BPMN-Based Model-Driven Testing of Service-Based Processes

Conference Paper · May 2017

DOI: 10.1007/978-3-319-59466-8_8

2 authors:

Daniel Lübke
Leibniz Universität Hannover
63 PUBLICATIONS **587** CITATIONS
SEE PROFILE

Tammo van Lessen
Universität Stuttgart
43 PUBLICATIONS **442** CITATIONS
SEE PROFILE

This version was submitted to BPMDS 2017. It does not contain changes made to the final version!

BPMN-based Model-Driven Testing of Service-based Processes

Daniel Lübke[1,2] and Tammo van Lessen[3]

[1] Leibniz Universität Hannover, FG Software Engineering, Germany
[2] innoQ Schweiz GmbH, Switzerland
[3] innoQ Deutschland GmbH, Germany
daniel.luebke@inf.uni-hannover.de, tammo.van-lessen@innoq.com

Abstract. Executable Business Processes realized in WS-BPEL and BPMN2 are used more and more for automating digitalized core processes in organizations. Due to their critical nature for the organization these processes need to be developed with high quality standards. Existing literature concentrates on testing such processes but do not offer integration into the development lifecycle and validation with other stakeholders. Our approach is based on Test Models that allow both the easier definition of automated test cases as well as discussion with non-technical stakeholders and thus can be used for business process validation and process modeling support. We define a meta-model for the BPMN-based Test Models that has been validated in a case study in an industrial project.
Keywords: BPMN, Model-Driven Testing, Business Process, Service Composition, Process Validation

1 Introduction

Executable Business Processes are becoming more and more common to implement the flow of business activities in business software systems. These languages, like WS-BPEL [5] and BPMN2 [20], are allow the visual modeling of process flow, data-flow and service invocations. Parallel execution, business transactions by compensation management, and message correlation are integral concepts mandated by the standards that are implemented in the middleware and need not be written manually by developers.

Executable Business Processes are also software artifacts and offer high expressiveness. They can carry complex process logic and data transformation logic that is critical from an availability point of view: Business-critical processes need to be available to the organization and software failures or unwanted behavior may cause huge losses.

The more business processes are automated in an organization and the more complex they get over time, the more important quality assurance of such executable business processes gets.

One important part of quality assurance for executable business processes is the development of automated tests. In development projects, different project

members develop tests for finding defects on different levels (unit tests, integration tests and system tests [9]). These tests focus on different error sources of the system. However, all tests need to be maintained alongside the business processes, which can take considerable effort in a complex system. Especially changes to service contracts lead to large maintenance efforts for migration of test definitions and test data.

Complexity also adds another challenge for project teams: They need to validate the process models with the business side. Depending on the business side's ability to understand and work with large business process models, this task can represent a challenge in itself. However, it is essential to agree on the functionality and necessary acceptance tests for the executable business processes. This can only be achieved, if the testing approach is combined with a review of the business processes – and even better the test cases are validated by the business side.

This paper builds upon an experience report published by the authors in [11]. In addition to the experiences this paper presents a meta-model, development process buildings blocks and a case study and therefore makes the approach applicable for others and provides initial empirical validation. This paper is therefore structured as follows: In the following section existing approaches for testing executable business processes is presented. In Section 3 the model-driven testing approach is presented with describing the underlying requirements (Section 3.1), the underlying meta-model (Section 3.2), a short description of the generator (Section 3.3) and a discussion on how the requirements are fulfilled (Section 4.1.) This main section is followed by Section 5 with discusses how the approach can be integrated into software development processes. An empirical evaluation of the presented approach in the form of a case study is described in Section 6 before the paper concludes.

2 Related Work

Research into BPEL and BPMN testing for service compositions is an active research area. Rusli et al. [18] conducted a mapping study and found that 58% of the test publications concentrate on Test Generation but they do not list profiles nor meta-models as research objectives. The only approach the authors identified to use profiles was by Rauf et al. [17], which uses UML profiles to help with test case generation.

For their testing approach Kaschner and Lohmann [6] informally introduce their specification models and public view models as examples. They use black-box pools for participants and describe the behavior of processes by giving abstracted process descriptions.

Tools for automating test execution against BPEL service compositions have been developed. At least two of those (Li et al. [8] and Mayer & L¨ubke [14]) provide a test framework that is capable of sending and receiving SOAP messages independently of BPEL. Due to this ability they can also be used to test

BPMN service orchestrations that utilize SOAP services and provide the technical infrastructure for running tests defined with our approach.

