Automatic Generation of a Data-Centered View of Business Processes[⋆]

Cristina Cabanillas[1], Manuel Resinas[1], Antonio Ruiz-Cortés[1], and Ahmed Awad[2]

[1] Universidad de Sevilla, Spain
{cristinacabanillas,resinas,aruiz}@us.es
[2] Hasso Plattner Institute at the University of Potsdam
ahmed.awad@hpi.uni-potsdam.de

Abstract. Most commonly used business process (BP) notations, such as BPMN, focus on defining the control flow of the activities of a BP, i.e., they are activity-centered. In these notations, data play a secondary role, just as inputs or outputs of the activities. However, there is an increasing interest in analysing the life cycle of the data objects that are handled in a BP because it helps understand how data is modified during the execution of the process, detect data anomalies such as checking whether an activity requires a data object in a state that is unreachable, and check data compliance rules such as checking whether only a certain role can change the state of a data object. To carry out such an analysis, it is very appealing to provide a mechanism to transform from the usual activity-centered model of a BP to the set of life cycles of all the data objects involved in the process (i.e., a data-centered model). Unfortunately, although some proposals describe such transformation, they do not deal with data anomalies in the original BP model nor include information about the activities of the BP that are executed in the state transitions of the data object, which limits the analysis capabilities of the life cycle models. In this paper, we describe a model-driven procedure to automatically transform from an activity-centered model to a data-centered model of a BP that solves the aforementioned limitations of other proposals.

Keywords: business process, data management, object life cycle, data anomalies, Petri net, reachability graph.

1 Introduction

It is widely known that business processes (BPs) involve different kinds of elements, to be named control flow, time, data and resources. However, most

⋆ This work has been partially supported by the European Commission (FEDER), Spanish Government under the CICYT project SETI (TIN2009-07366); and projects THEOS (TIC-5906) and ISABEL (P07-TIC-2533) funded by the Andalusian Local Government.

commonly used BP models and notations focus on the control flow and the timing of activities in the BP. As a consequence, in most BP models, data (e.g., documents, reports, invoices, emails and the like) play a secondary role, just as inputs or outputs of the activities of the process.

Nevertheless, understanding and analysing how data is modified during the execution of a BP is getting an increased interest from both industry and academy. For instance, BPMN, the de-facto standard for BP modelling, has incorporated more advanced constructs for data management in its last version [1]. In addition, there is an increasing number of research proposals to analyse the way data is used in a BP to detect anomalies [2,3,4] and to define data-aware compliance rules [5] for BPs. Therefore, providing a mechanism to transform from the usual activity-centered view of a BP to a data-centered view that focuses on the data handled during the process is very appealing to this goal of understanding and analysing how data is modified during the execution of a BP.

In this paper we describe a model-driven procedure based on Petri nets for carrying out this transformation automatically. In particular, the input of the procedure is a BP diagram expressed in BPMN 2.0 (cf. Figure 1). We use this notation because it is the de-facto standard for BP modelling. Such diagrams represent data objects connected to the BP activities that use them either to read them or write them, or for both things. A data object has a *type* and can have one or more *states* along the execution of a process. For instance, in the BP of opening a bank account, the data object *application* filled by the new customer could go through states *sent*, *accepted* and *stored*. The output of the procedure is a data-centered view composed of the set of object life cycles (OLCs) of all the data objects that are involved in a BP. They represent the allowed transitions between the states of the data object according to the BP diagram. In addition, these transitions also include information about the activities of the BP that are executed in the transition between states of the data object (cf. Figure 2). Furthermore our procedure also deals with some data anomalies that may appear in a BP model (cf. Section 4 for more details).

Our approach has the following advantages: (i) it is fully automated; (ii) it is based on Petri nets, which allows us to use efficient and well-tested Petri net algorithms; (iii) since it includes information about the activities that are executed in each transition, it provides the same full information required to understand BP execution as activity-centered process diagrams; and (iv) it is robust in the sense that it provides an accurate data-centered view despite having a BP with data anomalies as input. Moreover, it informs the user about these data anomalies.

