Weak Conformance between Process Models and Synchronized Object Life Cycles

Andreas Meyer and Mathias Weske

Hasso Plattner Institute at the University of Potsdam
{Andreas.Meyer,Mathias.Weske}@hpi.de

Abstract. Process models specify behavioral execution constraints between activities as well as between activities and data objects. A data object is characterized by its states and state transitions represented as object life cycle. For process execution, all behavioral execution constraints must be correct. Correctness can be verified via soundness checking which currently only considers control flow information. For data correctness, conformance between a process model and its object life cycles is checked. Current approaches abstract from dependencies between multiple data objects and require fully specified process models although, in real-world process repositories, often underspecified models are found. Coping with these issues, we apply the notion of weak conformance to process models to tell whether each time an activity needs to access a data object in a particular state, it is guaranteed that the data object is in or can reach the expected state. Further, we introduce an algorithm for an integrated verification of control flow correctness and weak data conformance using soundness checking.

1 Introduction

Business process management allows organizations to specify their processes structurally by means of process models, which are then used for process execution. Process models comprise multiple perspectives with two of them receiving the most attention in recent years: control flow and data [22]. These describe behavioral execution constraints between activities as well as between activities and data objects. It is usually accepted that control flow drives execution of a process model. While checking control flow correctness using soundness [1] is an accepted method, correctness regarding data and control flow is not addressed in sufficient detail. In this paper, we describe a formalism to integrate control flow and data perspectives that is used to check for correctness.

In order to achieve safe execution of a process model, it must be ensured that every time an activity attempts to access a data object, the data object is in a certain expected data state or is able to reach the expected data state from the current one, i.e., data specification within a process model must conform to relevant object life cycles, where each describes the allowed behavior of a distinct class of data objects. Otherwise, the execution of a process model may deadlock. To check for deadlock-free execution in terms of data constraints, the notion of object life cycle conformance [9, 20] is used. This approach has some restrictions with respect to data constraint specification, because each single change of a data object as specified in the object life cycle, we refer to as data state transition, must be performed by some activity. [21] relaxes this limitation such that several state changes can be subsumed within one activity. However, gaps within the data constraints specification, i.e., implicit data state transitions, are not allowed although some other process may be responsible of performing a state change of an object, i.e., these approaches can only check whether an object is in a certain expected state. We assume that implicit data state transitions get realized by an external entity or by detailed implementations of process model activities. In real world process repositories, usually many of those *underspecified* process models exist, which motivates the introduction of the notion of weak conformance [13]. It allows to also check underspecified models.

Additionally, in real world, often dependencies between multiple data objects exist; e.g., an order may only be shipped to the customer after the payment is recognized. Non of above approaches supports this. Thus, we utilize the concept of synchronized object life cycles that allows to specify dependencies between data states as well as state transitions of different object life cycles [16]. Based thereon, we extend the notion of weak conformance and describe how to compute it for a given process model and the corresponding object life cycles including synchronizations. We utilize the well established method of soundness checking [1] to check for process model correctness. For mapping a process model to a Petri net, we utilize an extension covering data constraints [16] to a widely-used control flow mapping [4] to enable an integrated checking of control flow and data correctness. Further, fundamentals and preliminaries required in the scope of this paper are discussed in Section 2 of our report [16].

The remainder is structured as follows. First, we discuss weak conformance in general and compare it to existing conformance notions in Section 2 before we introduce the extended notion of weak conformance in Section 3. Afterwards, we discuss the procedure for integrated correctness checking in Section 4. Section 5 is devoted to related work before we conclude the paper in Section 6.

2 Weak Conformance

The notion of weak conformance has been initially proposed in [13] as extension to the notion of object life cycle conformance [9, 20] to allow the support of underspecified process models. A fully specified process model contains all reads and writes of data nodes by all activities. Additionally, each activity reads and writes at least one data node except for the first and last activities, which may lack reading respectively writing a data node in case they only create respectively consume a data node.

In contrast, underspecified process models may lack some reads or writes of data nodes such that they are implicit, performed by some other process, or they are hidden in aggregated activities changing the state multiple times with respect to the object life cycle. Though, full support of underspecified process models requires that the process model may omit state changes of data nodes although they are specified in the object life cycle.

Table 1. Applicability and time complexity of data conformance computation algorithms

Attribute	[9, 20]	[21]	[13]	this
Full specification	+	+	+	+
Underspecification	-	o	+	+
Synchronization	-	-	-	+
Complexity	exp.	poly.	–	exp.

