Creating Declarative Process Models Using Test Driven Modeling Suite

Stefan Zugal, Jakob Pinggera, and Barbara Weber

University of Innsbruck, Austria
{stefan.zugal|jakob.pinggera|barbara.weber}@uibk.ac.at

Abstract. Declarative approaches to process modeling promise a high degree of flexibility. However, current declarative state-of-the-art modeling notations are, while sound on a technical level, hard to understand. To cater for this problem, in particular to improve the understandability of declarative process models as well as the communication between domain experts and model builders, Test Driven Modeling (TDM) has been proposed. In this tool paper we introduce Test Driven Modeling Suite (TDMS) which provides operational support for TDM. We show how TDMS realizes the concepts of TDM and how Cheetah Experimental Platform is used to make TDMS amenable for effective empirical research. Finally, we provide a brief example to illustrate how the adoption of TDMS brings out the intended positive effects of TDM for the creation of declarative process models.

Key words: Declarative Business Process Models, Test Driven Modeling, Test Driven Modeling Suite.

1 Introduction

In today's dynamic business environment the economic success of an enterprise depends on its ability to react to various changes like shifts in customer's attitudes or the introduction of new regulations and exceptional circumstances [1]. Process-Aware Information Systems (PAISs) offer a promising perspective on shaping this capability, resulting in growing interest to align information systems in a process-oriented way [2]. Yet, a critical success factor in applying PAISs is the possibility of flexibly dealing with process changes [1]. To address the need for flexible PAISs, competing paradigms enabling process changes and process flexibility have been developed, e.g., adaptive processes [3], declarative processes [4] and late binding and modeling [5].

Especially declarative processes have recently attracted the interest of researchers, as they promise a high degree of flexibility [4]. Although the benefits of declarative approaches seem rather evident [4], they are not widely adopted in practice yet. In particular, as pointed out in [4], [6] ,[7], understandability problems hamper the usage of declarative process models. An approach tackling these problems, the *Test Driven Modeling* (TDM) methodology, is presented in [7]. TDM aims at improving the understandability of declarative process models as

well as the communication between domain experts [8] and model builders [8] by adopting the concept of *testcases* from software engineering. This tool paper describes *Test Driven Modeling Suite* (TDMS)[1] that provides operational support for TDM.

The remainder of this tool paper is structured as follows: Section 2 briefly introduces TDM. Then, Section 3 discusses the software architecture and features of TDMS, while Section 4 illustrates the usage of TDMS by an example. Finally, Section 5 concludes with a summary and an outlook.

2 Test Driven Modeling

In this section we briefly sketch what constitutes a declarative process model and how TDM is intended to support the creation of declarative process models. Please note that we focus on TDMS and necessary backgrounds only. A discussion of, e.g., related approaches, is out of scope and can be found in [7].

A declarative process model is characterized by a set of activities and a set of constraints. In contrast to imperative process modeling languages like, e.g., BPMN, the control-flow is not explicitly, but implicitly defined through constraints which exclude forbidden behavior. For instance, a constraint in process model S might specify that activity A is not allowed to be executed more than once. Then, every process instance that contains not more than one execution of A is considered to be a valid instance of S—independent of when A has been executed. An exemplary declarative process model can be found in Fig. 4 (2).

While constraints focus on forbidden behavior, TDM introduces the concept of *testcases* to focus on *desired* behavior of the process model. In particular, a testcase consists of an *execution trace* (i.e., a sequence of activities that constitute a process instance) as well as a set of *assertions* (i.e., conditions that must hold at a certain state of the process instance) (cf. Fig. 1). The execution trace of a testcase thereby specifies behavior that must be supported by the process model, whereas assertions additionally allow to test for unwanted behavior, i.e., behavior that must be prohibited by the process model. A typical example for an assertion would be to check whether activity N is executable at time M.

Consider, for illustration, the testcase depicted in Fig. 1. It contains the execution trace <A,B> (1) as well as an *execution assertion* that specifies that A cannot be executed between the completion of A and the start of B (2) and *termination assertions* that specify that the process instance *cannot* be terminated before the completion of A (3), however, it must be possible to terminate after the completion of A (4). The times in Fig. 1 do not necessarily constitute *real* times, but rather provide a timeline to test for control-flow behavior, i.e., define whether activities can be executed subsequently or in parallel. Furthermore testcases are validated automatically, i.e., no user interaction is required to check whether the specified behavior is supported by the process model.

