Introducing Configurability into Scenario-Based Specification of Business Processes

Robert Prüfer and Jan Sürmeli

Humboldt-Universität zu Berlin, Germany {pruefer|suermeli}@informatik.hu-berlin.de

Abstract Process model configuration is an approach to model highly similar variants of a process. In a configurable process model, events can be hidden or blocked to characterize variants. However, it may be difficult to model large processes consisting of many interacting units. To this end, one may use scenario-based specification, specifying the process in comprehensible, reoccurring parts that describe interactional behavior. In this paper, we take a look at how configurability and scenario-based specification could be merged in one approach. We particularly focus on the impact of permitting events in scenarios to be hidden or blocked.

Keywords: Business Process Modeling, Process Configuration, Scenario-based Specification

1 Introduction

Business process modeling enables design, analysis, and optimization of existing and new processes. One approach is to start with a generic reference model, and then to refine the model iteratively until the desired level of detail is reached. During refinement many highly similar variants of the process arise. To capture all these variants, the modeler could create many similar models. This immediately leads to problems regarding the maintenance and refactoring of these models.

Process configuration [1,4,12] proposes to integrate all variants of a process in one single model, marking all possible variation points. A configurable process model M represents a finite set m_1, \ldots, m_n of highly similar process models. Each model m_i is the result of configuring M with a configuration c_i : A configuration interprets each variation point in M and thus yields a refined process model. Current approaches for configurable process models concentrate on classical formalisms to model concurrent processes such as workflow nets [2]. Such a process model captures the complete interaction of all units carrying out the process, e.g., people, web services, and information systems. Thus, it is difficult to model large processes, and to understand these models. Modeling each unit separately produces smaller models, and facilitates the analysis and implementation of each unit. However, the interaction of all units may still be hard to assess.

Scenario-based specification tackles this problem: A scenario describes a part of the interaction of many units. A specification consists of a set of scenarios,

N. Herzberg, M. Kunze (eds.): Proceedings of the 6th Central European Workshop on Services and their Composition (ZEUS 2014), Potsdam, Germany, 27-03-2014, published at http://ceur-ws.org

covering all desired interactions. Well-established scenario-based specification techniques include *High Level Message Sequence Charts* (HMSCs) [14] and *Live Sequence Charts* (LSCs) [13]. For some formalisms, there exist techniques to automatically derive a process model for each unit, bridging the scenario-based view with the classical view on distributed systems, facilitating the reuse of analysis techniques, and the implementation of each unit.

Our overall goal is to connect the clarity and intuitiveness of scenario-based specification with the advantages of configurable process models. In this paper, we discuss how the core concepts of process configuration may be introduced for scenario-based specifications. As in [3], we permit the three configuration options *allowing*, *blocking*, and *hiding* to be assigned to an event. We further propose possible meanings of these configuration options and compare them to the semantics of configurable workflow net models in [4].

We proceed as follows: We propose a syntax and discuss a possible semantics of configurable scenario-based specifications in Sect. 2. Afterwards, we discuss related work in Sect. 3. Finally, we conclude our paper and sketch possible future work in Sect. 4.

2 Configurable scenario-based specifications

In this section, we propose a syntax (Sect. 2.2) and discuss *possible semantics* (Sect. 2.3) for *configurable specifications*. First, we recall syntax and semantics of scenario-based specifications (Sect. 2.1). We mostly forgo formal definitions, and describe the concepts by means of the following running example adapted from [3]. We consider a business trip application process, roughly consisting of the following steps: Either an employee or a secretary prepares a form, which is then submitted to an administrator. Concurrently to the preparation, the employee arranges the travel. Upon receipt of a form, an administrator may either approve or reject the form, or request for changes. In the latter case, the employee updates the form, again arranges the travel and resubmits.

2.1 Syntax and semantics of scenario-based specifications

We informally recall the syntax and semantics of *distributed Life Sequence Charts* (DLSCs), a scenario-based specification language introduced in [10].

A partially ordered run (run, for short) is a set of events, partially ordered by causality. To each event e an activity $\alpha(e)$ is assigned. Figure 1 shows two runs: A labeled box represents an event e with its assigned activity $\alpha(e)$. The arrows model the causality relation. In run w_1 , first a start-event occurs causing an EArrange-event and an EPrepare-event. Once both events have occurred, an ESubmit-event occurs, causing an AApprove-event. In the following, the notions of predecessor and successor always refer to the causality relation. A prefix is a predecessor-closed set of events, a suffix is a successor-closed set of events.

