Diagnosing and Repairing Data Anomalies in Process Models

Ahmed Awad[1], Gero Decker[1], and Niels Lohmann[2]

[1] Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
{ahmed.awad,gero.decker}@hpi.uni-potsdam.de
[2] Institut für Informatik, Universität Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. When using process models for automation, correctness of the models is a key requirement. While many approaches concentrate on control flow verification only, correct data flow modeling is of similar importance. This paper introduces an approach for detecting and repairing modeling errors that only occur in the interplay between control flow and data flow. The approach is based on place/transition nets and detects anomalies in BPMN models. In addition to the diagnosis of the modeling errors, a subset of errors can also be repaired automatically.

1 Introduction

Process models reflect the business activities and their relationships in an organization. They can be used for analyzing cost, resource utilization or process performance. They can also be used for automation. Especially in the latter case, not only the control flow between activities must be specified but also branching conditions, data flow and preconditions and effects of activities.

The Business Process Modeling Notation (BPMN [1]) is the de-facto standard for process modeling. It provides support for modeling control flow, data flow and resource allocation. For facilitating the handover of BPMN models to developers or enabling the transformation of BPMN to executable languages such as the Business Process Execution Language (BPEL [2]), data flow modeling is an essential aspect. This is mainly done through so called *data objects* that are written or read by activities. On top of that, it can be specified that a data object must be in a certain state before an activity can start or that a data object will be in another state after it was written by an activity. This allows to study the interplay between control flow and data flow in a process.

When using process models for automation, correctness of the process models is of key importance. Many different notions of correctness have been reported in the literature, mostly concentrating on control flow only [3–7]. These techniques reveal deadlocks, livelocks and other types of unwanted behavior of process models. Although data flow modeling is as important when it comes to automation, only few corresponding verification techniques can be observed [8, 9].

In this context, the contribution of our paper is three-fold. (i) We provide a formalization of basic data object processing in BPMN based on place/transition nets [10], (ii) we define a correctness notion for the interplay between control flow and data flow that can be computed efficiently, and (iii) we provide a technique for automatically repairing some of the modeling errors. While we use BPMN as main target language of our approach, it could easily be applied to other languages as well.

The paper is structured as follows. The next section will provide an example that will be used throughout this paper. Section 3 introduces the mapping of BPMN to place/transition nets. Different types of data anomalies are discussed in Sect. 4. Section 6 reports on related work, before Sect. 7 concludes the paper and points to future work.

2 Motivating Example

Fig. 1 depicts a business process that handles insurance claims. The circles represent the start and end of the process, the rounded rectangles are activities and the diamond shapes are gateways for defining the branching structure. Data objects are represented by rectangles with dog-ears.

First, the new claim (the data object under study) is registered. It is checked whether the claim is fraudulent or not. In case it is fraudulent, an investigation is initiated to reveal this fraud. Otherwise, the claim is evaluated to be either accepted or rejected. Accepted claims are paid to the claimer. In all cases claims are closed at the end.

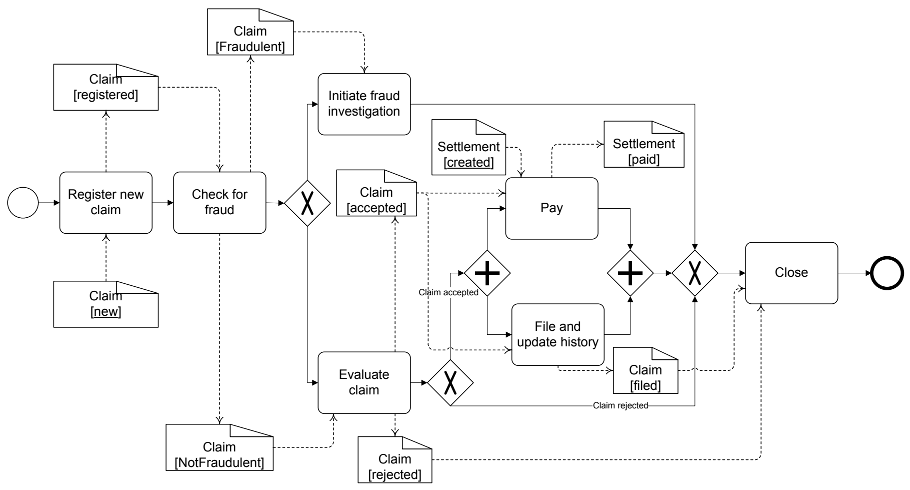


Fig. 1. A process model regarding data object handling adapted from [11]

From the control flow perspective, the process model is correct according to several correctness notions from the literature. This can be proved by translating the BPMN model to a place / transition net as described in [12]. The result of this transformation is shown in Fig. 2.