In this paper, we extend the notion of weak conformance to also support object life cycle synchronization. First, we compare different approaches to check for conformance between a process model and object life cycles. Table 1 lists the applicability and specifies the time complexity of the computation algorithms for approaches described in [9, 20], [21], [13], and this paper. The notion from [9, 20] requires fully specified process models and abstracts from inter-dependencies between object life cycles by not considering them for conformance checking in case they are modeled. Conformance computation is done in polynomial time. In [21], underspecification of process models is partly supported, because a single activity may change multiple data states at once (aggregated activity). Though, full support of underspecified process models would require that the process model may omit data state changes completely although they are specified in the object life cycle. Synchronization between object life cycles is not considered in that approach and complexity-wise, it requires exponential time. [13] supports fully and underspecified process models but lacks support for object life cycle synchronization, which is then solved by the extension described in this section. For [13], no computation algorithm is given such that no complexity can be derived. The solution presented in this paper requires exponential time through the Petri net mapping and subsequent soundness checking as described in Section 4. However, state space reduction techniques may help to reduce the computation time for soundness checking [6]. The choice of using soundness checking to verify weak conformance allows to check for control flow soundness as well as weak conformance in one analysis and still allows to distinguish occurring violations caused by control flow or data flow.

3 The Notion of Weak Conformance

Weak conformance is checked for a process model with respect to the object life cycles referring to data classes used within the process model. To such concept, we refer as *process scenario* $h = (m, \mathcal{L}, C)$, where m is the process model, $\mathcal{L}$ is the synchronized object life cycle, and C is the set of data classes. Next, we define several notions for convenience considerations before we introduce the notion of weak conformance. Let $f \in \mathfrak{F}_m$ be a data flow edge of process model m indicating either a data object read or write. With f_A and f_D, we denote the activity (A) and data node (D) component of f, respectively. For instance, if f is equal to (a, d), a read, or to (d, a), a write, then (in both cases) $f_A = a$ and $f_D = d$. With $\vartheta(f)$, we denote the data state r_d involved in a read ($f = (d, a) \in \mathfrak{F}$) or write ($f = (a, d) \in \mathfrak{F}$) operation. We denote the set of synchronization edges having data state r_d as target data state with SE_r. Further, $a \Rightarrow_m a'$ denotes that there exists a path in process model m which executes activity $a \in A_m$ before activity $a' \in A_m$. Analogously, $s \Rightarrow_{l_c} s'$ denotes that there exists a path in the object life cycle l_c of data class c which reaches state $s \in S_c$ before state $s' \in S_c$. Thereby, we assume trace semantics. Due to space limitations, details about the concepts utilized throughout this paper and especially in this section can be found in [16], where we introduce the corresponding fundamentals.

Definition 1 (Weak Data Class Conformance). Given process scenario $h = (m, \mathcal{L}, C)$, $m = (N, D, Q, \mathfrak{C}, \mathfrak{F}, type, \mu, \varphi)$ and $\mathcal{L} = (L, SE)$, process model m satisfies *weak conformance* with respect to data class $c \in C$ if for all $f, f' \in \mathfrak{F}$ such that $f_D = d = f'_D$

with d referring to c holds (i) $f_A \Rightarrow_m f'_A$ implies $\vartheta(f) \Rightarrow_{l_c} \vartheta(f')$, (ii) $\forall se \in SE_{\vartheta(f')}$ originating from the same object life cycle $l \in L : \exists \xi(se) == true$, and (iii) $f_A = f'_A$ implies f represents a read and f' represents a write operation of the same activity. $\diamond$

Given a process scenario, we say that it satisfies *weak conformance*, if the process model satisfies weak conformance with respect to each of the used data classes. Weak data class conformance is satisfied, (i),(iii) if for the data states of each two directly succeeding data nodes referring to the same data class in a process model there exists a path from the first to the second data state in the corresponding object life cycle and (ii) if the dependencies specified by synchronization edges with a target state matching the state of the second data node of the two succeeding ones hold such that all dependency conjunctions and disjunctions are fulfilled. Two data nodes of the same class are directly succeeding in the process model, if either (1) they are accessed by the same activity with one being read and one being written or (2) there exists a path in the process model in which two different activities access data nodes of the same class in two data states with no further access to a node of this data class in-between.

