Using the UML testing profile for enterprise service choreographies

Alin Stefanescu,
Sebastian Wieczorek
SAP Research, Darmstadt, Germany
name.surname@sap.com

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

Abstract

In this paper we present an approach of using model-driven technologies for testing of service component interactions. We report on an industrial experiment with a novel combination of existing UML standards, i.e., the UML Testing Profile (U2TP), in conjunction with proprietary domain specific languages (DSLs). Many model-based testing (MBT) approaches use the UML 2 standard, but very few of them use also U2TP. Moreover, in practice UML coexists with DSLs which makes the overall integration not easy. We present our experiences and challenges of a U2TP-enabled MBT approach for a DSL for enterprise service choreographies, which describe the communication protocols between service components. The proposed workflow directly translates choreographies into UML models augmented with U2TP stereotypes, which are further loaded into our FOKUS!MBT tool chain. The tool provides an implementation of the U2TP standalone meta-model along with test case and test data generators to describe a holistic test process within one dedicated meta-model for testing concerns.

1. Introduction

Model-based engineering is a recent approach for software design and development aiming to exploit high-level models to increase both productivity and quality. Model-based testing (MBT) is a recent approach that exploits models of the system under test (SUT) for the automated generation of test suites [21]. Whereas the current de facto standard for modeling is UML [15], there is no clear standard in the area of test modeling, i.e., a variety of test modeling languages coexist, making the language choice a difficult problem [9]. This problem would be mitigated if a test modeling standard would gain adoption.

Given the widespread popularity of UML, it was a natural step from the OMG[1] to support and standardize a UML profile for testing concepts [16]. U2TP was finalized in 2005 in an effort to add test specific concepts to UML 2.0 and its subsequent versions. However, since its release five years ago, we could not track a high industrial adoption of U2TP (we only identified two commercial tools supporting it [10,20] and an Eclipse implementation TPTP[2]) or a high interest in the academic communities (except from the contributors to the standard [1,4,3]). One reason could be that UML diagrams were already used as test models in different MBT commercial tools (e.g., QTronic[3] or TestDesigner[4]) and in academia [7,8], but with an implicit or ad-hoc understanding and semantics of the test configuration and directives. Given the potential of U2TP to be used together with UML models and the long term benefits of using a standard for test modeling, we decided to pragmatically investigate how U2TP could be applied in the new area of service engineering, focusing in this paper on enterprise service integration testing.

The recent service-oriented architecture (SOA) paradigm is gaining a quick adoption in a globalized and interconnected market, by promoting standardized interaction across industries via service consumption and composition. However, the flexibility and "open-world assumption" of service integration pose new challenges to the classical testing methodologies due to the lack of control or high dynamicity of the service-based applications [2]. In our case study, we address the service integration testing at the level of service choreographies, i.e., the high-level communication protocols between different enterprise components. Together with service orchestrations, choreographies are essential modeling blocks in Enterprise SOA [23]. Being the leader in the area of business software, SAP

[1] Object Management Group (OMG) - *http://www.omg.org*
[2] Eclipse Test & Performance Tools Platform (TPTP). *http://www.eclipse.org/tptp*
[3] *http://www.conformiq.com/qtronic.php*
[4] *http://www.smartesting.com*

also delivers SOA via its service-enabled software (SAP Business Suite, Business ByDesign) and SOA-based, open technology platform SAP NetWeaver[5]. Moreover, the SAP software development is based on model-driven engineering using a dozen of open but also domain-specific modeling languages (DSLs) [11]. While there are clear benefits of DSLs for internal development for a large software company like SAP, it is also important to have a good bridge to industry standards such that the integration of SAP and non-SAP applications and the communication with customers work seamlessly. Going in this direction, in [19] we presented a DSL for service choreographies called Message Choreography Models (MCM) and a transformation from MCM to UML, which enabled us to apply an MBT tool based on a model execution engine (executing UML state machines annotated with Java snippets). In this paper, we go a step further towards existing standards by adding U2TP information on top of the UML models and consequently we use another tool chain called FOKUS!MBT that supports U2TP natively.

Problem statement and contributions

Above we have identified a couple of areas where improvements are required. Thus, we have seen that:

- the test modeling area is rather heterogeneous and needs more standardization,
- U2TP is a good candidate but industrial application is still not widespread,
- industrial players also heavily use DSLs for SOA,
- more MBT tools supporting U2TP are needed.