So far we have introduced the concept of testcases, in the following we will sketch how their adoption intends to improve the communication between do-

[1] Freely available from: http://www.zugal.info/tdms

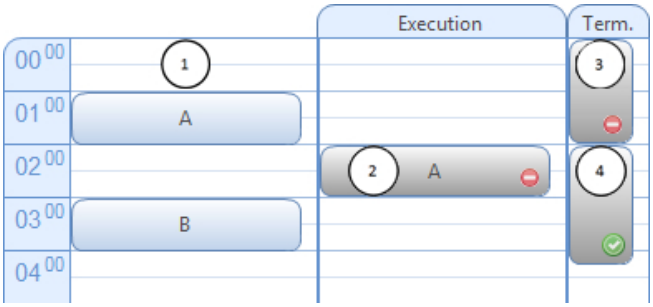

Fig. 1. A Simple Testcase

main expert (DE) and model builder (MB). Testcases provide information in a form that is not only understandable to the MB, but also understandable to the DE, who usually does not have the knowledge to read formal process models [8]. Usually the DE needs the MB to retrieve information from the model, cf. Fig. 2 (2) and (3). Since testcases are understandable to the DE, they provide an additional communication channel to the process model, cf. Fig. 2 (4) and (6). It is important to stress that TDM's intention is not to make the DE specify the testcases in isolation. Rather, testcases should be created by the DE and the MB together and provide a common basis for discussion.

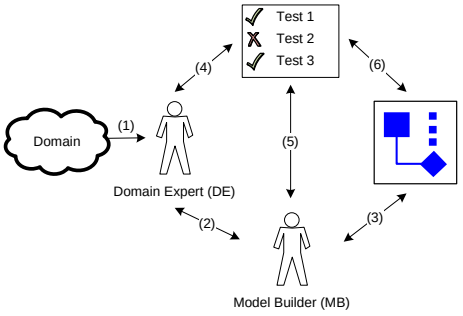

Fig. 2. Communication Flow

Besides improving the communication between DE and MB, testcases aim at improving the MB's understanding of the process model by providing an additional point of view. As pointed out in [7], especially so-called hidden dependencies [9], i.e., information that is not *explicitly* available in the process model can impede a model's understandability. An exemplary hidden dependency is shown in Fig. 3 (2): A must be executed exactly once (cf. cardinality constraint on A) and after A has been executed, B must be executed (cf. response constraint between A and B). Thus, B must be executed at least once for every process instance. However, this information is present in the process model implicitly only. Therefore the MB cannot rely on explicit information only, but has to inspect the model carefully for such hidden dependencies. Using TDM this

problem can be tackled by specifying a testcase that tests for this hidden dependency as shown in Fig. 3 (1): the testcase specifies that the process instance can only be terminated if B has been executed at least once. As soon as the MB conducts changes to the process model that violate the testcase, the automated validation of TDMS (cf. Section 3) immediately informs the MB.

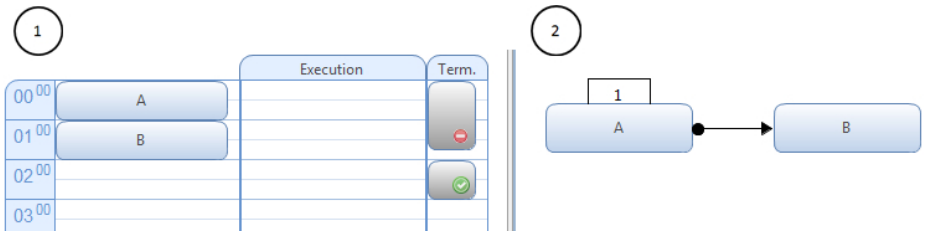

Fig. 3. Hidden Dependency

3 Test Driven Modeling Suite

Up to now we have introduced the concept of TDM. This section deals with Test Driven Modeling Suite (TDMS) which provides operational support for TDM. In particular, Section 3.1 discusses the features of TDMS in detail. Subsequently, Section 3.2 describes how TDMS is integrated with existing frameworks for empirical research and business process execution.

