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.

Keywords: 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], case handling [4], declarative processes [5] and late binding and modeling [6] (an overview is provided in [7]).

Although declarative processes promise a high degree of flexibility, avoid over-specification and provide more maneuvering for end-users [8], [5], they are not widely adopted in practice yet. In particular, as pointed out in [8], [9], [10], understandability problems hamper the usage of declarative process models. For instance, checking whether a process instance is supported by a process schema, is far from trivial. An approach tackling these problems, the *Test Driven*

Modeling (TDM) methodology, is presented in [10]. TDM aims at improving the understandability of declarative process models as well as the communication between domain experts [11] and model builders [11] by adopting the concept of *test cases* from software engineering. The contribution of this paper is to describe *Test Driven Modeling Suite* (TDMS)[1], i.e., the software that provides operational support for TDM.

The remainder of this paper is structured as follows: Section 2 briefly introduces declarative business process models, Section 3 shortly discusses TDM. Then, Section 4 describes the software architecture and features of TDMS, whereas Section 5 illustrates the usage of TDMS by an example. Finally, Section 6 deals with related work and Section 7 concludes with a summary and an outlook.

2 Declarative Process Models

There has been a long tradition of modeling business processes in an imperative way. Process modeling languages supporting this paradigm, like BPMN, EPC and UML Activity Diagrams, are widely used. Recently, *declarative approaches* have received increasing interest and suggest a fundamentally different way of describing business processes [8]. While imperative models specify exactly *how* things have to be done, declarative approaches only focus on the logic that governs the interplay of actions in the process by describing the *activities* that can be performed, as well as *constraints* prohibiting undesired behavior. An example of a constraint in an aviation process would be that crew duty times cannot exceed a predefined threshold. Constraints described in literature can be classified as execution and termination constraints. *Execution* constraints, on the one hand, restrict the execution of activities, e.g., an activity can be executed at most once. *Termination* constraints, on the other hand, affect the termination of process instances and specify when process termination is possible. For instance, an activity must be executed at least once before the process can be terminated. Most constraints focus either on execution *or* termination semantics, however, some constraints also combine execution and termination semantics (e.g., the succession constraint [8]).

To illustrate the concept of declarative processes, a declarative process model is shown in Fig. 1 a). It contains activities *A* to *F* as well as constraints *C1* and *C2*. *C1* prescribes that A must be executed at least once (i.e., *C1* restricts the termination of process instances). *C2* specifies that *E* can only be executed if *C* has been executed at some point in time before (i.e., *C2* imposes restrictions on the execution of activity E). In Fig. 1 b) an example of a process instance illustrates the semantics of the described constraints. After process instantiation, *A*, *B*, *C*, *D* and *F* can be executed. *E*, however, cannot be executed as *C2* specifies that *C* must have been executed before. This is indicated by the grey bar in Fig. 1 b) below “E”. Furthermore, the process instance cannot be terminated as *C1* is not satisfied, i.e., *A* has not been executed at least once. This is indicated

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

by the grey area in Fig. 1 b) below "Termination". The subsequent execution of B does not cause any changes as it is not involved in any constraint. However, after *A* is executed, *C1* is satisfied, i.e., *A* has been executed at least once and thus the process instance can be terminated (cf. Fig. 1 b). Hence, after *e4* the box below "Termination" is white. Then, *C* is executed, satisfying *C2* and consequently allowing *E* to be executed (the box below "E" is white after *e6* occurred). Finally, the execution of *E* does not affect any constraint, thus no changes with respect to constraint satisfaction can be observed. As all termination constraints are still satisfied, the process instance can still be terminated.

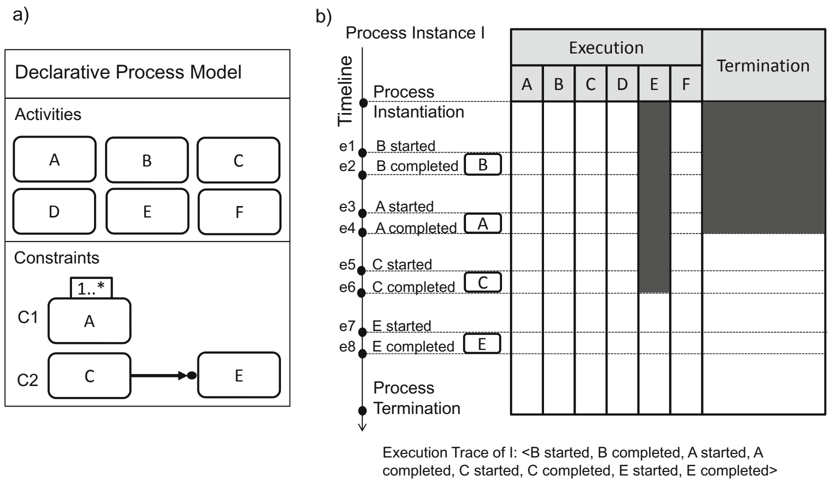


Fig. 1. Executing a declarative process

As illustrated in Fig. 1 b), a process instance can be specified through a list of *events* that describe changes in the life-cycle of *activity instances*, e.g., *"e1: B started"*. In the following, we will denote this list as *execution trace*, e.g., for process instance I: <*e1*, *e2*, *e3*, *e4*, *e5*, *e6*, *e7*, *e8*>. If events are non-overlapping, we merge subsequent start events and end events, e.g., <*B started*, *B completed*, *A started*, *A completed*> is abbreviated by <*B*, *A*>.