Our work will address these aspects by:

- showing a U2TP-based test generation process and its implementation,
- analyzing the strengths and weaknesses of current U2TP release based on the industrial evaluation,
- investigating the feasibility of using U2TP as a test modeling notation starting from a proprietary DSL from service engineering area, and
- presenting an MBT tool chain that relies on the U2TP standalone meta-model implementation and includes a constraint-based test data generator.

The paper is structured as follows. Section 2 discusses the implementation of the U2TP meta-model. Section 3 shows the proposed MBT approach for service choreography models and a concrete example. Section 4 describes the FOKUS!MBT tool chain that integrates test case and test data generators. Section 5 provides the related work and Section 6 concludes the paper.

[5] *http://www.sap.com/platform/soa*

2. U2TP-based Testing Metamodel

The UML2 specification [15] aims to support a model-driven software development process. Even though it is widely adopted in the software industry, UML2 is lacking native testing concepts, like test components, test verdicts, timers, etc. These concepts are relevant for systematic test activities required for the development of high-quality systems. Therefore, a consortium of the OMG specified a test-specific profile for UML2, called *UML2 Testing Profile* (U2TP)[6]. U2TP facilitates model-driven testing approaches and bridges the gap between system and test development by relying on the same notation. More details about the U2TP can be found in [1,3,4,24].

Besides the native UML profile, the U2TP specification provides a standalone MOF-based meta-model, too. This enables test developers to base their MBT approach on an OMG specification without a mandatory usage of UML. However, the standalone meta-model defines only abstract high-level concepts. For example, it defines the metaclass *Behavior* without specifying what kind of behavior it represents. The U2TP overcomes this lack of precision by offering a default *weak-typing mechanism*, where string attributes of the metaclasses may point to external artifacts, providing the required semantics. A more cohesive alternative of implementing the standalone metamodel is to avoid scattering the test-relevant information into separate artifacts by enhancing the metamodel with the precisely defined concepts explicitly. This can be achieved by using class inheritance, where the metaclass *Behavior* might be specialized by the concepts for interactions (e.g., an implementation of UML Interaction package). We refer to this as a *strong-typing mechanism*.

During the MODELPLEX project [8] we specified and realized a Testing Metamodel [14], henceforth referred as *TestingMM* that implements the U2TP standalone metamodel in the strong-typing manner. TestingMM includes a subset of several UML2 packages, in particular *UML::Kernel*, *UML::Component,* and *UML::Interaction*. One disadvantage of this cohesive approach is the complexity of the TestingMM due to the reused UML2 concepts. The TestingMM and the MODELPLEX Testing Metamodel are discussed more detailed in [21,14,22]. Additionally, we implemented the native U2TP profile in order to establish a full U2TP-based test generation process as described in the next sections, starting from annotated UML system models.

[6] There are two official and valid acronyms for the UML Testing Profile: U2TP and UTP.