Dong et al. [7] presented a testing approach for BPEL based on Petri Nets. The approach analyzes an existing BPEL process and deducts test cases from it. Maalej et al. [12,13] use model-based testing for checking conformance with times autonoma. Ji et al. [4] use data-flow analysis techniques to deduct regression tests from existing BPEL processes. Yuan et al. [21] propose the use of UML Activity Diagrams in conjunction with the BPEL specification.

All these approaches have in common that the test models are not to be understood by business stakeholders and are often even derived from the BPEL model itself. Thus, no validation of the test cases can occur and the tests can only be defined after the creation of the BPEL process has been finished.

3 Model-Driven Test Approach

3.1 Requirements

In order to enable a development project to define, develop and maintain executable test cases for executable business processes efficiently, we identified the following requirements when discussing testing challenges with an industrial project. Those were later amended during the case study:

R01: Test Cases must be reviewable by business stakeholders, business analysts, business process modelers, and developers. Because at least business stakeholders are often not familiar with technical notations, test cases must be defined in an easy and intuitive way.

R02: Test Cases must be deterministic, i.e. they must be defined so precisely that the same process flow will be executed every time the test case is executed in order to replicate error scenarios and re-run test cases for determining whether a defect has been fixed.

R03: Test Cases must express timing dependencies between different services (e.g. sub-processes or different participants) in order to express scenarios which are timing sensitive.

R04: Test Cases must be easy to maintain. Especially changes to the environment like new versions of service contracts must be achieved with as low effort as possible.

R05: Effort invested into the definition of test cases should not be lost, but the business-friendly description should be the basis for test case automation.

R06: Test Cases definitions should be "living" documents: Their usage should be for practical reasons mandatory or so useful that they are updated in the normal development activities.

R07: Test Cases should be derivable from the business process ("test-later" approach) or should be the basis for the business process model implementation ("test-first" approach [1]).

R08: Test Cases should ideally serve as process documentation allowing stakeholders to find specified behavior easily in a given context.

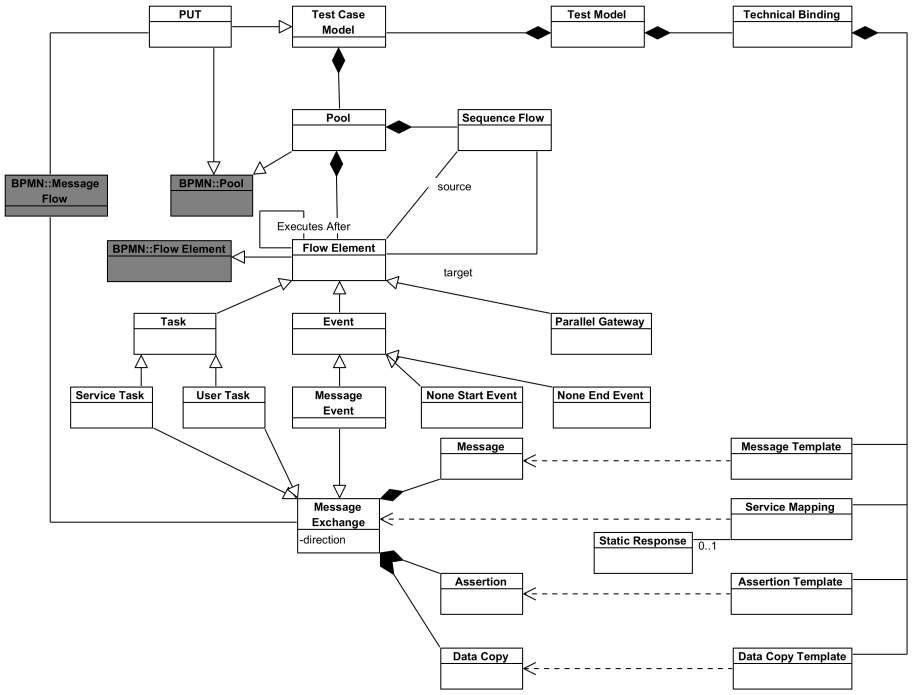

Fig. 1: A simplified Meta-Model of the BPMN Test Profile

R09: Test Cases need to serve as a manual for testers for conducting the tests manually. Besides automation, it is initially still necessary to test those functions manually that have user interactions to check for usability and formatting issues.