3 Test Driven Modeling

In the following, we briefly introduce TDM, the conceptual basis of TDMS. In particular, in Section 3.1 we will draw on concepts from cognitive psychology to shed light on possible causes of understandability problems related to declarative models. Subsequently, in Section 3.2, we discuss the most relevant concepts of TDM.

3.1 Cognitive Backgrounds

Declarative process models allow for the specification of flexible business processes [8], [12]. Still, as argued in [8], [9], [10], the understandability of respective models appears to be a hurdle for practical usage. Understandability thereby refers to how difficult it is to extract information from a process model. As detailed in [13], understandability is usually operationalized by asking questions about a process model. The more questions are answered correctly on average, the higher the understandability. Currently, it is still unclear for which reasons declarative models are harder to understand than imperative models. To provide a possible explanation, we would like build upon concepts from cognitive psychology. In particular, we identified that *computational offloading* [14], [15], [16] seems to play an essential role. In short, computational offloading allows the reader to *"offload"* computations to a diagram. In other words, the way how the diagram represents information allows the reader to quickly extract certain information. For instance, in a BPMN model control flow is explicitly represented by sequence flows (i.e., control edges) and gateways (e.g., AND gateway, OR gateway). Assume the reader wants to check whether a certain process instance is supported by an imperative process model. To this end, she may use the control edges to simulate the process instance by tracing through the process model. In this way, the model allows to offload the computation of the process instance. Contrariwise, one might describe the process model textually. Both representations (text and diagram) are information equivalent, i.e., the same information is present, however, the text does not allow the reader to quickly identify process instances, the reader has to simulate the process instance entirely in her head. Similarly, declarative process models do not provide explicit mechanisms to offload the computation of execution traces. Rather, as discussed in Section 2, the reader has to interpret the constraints in her mind. For a detailed discussion about computational offloading and related cognitive concepts, we refer to [17].

Assuming that computational offloading of computing process instances is not present in declarative models, i.e., reading imposes a high mental effort, repercussions on understandability, validation and maintainability can be expected. Since model understandability, as defined in [18], directly relates to reading and answering questions about a process model, as discussed above, a negative impact can be expected. Regarding validation, i.e., to check whether the model properly reflects the real-world business, an interesting insight is provided in [19]. The authors state that *"programmers rely heavily upon mental simulation for evaluating the validity of rules"*. Seen in the context of business process modeling, "mental simulation" refers to the "mental execution" of process instances. In other words, the person who validates the process model checks via "mental simulation" whether certain process instances, i.e., scenarios, are supported by a process model. As discussed, for a declarative business process model, the computation of process instances is far from trivial, hence a negative impact on validation can be expected. In further consequence, also the maintainability of declarative process models may be compromised, as argued in [10]. It is known that every change operation requires a *sense-making task*, i.e., determining what to change, as well as an *action task*, i.e., performing the change [20]. Compromised understandability supposedly compromises the sense-making task, which in turn impairs the change operation.

Basically, the idea of TDM is to provide computer-based support to compensate for the lack of computational offloading. In particular, *test cases* allow to capture and automatically validate scenarios, i.e., process instances, that should be supported by the process model. Likewise, test cases provide the option to specify anti-scenarios, i.e., behavior that should be forbidden by the process model.

3.2 Test Driven Modeling

Constraints, as introduced in Section 2, focus on forbidden behavior. TDM, however, introduces the concept of *test cases* to test for *desired* behavior of the process model. In particular, as illustrated in Fig. 2, TDM's meta model can be divided into two main parts: the specification of test cases (upper half) as well as the specification of the business process model (lower half). A *Test Driven Model* consists of exactly one *Declarative Process Model* and an arbitrary amount of *Test Cases.* A *Declarative Process Model*, as already discussed in Section 2, consists of at least one *Activity* as well as an arbitrary amount of *Constraints.* For the sake of brevity, Fig. 2 shows three constraints only, i.e., the *Response Constraint*, the *Precedence Constraint* and the *Coexistence Constraint.* TDMS actually supports all constraints described in [8], for a detailed description of the constraints we refer to [12]. Besides the specification of a *Declarative Process Model*, the meta model in Fig. 2 describes how *Test Cases* can be specified. In particular, a *Test Case* is built-up by a *Process Instance* (subsequently also referred to as *execution trace*) and an arbitrary number of *Assertions.* The *Process Instance*, in turn, consists of an arbitrary number of *Activity Instances.* For each *Activity Instance* a start event (i.e., when the activity instance enters the state *started* in its life-cycle) as well as and an end event (i.e., when the activity instance is *completed*) are defined. Similarly, each *Assertion* is defined for a certain window by its start- and end event. Within this window, a condition that is specified by the *Assertion* must hold. TDM thereby differentiates between two types of *Assertions*: an *Execution Assertion* can be used to verify whether a certain *Activity* is executable. The *positive* flag in *Assertion* thereby defines whether an *Activity* is expected to be *executable* or if the *Activity* is expected to be *non-executable.* Similarly, a *Termination Assertion* can be used to test whether the *Process Instance* can be terminated within a specified window.