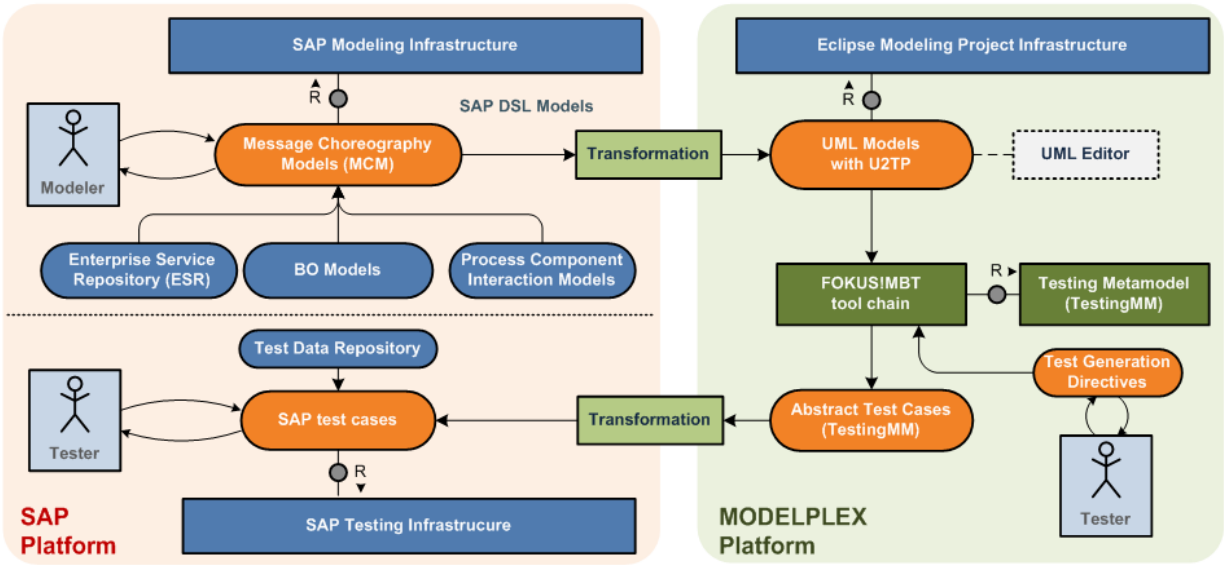

Figure 1. Overview of the proposed MBT process

3. Testing of Service Choreographies using UML+U2TP

During the three years of the MODELPLEX project [13], SAP Research together with the project partners Fraunhofer FOKUS and IBM developed a holistic MBT approach for message choreography models (MCM). The approach was implemented using Eclipse plugins integrated in the Eclipse-based MODELPLEX platform[7] (latest version released in March 2010). More precisely, FOKUS and IBM developed two MBT tools based on UML models, and SAP developed several adaptors and model transformations to export the MCM models into UML, the input format of both MBT tools. Afterwards, the results of the test generation are imported back into the SAP testing framework. The results of the SAP-IBM cooperation were presented in [19]. It focused on the transformations from MCM to UML models annotated with Java snippets, the application of a random test generator based on a dedicated UML execution engine, and the import of the resulted test cases in TPTP format back into the SAP framework. In this paper, we present the results of the SAP-FOKUS cooperation following a U2TP-based approach, which is sketched in Figure 1 (in there, the tag *R* on some transitions means *required*). There are two main differences to the previous approach. First, the U2TP was used on top of the generated UML models and second, a new U2TP-based test generator with test data generation facilities was implemented. The test data generation was not addressed in the previous approach. Moreover, whereas the generated test cases in [19] were in the TPTP format, now they are in the TestingMM format.

Figure 1 shows the overall process, which starts with SAP DSLs [11], i.e., Message Choreography Models that import information from other internal enterprise service models like Business Objects (BO) models, Process Component Interaction Models (PCIM) and service descriptions from an Enterprise Service Repository (ESR). An example of a PCIM is presented in Figure 2 (left), which has a Buyer and a Seller communicating with service-based messages like *Purchase Order Request* or *Purchase Order Confirmation*. An MCM is depicted in Figure 2 (right) provides a state-based model describing the protocol that the message exchange must obey. Although not visible in the figure, an MCM also contains detailed information about the guards for message sending and the effects of message receiving (similar to guards and effects in UML state machines). The message themselves are XML-like structured data. The message triggering is based on BO actions like BO creation or BO updating. There are also more advanced features of MCM which are out of scope here, but were presented in [19].

A transformation of MCM to UML with Java snippets was presented in [19]. A class diagram generated from MCM is presented in Figure 3 (left), e.g., there are classes defined for each of service components Buyer and Seller, but also for each type of available messages. Figure 3 (right) shows a UML state machine generated from a global choreography. The guards and effects of the MCM are now visible on the transitions of the state machine.

[7] *http://www.modelplex-platform.com*

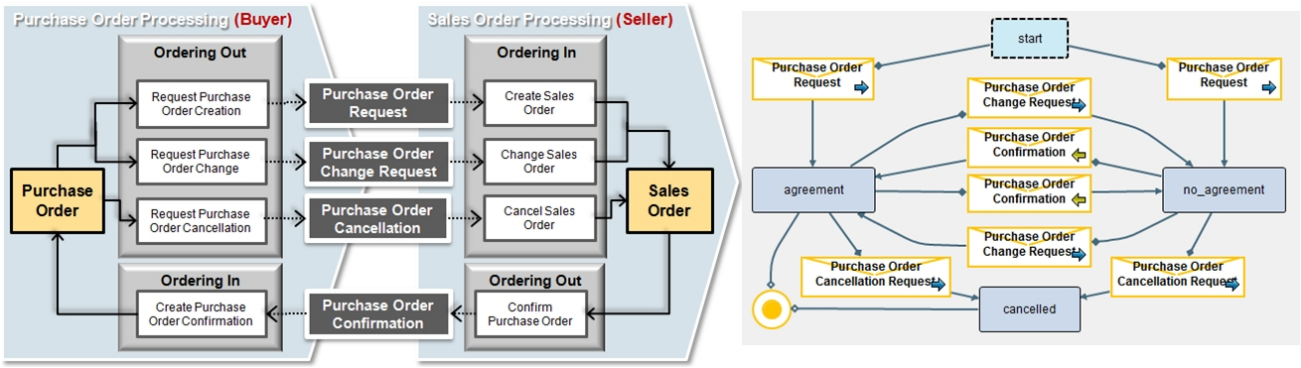

Figure 2. Static information of service components (left) and service choreography (right) of an MCM