3.1 Software Components

To provide an overview of TDMS' features, all integrated components are illustrated in Fig. 4; each component will be described in detail in the following. On the left hand side TDMS provides a graphical editor for editing testcases (1). To the right, a graphical editor allows for designing the process model (2). Whenever changes are conducted, TDMS immediately validates the testcases against the process model and indicates failed testcases in the testcase overview (3)—currently listing three testcases from which one failed. In addition, TDMS provides a detailed problem message about failed testcases in (4). In this example, the MB defined that the trace $<A,B,B,B,A,C>$ must be supported by the process model. However, as A must be executed exactly once (cf. the cardinality constraint on A), the process model does not support this trace. In TDMS the failed testcase is indicated by the activity highlighted in (1), the testcases marked in (3) and the detailed error message in (4).

Testcase Editor. As mentioned before, testcases are a central concept of TDM, have precise semantics for the specification of behavior and still should be understandable to domain experts. To this end, TDMS provides a calendar-like testcase editor as shown in Fig. 4 (1).

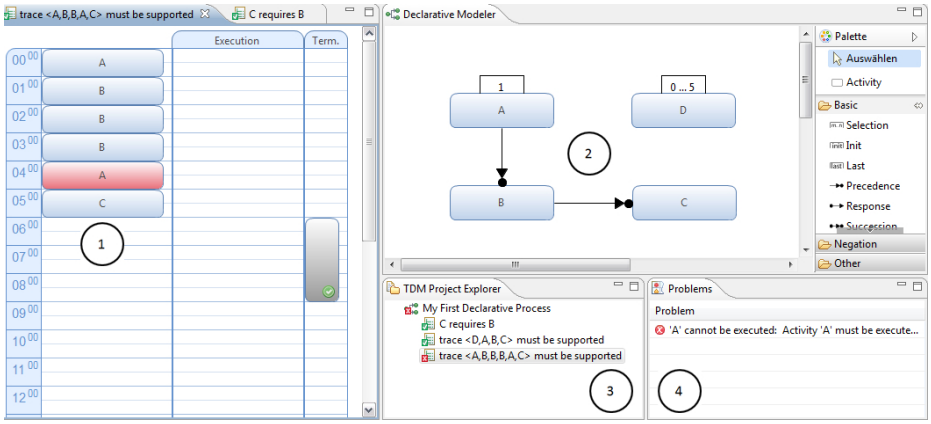

Fig. 4. Screenshot of TDMS

Declarative Process Model Editor. The declarative process model editor, as shown in Fig. 4 (2), provides a graphical editor for designing models in DecSerFlow [4], i.e., a declarative process modeling language.

Testcase Creation and Validation. In order to create new testcases or to delete existing ones, Fig. 4 (3) provides an outline of all testcases. Whenever a testcases is created, edited or deleted, or, on the other hand, the process model is changed, TDMS immediately validates all testcases and provides a detailed problem message in Fig. 4 (4) if a testcase failed. It is important to stress that the validation procedure is performed *automatically*, i.e, no user interaction is required to validate the testcases.

In order to ensure that all components work properly, TDMS has been developed using Test Driven Development, where applicable. In addition, researchers with different backgrounds, e.g., economics and computer sciences, have been included to develop an intuitive user interface. In a recent application of TDMS in a controlled experiment [10] no abnormal program behavior was observed. In addition, students considered TDMS as intuitive and easy to use.

3.2 Integration of Test Driven Modeling Suite

TDM, as introduced in Section 2, focuses on the modeling of declarative processes, TDMS provides the necessary operational support, i.e., tool support. To this end, TDMS makes use of Cheetah Experimental Platform's (CEP) [11] components for empirical research and integrates Declare [12] for workflow execution, as illustrated in Fig. 5 and detailed in the following.

