industry, we heard several times that a more concise domain-specific language would be more appropriate than UML. We do not agree with this statement in general. From our point of view, it is necessary to tackle the reluctance to UML by hiding combined complexity (and sometimes even the naming) of UML behind a tailored UI that shows only aspects and names known by the user. This was addressed by the multi-paradigmatic editor framework.

The flexibility the multi-paradigmatic editor framework grants to the user was highly appreciated by the users in past projects. We had developed several of form-based editor pages according to specific needs and wishes of the user. The benefits of form-based modeling are twofold: at first, it abstracts efficiently from combined complexity, and secondly, it allows a skill-oriented presentation of the information encoded in the test model by using well-known form-based widgets.

8.2.3 Flexibility in information visualization

We had once a discussion with a representative from industry about a particular editor page for expressing the logical interface of the SUT. We thought we had already sufficiently abstracted the combined complexity, however, the expert said it would be still too detailed. His argument was that a nurse which is in charge of developing test cases for a new bed occupancy system has very little knowledge about ports. We simply implemented a further simplified editor page with less information and less configurability, but hidden complexity in almost no time.

We share the assumption that the ability of being flexible with respect to information visualization is considered most critical for industrial acceptance and ultimately adoption [8].

8.2.4 Customization of Eclipse Papyrus

The internal architecture of Eclipse Papyrus is very well suited for user-specific customization. We believe that Papyrus has the potential to become the one Eclipse DSL modeling environment in future. The UI of Papyrus, however, is not as flexible as it should be for a GPME. For example, it is not possible to simply switch off entries in the context menus event though they are counteracting the idea of an MSME. For example, Papyrus will always show the elements that can be created in a certain user context. As a side effect, we have to continually explain to users of Fokus!MBT that the predefined commands by Papyrus must not be used in any case since the may corrupt the test model's integrity in terms of syntactical and semantic methodology. This is annoying and prone to errors, because users might be tempted to use the Papyrus commands instead. From a technical point of view, the visibility of any command in a context menu could be adequately managed with the UI configuration model of Fokus!MBT.

8.2.5 Industrial perception of Eclipse Papyrus

We underestimated how the reputation of Papyrus would influence the perception of Fokus!MBT. We encountered that the industry is still skeptic about the robustness and stability of Papyrus. When it came down to adoption of Fokus!MBT, we had several times to argue why we had built that tool on top of Papyrus, since it seemed far away from being a seriously applicable tool. This, however, had significantly changed most recently, since TOPCASED adopted Papyrus in a similar way to Fokus!MBT, i.e., as underlying modeling environment. TOPCASED, in contrast to Papyrus, is already and successfully applied to mission-critical projects. Therefore, we expect Papyrus to be further stabilized and to become a reliable and widely applied UML modeling environment based on Eclipse in the future. This is a confident finding and might lead to the decision to abandon the tool integration layer in future.

9. RELATED WORK

Since the term *MBT tool* is not precisely defined, any tool that supports the idea of MBT can be declared as MBT tool. This holds, of course, true for Fokus!MBT as well. For this paper we consider work as related to ours if they are dedicated to the development of test modeling environments, excluding, for example, work about test case or test data generators.

Representatives of commercial test modeling environments are Conformiq [3], CertifyIT [27] and Automatic Test Generator [10]. All are based on Eclipse and offer methodology-specific support to the user. Each of these tools comes along with a fix proprietary test generator. Fokus!MBT, in contrast, is designed to work with several test generators, if required. Besides ATG, none of these tools commercial are based on UTP.

Related academic toolings are TellingTestStories (TTS) [6] and MATERA [1]. TTS is a tool for a test-driven elicitation process of requirements and provides a test modeling environment for developing activity diagram-based test cases. The tool is also capable of test execution, which is not in scope of Fokus!MBT. TTS does not deal with test generation at all and was optimized for testing service-oriented architectures solely. Fokus!MBT, in contrast, does not restrict the domain it can be applied to per se. An essential difference is that TTS always requires a system model being present for test case generation.

MATERA was a research tool developed as plug-in for the commercial UML GPME magic draw. MATERA is consequently diagram-based, thus, it does not support multi-paradigmatic test modeling as Fokus!MBT. MATERA uses for test case generation the Conformiq engine, hence, it does not allow other test generator being integrated.