The remaining of the paper is organised as follows. Section 2 introduces a use case used to exemplify the output produced by the procedure. Section 3 contains the description of the whole procedure for OLC generation. In Section 4 the detection and handling of data anomalies is introduced. Section 5 contains a summary of related work and in Section 6 we draw a set of conclusions and outline some future work.

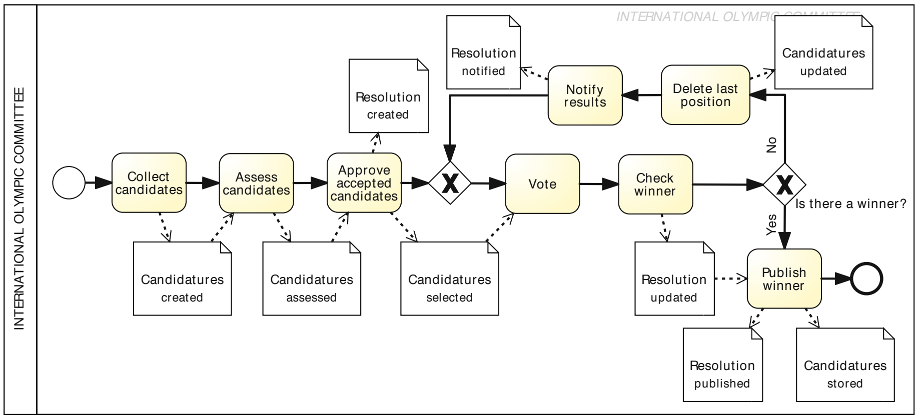


Fig. 1. Business process for assigning the venue for the Olympic Games

2 Use Case

To illustrate our approach we use the BP for assigning the venue for the Olympic Games (Figure 1) as use case in this paper[1]. The International Olympic Committee is in charge of this process. This committee first receives the applications of the cities that want to organize the Olympic Games. Each city is evaluated in order to keep only those which fulfill all the requirements. After this filter is applied, an approval of the final candidates is necessary. Once the list of candidates is ready, a secret voting is carried out. If there is consensus and only one city is selected, then the winner venue is published. Otherwise, the least voted city is eliminated from the list of candidates and a new voting is performed. This is repeated until there are only two cities left. Then, the city with a greatest number of votes wins.

There are two data objects in this BP model. Data object *Candidates* represents a document that contains a list of the cities that applied for the venue. The information of each candidate in the document includes the name of the city, its description, what it offers for each requirement needed, and the mark given by the committee to discern between accepted and rejected candidates. This document may be updated during the voting repetitive process. Data object *Resolution* represents the result of the voting and, thus, is a document with the same list of candidates and the number of votes each of them received. Again, this data object will be updated if more than one voting is performed. If there is no winner yet, the resolution is notified. Otherwise, the resolution is completed with the features of the final venue and published.

The output of the procedure presented in this paper is a set of finite-state machines (FSM) representing the life cycles of the data objects modelled in a

[1] Note that this process is used for illustration purposes only, so there may be differences with the actual process of the Olympic Games venue selection process.