4 Computation of Weak Conformance via Soundness Checking

A given process scenario $h = (m, \mathcal{L}, C)$ can be checked for weak conformance by applying the following four steps in sequence:

1. Map the process model m and the synchronized object life cycle $\mathcal{L}$ to Petri nets,
2. integrate both Petri nets,
3. post-process the integrated Petri net and transform it to a workflow net system, and
4. apply soundness checking to identify violations within the process scenario h.

Before we discuss these four steps, we recall the notions of preset and postset. A preset of a transition t respectively a place p denotes the set of all places respectively transitions directly preceding t respectively p. A postset of a transition t respectively a place p denotes the set of all places respectively transitions directly succeeding t respectively p.

1—Petri Net Mapping: The process model is mapped to a Petri net following the rules described in [4] for the control flow and in [16] for the data flow. The mapping of the synchronized object life cycle is split. First, each single object life cycle $l \in L$ is mapped to a Petri net, which than secondly are integrated utilizing the set of synchronization edges. The mapping of single object life cycles utilizes the fact that Petri nets are state machines, if and only if each transition has exactly one preceding and one succeeding place [2]. Thus, each state of an object life cycle is mapped to a Petri net place and each data state transition connecting two states is mapped to a Petri net transition connecting the corresponding places.

For each typed synchronization edge, one place is added to the Petri net. If two typed synchronization edges have the same source and the same dependency type, target the same object life cycle, and if the corresponding target states each have exactly one incoming synchronization edge, both places are merged to one. Similarly, two places are merged, if two typed synchronization edges have the same target, the same dependency type, and origin from the same object life cycle. The preset of an added place comprises all transitions directly preceding the places representing the source and the target data states of the corresponding synchronization edge. The postset of an added place comprises all transitions directly preceding the place representing the target state of the synchronization edge. For currently typed edges, the postset additionally comprises the set of all transitions directly succeeding the place representing the source state.

For each untyped synchronization edge, one transition is added to the Petri net. If $\bigcap_{se_T} \{src \cup tgt\} \neq \emptyset$ for two untyped synchronization edges, i.e., they share one data state, then both transitions are merged. The preset and postset of each transition comprise newly added places; one for each (transitively) involved synchronization edge for the preset and the postset respectively. Such preset place directly succeeds the transitions that in turn are part of the preset of the place representing the data state from which the data state transition origins. Such postset place directly precedes the transition representing the corresponding source or target transition of the typed synchronization edge.

2—Petri Net Integration: First, data states occurring in the object life cycles but not in the process model need to be handled to ensure deadlock free integration of both Petri nets. We add one place p to the Petri net, which handles all not occurring states, i.e., avoids execution of these paths. Let each q_i be a place representing such not occurring data state. Then, the preset of each transition t_j being part of the preset of q_i is extended with place p, if the preset of t_j contains a data state which postset comprises more than one transition in the original Petri net mapped from the synchronized object life cycle.

Each data state represented as place in the Petri net mapped from the process model consists of a control flow and a data flow component as visualized in Fig. 1 with C and D. Within the integrated Petri net, the control flow component is responsible for the flow of the object life cycle and the data flow component is responsible for the data flow in the process model. The integration of both Petri nets follows three rules, distinguishable with respect to read and write operations. The rules use the data flow component of data state places.

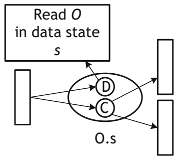

Fig. 1. Internal places for a place representing a data state

(IR-1) A place p from the object life cycle Petri net representing a data state of a data class to be read by some activity in the process model is added to the preset of the transition stating that this data node (object) is read in this specific state, e.g., the preset of transition *Read O in data state s* is extended with the place representing data state s of class *O*, and (IR-2) a new place q is added to the integrated Petri net, which extends the postset of the transition stating that the data node (object) is read in the specific state and which extends the preset of each transition being part of the postset of place p, e.g., the place connecting transition *Read O in data state s* and the two transitions succeeding the place labeled *O.s*. (IR-3) Let v be a place from the object life cycle Petri net representing a data state of a class to be written by some activity in the process model. Then a new place w is added to the integrated Petri net, which extends the preset of each transition being part of the preset of w and which extends the postset of the transition stating that the data node (object) is written in the specific state. the Petri net derived from the process model stating this write.