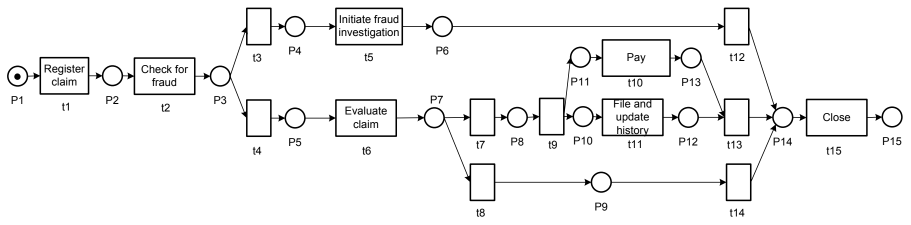


Fig. 2. Petri net representation of the example (considering the control flow only)

The Petri net is *sound* [13], implying that the process will terminate properly in all possible execution scenarios without leaving running activities behind. Therefore, we can conclude that the control flow of the given BPMN model is fine.

On the other hand, the BPMN model also contains information about data flow. As we did with the control flow aspects, we want to make sure that also the data flow is correct.

The use of BPMN data objects with defined states can be interpreted as preconditions and effects of activities. For instance, a registered claim exists after activity "register new claim" has completed. Activity "pay" can in turn only be executed if an accepted claim is present. Also branching conditions can be specified, as it is the case for distinguishing accepted and rejected claims for deciding to take the upper or lower branch.

The example contains a number of anomalies that only appear in the interplay between control flow and data flow. For instance, "pay" and "file and update history" are specified to run concurrently, while executing "file and update history" before starting "pay" would lead to a problem: The claim needs to be in state accepted but it was already set to filed. Further discussion about data anomalies is deferred to Sect. 4.

3 Mapping of Data Objects to Petri Nets

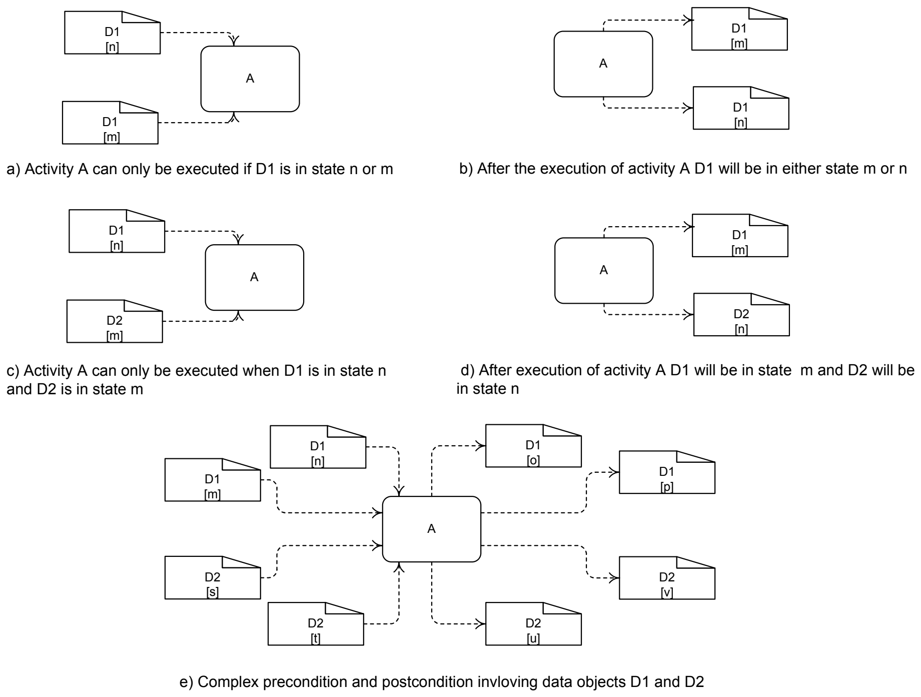
The BPMN specification considers data objects as a way to visualize how data is processed. The semantics of data objects remain unspecified and even left to the interpretation of the modeler (see [1] page 93). Therefore, we introduce a particular semantics of BPMN data objects in this paper as a necessary precondition for formal verification.

First of all, we assume a single copy of the data object that is handled within the process. This single copy is assumed to exist from the moment of process instantiation. Multiple data object shapes with the same label are considered to refer to the same data object. For instance, all shapes labeled "claim" refer to the same claim object (cf. Fig. 1). Exactly one claim object is assumed to exist for each instance of that process.

On the other hand, each data object is in a certain state at any time during the execution of the process. This state changes through the execution of activities. BPMN offers the possibility to specify allowed states of a data object. Even more so, it can be specified which state a data object must be in before an activity can start (precondition) and which state a data object will be in after having completed an activity (effect). This is represented via associations in BPMN. A directed association from a data object to an activity symbolizes a precondition and an association leaving an activity towards a data object symbolizes an effect.

While often it is required that a data object is in exactly one state before being able to execute an activity, it might also be allowed that the data object is in either one of a set of states. This is symbolized through multiple associations targeting an activity and that each originate in a data object shape referring to the same data object but in different states (cf. Fig. 3 a)). An analogous representation applies for alternative effects (cf. Fig. 3 b)).