In short, TDM allows for the specification of declarative process models and test cases. Each test case defines a certain scenario, i.e., process instance, that must be supported by the process model. Assertions can thereby be used to test for specific conditions, namely whether an activity is executable as well as whether the process instance can be terminated.

Consider, for illustration, the testcase depicted in Fig. 3. It contains the execution trace <*A*, *B*> (1) as well as an assertion that specifies that *A* cannot be executed between *e2* and *e3* (2) and assertions that specify that the process

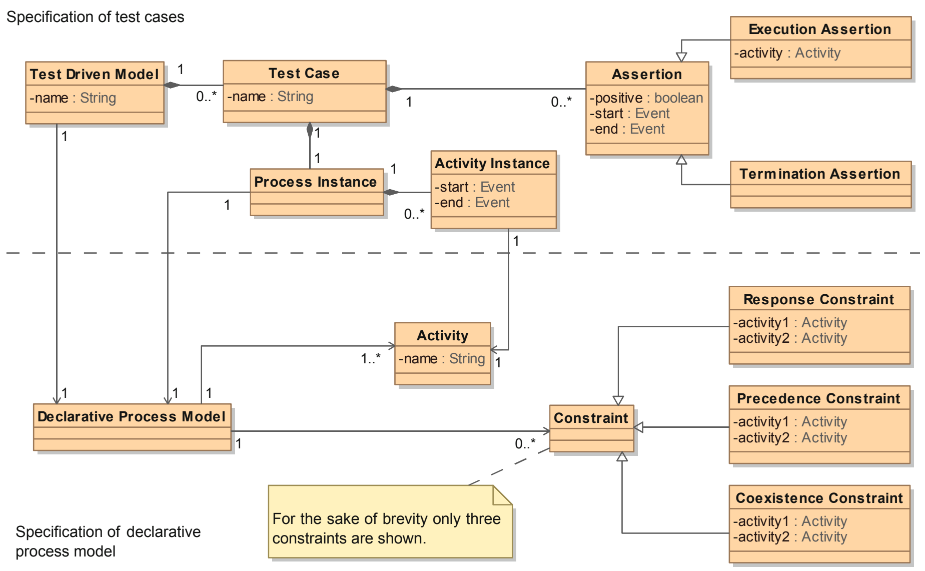


Fig. 2. Meta model of TDM

instance cannot be terminated before *e2* (3), however, it must be possible to terminate after *e2* (4). For the reason of simplicity, the example shows subsequent executions only. However, testcases can also be used to simulate parallel executions of activity. In this vein, also several different instances of the same activity may run at the same time—given that no constraint prohibits such behavior. The times in Fig. 3 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 test cases are validated automatically, i.e., no user interaction is required to check whether the specified behavior is supported by the process model.

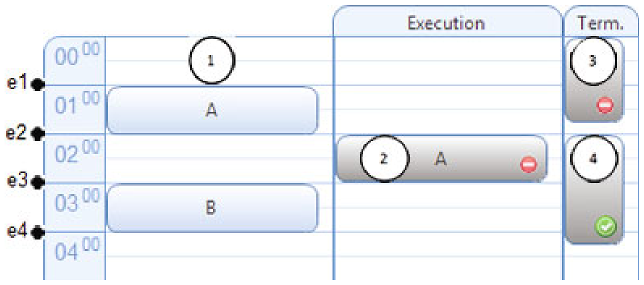


Fig. 3. A simple testcase

To illustrate how a test case may help to improve the understandability of a declarative process model, consider the test case illustrated in Fig. 4. The process model to the right (2) can be described in the following way: A must be executed exactly once (cf. cardinality constraint on A). After A has been executed, B must be executed (cf. response constraint between A and B). Thus, also B must be executed at least once for every process instance. However, this information is present in the process model implicitly only. Hence, the person who reads to model has to inspect the model carefully for such dependencies in order to properly understand the models' semantics—computational offloading is missing. According to [20], connections that are not directly visible in a model are referred to as *hidden dependencies*. As the name suggests, such dependencies are hard to see and detect, potentially misleading the reader and causing understandability problems. TDM allows the process modeler to actively resolve hidden dependencies by specifying a respective test case, thereby making the dependency explicit. To illustrate how this could be done for the given example, consider Fig. 4 (1): the test case specifies that the process instance can only be terminated if B has been executed at least once, making the hidden dependency explicit. As soon as the modeler conducts changes to the process model that violate the test case, the automated validation of TDMS (cf. Section 4) immediately informs the modeler, making her aware of the hidden dependency.