R10: Test Cases need to define a way to store values generated by the process, e.g. correlation tokens, in order to provide all information necessary for producing an executable test case.

R11: Allow the hiding of technical messages in the Test Models (relates to R01).

3.2 Meta-Model for Test Models and Test Case Models

In order to satisfy the requirements, our approach defines two sets of artifacts: Those defined from the business point of view and those defined from a technical point of view. This separation allows us to fulfill technical requirements and still have artifacts that business stakeholders can discuss and comprehend.

A simplified version of the meta-model is shown in Figure 1. The gray colored classes are part of the BPMN standard. The meta-models main class is the *Test Model* that contains a set of *Test Case Models* that are modeled in BPMN. The control-flow restrictions of these models are explained in further detail below and are enforced by only allowing a sub-set of BPMN to be used: Every BPMN Test

Case Model has one BPMN Pool that represents the process/system under test (PUT). All other *Pools* represent process participants that should be mocked during the test.

Pools contain *Flow Elements* that are derived from *BPMN Flow Elements*. The allowed Flow Elements contain only the necessary control-flow structures (*Tasks*, selected *Events* and selected *Gateways*) that are necessary to organize *Message Exchanges* between the PUT and another Pool. Every Message Exchange can have multiple *Assertions* for data that is sent to a participant and can define a *Message* that is sent back to the PUT. The cardinalities of assertions and messages are dependent on the service contract: In case of incoming (i.e. received by the participant) one-way operations, no returning message must be defined. In case of outgoing one-way operations, no assertions must be defined. For two-way operations, a message must be defined that is sent to the process.

Messages and Assertions are not defined absolutely but are references to *Templates* with parameters. Like in Behavior-Driven Development (BDD) [16] this is done with a structured text form that looks like normal language to readers but must conform to a template. This allows the reuse of information by referencing it from different Test Case Models. Templates are implementation specific, e.g. SOAP messages and XPath assertions might be defined for testing SOAP Web services.In addition to the Templates, there are technology-dependent *Service Mappings* that map message exchanges in the Test Case Model to physical operations, e.g. as defined in a WSDL. By doing this, all technical information is extracted from the Test Case Model itself and can be replaced by a different one. This allows the same set of Test Cases to be used for testing different service versions as long as the process stays the same: Templates and Service Mappings can be provided for both service versions.

The Test Case Models as illustrated as an example in Figure 3 are BPMN models that are required to use a subset of BPMN with one extension taken from [3]. The restrictions make the process flow deterministic in order to satisfy Requirement R02: The control-flow in BPMN is controlled by the use of gateways that can be defined explicitly with their own diamond-shaped symbol or implicitly by having multiple sequence flows (“arcs”) from a single activity. All implicitly defined control-flows are forbidden in our profile and only parallel gateways, i.e. those that trigger all paths after them, are allowed. Originally, no gateways were allowed but when testing our approach in an industry project we encountered situations that required parallel activities by process participants. We also found that process participants need to be synchronized (Requirement R03): An activity in one participant must only be executed if another participant completed another activity. Such cross-participant dependencies cannot be expressed in BPMN. Especially in order to test compensation in parallel processes, test cases needed to be able to express such dependencies in order to eliminate uncontrolled execution orders. In the case study we also found paper-based activities not controlled by the software that required the use of cross-participant synchronization for automating test cases.

