Weak Conformance of Process Models with respect to Data Objects

Andreas Meyer, Artem Polyvyanyy, and Mathias Weske

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2–3, D-14482 Potsdam, Germany
{Andreas.Meyer,Artem.Polyvyanyy,Mathias.Weske}@hpi.uni-potsdam.de

Abstract. Process models specify behavioral aspects by describing ordering constraints between tasks which must be accomplished to achieve envisioned goals. Tasks usually exchange information by means of data objects, i.e., by writing information to and reading information from data objects. A data object can be characterized by its states and allowed state transitions. In this paper, we propose a notion which checks conformance of a process model with respect to data objects that its tasks access. This new notion can be used to tell whether in every execution of a process model each time a task needs to access a data object in a particular state, it is ensured that the data object is in the expected state or can reach the expected state and, hence, the process model can achieve its goals.

1 Introduction

Process modeling usually comprises two aspects: The control flow perspective and the data flow perspective [1]. Control flow defines possible execution sequences of tasks, whereas data flow provides means for exchanging information between the tasks. Information gets passed between tasks of a process model by writing to and reading from data objects. A data object can be formalized as a set of data states and transitions between the data states, i.e., as a labeled transition system, which is usually referred to as an object life cycle. An object life cycle can be used to identify the current data state of the data object and the set of its reachable data states from the current one [2]. Similarly, the execution semantics of process models is often defined by employing the notion of a process state that defines a set of tasks which can be performed. A process state changes once a task gets accomplished. A process state together with all data states (one for each data object) collectively define a state of a process instance. It is usually accepted that control flow drives execution of process models, i.e., a change in a state of a process instance is triggered by a change of a process state, which in turn may activate changes of data states.

In order to achieve safe execution of a process model, it must be ensured that every time a task 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., object life cycles of data objects must conform to the process model; otherwise, the execution of a process model may deadlock, i.e., terminate prior to

reaching the goal state. In this paper, we propose a notion of weak conformance which allows for a precise characterization of the above described intuition, where "weak" reflects the fact that data states are required to be reachable via arbitrary number of data state transitions and not necessarily via a single one.

In a process model which satisfies weak conformance with respect to its data objects, it is assumed that implicit data state transitions get realized by an external entity or by detailed implementations of process model tasks. Relevance for the new notion is based on the need to check for conformance of underspecified process models where, e.g., external events, not captured in the process model, change states of data objects. Events and tasks, being part of but not modeled in the process model, may also change the states of data objects. These modeling artifacts are, for instance, hidden in subprocess structures, so that process models and object life cycles are specified at different levels of detail. Practically, process models still conform to their used data objects if the hidden state changes do not contradict against data object life cycles.

The remainder of the paper proceeds as follows: The next section describes process scenarios – a formalism which integrates control flow and data flow aspects of process modeling. In Section 3, we define the notion of weak conformance of the process model from a process scenario with respect to data objects it operates with. Section 4 is devoted to related works. Finally, Section 5 draws conclusion.

2 Process Scenarios

In this section, we propose *process scenarios* – a formalism for designing concurrent systems which integrates control flow and data flow perspectives. A process scenario consists of two parts: (i) a process model which orchestrates the execution of tasks, and (ii) data objects which describe what information do tasks require to be executed and/or what information do tasks produce. We start the discussion with the definition of the first part – a process model.

Definition 1 (Process model).
A *process model* is a tuple $M = (A, G, D, R, C, F, type, \mathcal{A}, \mu)$, where A is a finite set of *tasks*, G is a finite set of *gateways*, D is a finite set of *data objects*, R is a finite set of *data states*, $C \subseteq (A \cup G) \times (A \cup G)$ is the *control flow* relation, $F \subseteq (A \times (D \times R)) \cup ((D \times R) \times A)$ is the *data flow* relation, $type : G \to \{xor, and\}$ assigns to each gateway a type, $\mathcal{A}$ is a finite set of *names* such that $\tau \in \mathcal{A}$ (A, G, D, R, and $\mathcal{A}$ are pairwise disjoint), and $\mu : A \to \mathcal{A}$ assigns to each task a name.