If multiple data objects are required as input (as in Fig. 3 c)), then it is interpreted as a precondition that data object D1 is in state n *and* data object D2 is in state m. The same principle is applied in case of outputting multiple data objects Fig. 3 d). Fig. 3 e) describes the union of the other situations when an activity can have complex pre/postconditions on data objects. In that specific case, activity A requires that data object D1 to be in *either* state n or m *and* data object D2 to be in *either* state s or t. After activity A completes it changes the state of data object D1 to *either* o or p *and* changes the state of data object D2 to *either* state u or v.



a) Activity A can only be executed if D1 is in state n or m

b) After the execution of activity A D1 will be in either state m or n

c) Activity A can only be executed when D1 is in state n and D2 is in state m

d) After execution of activity A D1 will be in state m and D2 will be in state n

e) Complex precondition and postcondition invloving data objects D1 and D2

Fig. 3. Assumptions about BPMN data flow semantics

The mapping introduced in [12] covers BPMN control flow. We extend this mapping with data flow in the following way: First, we provide a separate data flow mapping for each data object. Each of these mappings represents preconditions and effects of tasks regarding the corresponding data object. In a second step, the control flow Petri net obtained through [12] is merged with all data flow nets.

Fig. 4 illustrates the data flow mapping. Each data object is mapped to a set of places. Each place represents one of the states the data object can be in. Activities with preconditions or effects are modeled as transitions. Depending on the kind of preconditions and effects, an activity can be represented by one or a set of transitions in the data flow model. Arcs connect these places with transitions, again depending on the preconditions and effects.

The simplest case is represented as case a). Here, an activity A reads a data object in a certain state and changes it to another state. This is represented as consuming a token from place *[Data object, state]* and producing a token on *[Data object, other_state]*. In case b), executing activity A does not have any effect on the data objects. However, it requires the data object to be in a certain state. This is modeled using a bi-flow in the data flow Petri net: The transition consumes and produces a token from the same place.

All other patterns require multiple transitions per activity. Case c) displays that the data object is changed to a certain state (*other_state*) when executing activity A. Multiple transitions are used as the data object can be in a number of previous states. For each previous state a transition models the change to the new state. In this case, it does not make any difference if the data object is used as input (without constraint on the state) or not at all.

Case d) shows how it is represented that the data object must be in either state n or state m, before activity A can start. While the state is actually not changed, it must still be ensured that activity cannot execute when the data object is in another state, say x. If an activity takes a data object as input but does not impose any constraint on its state, we normally would need to represent this by enumerating all states. However, as we know that the data object is guaranteed to be in one of the states, this would not realize any restrictions. Therefore, we simply do not reflect this case at all in the mapping.

Case e) shows that a number of different outcomes of activity A is also modeled using multiple transitions. Here, for each combination of input state and valid output state, a transition must be introduced.

Case f) illustrates how data object states can also be used in branching conditions. This case is actually quite similar to case d).

Fig. 5 shows the resulting data flow Petri net for the claim in our example. In addition to the control flow model we have already seen and this first data flow model, we need to generate a third Petri net covering the data flow regarding the settlement.

In a next step, the control flow and data flow Petri nets are composed. Composition of two Petri nets is done through transition fusion, i.e. for each pair of transitions from the two models that originate from the same activity in the BPMN model a transition is created in the resulting net. Those transitions in

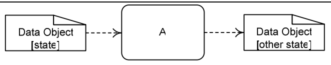
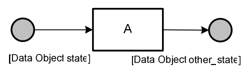
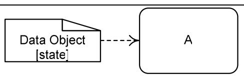
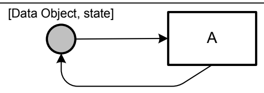
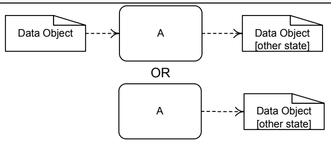
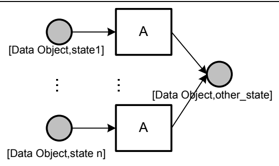
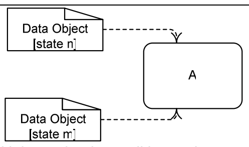
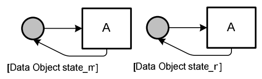
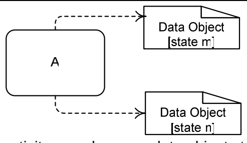
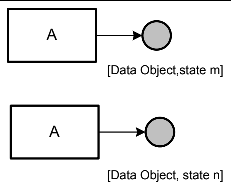
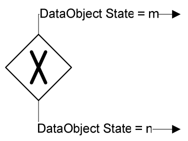
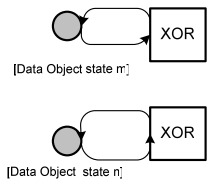
BPMN	Description	Petri Net
 a) Read-write with a condition on input state	Activity A has a precondition on the state of the data object at input. After completion, activity A changes the state of the data object.	
 b) Read-only with a condition on input state	Activity A conditionally reads data object i.e. data object has to be has to be in this specific state	
 Data Object A Data Object [other state] OR A Data Object [other state] c) Read-write without a condition on input state	Activity A has no specific condition on the state of data object as input. After completion, A will change the state of the data object to other state	 A [Data Object,state1] [Data Object,other_state] A [Data Object,state n]
 d) Multiple read-only condition on input state	Activity A accepts data object as input in any of the specified states. The data object can assume only one of these states when A is about of execute.	
 e) An activity can change a data object state to one of many states	Activity A can produce the data object after completion in only one of the specified states.	
 f) Explicit XOR arc conditions	Explicit XOR arc conditions based on data object states must be reflected in the resulting Petri net.	