3.3 Generator Implementation

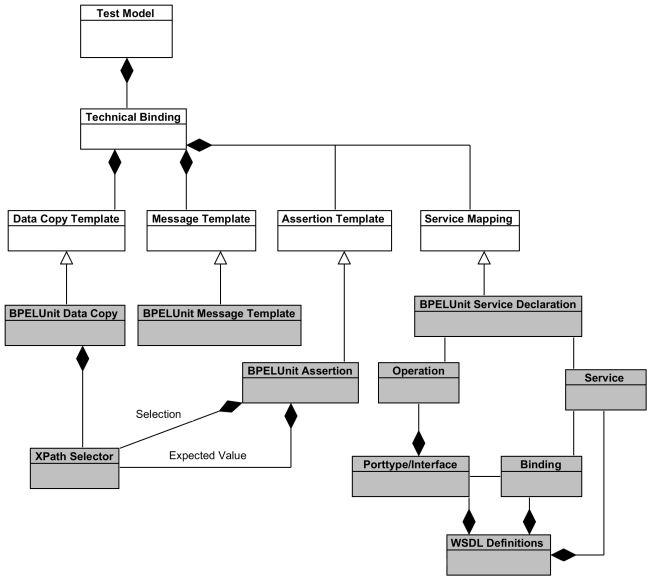

Fig. 2: Technical Binding to BPELUnit Elements

In order to roll out our approach in the case study project, we needed to provide a working tool chain. Thus, we developed a generator that reads BPMN models in the official OMG XML format and generates BPELUnit [14] test suites from it. By using the official XML format, our generator can be used with many BPMN tools and we avoided to develop an own test designer. BPELUnit was already used by the project and thus the new test cases could be integrated well into the development tool chain. It provides the infrastructure for testing and mocking arbitrary SOAP services – not only those implemented in BPEL. BPELUnit operates by sending predefined SOAP messages, evaluating incoming messages by the use of XPath assertions. In addition it supports run-time templating of both messages and assertions.

For using the same modeling tools like the case study project, the generator faced a problem: The test profile requires the use of an academic timing extension to BPMN, which is not availabe in the modeling tool used by the project nor in any commercially supported tool that we know of. As a consequence, we replicated the dependency by replacing it with a textual annotation, e.g. *Depends-On: Flight Booked*.

The generator reads a spreadsheet that contains various sheets for configuration purposes. A shortened version of the BPELUnit technical binding meta-

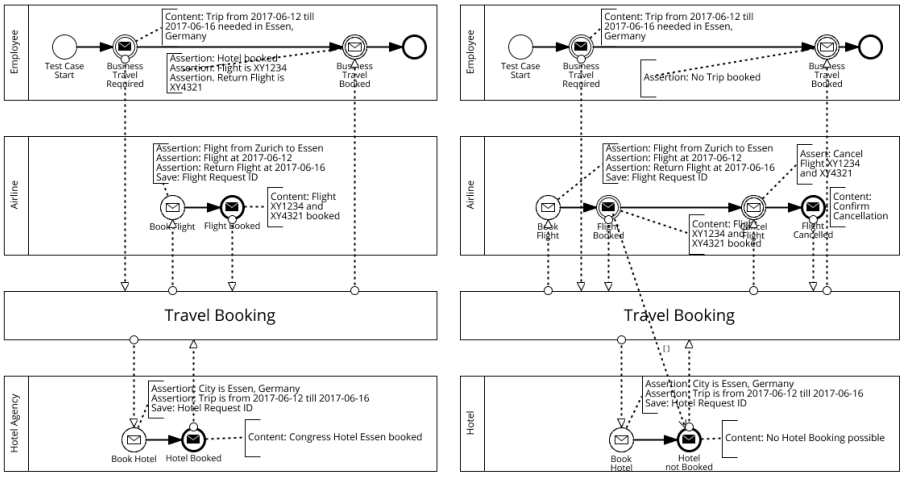

Fig. 3: Example Test Cases

model is shown in Figure 2 that defines the contents of the different sheets. The gray-colored classes are technology-specifc to WSDL and SOAP:

- Services in the Test Model are mapped to services, bindings and operations of WSDL service descriptions.
- Assertions are mapped to two XPath expressions: The first selects the value in the received message to be compared (e.g. //customerNo) and the second defines the expected value (e.g. '123-A-B').
- Message Templates are implemented by using XSLT transformations. The parameters from the templated text are passed into an XSLT templates, which returns an XML document that will be sent in the SOAP message.
- For Data Copies an XPath expression is defined that selects the value to be stored from a SOAP message. This value will be placed in a variable that can be managed by BPELUnit and injected into outgoing SOAP messages.

4 Example

In order to illustrate the usage of our approach, we define test cases for a very simple example travel process:A travel is booked that consists of a flight and a hotel. If either booking fails, compensation for already booked items will be triggered, i.e. the bookings will be canceled.

For illustrating the testing of this process, we define two Test Case Models (see Figure 3): One that successfully books a hotel and a flight, and another one that successfully books the hotel but fails to book the flight. Because the bookings for the flight and the hotel are done in parallel, we need a synchronization for the error cases in order to guarantee that one item is already booked and thus needs to be compensated.

The example diagrams contain all information as BPMN comments for illustrative purposes; in real models assertions and messages are stored in the (invisible) BPMN documentation elements. While initially we used the BPMN comments, it became clear during the case study (see Section 6 that diagrams became too hard to read and cluttered.