A scenario is a finite run r distinctly partitioned into its *prechart* and its *mainchart*. The prechart contains at least the minimal events of r, that is, the

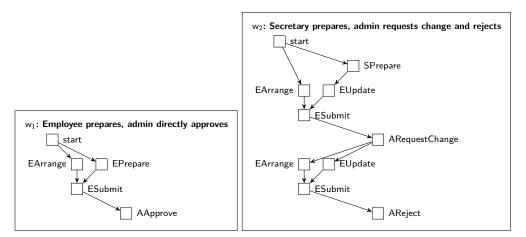


Figure 1: Two runs w_1 and w_2 of specification S in Fig. 2

events without predecessors. The main charts contains the remaining events. Graphically, we separate prechart and main chart by a dashed line. Ignoring the additional annotations in curly brackets, Fig. 2 shows six scenarios s_1, \ldots, s_6 . A *specification* S is a finite set of scenarios together with an *initial run* i. The scenarios s_1, \ldots, s_6 and the initial run i in Fig. 2 form the specification S.

The semantics of a specification S is the set $\mathcal{R}(S)$ of its maximal runs. We first introduce the notion of a run of S, then we define the notion of maximal runs. Intuitively, we construct runs as follows: We begin with the initial run, and subsequently append the main chart of a scenario whose prechart matches the suffix of the currently constructed run. Formally, we define the notion of a run of S recursively. As base case, the initial run of S is a run of S. Let w be a run of S, and s be a scenario of S, such that the prechart of s is a suffix (up to isomorphism) of w. Then, appending the main chart of s to w yields a run of S.

A run w of S is maximal, if it is not a prefix of any other run of S. That is, there is no prechart of a scenario in S which is a suffix (up to isomorphism) of w. As an example, in Fig. 1, w_1 starts with i. As i is isomorphic to the prechart of s_2 , we append the main chart of s_2 . The prechart of s_4 is now isomorphic to a suffix of the current run. We append s_4 , yielding w_1 . Similarly, we construct w_2 by appending the main charts of s_1 , s_3 , s_6 , s_3 , and s_5 to i. Both runs are maximal, therefore $w_1, w_2 \in \mathcal{R}(S)$.

2.2 Syntax of configurable specifications

We consider the *configuration options* of *allowing*, *blocking*, and *hiding*, denoted by \mathcal{A} , \mathcal{B} , and \mathcal{H} , respectively. According to [3], *allowing* means to not change behavior, *blocking* removes the event together with all its successors, and *hiding* an event means to skip it while preserving remaining behavior.

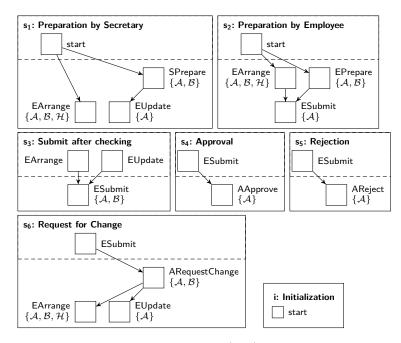


Figure 2: A *configurable specification* (S, C) consisting of the specification S with scenarios s_1, \ldots, s_6 and initial run i; C is defined by annotations in curly brackets.

To make a specification *configurable*, one chooses a set of configuration options for each event in a main chart. Thus, a *configurable specification* (S, C) is a specification S together with a function C mapping each event e of a main chart of a scenario in S to its *configuration options* $C(e) \subseteq \{\mathcal{A}, \mathcal{B}, \mathcal{H}\}$. In the following, we restrict ourselves to configurable specifications where $\mathcal{A} \in C(e)$ for each event e. We depict C as annotations to events. We omit \mathcal{A} , and if $C(e) = \{\mathcal{A}\}$, we completely omit the annotation. Figure 2 shows a configurable specification (S, C). In the following example, we write x.y for the y-event in scenario x. E.g., \mathcal{A} and \mathcal{B} are the configuration options of s_1 .SPrepare in Fig. 2.

A configuration c of a configurable specification (S, C) is a function mapping each event e of a main chart of S to $c(e) \in C(e)$. That is, a configuration chooses a configuration option for each event. We observe that c is properly determined by its blocked and hidden events. Thus, the following is a well-defined configuration of (S, C) in Fig. 2: $c_1 = \{s_1.SPrepare, s_3.ESubmit, s_6.ARequestChange \mapsto B\}$. In contrast to that, $c_2 = \{s_1.SPrepare, s_1.EUpdate, s_1.ARequestChange \mapsto B\}$ is not a configuration of (S, C), because $c_2(s_1.EUpdate) = B \notin C(s_1.EUpdate)$.