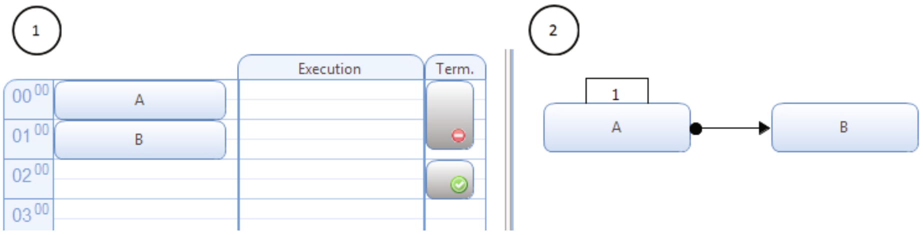


Fig. 4. Hidden dependency

So far we have introduced the concept of test cases and the intended impact on model understandability, in the following we will sketch how their adoption intends to improve the communication between domain expert (DE) and model builder (MB). First, it is worthwhile to note that test cases and process model are not meant to be created in isolation. Rather, as inspired by Test Driven Development [21], test cases and process model should be created interwoven (for a detailed discussion we refer to [10]). Thereby, test cases provide information in a form that is not only understandable to the MB, but also understandable to the DE, who normally does not have the knowledge to read formal process models [11]. Usually the DE needs the MB to retrieve information from the model, cf. Fig. 5 (2) and (3). Since test cases are understandable to the DE, they provide an additional communication channel to the process model, cf. Fig. 5 (4) and (6). It is important to stress that TDM's intention is not to make the DE specify the test cases in isolation. Rather, test cases should be created by the DE and the MB together and provide a common basis for discussion.

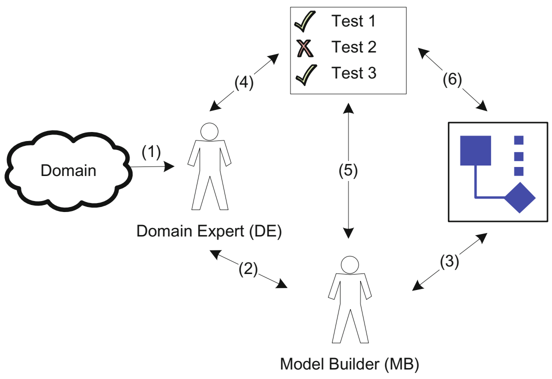


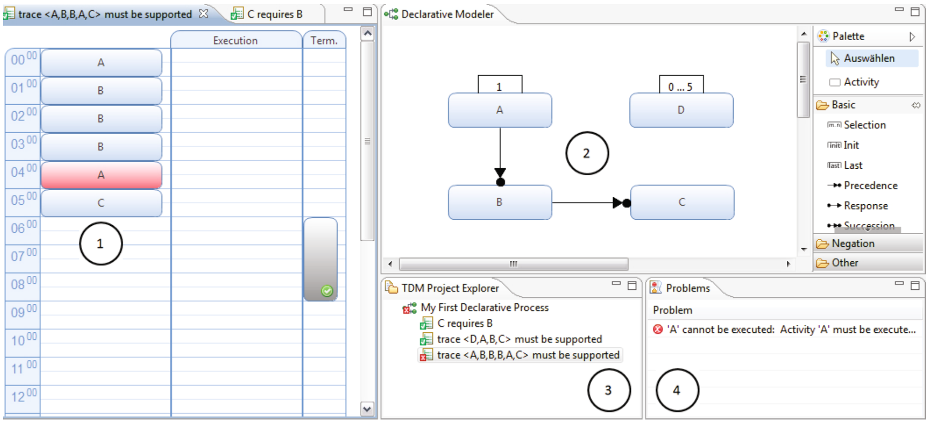
Fig. 5. Communication flow

4 Test Driven Modeling Suite

Up to now we have introduced the concept of TDM. This section deals with Test Driven Modeling Suite (TDMS) that provides operational support for TDM. In particular, Section 4.1 discusses the features of TDMS in detail. Subsequently, Section 4.2 describes how TDMS is integrated with existing frameworks for empirical research and business process execution.