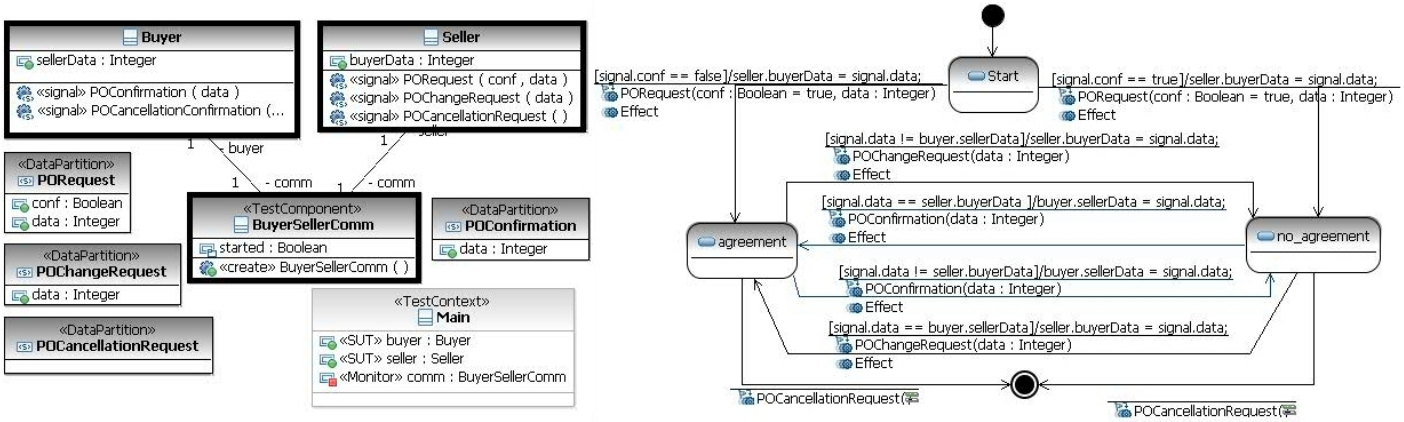

Figure 3. UML diagrams for the use case – class diagram with U2TP stereotypes (left) and state machine (right)

The generated UML entities are annotated with U2TP stereotypes. For instance, the Buyer and Seller instances are tagged as *SUT*, whereas the message classes are tagged as *Data Partitions*. The UML state machine for the choreography is contained in a class called *BuyerSellerComm*, which is stereotyped as *Test Component* (this is not visible in the class diagram). Moreover, since the state machine associated to the global choreography is used for the test generation, it is stereotyped as *Test Case* (note that this stereotype does not imply only one test case, but a possible test suite). The generated test cases including the generated test data from FOKUS!MBT (see Figure 5) are transformed from their TestingMM format into SAP test cases using Ecore tooling from the Eclipse Modeling Framework (EMF). The generated SAP test cases usually need extra test data information to be added by SAP testers in order to trigger the call of some complex services. This is due to the fact that we only partially model the SAP test data pools. However the verification checks, e.g., observing the local states of the components, are performed automatically in the SAP testing framework. Moreover, although the test generation directives (e.g., transition coverage) are applied on the global model, the test cases are executed in the SAP backend directly on the local components.

4. Using the FOKUS!MBT Tool Chain

FOKUS!MBT is a flexible, yet extensible tool chain that facilitates the development of MBT scenarios for heterogeneous application domains. Internally, it is based on the TestingMM as a canonical data model that allows us to store all information of the test process in one source artifact.

Process Overview

FOKUS!MBT allows manual test modeling as well as automated generation of test artifacts from any input source. Currently, a major focus lies on test case generation based on UML system models annotated with U2TP stereotypes, but it is not limited to that. Currently, we focus in particular the generation of test cases out of UML state machines and interactions. The resulting test cases are expressed by TestingMM interactions, which are similar to UML interactions, but not identical.