Finally, MDTester [13] needs to be mentioned as another solution developed by Fraunhofer FOKUS. This tool, however, was superseded by Fokus!MBT, thus, it is not any more related to but substantially integrated by Fokus!MBT. MDTester was based on a proprietary testing metamodel called Unified Test Modeling Language (UTML) and supported a pattern-based approach to test modeling. Parts of the test patterns specified by MDTester have been integrated into Fokus!MBT.

10. CONCLUSION AND FUTURE WORK

In this paper we have described our tool contribution to the MBT community in form of the test modeling environment Fokus!MBT. Due to page limitation we concentrated to the most significant and illustrative parts of Fokus!MBT, thus, we have emphasized its principles, its technical architectures and its way to overcome combined complexity.

Future technical work will address the extraction of the UI core extension service layer into an autarkic and reusable component called Unicorn. Additionally, we strive to support the idea of engineering viewpoints with Fokus!MBT to further align the UI with the needs of the user. Future conceptual work will address several topics such as application of Fokus!MBT in different contexts like security testing or Cloud testing. We are looking for more industrial case studies to further strengthen the capabilities of Fokus!MBT and to get new requirements in order to address relevant challenges of the industry as soon as possible.

The experiences we made during the development and application of Fokus!MBT have motivated and guided or work on the UTP at OMG. Fokus!MBT acted as kind of a proof-of-concept to prove the feasibility of UTP in real world scenarios. Besides, our work on providing a dedicated UTP modeling environment was an important experience for initiating endeavors at OMG regarding a successor specification. We contributed back to the Eclipse modeling community about 30 to 40 issues to various modeling

projects. Besides, we initiated a new Eclipse modeling project called UML Profile Repository (UPR) together with the Eclipse Papyrus development team. This project strives to provide an Eclipse-centralized repository of standardized UML profiles to prevent uncontrolled growth of semantically equivalent but technically incompatible implementations. We are certain that this project will further foster interoperability of EMF-based UML tools. Finally, we are keen to contribute some of the rather general parts of Fokus!MBT (in particular the flexible UI configuration) to the Papyrus project, because we are convinced that Papyrus would benefit from a more configurable UI.

11. ACKNOWLEDGMENTS

The work on the first two generations of Fokus!MBT was partially funded by the projects MODELPLEX (#IST-34081) and VERDE (ITEA 2~08020). In REMICS (#257793), MIDAS (#318786) and RASEN (#316853) the third generation is/was developed.

12. REFERENCES

[1] Backlund, A and Trusca, D, MATERA – An Integrated Framework for Model-Based Testing, 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems ECBS 2010, Oxford, England, IEEE, 2010, pp. 329-334

[2] Bureck, M. et al., "Model-Based Test Automation with Fokus!MBT and ModelBus," in 1st ETSI Model-Based Testing User Conference 2011, Berlin, 2011. URL: http://www.model-based-testing.de/mbtuc11. Last visit: 6th May, 2013.

[3] Conformiq, http://www.conformiq.com. Last visit: 6th May, 2013

[4] European Telecommunications Standards Institute (ETSI), "ES 201 873-1 V3.2.1 (2007-02): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language," 2007.

[5] European Telecommunications Standards Institute (ETSI), ES 201 873-6 V3.4.1 (2008-09) The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface, 2007.

[6] Felderer, M. et al, Model-driven System Testing of Service-Oriented Systems, in: Proceedings of the 9th International Conference on Quality Software, QSIC 2009, Jeju, Korea.

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

[8] Grieskamp, W., Multi-paradigmatic Model-Based Testing, in: Proceedings of First Combined Internation Workshops, FATES 2006 and RV 2006, Seattle, USA, 2006.

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

[10] IBM Rational Rhapsody Automatic Test Generation Add-on, http://www.btc-ag.com/de/SID-B0230587-9D415B19/3006.htm, last visit: 6th May, 2013

[11] International Software Testing Qualifications Board (ISTQB): ISTQB/GTB standard glossary for testing terms. http://www.software-tester.ch/PDF-Files/CT_Glossar_DE_EN_V21.pdf. Last visit: 6th May, 2013

[12] Microsoft Spec Explorer: http://research.microsoft.com/en-us/projects/specexplorer/. Last visit: 6th May, 2013.