4.1 Software Components

To give an overview of TDMS' features, a screenshot is provided in Fig. 6; each component will be described in detail in the following. On the left hand side TDMS offers a graphical editor for editing test cases (1). To the right, a graphical editor allows for designing the process model (2). Whenever changes are conducted, TDMS immediately validates the test cases against the process model and indicates failed test cases in the test case overview (3). In this case, it lists three test cases from which one failed. In addition, TDMS provides a detailed problem message about failed test cases 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 test case is indicated by the activity highlighted in (1), the test cases marked in (3) and the detailed error message in (4).



My First Declarative Process
C requires B
trace <D,A,B,C> must be supported
trace <A,B,B,B,A,C> must be supported

'A' cannot be executed: Activity 'A' must be execute...

Fig. 6. Screenshot of TDMS

Testcase Editor. As discussed in Section 3, test cases 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 test case editor as shown in Fig. 6 (1). Whether the user interface is indeed as intuitive, i.e., self-explanatory, is not entirely clear yet. So far we know that a group of students was able to use it after a short introduction [22]. In addition, further investigations into its usability are planned, cf. Section 7.

Declarative Process Model Editor. The declarative process model editor, as shown in Fig. 6 (2), provides a graphical editor for designing models in DecSerFlow [8], a declarative process modeling language.

Testcase Creation and Validation. In order to create new test cases or to delete existing ones, Fig. 6 (3) provides an outline of all test cases. Whenever a test cases is created, edited or deleted, or, on the other hand, the process model is changed, TDMS immediately validates all test cases. For the case a test case fails, TDMS provides a detailed problem message in Fig. 6 (4). It is important to stress that the validation procedure is performed *automatically*, i.e., no user interaction is required to validate the test cases. To this end, TDMS provides a *test engine* in which test cases are executed, as shown in Fig. 7. Basically, the test engine consists of a declarative process instance that is executed on a declarative workflow engine within a test environment. Thereby, TDMS' process model provides the basis for the process instance. The test cases steer the execution of the process instance, e.g., instantiating the process instance, starting activities or completing activities. In addition, test cases may also check the state of the process instance in the course of evaluating execution- or termination assertions. For a detailed description of test case validation, we refer to [10].

As pointed out in Section 3, the TDM methodolog is iterative, hence TDMS must also provide respective support. In particular, the iterative creation of

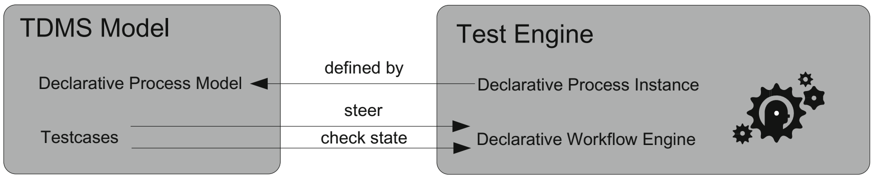

TDMS Model
Declarative Process Model
Testcases
defined by
steer
check state
Test Engine
Declarative Process Instance
Declarative Workflow Engine

Fig. 7. Testing framework

the process model poses a significant challenge, as any relevant change of the process model[2] requires the validation of testcases. However, existing declarative approaches either lead to exponential runtime for schema adaptations [8] or do not support workflow execution [12]. In order to tackle these problems, TDMS provides an own declarative workflow engine. Similar to Declare, where constraints are mapped to LTL formulas [23], TDMS' workflow engine maps constraints to Java[3] objects. In addition, for each process instance, the workflow engine keeps a list of events to describe its current state, as described in Section 2. The enablement of an activity can then be determined as detailed in the following. Based on the current process instance, a constraint is able to determine whether it restricts the execution of an activity. The workflow engine consults all defined constraints and determines for each constraint whether it restricts the execution. If no constraint vetos, the activity can be executed. For determining whether the process instance can be terminated, a similar strategy is followed. However, in this case constraints are asked whether they restrict the termination of the process instance instead.

Whenever a constraint should be added to the process model it is then sufficient to add this constraint to set of constraints to be checked. Similarly, when removing a constraint, the workflow engine does not consider the respective constraint anymore. While such an approach allows for efficient schema adaptations, it does not support verification mechanisms as provided in, e.g., Declare [23]. To compensate for this shortcoming, TDMS provides an interface to integrate third party tools for verification (cf. Section 4.2).

In order to ensure that all components work properly, TDMS has been developed using Test Driven Development [21], where applicable. In addition, researchers with different backgrounds, e.g., economics and computer science, have been included to develop an intuitive, i.e., self-explanatory, user interface. To validate whether our efforts succeeded, we used TDMS to teach declarative process modeling. In particular, we made use of TDMS' validation of test cases to allow students to interactively explore the semantics of a declarative process model. After a short introduction, students were able to work independently, indicating that operating TDMS, i.e., using the software, is easy to learn. Regarding the quality of TDMS, we would like to refer to a controlled experiment we recently

[2] Layouting operations, for instance, can be ignored here as they do not change the semantics of the process model.