3—Workflow Net System: Soundness checking has been introduced for workflow net systems [1,12]. Workflow nets are Petri nets with a single source and a single sink place and they are strongly connected after adding a transition connecting the sink place with the source place [1]. The integrated Petri net needs to be post-processed towards these properties by adding *enabler* and *collector* fragments. The enabler fragment consists of the single source place directly succeeded by a transition y. The postset of y comprises all places representing an initial data state of some object life cycle and the source place of the process model Petri net. The preset of each place is adapted accordingly.

The collector fragment first consists of a transition t preceding the single sink node. For each distinct data class of the process scenario, one place p_i and one place q_i are added to the collector. Each place p_i has transition t as postset[1]. Then, for each final data state of some object life cycle, a transition u_i is added to the collector. Each transition u_i has as preset the place representing the corresponding data state and some place q_i referring to the same data class. The postset of a transition u_i is the corresponding place p_i also referring to the same data class. Additionally, a transition z succeeded by one place is added to the collector. The place's postset is transition t. The preset of z is the sink place of the process model Petri net. The postset of z is extended with each place q_i.

Next, the synchronization places need to be considered. If a typed synchronization edge involves the initial state of some object life cycle as source, then the corresponding place is added to the postset of transition y of the enabler fragment. For all synchronization edges typed previously, the postset of the corresponding place is extended with transition t of the collector. If a currently typed synchronization edge involves a final state of some object life cycle as source, then the corresponding place is added to the postset of the corresponding transition u_i of the collector fragment. Finally, the semaphore places need to be integrated. Therefore, for each semaphore place, the preset is extended with transition y from the enabler and the postset is extended with transition t from the collector fragments. Now, connecting sink and source node, the workflow net is strongly connected. A workflow net system consists of a workflow net and some initial marking. The workflow net is given above and the initial marking puts a token into the single source place and nowhere else.

4—Soundness Checking: Assuming control flow correctness, if the workflow net system satisfies the soundness property [1], no contradictions between the process model and the object life cycles exist and all data states presented in all object life cycles are implicitly or explicitly utilized in the process model, i.e., all paths in the object life cycles may be taken. If it satisfies the weak soundness property [12], no contradictions between the process model and the object life cycles exist but some of the data states are never reached during execution of the process model. In case, control flow inconsistencies would appear, places and transitions representing the control flow would cause the violation allowing to distinguish between control flow and data conformance issues.

Validation. The described approach reliably decides about weak conformance of a process scenario. It takes sound Petri net fragments as input and combines them with

[1] Generally, we assume that addition of one element a to the preset of another element b implies the addition of b to the postset of a and vice versa.

respect to specified data dependencies. Single source and sink places are achieved through the addition of elements either marking the original source places or collecting tokens from the original final places. Thus, they do not change the behavior of the process model and the object life cycles, i.e., they do not influence the result.

5 Related Work

The increasing interest in the development of process models for execution has shifted the focus from control flow to data flow perspective leading to integrated scenarios providing control as well as data flow views. One step in this regard are object-centric processes [3,17,23] that connect data classes with the control flow of process models by specifying object life cycles. [8] introduces the essential requirements of this modeling paradigm. [9,20] present an approach, which connects object life cycles with process models by determining commonalities between both representations and transforming one into the other. Covering one direction of the integration, [10] derives object life cycles from process models. Tackling the integration of control flow and data, [14,15] enable to model data constraints and to enforce them during process execution directly from the model. Similar to the mentioned approaches, we concentrate on integrated scenarios incorporating process models and object life cycles removing the assumption that both representations must completely correspond to each other. Instead, we set a synchronized object life cycle as reference that describes data manipulations allowed in a traditional, i.e., activity-driven, modeled process scenario, e.g., with BPMN [18].

The field of compliance checking focuses on control flow aspects using predefined rule sets containing, for instance, business policies. However, some works do consider data. [11] applies compliance checking to object-centric processes by creating process models following this paradigm from a set of rules. However, these rules most often specify control flow requirements. [7] provides a technique to check for conformance of object-centric processes containing multiple data classes by mapping to an interaction conformance problem, which can be solved by decomposition into smaller sub-problems, which in turn are solved by using classical conformance checking techniques. [23] introduces a framework that ensures consistent specialization of object-centric processes, i.e., it ensures consistency between two object life cycles. In contrast, we check for consistency between a traditional process model and an object life cycle. Eshuis [5] uses a symbolic model checker to verify conformance of UML activity diagrams [19] considering control and data flow perspectives while data states are not considered in his approach. [9] introduces compliance between a process model and an object life cycle as the combination of object life cycle conformance (all data state transitions induced in the process model must occur in the object life cycle) and coverage (opposite containment relation). [21] introduces conformance checking between process models and product life cycles, which in fact are object life cycles, because a product life cycle determines for a product the states and the allowed state transitions. Compared to the notion of weak conformance, both notions do not support data synchronization and both set restrictions with respect to data constraints specification in the process model.