2.3 Semantics of configurable specifications

We propose the semantics of a configurable specification (S, C) together with a configuration c of (S, C) to be a specification $[\![(S, C)]\!]_c$. Thus, (S, C) represents

46 Robert Prüfer and Jan Sürmeli

a set of specifications. Following the semantics in [3], we propose that *allowing* an event does not interfere with its occurrence. Therefore, allowing each event in (S, C) yields S. In the following, we say that c introduces or removes behavior, if $\mathcal{R}(\llbracket(S, C)\rrbracket_c) \setminus \mathcal{R}(S) \neq \emptyset$ or $\mathcal{R}(S) \setminus \mathcal{R}(\llbracket(S, C)\rrbracket_c) \neq \emptyset$, respectively. In the remainder, we separately discuss possible semantics of *blocking* and *hiding*.

Blocking events. Intuitively, a blocked event and all its successors must not occur. We can think of two semantics $\mathcal{B}_{\text{full}}$ and $\mathcal{B}_{\text{part}}$ for blocking an event e in a scenario s: Under $\mathcal{B}_{\text{full}}$ -semantics, the whole scenario s cannot occur and therefore is removed from the specification. Under $\mathcal{B}_{\text{part}}$ -semantics e and all its successors are removed from s. For example, consider scenario s_1 in Fig. 2 and example configuration c_1 from Sect. 2. According to $\mathcal{B}_{\text{full}}$ -semantics, s_1 would be removed. According to $\mathcal{B}_{\text{part}}$ -semantics, only the events s_1 .SPrepare and s_1 .EUpdate would be removed from s_1 . We observe that both semantics $\mathcal{B}_{\text{full}}$ and $\mathcal{B}_{\text{part}}$ in general remove behavior: For example, under each of both semantics, w_2 in Fig. 1 is not a run of $[[(S, C)]]_{c_1}$ although it is a run of S. However, the $\mathcal{B}_{\text{part}}$ -semantics has an additional impact: Removing an event with its successors in a scenario in general also *introduces* behavior: The construction of a run may yield prefixes, and thus may even allow to append new scenarios.

Hiding events. According to [3], hiding an event e intuitively means to skip it while preserving all other behavior. Especially, the successors of e still can occur. Technically, $\alpha(e)$ is set to τ , neither removing nor introducing behavior.

The crucial point with hiding events in scenarios is the different *precondition* for the occurrence of an event and of a scenario, respectively. In [3], the occurrence of an event is determined by the *state* of the process. As changing the activity of an event e has no impact on the state reached, hiding does not influence occurrence of any event. In contrast, occurrence of a scenario is determined by its prechart, i.e., whether a partial order of activities occurred. Therefore, changing the activity of an event in some scenario s can influence whether another scenario s' can occur. Thus, changing the activity of an event can introduce or remove behavior.

Consequently, it must be discussed how $[\![(S,C)]\!]_c$ can be defined such that behavior is preserved. This could range from simple changes in single precharts to the insertion of new scenarios. An elaborate discussion of this definition is out of the scope of this paper and is left for future work.

3 Related work

We chose *distributed life sequence charts* (pLSCs)[10] as underlying formalism for scenarios, because (1) pLSCs are based on partially ordered runs of events, which easily allow to add concepts of configurability to distributed systems, and (2) there exist techniques [9,10] to synthesize distributed components out of a pLSC specification. Additionally, pLSCs adopt the concepts of *prechart* and *main chart* from LSCs [13] for composition of scenarios. Hence, a single scenario in form of a

DLSC describes a *self-contained story*, which may be advantageous compared to the automata-based composition mechanism of HMSCs [14] as discussed in [11].

Similar to the scenario-based approach, the idea of *Process Fragments* [8] is to model small pieces of a process and to compose them. In contrast to scenarios, in this approach processes are assumed to be acyclic. Further, the approach does not enable the modeler to specify a distinguished precondition for occurrence of a process fragment. To compose two process fragments p_1 and p_2 , it must be explicitly specified which activities of p_1 are to be connected to which activities of p_2 [7].

The configuration options – allowing, blocking and hiding of events – are introduced in [4]. In [3,4], the authors tackle the problem of finding and characterizing the set of all configurations leading to a *behaviorally correct* process model. In [6], the authors describe techniques to *discover* a configurable process model from an event log. In [15], the authors describe how configurable process models may be created by *merging* process models. Both discovery and merging thus can be seen as alternative approaches yielding configurable process models. As an alternative to creating a configurable process model, an approach to improve an existing reference model is described in [16]. For an overview of approaches to cope with variability in Business Processes, see [5].

In Software Product Lines Engineering (SPLE) [17], a feature model represents variants of a product. Whereas the main purpose of a feature model is to characterize valid product lines, modeling variants of a business process aims at verifying behavioral properties of this process.