[3] http://java.sun.com

performed [22]. Thereby, 12 students used TDMS for about 2 hours to adapt 2 declarative process models, i.e., a total of 24 process models were adapted. Throughout the experiment, no abnormal program behavior was observed. Apparently this does not mean that TDMS has industrial quality, however, TDMS meets the requirement for academic purposes as intended.

4.2 Integration of Test Driven Modeling Suite

TDM, as introduced in Section 3, 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) [24] components for empirical research and integrates Declare [23] for workflow execution and process model verification, as illustrated in Fig. 8 and detailed in the following.

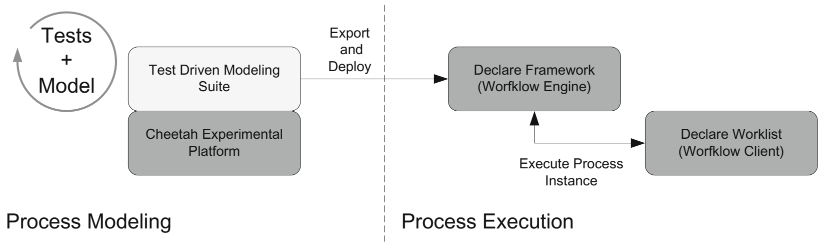

Tests
+
Model
Test Driven Modeling Suite
Cheetah Experimental Platform
Export and Deploy
Declare Framework (Worfklow Engine)
Declare Worklist (Worfklow Client)
Execute Process Instance
Process Modeling
Process Execution

Fig. 8. Interplay of TDMS, CEP and Declare

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. In addition, data should be easy to collect and analyze. For this purpose, TDMS was implemented as an experimental workflow activity of CEP, allowing TDMS to be integrated in any experimental workflow (i.e., a sequence of activities performed during an experiment, cf. [24]). Furthermore, 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 test cases step-by-step. Or, even more sophisticated, such a fine-grained instrumentation allows researchers and practioners to closely monitor the *process of process modeling*, i.e., the creation of the process model, using *Modeling Phase Diagrams* [25].

To illustrate how using CEP as basis for TDMS is beneficial for empirical research, we would like to refer to a recently performed experiment [22]. Therein, we investigated the impact of testcases on the maintainability of declarative process models. To this end, we provided students with two modeling assignments. For one of the modeling assignments, the full support of TDMS was available. For the other modeling assignment, only the process model editor was available.

In addition, we used a survey to assess demographic data such as modeling experience or education (for details we refer to [22]). The first benefit of CEP is that all these tasks are *automatically* presented to the students. Hence, no student could accidentally forget to fill out the demographic survey or to perform a modeling task. In other words, TDMS can seamlessly be integrated in such an experimental workflow. Thereby, all data is automatically collected and stored in a database. The second benefit of CEP comes out when evaluating the data gathered during the experiment. On the one hand, data can be exported *in an automated way* to comma-separated value files, which can then directly be analyzed using statistics software. On the other hand, collected data is fine-grained and therefore allows for in-depth evaluation. In this sense, we could show that with testcases at hand, twice as many constraints were added or deleted [22].

Process Model Verification and Execution. As discussed, the internal workflow engine of TDMS does not support the verification of declarative process models. However, it is known that the combination of constraints may lead to activities that can not be executed [8]. In order ensure that the process model is free from such *dead activities*, we make use of the verification provided in Declare [23]. In particular, as illustrated in Fig. 8, the process model is iteratively created in TDMS. For the purpose of verification, the process model is then converted into a format that can be read by the Declare framework. Similarly, this export mechanism can be used to execute the process model in Declare's workflow engine.

5 Example

A preliminary empirical evaluation shows a positive influence of TDM during model maintenance [22]. First, mental effort decreased, i.e., less cognitive resources were needed to conduct the change. Second, perceived quality increased, i.e., modelers were more confident about their changes—even though the quality of changes did not differ significantly. 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 test cases describing of how to supervise a master thesis (cf. Fig. 9–11). 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 test case's execution trace (cf. Fig. 9). TDMS automatically creates respective activities in the process model and the DE and MB run the test case. As the specified execution trace is supported by the process model, the test case passes.

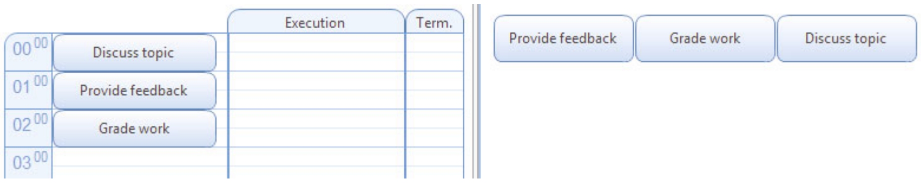


Fig. 9. Testcase 1: <*D*,*P*,*G*> proposed by the DE

Subsequently, the DE and MB engage in a dialogue of questioning and answering [26]—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 test case capturing this requirement and run it. Apparently, the test case fails as there are no constraints in the model yet. The MB inserts an init constraints on D (i.e., D must be the first activity in every process instance); now the test case passes (cf. Fig. 10).