6 Conclusion

In this paper, we presented an approach for the integrated verification of control flow correctness and weak data conformance using soundness checking considering dependencies between multiple data classes, e.g., an order is only allowed to be shipped after the payment was received but needs to be shipped with an confirmed invoice in one package. Therefore, we utilized the concept of synchronized object life cycles. For checking data correctness, we use the notion of weak conformance and extended it with means for object life cycle synchronization. Additionally, we utilized a mapping of a process model with data constraints to a Petri net and described a mapping of a synchronized object life cycle to a Petri net. Both resulting Petri nets are combined for an integrated control flow and data conformance check based on the soundness criterion. With respect to the places or transitions causing soundness violations, we can distinguish between control flow and data flow issues and therefore, we can verify the notion of weak conformance. Revealed violations can be highlighted in the process model and the synchronized object life cycle to support correction. In this paper, we focused on the violation identification such that correction is subject to future work.

References

1. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
2. van der Aalst, W.M.P.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
3. Cohn, D., Hull, R.: Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. IEEE Data Engineering Bulletin 32(3), 3–9 (2009)
4. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information & Software Technology 50(12), 1281–1294 (2008)
5. Eshuis, R.: Symbolic Model Checking of UML Activity Diagrams. ACM Transactions on Software Engineering and Methodology (TOSEM) 15(1), 1–38 (2006)
6. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous Soundness Checking of Industrial Business Process Models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009)
7. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking of Interacting Processes with Overlapping Instances. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 345–361. Springer, Heidelberg (2011)
8. Künzle, V., Weber, B., Reichert, M.: Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches. IJISMD 2(2), 19–46 (2011)
9. Küster, J.M., Ryndina, K., Gall, H.C.: Generation of Business Process Models for Object Life Cycle Compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
10. Liu, R., Wu, F.Y., Kumaran, S.: Transforming Activity-Centric Business Process Models into Information-Centric Models for SOA Solutions. J. Database Manag. 21(4), 14–34 (2010)
11. Lohmann, N.: Compliance by design for artifact-centric business processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 99–115. Springer, Heidelberg (2011)

12. Martens, A.: On Usability of Web Services. In: Web Information Systems Engineering Workshops, pp. 182–190. IEEE (2003)
13. Meyer, A., Polyvyanyy, A., Weske, M.: Weak Conformance of Process Models with respect to Data Objects. In: Services and their Composition (ZEUS), pp. 74–80 (2012)
14. Meyer, A., Pufahl, L., Batoulis, K., Kruse, S., Lindhauer, T., Stoff, T., Fahland, D., Weske, M.: Automating Data Exchange in Process Choreographies. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 316–331. Springer, Heidelberg (2014)
15. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and Enacting Complex Data Dependencies in Business Processes. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 171–186. Springer, Heidelberg (2013)
16. Meyer, A., Weske, M.: Weak Conformance between Process Models and Object Life Cycles. Tech. rep., Hasso Plattner Institute at the University of Potsdam (2014)
17. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
18. OMG: Business Process Model and Notation (BPMN), Version 2.0 (January 2011)
19. OMG: Unified Modeling Language (UML), Version 2.4.1 (August 2011)
20. Ryndina, K., Küster, J.M., Gall, H.C.: Consistency of Business Process Models and Object Life Cycles. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 80–90. Springer, Heidelberg (2007)
21. Wang, Z., ter Hofstede, A.H.M., Ouyang, C., Wynn, M., Wang, J., Zhu, X.: How to Guarantee Compliance between Workflows and Product Lifecycles? Tech. rep., BPM Center Report BPM-11-10 (2011)
22. Weske, M.: Business Process Management: Concepts, Languages, Architectures, 2nd edn. Springer (2012)
23. Yongchareon, S., Liu, C., Zhao, X.: A Framework for Behavior-Consistent Specialization of Artifact-Centric Business Processes. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 285–301. Springer, Heidelberg (2012)