The main steps of the tool are depicted in Figure 4 and described below:

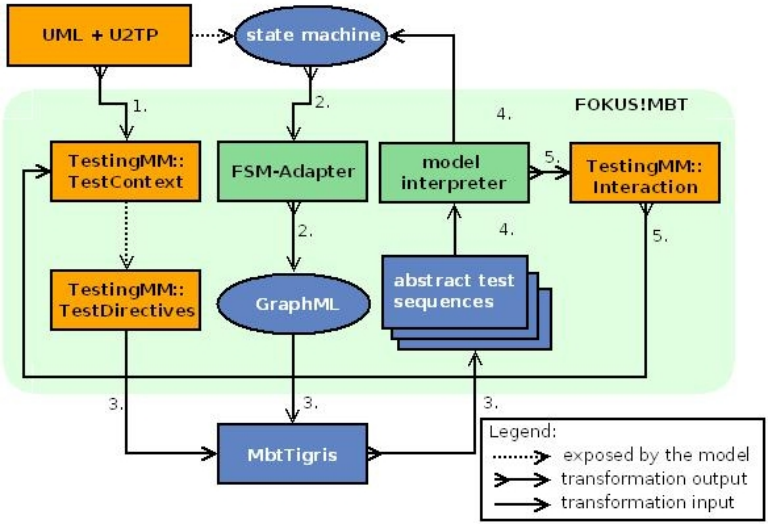

Figure 4. Workflow and dataflow of the FOKUS!MBT tool

1. Translate the structural aspects of the UML model into an appropriate TestingMM architecture.
2. Convert the state machine marked as *Test Case* into an appropriate format (depends on an external test case generator).
3. Invoke the external test case generator to traverse the system model behavior and produce abstract test sequences with respect to the test directives.
4. Generate test data with respect to guards and effects by interpreting the model for the sequences generated in step 3.
5. Transform the set of concrete test sequences into TestingMM interactions.

Generating the Test Architecture

The test architecture defines the structural definition of the test model. Figure 5 (left) shows the test architecture generated out of the class diagram (Figure 3, left). Connections between the components are established via ports. Similar to UML, a port provides and requires a set of interfaces. This is depicted as *provided* and *use* dependencies in the diagram.

The interfaces provided by each component are derived from the UML class diagram. The Buyer and Seller class provide UML reception elements, indicating the possibility to receive a specific signal type. These elements are directly transformed into TestingMM receptions. This information is later used for the derivation of sender and receiver of an exchanged message within the state machine.

FSM-based Test Case Generator

Currently, the test case generation of FOKUS!MBT is based on a tool called *Model Based Testing*[8]. To avoid name clashes, in this paper we refer to this tool as *MbtTigris*. *MbtTigris* finds paths through a given input graph. Natively, it provides a set of structural coverage criteria like state and transition coverage as well as different generation algorithms (like A*, Dijkstra, or random). The required input format for *MbtTigris* is GraphML[9], an XML-based representation for graphs.

Therefore, we have translated the state machine into a GraphML instance (step 2) and configured the traversal engine to achieve the desired coverage goal (step 3). The invocation of *MbtTigris* returns a set of abstract test sequences of fired transitions (step 4). FOKUS!MBT can instruct *MbtTigris* to generate at least a certain number of different abstract test sequences. Such a sequence consists of single events, indicating a message exchange between the components. The set of abstract test sequences are subsequently passed to model interpreter.

Test Data Generation

The model interpreter traces each event of an abstract test sequence in the state machine in order to concretize and transform them into an executable test case. The receiver and sender of a message are derived from the provided receptions of each component's interface.