4 Conclusion and future work

In this paper, we proposed syntax and possible semantics of configurable scenariobased specifications as a method to model variants of a process. We adopted an existing approach for configurable process models [4] to the scenario-based specification formalism of pLSCs [10]. As a proper semantics for hiding needs to be discussed carefully, this is an immediate starting point for future work. We restricted ourselves to a pure control flow view of a process. We believe that it is interesting to investigate *data-dependent configurability*. The approach to integrate data in scenarios in [11] could serve as a useful formal basis. This approach also introduces *abstraction*, allowing to specify optional behavior, which is interesting in combination with hiding. Further, a method to synthesize a configurable process model out of a configurable specification would allow to reuse techniques from [3] to characterize behaviorally correct process models. Whereas configurable process models have been assessed in case studies such as [12], the proposed formalism in this paper still needs evaluation. Here, we plan a case study in the healthcare sector. We believe that this is a reasonable application domain, because (1) healthcare processes consist of different actors performing complex interactional behavior, and (2) as individual treatment of patients inherently leads to different variants of one process, treatment of a significant number of patients could be captured by a configurable process model. 48 Robert Prüfer and Jan Sürmeli

References

- 1. Process configuration, http://www.processconfiguration.com
- van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: Classification, decidability, and analysis. Form. Asp. Comput. 23(3), 333–363 (May 2011)
- 3. van der Aalst, W.M.P., Lohmann, N., La Rosa, M.: Ensuring correctness during process configuration via partner synthesis. Inf. Syst. 37(6), 574–592 (2012)
- van der Aalst, W., Dumas, M., Gottschalk, F., ter Hofstede, A., La Rosa, M., Mendling, J.: Preserving Correctness During Business Process Model Configuration. Formal Aspects of Computing (2010)
- Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: Enhancing modeling and change support for process families through change patterns. In: Nurcan, S., Proper, H.A., Soffer, P., Krogstie, J., Schmidt, R., Halpin, T.A., Bider, I. (eds.) BMMDS/EMMSAD. Lecture Notes in Business Information Processing, vol. 147, pp. 246–260. Springer (2013)
- Buijs, J., Dongen, B., van der Aalst, W.: Mining configurable process models from collections of event logs. In: Daniel, F., Wang, J., Weber, B. (eds.) Business Process Management, Lecture Notes in Computer Science, vol. 8094, pp. 33–48. Springer Berlin Heidelberg (2013)
- Eberle, H., Leymann, F., Schleicher, D., Schumm, D., Unger, T.: Process fragment composition operations. In: APSCC. pp. 157–163. IEEE Computer Society (2010)
- Eberle, H., Unger, T., Leymann, F.: Process fragments. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM Conferences (1). Lecture Notes in Computer Science, vol. 5870, pp. 398–405. Springer (2009)
- Fahland, D.: From Scenarios To Components. Ph.D. thesis, Humboldt-Universität zu Berlin (2010), http://repository.tue.nl/685341
- Fahland, D., Kantor, A.: Synthesizing decentralized components from a variant of live sequence charts. In: Hammoudi, S., Pires, L.F., Filipe, J., das Neves, R.C. (eds.) MODELSWARD. pp. 25–38. SciTePress (2013)
- Fahland, D., Prüfer, R.: Data and Abstraction for Scenario-Based Modeling with Petri Nets. In: Haddad, S., Pomello, L. (eds.) Petri Nets 2012. Lecture Notes in Computer Science, vol. 7347, pp. 168 – 187. Hamburg, Germany (June 2012)
- Gottschalk, F., Wagemakers, T., Jansen-Vullers, M., van der Aalst, W., La Rosa, M.: Configurable process models: Experiences from a municipality case study. In: Proceedings of the 21st International Conference on Advanced Information Systems (CAiSE 09). Lecture Notes in Computer Science, vol. 5565, pp. 486–500. Springer Verlag, Berlin Heidelberg (June 2009)
- 13. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2003)
- 14. ITU-T: Message Sequence Chart (MSC). Recommendation Z.120, International Telecommunication Union, Geneva (2004)
- La Rosa, M., Dumas, M., Uba, R., Dijkman, R.: Business process model merging: An approach to business process consolidation. ACM Trans. Softw. Eng. Methodol. 22(2), 11:1–11:42 (Mar 2013)
- Li, C., Reichert, M., Wombacher, A.: The minadept clustering approach for discovering reference process models out of process variants. International Journal of Cooperative Information Systems 19(03n04), 159–203 (2010)
- 17. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering -Foundations, Principles, and Techniques. Springer (2005)