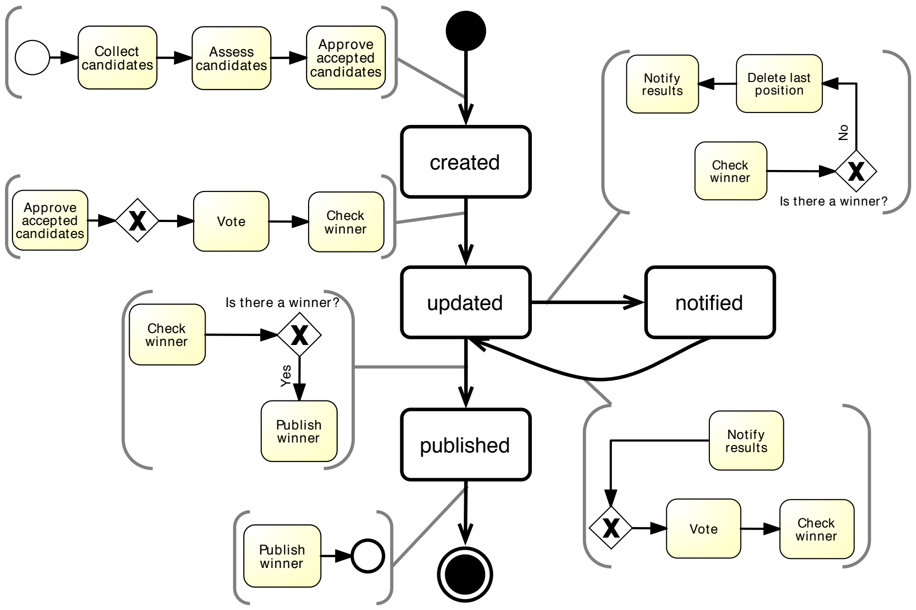


Fig. 2. Object life cycle of data object *Resolution* of the business process in Fig. 1

BP. Figure 2 depicts the life cycle of data object *Resolution* of our use case. The life cycles of a data object have one *start state* (represented with a filled circle), one *final state* (represented with a semi-filled circle), and one or more *intermediate states* (represented with a rectangle) that correspond with states of the data object in the BP model. Transitions (represented with directed arrows) connect two states and contain the parts of the BP that are executed in the transition between states of the data object.

3 BP2OLC Procedure

BP2OLC is our approach to automatically generate the OLCs of the data objects represented in a BPMN model[2]. As depicted in Figure 3, it is a three-step procedure based on model transformations which involves four different models. The procedure must be carried out for each *data object type* present in the BP model. We assume the source BP model has the following features:

1. As far as control flow is concerned, the BP model is sound, which basically means it has no control flow deadlocks and terminates properly [6].
2. There is only one copy of each data object in each instance of the process, e.g., there is only one data object *Resolution* in one instance of the process.

[2] All the terms referring to elements of a BP model are used in the same sense as in the BPMN 2.0 specification [1].

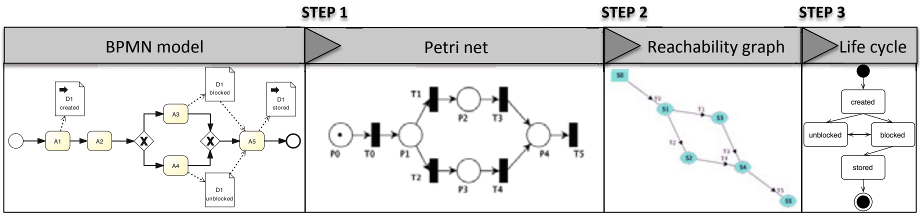


Fig. 3. Overview of the BP2OLC procedure

Besides, data objects are created within the BP instance that uses them (i.e. data objects created outside of the process are not considered).

3. Each data object has always a state. In case an appearance of a data object in the BP model is not associated with any state, this appearance will be ignored.
4. The BP model can contain data objects connected to any kind of activity (sub-processes are treated like task activities). Only XOR gateways can be used.

Assumption 1 is made because control-flow soundness is out of the scope of this paper. Assumptions 2 and 3 are reasonable and have also been made elsewhere [2]. The last assumption is related to the reach of the current approach.

3.1 Step 1. From BPMN Model to Petri Net

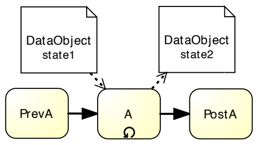
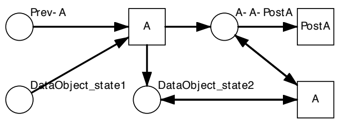
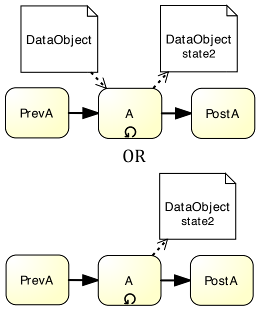
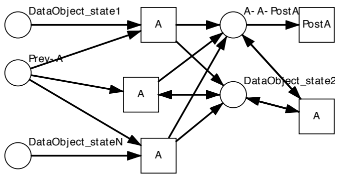
We believe that providing a semantic mapping [7] between a BPMN model and a target domain such as Petri nets, whose semantics has been formally defined, is a good approach because it allows one to use the techniques specific to the target semantic domain for analysing the source models. We chose Petri nets for two reasons: (i) plenty of processing algorithms on Petri nets have already been developed and can be useful for our purpose [6,8]; and (ii) the transformation of the control flow of a BP model into an equivalent Petri net has already been described in [6].