We use subscripts, e.g., A_M, G_M, and μ_M, to denote the relation of the sets and functions to process model M, and omit subscripts where the context is clear. We refer to the set $A \cup G$ as *nodes* of process model M. If $\mu(a) \neq \tau$, $a \in A$, then a is *observable* in M; otherwise a is *silent* in M. We expect that every process model M fulfills basic structural correctness requirements: (i) every task of M has at most one incoming and at most one outgoing control flow edge, (ii) every gateway has at least three incident control flow edges, (iii) M has exactly one *source* task and at least one *sink* task (the source has exactly one outgoing and no incoming control flow edges, while each sink has exactly one incoming and no

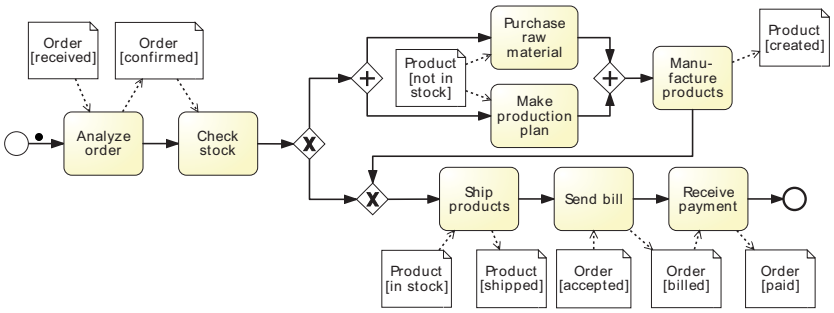
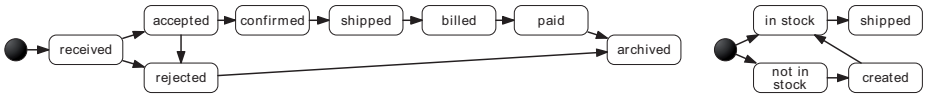


Fig. 1. A simple "order delivery and payment" process model

outgoing control flow edges), (iv) the source and all sinks of M are silent tasks, whereas all other tasks are observable, (v) every node of M is on a path from the source to some sink, and (vi) there exist no data flow edges which involve silent tasks, i.e., $\nexists(a,(d,r)) \in F : \mu(a) = \tau$ and $\nexists((d,r),a) \in F : \mu(a) = \tau$.

We adapt the notation similar to BPMN [3] for visualization of process models. An observable task is drawn as a rectangle that has rounded corners with its name inside. Source and sink tasks are visualized as start and end BPMN events, respectively. Gateways are drawn as diamonds. We call a gateway $g \in G_M$ of M an *xor* (an *and*) gateway, if $type_M(g) = xor$ ($type_M(g) = and$). An *xor* (an *and*) gateway uses a marker which is shaped like "×" ("+") inside the diamond shape. A data object (in a particular data state) is visualized as a BPMN data object. A data object $d \in D_M$ can appear multiple times in the visualization of the process model (also when in a particular data state $r \in R_M$). Control flow and data flow edges are drawn as solid and dashed directed edges, respectively.

The semantics of a process model is defined as a token game. A marking of a process model is represented by tokens on its control flow edges. Given process model M, a *marking* (or a *process state*) of M is a mapping $m : C_M \to \mathbb{N}_0$ ($\mathbb{N}_0$ is the set of natural numbers including zero). Fig. 1 shows a process model in its *initial* process state – a process state which puts one token on the only outgoing control flow edge of the source and no tokens elsewhere. Every node of a process model (except silent tasks) can be executed. The execution of an observable task removes one token from its only incoming and adds one token on its only outgoing control flow edge. The execution of an *and* gateway removes one token from each of its incoming control flow edges and then adds one token on each of its outgoing control flow edges. The execution of an *xor* gateway removes one token from one of its incoming control flow edges and afterwards adds one token on one of its outgoing control flow edges; the choice of the incoming edge as well as of the outgoing edge is done nondeterministically. Observe that we abstract from data-based decisions that are usually used to control the semantics of *xor* gateways. Let m and m' be two markings of M. We write $m \xrightarrow{x} m'$ to denote that m changes to m' by executing node x of M. If $\sigma = a_1 a_2 \dots a_n$, $n \in \mathbb{N}_0$, is a sequence of nodes of M, $m \xrightarrow{\sigma} m'$ denotes the fact that there exists a sequence of process states $m_1 m_2 \dots m_{n-1}$ such that $m \xrightarrow{a_1} m_1 \xrightarrow{a_2} \dots m_{n-1} \xrightarrow{a_n} m'$. We call



(a) (b)

Fig. 2. Object life cycles of (a) "Order" and (b) "Product" data objects

σ an *execution sequence* of M which starts with m. Let a and a' be two nodes of M. With $a \Rightarrow_M a'$ we denote the predicate which evaluates to `true` if $a = a'$ or there exists an execution sequence of M which starts with the initial marking and executes a before a'; otherwise $a \Rightarrow_M a'$ evaluates to `false`.