Cheetah Experimental Platform as Basis. One of the design goals of TDMS was to make it amenable for empirical research, i.e., it should be easy to employ in experiments; data should be easy to collect and analyze. For this purpose, TDMS was implemented as an experimental workflow activity of CEP, allowing

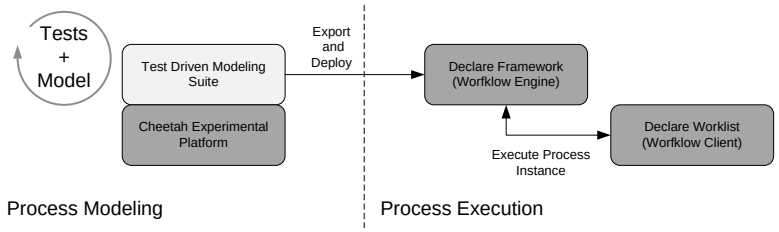

Fig. 5. Interplay of TDMS, CEP and Declare

TDMS to be integrated in any experimental workflow (i.e., a sequence of activities performed during an experiment, cf. [11]). In addition, we use CEP to instrument TDMS, i.e., to log each relevant user interaction to a central data storage. This logging mechanism, in combination with CEP's replay feature, allows the researcher to inspect in detail how TDMS is used to create process models and testcases by watching the process of modeling step-by-step.

Business Process Execution. In order to allow for the execution of declarative process models created in TDMS, an export mechanism to Declare [12] is provided. As illustrated in Fig. 5, testcases and process models are iteratively created in TDMS. For deployment, the process model is converted into a format that can be directly fed into the Declare framework, i.e., workflow engine. Then, the Declare worklist allows for the execution of the process instance.

4 Example

A preliminary empirical evaluation shows the positive influence of TDM on cognitive load and perceived quality during model maintenance [10]. To illustrate the influence of TDMS on process modeling, we provide an example that shows how a DE and a MB could use TDMS to create a process model and respective testcases describing of how to supervise a master thesis (cf. Fig. 6–8). For the sake of brevity, the example is kept on an abstract level and the following abbreviations are used:

D: *Discuss topic* **P:** *Provide feedback* **G:** *Grade work*

Starting from an empty process model, the DE lines out general properties of the process: *"When supervising a master thesis, at first the topic needs to be discussed with the student. While the student works on his thesis, feedback may be provided at any time. Finally, the thesis needs to be graded."*. Thus, possibly with help of the MB, the DE inserts activities *D*, *P* and *G* in the testcase's execution trace (cf. Fig. 6); TDMS automatically creates respective activities in the process model. Now, the DE and MB run the testcase and the test engine reports that the testcase passes.

Subsequently, the DE and MB engage in a dialogue of questioning and answering [13]—the MB challenges the model: *"So every thesis must start by discussing the topic?"*. *"Yes, indeed—you need to establish common knowledge first."*, the DE replies. Thus, they create a new testcase capturing this requirement and run it. Apparently, the testcase fails as there are no constraints in the

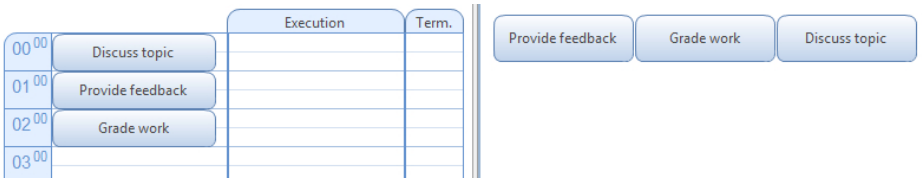

Fig. 6. Testcase 1: <*D,P,G*> Proposed by the DE

model yet. The MB inserts an init constraints on D (i.e., D must be the first activity in every process instance); now the testcase passes (cf. Fig. 7).

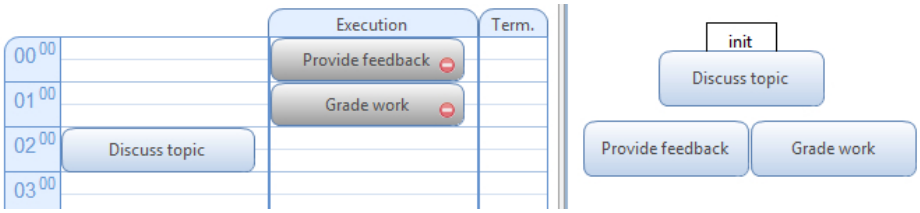

Fig. 7. Testcase 2: Introduction of Init on D

Again, the MB challenges the model and asks: *"Can the supervisor grade a thesis multiple times?"*. The DE replies: *"No, of course not, each thesis must be graded exactly once."* and together they specify a *third testcase* that ensures that G must be executed exactly once. By automatically validating this testcase, it becomes apparent that the current model allows G to be executed several times. Thus, the MB introduces a cardinality constraint on G (cf. Fig. 8).