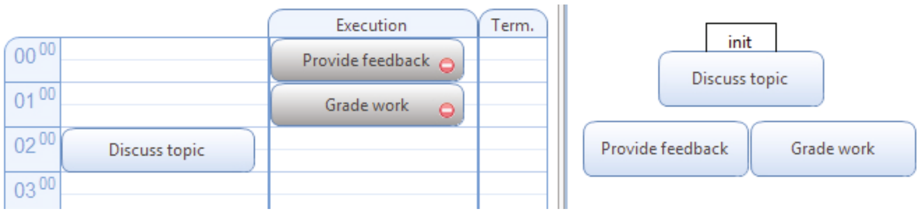


Fig. 10. Testcase 2: Introduction of Init on D

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 test case* that ensures that G must be executed exactly once. By automatically validating this test case, 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. 11).

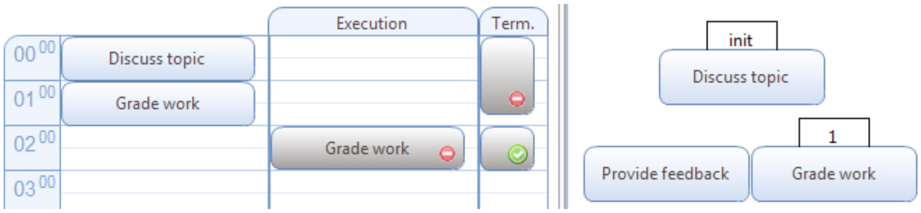


Fig. 11. Testcase 3: Introduction of Cardinality on G

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 [11], is not required to modify the model itself, rather he defines behavior through the specification of test cases (possibly with the help of the MB). Second, test cases provide a common basis for understanding, thus supporting communication between the DE and MB. Third, behavior that is specified through test cases is validated automatically by TDMS, thereby ensuring that model changes do not violate desired behavior. In this sense, test cases can be seen as a computer-supported kind of *modeling minutes* [27] that can be automatically validated against the process model.

6 Related Work

TDMS, as described in this work, allows for the interweaved creation of test cases and process models. The combination of conceptual models and test cases is far from new. For instance, in [28], [29], a language supporting automated test cases for conceptual schemas is presented. In contrast to this work, the language is not designed to be understood by the DE. In so-called *scenario-based approaches* the goal is to synthesize a conceptual model from a set of *scenarios*, i.e., test cases. The main difference to this work is the way how models are created. In our work, it is the responsibility of the MB to create the model. In the approaches described in the following, the model is *automatically* synthesized from scenarios. This way of synthesizing models is applicable to a variety of modeling languages and domains, as shown in [30], [31]. For instance, scenarios may be specified in Message Sequence Charts, Sequence Diagrams, Collaboration Diagrams [31] or Petri nets [32], [33]. Scenarios can then be synthesized to, e.g., Statecharts, Automatons [31] or Petri nets [32]. In principle, such approaches may be also applied to declarative process modeling. In this vein, Lamma et al. [34] describe how to extract declarative process models from process logs. While automated synthesis of declarative process models is certainly a viable approach to follow, as pointed out in [35], it is questionable in how far models that have been synthesized automatically are readable.

7 Summary and Outlook

In this work we started by introducing declarative business process models and associated problems. In particular, we lined out how the lack of computational offloading as well as the presence of hidden dependencies compromises model understandability, validation and maintainability. Subsequently, we sketched the most important concepts of TDM and discussed how it intends to improve the understandability of declarative process models and supports the communication between DE and MB. Then, we described TDMS that provides operational support for TDM. Thereby, we sketched how we employ CEP as basis to make TDMS amenable for empirical research and showed how Declare 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 test cases and process model, with the help of a small example.

We acknowledge that it is not yet entirely clear whether TDM and TDMS in particular help to foster the communication between DE and MB as well as improve the understandability and maintainability of declarative process models. Still, regarding maintainability, we would like to briefly sketch the findings from a controlled experiment [22]. The results show that test cases are able to lower mental effort during model adaptations and to improve perceived quality of the resulting models. For the quality of resulting models, however, no effects could be observed. As we argue, this does not necessarily imply that test cases are not able to improve quality. Rather, a certain model complexity is required for test cases to be beneficial.

In order to investigate whether test cases are indeed beneficial above a certain model complexity, we are currently preparing a replication of the experiment described in [22]. In addition, we are preparing a case study in which TDMS will be applied in real-world modeling scenarios. Therein, we will investigate in how far the adoption of TDMS influences the communication between DE and MB.