Next, we proceed with the definition of an object life cycle.

Definition 2 (Object life cycle).
An *object life cycle* is a tuple $L = (S, \Sigma, \mapsto, i)$, where S is a finite set of *data states*, Σ is a finite set of *actions* (S and Σ are disjoint), $\mapsto \subseteq S \times \Sigma \times S$ is the *data state transition* relation, and $i \in S$ is the *initial* data state.

We use subscripts S_L, Σ_L, $\mapsto_L$, and i_L, to denote the relation of the elements to the object life cycle L. Note that we omit subscripts where the context is clear. For $s, s' \in S$ and $a \in \Sigma$ we denote by $s \overset{a}{\longmapsto} s'$ the fact that $(s, a, s') \in \mapsto$. If $\sigma = a_1 a_2 \dots a_n$, $n \in \mathbb{N}_0$, is a sequence of actions, $s \overset{\sigma}{\longmapsto} s'$ denotes the fact that there exists a sequence of data states $s_1 s_2 \dots s_{n-1}$ such that $s \overset{a_1}{\longmapsto} s_1 \overset{a_2}{\longmapsto} \dots s_{n-1} \overset{a_n}{\longmapsto} s'$. We call σ an *execution sequence* of L which starts with s, and s' *is a reachable data state from data state* s *via* σ. With $s \Rightarrow_L s'$ we denote the predicate which evaluates to `true` if $s = s'$ or there exists an execution sequence of L which starts with i_L and reaches s before s'; otherwise $s \Rightarrow_L s'$ evaluates to `false`.

Finally, a process scenario is defined as follows.

Definition 3 (Process scenario).
A *process scenario* is a tuple $H = (M, \mathcal{L}, \omega)$, where M is a process model, $\mathcal{L}$ is a finite set of object life cycles, and $\omega : D_M \to \mathcal{L}$ assigns to each data object of M an object life cycle.

Note that we assume that for a process scenario $H = (M, \mathcal{L}, \omega)$ it holds that ω is injective and $\bigcup_{L \in \omega(D_M)} S_L \subseteq R_M$. Fig. 1 and Fig. 2 visualize a process scenario. The process model of the scenario is given in Fig. 1. It contains two data objects: "*Order*" and "*Product*". The life cycles of these data objects are shown in Fig. 2(a) and Fig. 2(b), respectively.

3 Weak Conformance

Prior to proceeding with the definition of weak conformance, we define several notions for convenience considerations. Let $f \in F_M$ be a data flow edge of process model M. With f_A, f_D, and f_R we denote the task, data object, and data state component of f, respectively. For instance, if f is equal to $(a, (d, r))$ or to $((d, r), a)$, then (in both cases) $f_A = a$, $f_D = d$, and $f_R = r$. We call f an *input* data flow edge if $f \in ((D \times R) \times A)$, and an *output* data flow edge if $f \in (A \times (D \times R))$.

Definition 4 (Weak data object conformance).
Given process scenario $H = (M, \mathcal{L}, \omega)$, $M = (A, G, D, R, C, F, type, \mathcal{A}, \mu)$, M *satisfies weak conformance with respect to data object* $d \in D$ if for all $f, f' \in F$ such that $f_D = d = f'_D$ holds $f_A \Rightarrow_M f'_A$ implies $f_R \Rightarrow_{\omega(d)} f'_R$, and $f_A = f'_A$ implies f is an input edge and f' is an output edge.

Given a process scenario, we say that the process model satisfies weak conformance, if it satisfies weak conformance with respect to each of its data objects. Weak data object conformance is satisfied if for each two succeeding data states of a data object there exists an execution sequence from the first to the second data state in the corresponding object life cycle. Two data states are succeeding in the process model if either (i) they are accessed by the same task with one being part of an input and one being part of an output data flow edge, or (ii) there exists an execution sequence in the process model in which two different tasks access the same data object in two data states.

The process model in Fig. 1 satisfies weak conformance with respect to data object "*Product*" and does not satisfy weak conformance with respect to data object "*Order*". Indeed, there exists an execution sequence which visits task "*Analyze order*" before task "*Send bill*", which access data object "*Order*" in data states "*confirmed*" and "*accepted*", respectively. However, data state "*accepted*" is not reachable in the object life cycle in Fig. 2(a) via data state "*confirmed*". One can fix this flaw, for instance, by changing the data state of the only input data flow of "*Send bill*" task from "*accepted*" to "*confirmed*", which also modifies the process model to satisfy weak conformance.