Fig. 4. Mapping options for data objects association with activities

any of the two models without partners in the other model are simply copied, as are all places. The arcs are set accordingly and the initial marking is the composition of the two initial markings. The result of the composition of two models is then again composed with the third model. The result for our three models is displayed in Fig. 6.

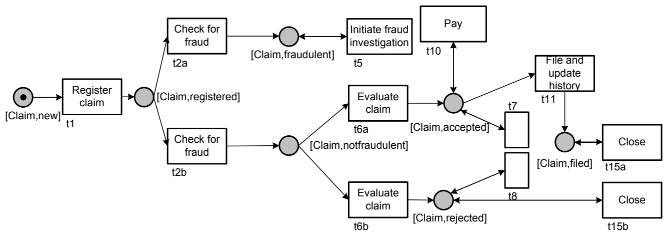


Fig. 5. Claim data object flow representation

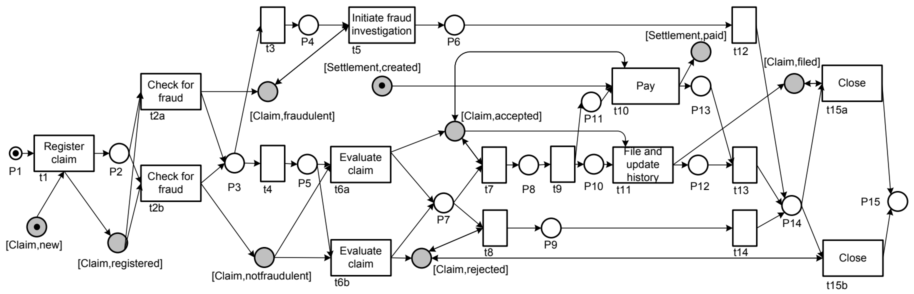


Fig. 6. Petri net representation of claim handling process

4 Classification of Anomalies and Resolution Strategies

The anomalies we are dealing with center around preconditions that keep a process from executing. While from a control flow perspective the execution of the activity would be possible, the precondition leads to a deadlock situation. Already in the example we have seen different kinds of anomalies that need to be dealt with in different ways.

1. *Too Restrictive Preconditions.* This is the general case of data anomalies. It occurs when an activity, from the control flow only perspective, is ready to execute but the on the data side data object(s) are not in the expected state.
2. *Implicit Routing.* Alternative branches can be taken in the process where each branch requires the data objects to be in certain states. However, while the branching condition does not impose these restrictions, they only become apparent as the activities have corresponding preconditions. In our example, this could occur when a claim is fraudulent and still the lower branch leading to “evaluate claim” could be chosen.
3. *Implicit Constraints on the Execution Order.* The example has shown that although the control flow allowed concurrency between the pay and file and update activities, the data flow demanded that pay cannot start after file and update history has already completed. Therefore, this data flow constraint could be interpreted as constraint on the execution order that would normally need to be reflected in the control flow.

These anomalies correspond to possible deadlocks in the composed Petri net where no reachable marking has a token in a place corresponding to a required object state. The problem of *Implicit Constraints on the Execution Order* occurs when it comes to firing transitions "Pay" and "File and update history". Both transitions are enabled but a deadlock occurs if the transition "Fire and update history" fires first. If the firing sequence is that "Pay" fires first no deadlock occurs. The *Too Restrictive Preconditions* anomaly takes place when a claim is declared to be fraudulent, at this time the transition "Close" is waiting for a token in place *[Claim, filed]* or place *[Claim, rejected]* so this transition deadlocks and that instance is not able to terminate.

While we devote Sect. 5 to provide diagnosis and resolution in a formal way, in the rest of this section we informally list the different anomalies discussed above and propose a set of resolution strategies for them if there are any.

Too Restrictive Preconditions This problem occurs when an activity in the BPMN model has a precondition on a certain data object state but this state is not reachable at the time the activity is ready to execute. Solutions to this situation could be:

– Remove the precondition: this is the naive solution in which the dependency on this specific object state is removed.
– Loosening the precondition: by looking at what are the available states of the data object at the time the activity is ready to execute and add them to the preconditions as alternative acceptable states. The "Close" activity in Fig. 1 is expecting the "Claim" data object to be in either state *rejected* or *filed* while it is possible that when it is activated; the "Claim" is in state *fraudulent.*

Implicit Routing This could be seen as a special case of the *too restrictive preconditions* anomaly where the data precondition is missed for some activity due to improper selection of the path to go. For instance, in Fig. 1 activity "Check for fraud" could produce the "Claim" is state *not fraudulent* but still the control flow could be routed by the XOR-Split to activity "Initiate fraud investigation". This happened because the branches of the XOR-Split did not have explicit conditions. The solution for this problem is to *avoid* this unwanted routing by discovering the branching conditions of the XOR-Split and updating the model with them.