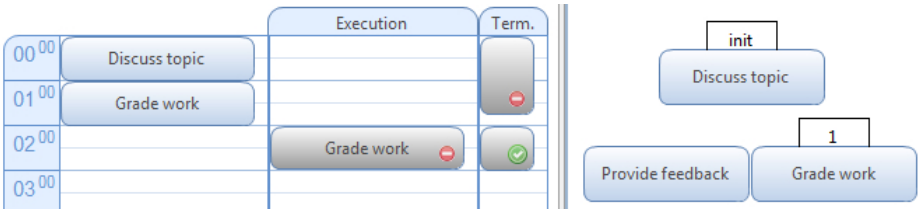

Fig. 8. Testcase 3: Introduction of Cardinality on G

While this example is kept small for the sake of brevity, it illustrates the benefits of using TDMS for modeling. First, the DE, who is usually not trained in reading or creating formal process models [8], is not required to modify the model itself, rather he defines behavior through the specification of testcases (possibly with the help of the MB). Second, testcases provide a common basis for understanding, thus supporting communication between the DE and MB. Third, behavior that is specified through testcases is validated automatically by TDMS, thereby ensuring that model changes do not violate desired behavior.

5 Summary and Outlook

TDMS, as described in this tool paper, provides operational support for the TDM methodology. More specifically, TDMS allows for a tight integration of declarative process models and testcases, thereby aiming at improving the communication between domain expert and model builder as well as resolving hidden dependencies. In addition, we sketched how we employ CEP as basis to make TDMS amenable for empirical research and showed how the Declare system is employed for the execution of declarative processes modeled in TDMS. Finally, we illustrated the intended usage of TDMS, in particular the iterative development of testcases and process model, with the help of a small example.

Future work focuses on further empirical validation: TDMS will be used in case studies to investigate whether the proposed methods are feasible in practice. In addition, TDMS will be employed in further controlled experiments to complement the case studies' results with quantitative data.

References

1. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. DKE **61** (2007) 39–58
2. Dumas, M., van der Aalst, W.M., ter Hofstede, A.H.: Process Aware Information Systems: Bridging People and Software Through Process Technology. Wiley-Interscience (2005)
3. Reichert, M., Dadam, P.: ADEPTflex: Supporting Dynamic Changes of Workflow without Losing Control. JIIS **10** (1998) 93–129
4. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, TU Eindhoven (2008)
5. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. ISJ **30** (2005) 349–378
6. Weber, B., Reijers, H.A., Zugal, S., Wild, W.: The Declarative Approach to Business Process Execution: An Empirical Test. In: Proc. CAiSE '09. (2009) 270–285
7. Zugal, S., Pinggera, J., Weber, B.: Toward Enhanced Life-Cycle Support for Declarative Processes. JSME (accepted)
8. van Bommel, P., Hoppenbrouwers, S., Proper, E., van der Weide, T.: Exploring Modelling Strategies in a Meta-modelling Context. In: Proc. OTM '06. (2006) 1128–1137
9. Green, T.R., Petre, M.: Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. JVLC **7** (1996) 131–174
10. Zugal, S., Pinggera, J., Weber, B.: The impact of testcases on the maintainability of declarative process models. In: Proc. BPMDS '11. (to appear)
11. Pinggera, J., Zugal, S., Weber, B.: Investigating the process of process modeling with cheetah experimental platform. In: Proc. ER-POIS '10. (2010) 13–18
12. Pesic, M., Schonenberg, H., van der Aalst, W.: DECLARE: Full Support for Loosely-Structured Processes. In: Proc. EDOC '07. (2007) 287–298
13. Hoppenbrouwers, S.J., Lindeman, L., Proper, E.H.: Capturing Modeling Processes - Towards the MoDial Modeling Laboratory. In: Proc. OTM '06. (2006) 1242–1252