References

1. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. DKE 61, 39–58 (2007)
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, 93–129 (1998)
4. van der Aalst, W.M.P., Weske, M.: Case handling: a new paradigm for business process support. DKE 53, 129–162 (2005)
5. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-Based Workflow Models: Change Made Easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
6. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. ISJ 30, 349–378 (2005)
7. Weber, B., Reichert, M., Rinderle, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. DKE 66, 438–466 (2008)
8. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, TU Eindhoven (2008)
9. Weber, B., Reijers, H.A., Zugal, S., Wild, W.: The Declarative Approach to Business Process Execution: An Empirical Test. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 470–485. Springer, Heidelberg (2009)
10. Zugal, S., Pinggera, J., Weber, B.: Toward Enhanced Life-Cycle Support for Declarative Processes. JSME (2011), doi:10.1002/smr.554
11. van Bommel, P., Hoppenbrouwers, S.J.B.A., Proper, H.A(E.), van der Weide, T.P.: Exploring Modelling Strategies in a Meta-modelling Context. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1128–1137. Springer, Heidelberg (2006)

12. Montali, M., Pesic, M., van der Aalst, W., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Trans. Web 4, 1–62 (2010)
13. Zugal, S., Pinggera, J., Weber, B., Mendling, J., Reijers, H.A.: Assessing the impact of hierarchy on model understandability—a cognitive perspective. In: Proc. EESSMod 2011, pp. 18–27 (2011)
14. Scaife, M., Rogers, Y.: External cognition: how do graphical representations work? Int. J. Human-Computer Studies 45, 185–213 (1996)
15. Zhang, J., Norman, D.A.: Representations in distributed cognitive tasks. Cognitive Science 18, 87–122 (1994)
16. Zhang, J.: The nature of external representations in problem solving. Cognitive Science 21, 179–217 (1997)
17. Zugal, S., Pinggera, J., Weber, B.: Assessing process models with cognitive psychology. In: Proc. EMISA 2011, pp. 177–182 (2011)
18. Reijers, H.A., Mendling, J.: A Study into the Factors that Influence the Understandability of Business Process Models. IEEE Transaction on Systems Man & Cybernetics, Part A 41, 449–462 (2011)
19. Kim, J., Lerch, F.J.: Why Is Programming (Sometimes) So Difficult? Programming as Scientific Discovery in Multiple Problem Spaces. ISR 8, 25–50 (1997)
20. Green, T.R., Petre, M.: Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. JVLC 7, 131–174 (1996)
21. Beck, K.: Test Driven Development: By Example. Addison-Wesley (2002)
22. Zugal, S., Pinggera, J., Weber, B.: The Impact of Testcases on the Maintainability of Declarative Process Models. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) BPMDS 2011 and EMMSAD 2011. LNBIP, vol. 81, pp. 163–177. Springer, Heidelberg (2011)
23. Pesic, M., Schonenberg, H., van der Aalst, W.: DECLARE: Full Support for Loosely-Structured Processes. In: Proc. EDOC 2007, pp. 287–298 (2007)
24. Pinggera, J., Zugal, S., Weber, B.: Investigating the process of process modeling with cheetah experimental platform. In: Proc. ER-POIS 2010, pp. 13–18 (2010)
25. Pinggera, J., Zugal, S., Weidlich, M., Fahland, D., Weber, B., Mendling, J., Reijers, H.A.: Tracing the Process of Process Modeling with Modeling Phase Diagrams. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 370–382. Springer, Heidelberg (2012)
26. Hoppenbrouwers, S.J.B.A(S.), Lindeman, L(L.), Proper, H.A(E.): Capturing Modeling Processes – Towards the MoDial Modeling Laboratory. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1242–1252. Springer, Heidelberg (2006)
27. Hoppenbrouwers, S.J., Proper, E.H., van der Weide, T.P.: Formal Modelling as a Grounded Conversation. In: Proc. LAP 2005, pp. 139–155 (2005)
28. Tort, A., Olivé, A.: An approach to testing conceptual schemas. DKE 69, 598–618 (2010)
29. Tort, A., Olivé, A.: First Steps Towards Conceptual Schema Testing. In: Proc. CAiSE Forum 2009, pp. 1–6 (2009)
30. Amyot, D., Eberlein, A.: An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development. Telecommunication Systems 24, 61–94 (2003)

31. Liang, H., Dingel, J., Diskin, Z.: A comparative survey of scenario-based to state-based model synthesis approaches. In: Proc. SCESM 2006, pp. 5–12 (2006)
32. Fahland, D.: From Scenarios To Components. PhD thesis, Humboldt-Universität zu Berlin (2010)
33. Fahland, D.: Oclets – Scenario-Based Modeling with Petri Nets. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 223–242. Springer, Heidelberg (2009)
34. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing Declarative Logic-Based Models from Labeled Traces. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 344–359. Springer, Heidelberg (2007)
35. Glinz, M., Seybold, C., Meier, S.: Simulation-Driven Creation, Validation and Evolution of Behavioral Requirements Models. In: Proc. MBEES 2007, pp. 103–112 (2007)