Implicit Constraints on the Execution Order Whenever there are two activities running in parallel and they have the same precondition(s), the anomaly occurs when at least one of these activities updates the state of the data object. If there is only one activity that changes the state of the object while the rest only read it, the best solution to the problem could be to force the updating activity to execute last after all read only activities have executed. Unfortunately, making local changes by forcing sequentialization between these parallel activities might lead to further anomalies. On the other hand, if more than one activity change the state of the object then forcing sequentialization will not solve the problem. A less than optimal solution is to apply resolution strategies for the *too restrictive preconditions* anomaly by loosening the precondition for the deadlocking activity.

5 Resolution of Data Anomalies

5.1 Notations and Basic Definitions

To formally reason about the behavior of a BPMN model with data objects, we use classical Petri nets [10] with their standard definitions: A *Petri net* is a tuple $[P, T, F, m_0]$ where P and T are two finite disjoint sets of *places* and *transitions*, respectively, $F \subseteq \big((P \times T) \cup (T \times P)\big)$ is a *flow relation*, and $m_0 : P \to \mathbb{N}$ is an *initial marking*. For a node $x \in P \cup T$, define ${}^\bullet x = \{y \mid [y, x] \in F\}$ and $x^\bullet = \{y \mid [x, y] \in F\}$. A transition t is *enabled* by a marking m (denoted $m \xrightarrow{t}$) if $m(p) > 0$ for all $p \in {}^\bullet t$. An enabled transition can *fire* in m (denoted $m \xrightarrow{t} m'$), resulting in a successor marking m' with $m'(p) = m(p) + 1$ for $p \in t^\bullet \setminus {}^\bullet t$, $m'(p) = m(p) + 1$ for $p \in {}^\bullet t \setminus t^\bullet$, and $m'(p) = m(p)$ otherwise.

Let $N_c = [P_c, T_c, F_c, m_{0_c}, \mathit{final}_c]$ be the Petri net translation of the control flow aspects of the BPMN process under consideration (using the translation from [12]). Similarly, let $N = [P, T, F, m_0, \mathit{final}]$ be the Petri net translation including control flow and data flow (using the patterns from Fig. 4). We assume that the set of places P is a disjoint union of *control flow places* P_c and *data places* $P_d \subseteq \mathcal{D} \times \mathcal{S}$; that is, a data place is a pair of a data object and a data state thereof. Furthermore, there is a surjective labeling function $\ell : T \to T_c$ that maps each transition of N to a transition of N_c. For two transitions $t_1, t_2 \in T$ we have $\ell(t_1) = \ell(t_2)$ iff t_1 and t_2 model the same activity, but t_1 is connected to different data places as t_2, i.e. ${}^\bullet t_1 \cap P_c = {}^\bullet t_2 \cap P_c$ and ${t_1}^\bullet \cap P_c = {t_2}^\bullet \cap P_c$.

We further extend the standard definition of Petri nets by defining a finite set *final* of *final markings* to distinguish desired final states from unwanted blocking states. We use the term *deadlock* for a marking $m \notin \mathit{final}$ which does not enable any transition.

Example. For the net N_c in Fig. 2, the marking $[\mathsf{p_{15}}]$ is the only final marking. When also considering data places for net N in Fig. 6, any marking m with (i) $m(\mathsf{p_{15}}) = 1$, (ii) $m(p) = 0$ for all other control places $p \in P_c \setminus \{\mathsf{p_{15}}\}$, and (iii) for each data object $d \in \mathcal{D}$, marks exactly one place $[d, s]$. For instance, the marking $[\mathsf{p_{15}}, [\mathsf{Settlement}, \mathsf{paid}], [\mathsf{Claim}, \mathsf{filed}]]$ is a final marking of N.

As a starting point of the analysis, we assume that the control flow model N_c is *weakly terminating*. That is, from each marking m reachable from the initial marking m_{0_c}, a final marking $m_f \in \mathit{final}_c$ is reachable. A control flow model that is not weakly terminating can deadlock or livelock, which is surely undesired. Such flaws need to be corrected even before considering data aspects and are therefore out of scope of this paper.

5.2 Resolving Too Restrictive Preconditions

If the model of both the control flow and the data flow deadlocks, such a deadlock m of N marks control flow places that enable transitions in the pure control flow model N_c. These transitions can be used to determine which data flow tokens are missing to enable a transition in m.

Definition 1 (Control-flow enabledness). *Let m be a deadlock of N and let m_c be a marking of N_c with $m_c(p) = m(p)$ for all $p \in P_c$. Define the set of* control-flow-enabled transitions *of m as $T_{cfe}(m) = \{t' \in T \mid m_c \xrightarrow{t}_{N_c} \wedge \ell(t') = t\}$.*

As N_c weakly terminates, $T_{cfe}(m) \neq \emptyset$ for all deadlocks of N. We now can examine the preset of control-flow enable transitions to determine which data flow tokens are missing.

Definition 2 (Missing data states). *Let m be a deadlock of N and $t \in T_{cfe}(m)$ a control-flow-enabled transition. Define the* missing data states of t in m *as $P_{mdi}(t, m) = ({}^\bullet t \cap P_d) \setminus \{p \mid m(p) > 0\}$.*

For a control-flow-enabled transition t in m, $P_{mdi}(t, m)$ consists of those data places that additionally need to be marked to enable t. This information can now be used to fix the model as sketched in Sect. 4: (1) to change the model such that these tokens are present; (2) to add an additional transition t' with $\ell(t) = \ell(t')$ that can fire in the current data state.