4.1 Mapping to Initial Requirements

Within this section we map the different model elements and generator features to the requirements initially outlined in order to show that our approach satisfies all requirements in general.

R01: The Test Cases are defined as a simple BPMN subset that is understandable by business stakeholders and does not contain any complex constructs.
R02: Because the control-flow in the Test Models is sequential except for parallel flow where necessary, and additional cross-participant contraints can be enforced, test designers and process modelers can model completely deterministic test cases.
R03: This requirement was a special case to R02 and is satisfied by the introduction of cross-participant dependencies.
R04: New Service Contracts only require a new set of technical bindings that are centralized. No changes to the Test Models are necessary.
R05: The BPMN test cases are easy to understand and also work as scenarios for discussing complex business processes and special corner cases.
R06: Because test cases are integrated into the development process and are the source of generation, they need to be current and maintained. As such they must be kept up-to-date by the development team.
R07: Currently, we support the "Test-First" Approach. However, there are approaches available (e.g. by Ni et al. [15]) which allow the deduction of scenarios from business process models that can be enhanced to derive Test Models later on.
R08: Test Cases are artifacts that are stored and can be made available. Due to being up-to-date they can serve as a complementary business process documentation. Empirical validation of this point is however needed.
R09: Testers can use the Test Models to conduct manual tests.
R10: The Data Copy facilities allow the extraction of Correlation Tokens and other process-generated data to be sent back later on.
R11: Due to the possibility of attaching default technical messages to services, the hiding of technical messages is possible.

5 Inclusion in the Development Process

Our approach does not require a certain development process. However, because the two domains of business process modeling and software engineering (especially the fields of requirements engineering and testing) are concerned and must be interlinked, we define four Development Process Building Blocks that are shown as information flows using the FLOW notation [19] in Figure 4:

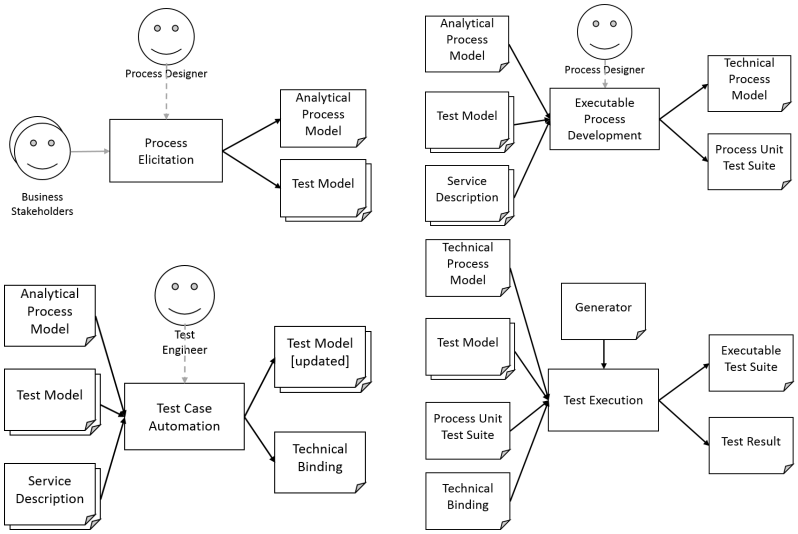

Fig. 4: Development Process Building Blocks

Process Elicitation: The goal of this development activity is to create and document an understanding of the project's underlying business process(es). Within this activity at least an analytical process model and the test models as per our approach are created. The elicitated Test Models are used in all other Development Process Building Blocks.

Executable Process Development: This Development Process Building Block uses the Test Case Models that provide smaller grained units that can be used for developing stories or smaller project tasks and help the Process Designer in understanding the overall model. Deliverables of this building block is the executable business process model as well as the unit tests covering it.

Test Case Automation: The Test Engineer role is responsible for adding the necessary technical bindings to the Test Models. This might require refinements of the test cases and the standardization of the messages and assertions.

Test Case Execution: This Development Process Building Block is fully automated and uses the generator to create a test suite out of the provided artifacts and executes it in order to create a test result.

This Development Process Buildings Blocks can be used in different arrangements to be used with different project methodologies, e.g. more waterfall-based approaches (strictly following the order of Process Elicitation, Executable Process Development, Test Case Automation, Test Case Execution) or agile ones (in which these four phases are repeated and done iteratively.) The project in the case study already used different arrangements of the building blocks in order to achieve certain project goals.