Comparing the proposed notion of weak conformance to the one introduced in [4], the given process model would not satisfy conformance with respect to data object "*Product*". In [4], the authors rely on process models with fully specified data information. For instance, task "*Ship products*" reads data object "*Product*" in data state "*in stock*". However, in [4], it is required that there exists a preceding task which writes "*Product*" in that state. As such task does not exist, conformance is not satisfied.

4 Related Work

Process models which follow on the imperative design paradigm have been studied extensively [1]. The increasing interest in the development of process models for execution has shifted the focus from control flow to data flow perspective. The first step in this regard are artifact-centric processes introduced in [5]. Artifact-centric processes connect data objects with the control flow of process models by specifying object life cycles which represent data dependencies and based thereon, the order of task execution. In [6,4], the authors present an approach which connects object life cycles with process models by determining commonalities between both representations and transforming one into the other. In [7], a rule-based approach is described; it allows to connect control flow with data flow and, thus, to automatically create data-driven executable process models. In terms of data-driven execution, case handling [8] plays a major role, as in case

handling data dependencies solely determine the order of task execution. In this paper, we also concentrate on the integrated scenarios which incorporate process models and object life cycles. However, we remove the assumption that all the approaches mentioned above follow, i.e., both representations must completely correspond to each other. Instead, we set object life cycles of data objects as references that describe what can be utilized by process models.

Compliance, or correctness, in process models mostly refers to checks of the process model with respect to a defined rule set containing, for instance, business policies. The field of compliance is well researched [9,10,11,12] and has already been tackled for artifact centric processes, e.g., [13]. A different type of compliance is introduced in [4]. There, compliance between a process model and an object life cycle of one data object used in the process model is defined 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). In this paper, we proposed the definition of a similar type of compliance. As we set object life cycles to be the reference, we assume them to be correct and, therefore, we can restrict the compliance check to conformance only. For conformance, instead of working with direct data state transitions we rely on data state reachability.

5 Conclusion

In this paper, we proposed a notion to check for weak conformance between a process model and object life cycles of its utilized data objects. A process model satisfies weak conformance if every time it is allowed to access states of a data object in a specific order (in the process model), these data states can also be reached in an object life cycle of the data object in the very same order.

In future works, we plan to propose an algorithm to perform analysis checks based on the notion of weak conformance introduced in this paper. For process models which do not satisfy weak conformance, one can suggest, whenever applicable, changes to the process model so that the resulting model conforms to its data objects. Process model modifications may also be applicable to already conforming process models in order to simplify their structure while preserving the conformance property. Furthermore, in process scenarios with "large" object life cycles, a conforming process model can determine the relevant aspects so that the object life cycles get tailored towards the specific needs of process scenarios and, in this way, become better understandable.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)
2. Booch, G.: Object-Oriented Analysis and Design with Applications (3rd Edition). Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004)
3. OMG: Business Process Model and Notation (BPMN), Version 2.0 (January 2011) `http://www.omg.org/spec/BPMN/2.0/` accessed March 1, 2012.

4. Küster, J., Ryndina, K., Gall: Generation of Business Process Models for Object Life Cycle Compliance. In: Business Process Management, Springer (2007) 165–181
5. Nigam, A., Caswell, N.: Business artifacts: An approach to operational specification. IBM Systems Journal **42**(3) (2003) 428–445
6. Ryndina, K., Küster, J., Gall: Consistency of Business Process Models and Object Life Cycles. In: MoDELS Workshops, Springer (2006) 80–90
7. Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In: Advanced Information Systems Engineering, Springer (2008) 48–63
8. van der Aalst, W., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data and Knowledge Engineering **53**(2) (2005) 129–162
9. Awad, A.: A Compliance Management Framework for Business Process Models. PhD thesis, Hasso Plattner Institute (2011)
10. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permissions. In: BPM Workshops, Springer (2006) 5–14
11. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: EDOC, IEEE (2006) 221–232
12. Agrawal, R., Johnson, C., Kiernan, J., Leymann, F.: Taming Compliance with Sarbanes-Oxley Internal Controls Using Database Technology. In: International Conference on Data Engineering, IEEE (2006) 92–101
13. Lohmann, N.: Compliance by design for artifact-centric business processes. In: Business Process Management, Springer (2011) 99–115