[8] *http://mbt.tigris.org*
[9] *http://graphml.graphdrawing.org*

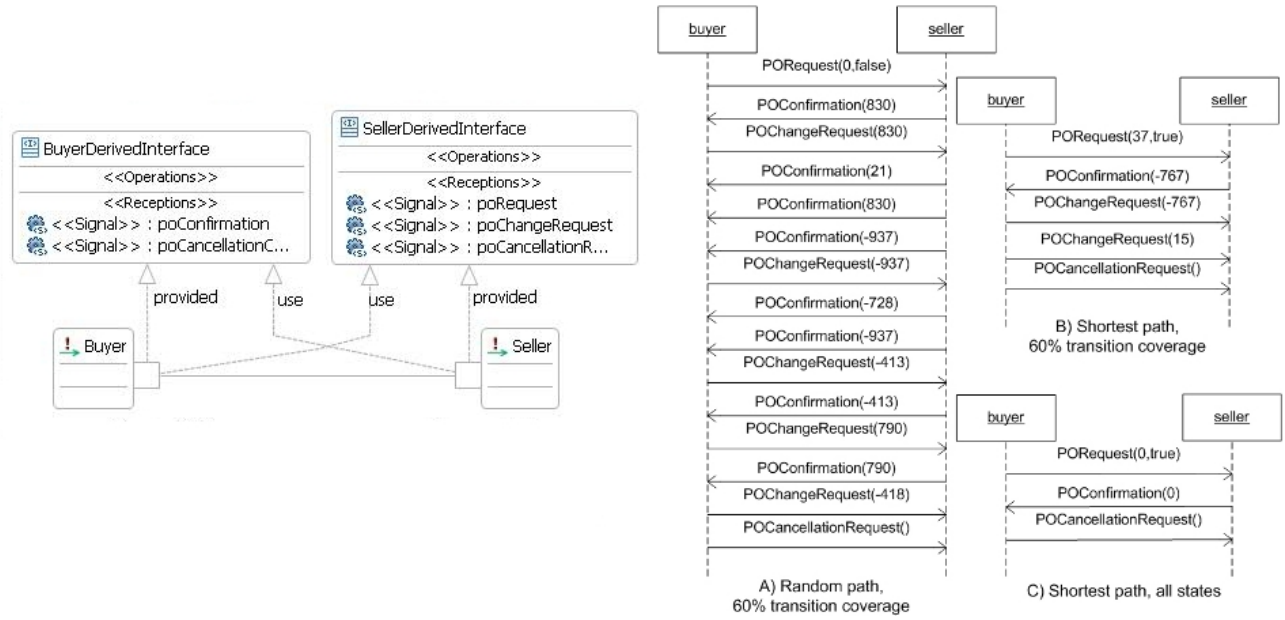

Figure 5. Generated test architecture (left) and test cases (right)

By translating the UML state machine into an FSM, we lost the information of a transition's triggers, guards and effects (as defined in [15]). In order to generate concrete test data, this information must be taken into account in the following order:

1. Creating a trigger event for the transition to be fired. The trigger exposes information about the sender and receiver component.
2. If there are constraints defined for the receiver object, use them to restrict the possible values of the signal attributes, associated with the trigger.
3. If it is a guarded transition, include the guard condition into the signal value calculation.
4. If the transition defines an effect, execute it to manipulate the associated object.

Therefore, FOKUS!MBT integrates a test data generator, which is based on the *Choco constraint solver*[10]. Constraint Programming aims for solving mathematical problems where the user states a problem and the solver searches for valid solutions. *Choco* implements a set of predefined constraints, mostly based on integer values. For example, if the range of *POConfirmation::data* is between -50 and 50 and *Seller::buyerData* has value 3, the test data generator assembles a constraint like (*data <> 3 AND data > -50 AND data < 50*) to satisfy the guard condition *signal.data != seller.buyerData.*

The test data generator transforms the problem description into adequate input for *Choco*. Afterwards, the solver returns a solution to the model interpreter if there is at least one solution, otherwise the whole sequence will be removed from the transformation process, because it represents an infeasible path. If a solution was found, a concrete signal object will be created and placed it into an event queue. This queue contains the events delivered by *MbtTigris,* augmented with concrete test data. The model interpreter returns the set of concretized test sequences to FOKUS!MBT where they are transformed into executable test cases (step 5. from Figure 4), represented as TestingMM interactions. The generated test data for this event is used as a message argument. By doing so, one interaction per concrete test sequence, returned by the model interpreter, is being created. Figure 5 depicts three test cases generated with different algorithms and coverage criteria (offered by *MbtTigris*).

Limitations and Workarounds

One technical problem with *MbtTigris* is that it only generates one path per invocation and the coverage goal is calculated for this path solely. If one needs to obtain a certain coverage for the whole test suite, some customization coding is needed. Another *MbtTigris* issue is the missing functionality of expressing triggering events, i.e., supporting situations where data of the signals is assigned to the SUT. Given also the limited support for EFSM concepts, in our experiments we used *MbtTigris* only as a FSM (finite state machine) traversal engine and we tackled ourselves the test information on the transitions (carried as properties of received signal objects) as shown above. Of course, using an FSM-based approach may introduce

[10] *http://choco.emn.fr*

infeasible paths (i.e., test cases that cannot be made executable). Although the problem of infeasible paths is undecidable in general, our constrain-based test data generator (see above) may provide hints on infeasible paths when no solutions are found by the constraint solver.

5. Related Work

Except the U2TP specification [16] and the examples therein, there only few publications on U2TP [1,4, 3,24] and even fewer references on industrial experiences. The reference book [1] provides an introduction into the common concepts of the U2TP and some possible applications in different testing areas. Its scenario for SOA testing covers only testing of individual or orchestrated services, but not of service choreographies. Furthermore, automated test case or test data generation is outside of scope.