6 Case Study

6.1 Case Description

Terravis [2, 11] is a large-scale process integration platform for conducting land registry-related business between land registries, banks, notaries, geometers, pension funds and other parties. Terravis has been developed since 2011 with the first services going productive in 2012. It offers 15 core end-to-end processes with many sub-variants and 12 administrative processes for managing mortage depots.

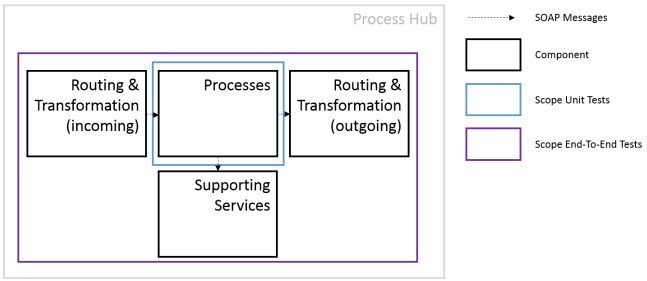

Fig. 5: Test Scope of the Model-Driven Tests

The analytical process models for documentation purposes and discussion are created using BPMN2 and the executable process models are realized using WS-BPEL. All service interactions between parties and Terravis are based on SOAP Messages.With increasing success, Terravis functionality was extended by both new processes as well as new variants in existing processes, which led to more process models and more complex process models [10].

Because Terravis is in the center of many parties that have different software lifecycles for their systems, it needs to offer multi-version support: Different service versions have to be offered in parallel to give all parties a chance to migrate to the newest version within their lifecycle model. Also a multi-channel approach for using a portal and integrated systems in parallel was added. These developments made manual testing overly complicated because of many new test cases and required more test automation to keep the desired quality level.

The research questions to be answered in this case study with regard to the model-driven testing approach described in this paper are:

RQ1: Is the test profile expressive enough to test real-world processes/services?
RQ2: Does the generator approach save testing effort by increasing reuse?
RQ3: Can the model-based approach improve the communication and the understanding within the development team by utilizing BPMN scenarios?
RQ4: Can the same Test Models be reused for testing different service versions of the same executable business process?

The Architecture of the Process Hub of Terravis is shown in Figure 5: The Process Hub is a backend component that consists of business processes run on a BPMS that are calling supporting service, e.g. for generating documents. The BPMS is accompanied by two routing and transformation chains that shield it from changes in the environment, e.g. service version changes.

The already existing unit test cases cover only the executable processes directly and in isolation. The goal for creating the new test cases was to cover the message flow end-to-end, i.e. call a process via the same infrastructure components like external parties do. This tests the integration of the routing and transformation chains with the processes as well as the supporting services. Because the new test cases only cover the externally visible interfaces they are smaller in size compared to the unit tests which mock all services regardless of their external visibility.

For creating the needed automated regression tests, an analysis of process instance frequency was conducted,the 12 most frequent process variants were selected, and one Test Case Model was created for each. The Development Process Buildings Blocks “Test Case Automation” and “Test Execution” were used to implement regression tests for already specified and implemented processes.

6.2 RQ1: Expressiveness of the Test Model

While implementing the set of regression tests, a mechanism for synchronizing the timings of different Process Participants (pools) was needed. As a result, requirement R03 was added and both the meta-model as well as the generator extended. With this expressiveness all test cases could be modeled and executed. This means that for this case study, the answer to RQ1 is that all processes could be successfully tested with the extended expressiveness of this profile.

6.3 RQ2: Effort of the Generation Approach

Element	Static Count	Usage	Avg. Usage
Test Cases	12	-	-
Message Template	20	45	2.25
Assertion Template	21	135	6.42
Data Copy Template	2	13	7.50
Service Mapping	35	91	2.60

Table 1: Static Metrics for the Test Suites

For answering RQ2, we analyzed the Test Case Models and technical bindings in order to compute the metrics presented in Table 1: All elements of the technical binding are – on average – referenced multiple times from the test case models. For example, on average every message template is used 2.25 times,

every assertion template 6.42 times, and every data copy template 7.5 times in the case study project. By reusing these elements multiple times from the set of test case models, developers and testers should be able to better handle changes to the service because only the templates need to be changed and not all occurrences. All in all, the answer to RQ2 is that all element types are highly reused.