Example. Consider the deadlock $[\mathsf{p_{14}}, [\mathsf{Claim}, \mathsf{fraudulent}], [\mathsf{Settlement}, \mathsf{created}]]$ of net N in Fig. 6. The transitions $\mathsf{t_{15a}}$ and $\mathsf{t_{15b}}$ are control flow enabled. The missing data inputs are $[\mathsf{Claim}, \mathsf{filed}]$ and $[\mathsf{Claim}, \mathsf{rejected}]$, respectively. The deadlock can be resolved by either relaxing a transition's input condition (e.g. by removing the edge $[[\mathsf{Claim}, \mathsf{filed}], \mathsf{t_{15a}}]$ or to add a new "Close" transition t_{15c} to the net with ${}^\bullet t_{15c} = \{\mathsf{p_{14}}, [\mathsf{Claim}, \mathsf{fraudulent}]\}$ and $t_{15c}{}^\bullet = \{\mathsf{p_{15}}, [\mathsf{Claim}, \mathsf{fraudulent}]\}$ (see Fig. 7.

5.3 Resolving Implicit Routing

In this section, we introduce the notion of data equivalence which can be used to classify occurring deadlocks. The classification then can be used to propose resolution strategies for these deadlocks.

Definition 3 (Data invariance, data equivalence). *A transition $t \in T$ is* data invariant*, iff $({}^\bullet t \cup t^\bullet) \cap P_d = \emptyset$. Let m_1, m_2 be markings of N. m_1 and m_2 are* data equivalent*, denoted $m_1 \sim_d m_2$, iff (i) $m_1(p) = m_2(p)$ for all $p \in P_d$ and (ii) there is a data invariant transition sequence σ such that $m_1 \xrightarrow{\sigma} m_2$ or $m_2 \xrightarrow{\sigma} m_1$. For a marking m of N, define the* data equivalence class *of m as $[\![m]\!] = \{m' \mid m \sim_d m'\}$.*

Obviously, $\sim_d$ is an equivalence relation that partitions the set of reachable markings of N into data equivalence classes. These equivalence classes can be used to classify deadlocks of N.

Definition 4 (Unsynchronized deadlock). *Let m be a deadlock of N. m is an* unsynchronized deadlock *if there exists a marking $m' \in [\![m]\!]$ such that $m' \xrightarrow{*} m''$ with $[\![m'']\!] \neq [\![m]\!]$ or $m'' \in final$.*

Let $m^* \in [\![m]\!]$ be a marking such that the transition sequence $m^* \xrightarrow{\sigma} m$ is minimal and $m^* \xrightarrow{*} m''$. m^* enables at least two mutually exclusive transitions t_1 and t_2. Let t_1 be the first transition of σ. To avoid the deadlock m, this transition must not fire in the data state of $[\![m]\!]$. Hence, the BPMN model has to be changed such that the XOR-split modeled by t_1 is refined by an appropriate branching condition.

Example. Consider the deadlock $[\mathsf{p_5}, [\mathsf{Claim}, \mathsf{fraudulent}], [\mathsf{Settlement}, \mathsf{created}]]$ of the net of Fig. 6. There exists a data-equivalent marking $[\mathsf{p_4}, [\mathsf{Claim}, \mathsf{fraudulent}],$ $[\mathsf{Settlement}, \mathsf{created}]]$ which activates transition "Initiate fraud investigation". To avoid the deadlock, transition t_4 must not fire in this data state $[\mathsf{Claim}, \mathsf{fraudulent}]$. This can be achieved by adding explicit XOR split branching conditions.

Branching conditions are realized on the Petri net level using bi-flows, as already illustrated in Fig. 4. Refining branching conditions means in this context that certain combinations of data object states should be excluded. We need to distinguish two cases in this context. Either there is currently no restriction on the data objects in question or there is one. The latter case is easier as we can simply delete the corresponding transition from the Petri net. If no transition is left for the XOR-split we know that we cannot find any branching condition that would guarantee proper behavior. In the first case, we need to enumerate all combinations of data object states except for the current combination (see Fig. 7.

5.4 Resolving Implicit Constraints on the Execution Order

The case of "implicit constraints on the execution order" occurs when a deadlock is reached from a point where two or more transitions can run in parallel and they have common input data places.

Definition 5 (Implicit Constraints on the Execution Order). *Let m be a deadlock of N. m is called a data-flow/control-flow conflict iff $\exists\, m'$ where $m' \xrightarrow{t_1} m \wedge m' \xrightarrow{t_2} m'' \wedge ({}^\bullet t_1 \cap P_c) \cap ({}^\bullet t_2 \cap P_c) = \emptyset \wedge ({}^\bullet t_1 \cap P_d) \cap ({}^\bullet t_2 \cap P_d) \neq \emptyset \wedge ({}^\bullet t_1 \cap P_d) \cap (t_2{}^\bullet \cap P_d) = \emptyset$*

Example. The deadlock $[\mathsf{p_{11}}, \mathsf{p_{12}}, [\mathsf{Claim}, \mathsf{filed}], [\mathsf{Settlement}, \mathsf{created}]]$ of net N in Fig. 6 occurred because there was a previous marking $[\mathsf{p_{10}}, \mathsf{p_{11}}, [\mathsf{Claim}, \mathsf{accepted}],$ $[\mathsf{Settlement}, \mathsf{created}]]$ where t_{10} and t_{11} were enabled. Both t_{10} and t_{11} are running in parallel and share a common data place $[\mathsf{Claim}, \mathsf{accepted}]$.