Authors of [4] show how to generate a test architecture described with U2TP, from of a system model for a Bluetooth application case study. This is achieved by introducing external test directives, which control the test model generation by references to elements of the UML system model. Directives are used to group, rename, or add elements for the test model. Test behaviors, expressed as sequence diagrams, are directly imported from the system model. A similar approach to [4] is [18], which relies solely on the QVT specification to define the transformation steps from the system model to test model. The external test directives are hardcoded in the QVT transformation rules. Furthermore, [24] demonstrates a straightforward derivation of executable TTCN-3 test scripts from UML models with U2TP stereotypes. They present only the case of single sequence diagrams that are one-to-one translated to test cases, so no test case or test data generations are deployed. Finally, [12] shows the usage of U2TP together with an MBT tool (Smart MBT) including several generation algorithms.

Compared with our methodology, none of the above (except [12]) fully addresses the test case generation and test data generation from state machines. Also, the definition of model coverage criteria was not taken into account. Moreover, all the above (including [12]), start with a UML system model from which U2TP testing models are generated, whereas we start with a SOA DSL from which we directly generate UML models with U2TP stereotypes.

Regarding commercial tools, a recent MBT survey study [6] found out that among 9 surveyed tools, only Rhapsody ATG [10] and TTWorkbench [20] currently implement the U2TP. While ATG uses primarily only the test architectural concepts of U2TP, TTWorkbench addresses the execution of U2TP test models, using methods similar to [24].

6. Concluding Discussion

In this paper, we presented a concrete SOA testing approach that aims to show the cross-fertilizing benefits of complementary domains along different dimensions: model-driven and service-oriented paradigms, academic-industrial collaboration, standard modeling languages vs. DSLs, high-level MBT processes and their concrete tool chain implementations.

In a nutshell (see also Figure 1), we start with a transformation from choreography models into UML models, containing both the architecture of the scenario as well as the behavior expressed as a UML state machine. During the transformation, U2TP stereotypes are applied to the system model so that FOKUS!MBT is capable to generated a test suite for the given coverage criteria. Test data for the test cases are generated automatically by evaluating the model's constraints. These constraints are assembled into a suitable input to the *Choco* constraint solver that finds adequate test data respecting the constraints. Finally, test cases in TestingMM format are transformed into test case instances of the SAP testing infrastructure, which is not discussed in this paper since it represents a straight-forward model-to-text transformation.

Given the fact that in our use case, the SAP modelers and testers were supposed to be proficient only in SAP DSLs, it was important that the above MBT process via U2TP is highly automated, i.e., with minimal user intervention. This implied that we avoided the existing approaches based on U2TP [1,4] which require manual addition of test relevant elements into new packages or programming of model transformation rules. Our solution was thus based on automated generation of U2TP stereotypes during the model transformation step. However, manual intervention was needed for giving the test directives (see Figure 5), because they are natively missing in U2TP standard (also identified by [4]). This was solved by providing an input method through the FOKUS!MBT user interface.

To conclude, there are general benefits of using a standard like U2TP. First of all, the U2TP specification document is quite concise, with a positive effect on the learning curve. By adding additional elements into the system model, existing behavioral and structural aspects of the system can be reused for the test model [1,4], thus reducing the test development time. Secondly, U2TP doubtlessly supports the model-based development of test suites in a very intuitive way [24] as long as a test developer has enough experience with

UML. This is one of the problems of U2TP at the same time, since UML is quite complex and not as accepted in testing domain as in system development domain. Moreover, there is still a lack of best-practice methodologies and tools, which could have contributed to the current rather low visibility and industrial application of U2TP.

The currently available version 1.0 of U2TP was finalized in 2005 and it contains now some inconsistencies to the newest UML2 version. There are signals from the U2TP consortium that a new release is planned and soon to be discussed. We have concrete plans to actively participate and use the experience presented in this paper as feedback to improve the standard in its future release.

Given the complexity of MBT for SOA domain, there are still things to be improved in this domain in the future. For instance, we plan to better support test data generation for the first-order logic expressions in our DSL. Also in the area of Enterprise SOA, better methods for system testing based on business scenarios and UI models are envisaged. Finally, as there are some difficulties in translating SOA concepts in UML (e.g., local and global viewpoints, channel reliability properties), we also plan to investigate the use of SOA profiles for UML (e.g., SoaML and UML4SOA) in conjunction with U2TP.

Acknowledgments. This work was partially supported by the projects MODELPLEX [13] and DEPLOY [5].