6.4 RQ3: Improvement of Communication

The metrics also show that the introduced facility for returning static XML content in case of empty messages was frequently used. Although 91 services were used (the same as the number of service mapping usages), only 45 message templates were used. This means that nearly the half of all (response) messages contained no business-viable content.

The project had to develop one new end-to-end process, which is comparably small compared to the other end-to-end processes. Also a change to a sub-process used by 7 end-to-end processes was required. Besides the full process models, Test Case Models conforming to our meta-model were created prior as part of the process and requirements analysis. Both the new end-to-end process and the sub-process required 8 new test case models. However, the sub-process development and definition has not yet been completed as the time of writing: The sub-process test cases need to be integrated into the test cases of the 7 end-to-end processes yet.

The project team concerned with the development of the new process and the extension of the existing processes used all four Development Process Building Blocks and was asked whether the scenarios were helping in the discussion and with the understanding of the process and its variants. All team members answered with yes. Thus, RQ3 can be cautiously answered that the approach improves communication and understanding of business process requirements in a development team.

6.5 RQ4: Effort with Service Version Changes

After introducing the initial regression test suite, a new version of the bank-side interfaces was rolled out and required a transformation component in the outer layers of the process solution. The Test Case Models could be kept and only a new set of technical bindings including the SOAP messages and assertions with the new schema was defined. This set of automated tests was run against the process solution and early defects could be spotted easily and be verified by the developers because the whole test generation and execution process was automated and made available on the Build Server. This reduced the time for testing the transformations compared to previous transformations for prior service version changes. Unfortunately, we could not measure the testing and integration effort required for the already existing mapping components but developers said that the new test cases saved time. Having made the service version transition

without changes to the Test Models, we can answer RQ4 with yes. However, we unfortunately lack the data to estimate the saved effort.

6.6 Improvements to the Original Approach

As part of the case study we could identify practical shortcomings together with the development team. These led to the following enhancements:

Usage of Documentation Elements: Our initial approach used BPMN comments for storing the message content and assertion text. However, these grew too large when used on real messages because in services much business payload is transferred. As a countermeasure we moved the contents into the non-visibile BPMN Documentation elements during our case study.

Cross-Participant Synchronization: We encountered the need for synchronizing two participants in the Test Model because information is exchanged via paper in the real-world imposing a certain order of service calls. To support this constellation we added requirement R03.

Participant Propagation for Templates: SOAP messages contain technical fields, especially information about which participant sends or receives a message. Because messages might contain the same business information but are sent by different participants, the associated participant name is passed as a parameter to the message templates reducing the number of necessary templates.

Static Response: The project used many two-way operations with empty responses for indicating the successful transmission of the request. The empty responses contain no business information and as such testers and business stakeholders do not want to see them in the Test Case Models. Also technical stakeholders did not like to declare empty message contents repeatedly. In order to address this issue, we added requirement R11 and allowed service bindings to specify a default message. This default message is used when no other message content is specified on the test activity.

Because this case study was only conducted within one project, its findings cannot be generalized easily before further empirical validation has been conducted, e.g. by further case studies or experiments. However, the case study shows that the application of our approach is likely to positively influence stakeholder communication and reduce the effort for test creation and maintenance.

7 Conclusions & Outlook

Within this paper we presented a model-driven testing approach utilizing BPMN. The approach allows development projects of executable business processes to define test cases and validate both the test cases and the overall process as a set of scenarios with various stakeholders. It basically transfers the idea of Behavior-Driven Development to the area of Executable Business Process Development

combined with Model-Driven Testing. In contrast to "traditional" BDD projects, customized BPMN is used and not a custom text-only domain-specific language. This allows the usage of BPMN as the "lingua franca" for all project participants and the reuse of already existing editors and repositories. Using the same modeling language lowers the entry barrier for the different stakeholders and allows the same tooling to be used.

Consequently, the required test case generator can be implemented based on the official BPMN XML Schema. We implemented a generator that creates BPELUnit test suites for testing SOAP-based business processes, although our approach is independent of the underlying test execution infrastructure.

Our presented approach also builds a bridge between the two domains of Business Process Modeling and Software Engineering. Both domains are highly related in business process executation projects that will be more common as part of further digitalization of organisations. In order to support the cooperation of roles from these domains, we specified Development Process Building Blocks that can be used to add the presented approach to software development processes. We hope to see more inter-domain research for building improved and demonstratedly better solution development methodologies.