Resolving implicit constraints on the execution order requires the modification of the control flow logic of the process. On a reachability graph level, state transitions from "good" states to "bad" states need to be removed. "Good" states would be those states from where a valid final state is still reachable and "bad" states from where no valid final state is reachable. While the problematic state transitions are identified in the combined model for control flow and data flow, corresponding state transitions would then be removed from the control flow model.

While this is feasible on the state transition / reachability graph level, projecting a corresponding change back to Petri nets (and finally back to BPMN) is very challenging. This is due to the fact that one transition in the Petri net corresponds to potentially many state transitions in the reachability graph. Therefore, removing individual state transitions from the reachability graph typically results in heavy modifications of the Petri net structure rather than just removing individual nodes. There exist techniques for generating Petri nets for a given automata [14]. However, this clearly goes beyond the scope of the paper. Projecting the modification of the Petri net back to a modification of the initial BPMN model is then the next challenge that would need to be tackled.

Therefore, our approach simply suggests to the modeler that the execution order must be altered in order to avoid a certain firing sequence of transitions. It is then up to the modeler to perform modifications that actually lead to a resolved model.

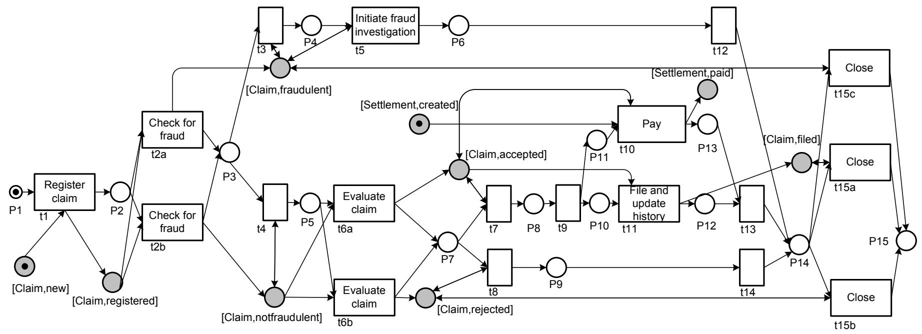


Fig. 7. Modified net with data anomalies corrected

6 Related Work

To the best of our knowledge, there is no research conducted to formalize the data flow aspects in BPMN. On the other hand, data flow formalization in process modeling has been recently addressed from different points of view.

The first discussion of the importance of modeling data in process models is in [8]. They identify a set of data anomalies as redundant data, lost data, missing data, mismatched data, inconsistent data, misdirected data, and insufficient data. However, the paper just signaled these types of anomalies without an approach to detect them. A refinement of the aforementioned set of data anomalies was given in [9]. In addition, authors provide a formal approach that extends UML ADs to model data flow aspects and detect anomalies. An earlier work to formalize data flow semantics in UML ADs was discussed in [15, 16] where Colored Petri Nets CPNs were used as the formal background.

Van Hee et al. [17] present a case study how consistency between models of different aspects of a system can be achieved. They model object life cycles derived from CRUD matrices as workflow nets and later synchronized with the control flow. Their approach is, however, does not present strategies how inconsistencies between data flow and control flow can be removed.

Verification of data flow aspects in BPEL was discussed in [18]. Authors use a modern compiler technology to discover the data dependency between interacting services as a step to enhance control-flow only verification of BPEL processes. Later on, the extracted data dependency enriches the underlying Petri net representing the control flow with extra places and transitions. The approach maps only those messages that affect the value of a data item used in decision points. Model checking data aware message exchange with web services was discussed in [19] where authors have extended Computational Tree Logic CTL with first order quantification in order to verify that sequence of messages exchanged will satisfy certain properties based on data contents of these messages.

Recent data-centric business process modeling approach let data dependency drive control flow among activities in a process [20]. A business artifact is an abstraction of a document that can be consumed and produced by tasks of the process. Nine patterns of processing such business artifacts have been defined. However, the notion of a state of an artifact is defined by means of repositories. The approach to a large extent is similar to ours, however, in their verification, deadlock scenarios and their resolution as we discussed were not addressed. Work in [11, 21] has addressed the consistency between business process models and lifecycle of business objects processed in these models. A more recent approach for making data dependencies first class citizens is discussed in [22]. [23] comes to formalize the aspects of data-centric business process modeling and develops an algorithm to derive data-centric models from activity-centric ones.

In [24], object life cycles are studied in the context of inter-organizational business processes implemented as services. The synchronization between control flow and data flow not only confines the control flow of a service, but also its interaction with other services. It is likely that the results of this paper can be adapted to inter-organizational business processes.