References

1. Baker, P., Dai, Z.R., Grabowski, J., Haugen, Ø., Schieferdecker, I. Williams, C.: Model-driven testing – using the UML testing profile. Springer (2007)
2. Canfora, G., Penta, M. D.: Service-oriented architectures testing: A survey. In: Software Engineering: International Summer Schools, ISSSE 2006-2008, Revised Tutorial Lectures, pp. 78–105. Springer (2009)
3. Dai, Z.R.: An Approach to Model-Driven Testing – Functional and Real-Time Testing with UML 2.0, U2TP and TTCN-3. PhD thesis, TU Berlin (June 2006)
4. Dai, Z.R, Grabowski, J., Neukirchen, H., Pals, H.: From Design to Test with UML – Applied to a Roaming Algorithm for Bluetooth Devices. In Testing of Communicating Systems, LNCS, vol. 2978, pp. 33-49. Springer (2004)
5. DEPLOY Project. Funded by European Commission, FP7, 2008-2012. Grant no. 214158. Webpage: *http://www.deploy-project.eu*
6. Götz H., Nickolaus M., Roßner T., Salomon K.: Modellbasiertes Testen. iXStudie. Heise Zeitschriften Verlag, (2009)
7. Hartman, A., Nagin, K.: The AGEDIS Tools for Model Based Testing. In: UML Satellite Activities 2004, LNCS, vol. 3297, pp. 277–280. Springer (2004)
8. Hartmann, J., Imoberdorf, C., Meisingerm M.: UML-Based Integration Testing. In: Proc. of ISSTA 2000, pp. 60-70. ACM Press (2000)
9. Hartman, A., Katara, M., Olvovsky, S.: Choosing a Test Modeling Language: A Survey. In: Haifa Verification Conference 2006, LNCS, vol. 4383, pp. 204–218. Springer (2006)
10. IBM. Rational Rhapsody Automatic Test Generation Add On. Online at: *http://www.ibm.com/software/rational/products/rhapsody/developer/features/test.html*
11. Kätker, S., Patig, S.: Model-driven development of service-oriented business application systems. In: „Business Services: Konzepte, Technologien, Anwendungen“, Wirtschaftsinformatik. Band 1, pp. 171-180. Österreichische Computer Gesellschaft (2009)
12. P. Krishnan and P. Pari-Salas: Model-based testing and the UML testing profile. In: Semantics and Algebraic Specification, LNCS, vol. 5700, pp. 315-328. Springer (2009)
13. MODELPLEX Project. Funded by European Commission, FP6, 2006-2010. Grant no. 034081. Webpage: *http://www.modelplex.org*
14. MODELPLEX Project. Testing Metamodel. Deliverable 4.4a, December 2007. Online at: *http://www.modelplex.org/index.php?option=com_remository&func=fileinfo&id=189*
15. Object Management Group (OMG). Unified Modelling Language (UML) Specification. Version 2.2, February 2009. Available at: *http://www.omg.org/spec/UML/2.2/*
16. Object Management Group (OMG). UML 2.0 Testing Profile, Final Adopted Specification. Version 1.0, July 2005. Available at: *http://www.omg.org/spec/UTP/1.0/*
17. Object Management Group (OMG). Object Constraint Language 2.0, 2006. *http://www.omg.org/spec/OCL/2.0/*
18. Pérez Lamancha, B., Reales Mateo, P., Rodríguez de Guzmán, I., Polo Usaola, M., Piattini Velthius, M.: Automated Model-based Testing using the UML Testing Profile and QVT. In: Proc. of Workshop MoDEVVa’09. ACM Int. Conf. Proc. Series, vol. 413. ACM (2009)
19. Stefanescu, A., Wieczorek, S., Kirshin, A.: MBT4Chor: A Model-Based Testing Approach for Service Choreographies. In: Proc. of ECMDA-FA’09, LNCS, vol. 5562, pp. 313-324. Springer (2009)
20. Testing Technologies. TTWorkbench Product. Online at: *http://www.testingtech.com/products/ttworkbench.php*
21. Utting, U., Legeard, B.: Practical Model-Based Testing – A Tools Approach. Morgan Kaufmann Publ. (2007)
22. Wendland, M.-F., Großmann, J., Hoffmann A.: Establishing a Service-Oriented Tool Chain for the Development of Domain-Independent MBT Scenarios. In Proc. of 7th Workshop System Testing and Validation (STV’10), pp. 329-334. IEEE Press (2010)
23. Woods, D., Mattern, T.: Enterprise SOA – Designing IT for Business Innovation. O'Reilly (2006)
24. Zander, J., Dai, Z.R., Schieferdecker, I., Din, G.: From U2TP Models to Executable Tests with TTCN-3 - An Approach to Model Driven Testing. In: Proc. of TESTCOM’05, LNCS vol. 3502, pp. 289-303. Springer (2005)