[13] MDTester, http://www.fokus.fraunhofer.de/distrib/motion/utml/, last visit: 6th May, 2013

[14] MIDAS project, http://www.midas-project.eu. Last visit: 6th May, 2013.

[15] Noyrit, F, Gérard, S., and Selic, B., "FacadeMetamodel: Masking UML," in proceeding of: ACM/IEEE 15th International Conference on Model Driven Engineering Languages & Systems (MODELS 2012)ACM/IEEE 15th International Conference on Model Driven Engineering Languages & Systems (MODELS 2012), Insbruck, Austria, 2012.

[16] Object Management Group (OMG), "Unified Modeling Language (UML)," http://www.omg.org/spec/UML/. Last visit: 6th May, 2013.

[17] Object Management Group (OMG), "UML Testing Profile," http://www.omg.org/spec/UTP/., Last visit: 6th May, 2013.

[18] Object Management Group (OMG), "UML Profile for Modeling and Analysis of Real-time Embedded Systems (MARTE)," formal/2011-06-02, http://www.omg.org/spec/MARTE/1.1/PDF/, 2011.

[19] Object Management Group (OMG), "Systems Engineering Modeling Language (SysML)," http://www.omgsysml.org. Last Visit: 6th May 2013.

[20] Pretschner, A. and Phillips, J., "Methodological Issues in Model-based Testing. In: Model-based Testing of Reactive Systems," LNCS 3472, pp. 281 – 291, 2004. ISBN: 978-3-540-26278-7

[21] RASEN project, http://www.rasen-project.eu. Last visit: 6th May, 2013.

[22] ReMICS project, http://www.remics.eu. Last visit: 6th May, 2013.

[23] Sadovykh, A. et al, Architecture Driven Modernization in Practice – Study Results: in *14th IEEE International Conference on Engineering of Complex Computer Systems*, Paris, France, 2009.

[24] Sadovykh, A. et al., "On Study Results: Round Trip Engineering of Space Systems." Fifth European European Conference on Model-Driven Architecture Foundations and Applications (ECMDA) 2009, Twente, Netherlands.

[25] Schieferdecker, I., "Model-Based Testing," IEEE Software, vol. 29, no. 1, pp. 14-18, January/February, 2012

[26] Schieferdecker, I., "Modellbasiertes Testen, " OBJEKTSpektrum 3/07, S. 39-45, 2007 (in German).

[27] Smartesting CertifyIT, http://www.smartesting.com/index.php/cms/en/product/certify-it . Last visit: 6th May, 2013

[28] Spillner, A. et al., "Wie wird in der Praxis getestet? Umfrage in Deutschland, Schweiz und Österreich," *ObjektSpektrum,* May 2011 (in German); www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2011/Testing/spillner_vosseberg_OS_testing_11.pdf Last visit: 6th May, 2013.

[29] TOPCASED GenDoc: http://www.topcased.org/index.php?idd_projet_pere=102. Last visit: 30th December, 2012.

[30] 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.

[31] Stefanescu, A., Wendland, M.-F. and Wieczorek, S., "Using the UML testing profile for enterprise service choreographies." 36th Euromicro Conference on Software Engineering and Advanced Applications, (SEAA 2010), Lille, France, 2010.

[32] Ulrich, S., Formalisierung von Testausführungsergebnissen in Konzepte des UML Testing Profils. Bachelor thesis at University of Applied Sciences Berlin, 2012.

[33] VERDE project, http://www.itea-verde.org/. Last visit: 6th May, 2013.

[34] Wendland, M.-F., "Fokus!MBT - A flexible and extensible toolset for Model-based testing approaches," in 3rd Proceedings of Model-based Testing in Practice Workshop (MoTIP) 2010, in Conjunction with the 6th European Conference on Modelling Foundations and Applications (ECMFA) 2010, Paris, 2010.

[35] Wendland, M.-F., Schieferdecker, I., and Hoffmann, A., "Modellbasiertes Testen mit Fokus!MBT" OBJEKTSpektrum, Online-Themenspecial, 2010. (in German)

[36] Wendland, M.-F., and Bureck, M., "Model-based Test Automation – Fokus!MBT," in 2nd Eclipse Integrated Development Day, Fraunhofer FOKUS Berlin, 2011. http://wiki.eclipse.org/Eclipse_IDD_Berlin2011. Last visit: 6th May, 2013.