Dual Workflow Nets [25] Dual Workflow Nets DWF-net is a new approach that enables the explicit modeling of data flow along with the control flow. As DWF-net introduces new types of nodes and differentiates between data and control tokens, specific algorithms for verification of such types of nets is developed. In comparing to our approach, we can model the same situations DWF-net is designed to model using only P/T nets.

7 Conclusion

An approach to detect and resolve data anomalies in process models was introduced. This approach is based on an extension of formalization of BPMN using Petri nets. Three categories of data anomalies, based on the formalization we introduced, were identified. Moreover, resolution strategies to some of these anomalies where discussed.

The proposed resolution of deadlocks coming up from the different data anomalies by loosening preconditions/discovering routing conditions by introducing/removing a set of transitions and flow relations do not prohibit a backward mapping from the modified control and data flow Petri net to a modified BPMN model.

The contribution in this paper can be seen as a step forward in verification of process models. Moreover, automated remedy of anomalies is possible. In case that the anomaly is not automatically resolvable, the modeler is aware of the part of the model where the problem is so that corrective actions can take place.

A prototypical implementation of the proposed approach is currently an ongoing activity within the context of our open source web based modeling tool Oryx [26]

References

1. OMG: Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG) (2006) http://www.bpmn.org/.
2. Fallside, D.C., Walmsley, P.: Web Services Business Process Execution Language Version 2.0. Technical report (2005) http://www.oasis-open.org/apps/org/workgroup/wsbpel/.
3. Aalst, W.v.d.: Verification of Workflow Nets. In: Application and Theory of Petri Nets 1997, volume 1248 of LNCS, Berlin, Springer Verlag (1997) 407–426
4. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: Business Process Management, volume 4102 of LNCS, Berlin, Springer Verlag (2006) 145–160
5. Sadiq, W., Orlowska, M.E.: Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In: CAiSE '99: Proceedings of the 11th International Conference on Advanced Information Systems Engineering, London, UK, Springer-Verlag (1999) 195–209
6. Dongen, B.F.v., Mendling, J., Aalst, W.v.d.: Structural Patterns for Soundness of Business Process Models. In: EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06), Washington, DC, USA, IEEE Computer Society (2006) 116–128
7. Awad, A., Puhlmann, F.: Structural Detection of Deadlocks in Business Process Models. In: BIS. Volume 7 of LNBIP., Springer (2008) 239–250
8. Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C.: Data Flow and Validation in Workflow Modeling. In: ADC '04: Proceedings of the 15th Australasian database conference, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2004) 207–214
9. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the Data-Flow Perspective for Business Process Management. Info. Sys. Research **17** (2006) 374–391
10. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science edn. Springer (1985)
11. Ryndina, K., Küster, J.M., Gall, H.C.: Consistency of Business Process Models and Object Life Cycles. In: Models in Software Engineering. Number 4364 in LNCS, Springer (2007) 80–90

12. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50** (2008) 1281–1294
13. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems (Cooperative Information Systems). The MIT Press (2002)
14. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A., England, N.R.: Petrify: a Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers. IEICE Transactions on Information and Systems **80** (1997) 315–325
15. Stoerrle, H.: Semantics and Verification of Data Flow in UML 2.0 Activities. In: Workshop on Visual Languages and Formal Methods (VLFM 2004). (2004) 35–52
16. Stroelle, H.: Semantics and Verification of Data Flow in UML 2.0 Activities. Electronic Notes in Theoretical Computer Science **127** (2005) 35–52
17. Hee, K.M.v., Sidorova, N., Somers, L.J., Voorhoeve, M.: Consistency in Model Integration. Data Knowl. Eng. **56** (2006) 4–22
18. Moser, S., Martens, A., Gorlach, K., Amme, W., Godlinski, A.: Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-based Data Flow Analysis. In: IEEE SCC, IEEE Computer Society (2007) 98–105
19. Hallé, S., Villemaire, R., Cherkaoui, O., Ghandour, B.: Model Checking Data-Aware Workflow Properties with CTL-FO+. In: EDOC, IEEE Computer Society (2007) 267–278
20. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling Business Contexture and Behavior Using Business Artifacts. In: CAiSE. Volume 4495 of LNCS., Springer (2007) 324–339
21. Küster, J.M., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: BPM. Volume 4714 of LNCS., Springer (2007) 165–181
22. Sun, S.X., Zhao, J.L.: Developing a Workflow Design Framework Based on Dataflow Analysis. In: HICSS '08: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, Washington, DC, USA, IEEE Computer Society (2008) 19
23. Kumaran, S., Liu, R., Wu, F.Y.: On the Duality of Information-Centric and Activity-Centric Models of Business Processes. In: CAiSE. Volume 5074 of LNCS., Springer (2008) 32–47
24. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral Constraints for Services. In: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24–28, 2007, Proceedings. (2007)
25. Fan, S., Dou, W.C., Chen, J.: Dual Workflow Nets: Mixed Control/Data-Flow Representation for Workflow Modeling and Verification. In: APWeb/WAIM Workshops. Volume 4537 of LNCS., Springer (2007) 433–444
26. Decker, G., Overdick, H., Weske, M.: Oryx - An Open Modeling Platform for the BPM Community. In: BPM. Volume 5240 of LNCS., Springer (2008) 382–385