The validation in a case study showed positive effects on communication, test creation, and test maintenance. However, future research will need to provide further and stronger validation of these results in order to demonstrate advantages in a more generalizable manner.During the conduction of the case study we already seen the need for small improvements that greatly improved the practical applicability both to the meta-model as well as to the tooling. Future Research should also identify further problems and provide improvements to our approach.

References

1. Kent Beck. *Test-Driven Development by Example*. Addison-Wesley, 2003.
2. Walter Berli, Daniel L¨ubke, and Werner M¨ockli. Terravis – large scale business process integration between public and private partners. In Erhard Pl¨odereder, Lars Grunske, Eric Schneider, and Dominik Ull, editors, *Lecture Notes in Informatics (LNI), Proceedings INFORMATIK 2014*, volume P-232, pages 1075–1090. Gesellschaft f¨ur Informatik e.V., Gesellschaft f¨ur Informatik e.V., 2014.
3. Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. Toward a Time-centric Modeling of Business Processes in BPMN 2.0. In *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, IIWAS '13, pages 154:154–154:163, New York, NY, USA, 2013. ACM.
4. Shunhui Ji, Bixin Li, and Pengcheng Zhang. Test case selection for data flow based regression testing of bpel composite services. In *Services Computing (SCC), 2016 IEEE International Conference on*, pages 547–554. IEEE, 2016.
5. Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Gu´ızar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter K¨onig, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri,

and Alex Yiu. *Web Services Business Process Execution Language Version 2.0*. OASIS, April 2007.
6. Kathrin Kaschner and Niels Lohmann. Automatic test case generation for interacting services. In *International Conference on Service-Oriented Computing*, pages 66–78. Springer, 2008.
7. W. l. Dong, H. Yu, and Y. b. Zhang. Testing bpel-based web service composition using high-level petri nets. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 441–444, Oct 2006.
8. Zhongjie Li, Wei Sun, Zhong Bo Jiang, and Xin Zhang. BPEL4WS Unit Testing: Framework and Implementation. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 103–110,Washington, DC, USA, 2005. IEEE Computer Society.
9. Daniel L¨ubke. *Test and Analysis of Service-Oriented Systems*, chapter Unit Testing BPEL Compositions. Springer, 2007.
10. Daniel L¨ubke. Using Metric Time Lines for Identifying Architecture Shortcomings in Process Execution Architectures. In *Software Architecture and Metrics (SAM), 2015 IEEE/ACM 2nd International Workshop on*, pages 55–58. IEEE, 2015.
11. Daniel L¨ubke and Tammo van Lessen. Modeling Test Cases in BPMN for Behavior-Driven Development. *IEEE Software*, Sep/Oct 2016:17–23, Sep/Oct 2016.
12. A. J. Malej, M. Krichen, and M. Jmael. Model-based conformance testing of ws-bpel compositions. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, pages 452–457, July 2012.
13. A. J. Malej, M. Krichen, and M. Jmael. Model-based conformance testing of ws-bpel compositions. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, pages 452–457, July 2012.
14. Philip Mayer and Daniel L¨ubke. Towards a BPEL Unit Testing Framework. In *TAV-WEB '06: Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications, Portland, USA*, pages 33–42, New York, NY, USA, 2006. ACM Press.
15. Yitao Ni, Shan-Shan Hou, Lu Zhang, Jun Zhu, Zhong Jie Li, Qian Lan, Hong Mei, and Jia-Su Sun. Effective message-sequence generation for testing bpel programs. *IEEE Transactions on Services Computing*, 6(1):7–19, 2013.
16. D. North. Introducing BDD. http://dannorth.net/introducing-bdd, 2006.
17. I. Rauf, M.Z.Z. Iqbal, and Z.I. Malik. Model based testing of Web service Composition using UML profile. In *Proceedings of the 2nd Workshop on Model-based Testing in Practice*, 2009.
18. H.M. Rusli, S. Ibrahim, and M. Puteh. Testing Web Services Composition: A Mapping Study. 2011.
19. Kurt Schneider. Software Process Improvement from a FLOW Perspective. In Andreas Birk, editor, *Workshop on Learning Software Organizations (LSO 2005)*, 2005.
20. Bruce Silver and Bruce Richard. *BPMN method and style*, volume 2. Cody-Cassidy Press Aptos, 2009.
21. Qiulu Yuan, Ji Wu, Chao Liu, and Li Zhang. A model driven approach toward business process test case generation. In *2008 10th International Symposium on Web Site Evolution*, pages 41–44, Oct 2008.