Definition 1. *A* Petri net *is a 3-tuple* $PN = (T_{PN}, P, F)$*, where:*

- $T_{PN} = \{t_1, t_2, ..., t_n\}$ *is the set of transitions of the Petri net, represented graphically as rectangles.*
- $P = \{p_1, p_2, ..., p_n\}$ *is the set of places of the Petri net, represented graphically as circles.*
- $F \subseteq (P \times T_{PN}) \bigcup (T_{PN} \times P)$ *is the set of arcs of the Petri net (flow relation), represented as arrows.*

A *marking (state)* or *markup* assigns a nonnegative integer to each place of a Petri net. If it assigns to place p a nonnegative integer k, we say that p is marked with k tokens. Pictorially, we place k black dots (tokens) in place p. A markup

Table 1. Mapping for data objects association with loop activities

BPMN	Description	Petri Net
 DataObject state1 DataObject state2 PrevA A PostA a) Read-write loop activity with a condition on input state	Loop activity A has both a precondition and a postcondition on the state of the data object. The precondition will be considered only for the first execution of A. Then the object is assumed to be in state2.	 Prev- A A A- A- PostA PostA DataObject_state1 DataObject_state2 A
 DataObject DataObject state2 PrevA A PostA OR DataObject state2 PrevA A PostA b) Read-write loop activity without a condition on input state	Loop activity A has no specific condition on the state of data object as input. We assume that after first completion of A the data object will always have state2.	 DataObject_state1 A A- A- PostA PostA Prev-A A DataObject_state2 A DataObject_stateN A

is denoted by M, an m-vector, where m is the total number of places. The pth component of M, denoted by $M(p)$, is the number of tokens in place p. The firing of an enabled transition will change the token distribution (marking) in a net [8].

We use the set of rules introduced by Awad et al. [2] to do the semantic mapping between elements of a BP model with data objects and elements of a Petri net. Let E_{BP} be the set of flow nodes of a BP (model), i.e. activities, gateways and events, D_{BP} the set of states of a data object of that BP, and $WRITERS_{BP} \subseteq E_{BP}$ be the set of activities of the BP that write that data object. The result of the semantic mapping is a Petri net with the following characteristics:

– The places of the Petri net are of two different kinds: control places P_C and data places P_D. Therefore $P = P_C \bigcup P_D$ and $P_C \bigcap P_D = \emptyset$.
  - $P_C = \{pc_1, pc_2, ..., pc_n\}$ corresponds to those places that represent sequence flow elements (arrows) of the business process. Each $pc_i = (ei_i, eo_i)$, where $ei_i, eo_i \in E_{BP}$ is a pair of values composed of the two flow nodes of the business process that the sequence flow element connects.
  - $P_D = \{pd_1, pd_2, ..., pd_n\} = D_{BP}$ corresponds to those places that represent states of the data object whose object life cycle we are generating. There is exactly one data place for each possible state of the data object.

– The transitions of the Petri net represent flow nodes of the business process model. It follows an $n : 1$ relationship, i.e., each transition represents only one flow node of the business process and a flow node may appear several times in a Petri net. Function $elem : T_{PN} \rightarrow E_{BP}$ represents such relation.

An example of the transformation rules is depicted in Table 1, which illustrates an extension of the catalogue of transformations proposed in [2] to deal with loop activities. As stated in [1], a loop activity executes the inner activity as long as a loop condition evaluates to true. An attribute can be set to specify a maximal number of iterations. An example of loop activity is an activity *Update order* that updates an order in a restaurant (by customer's command) until an event or a received message indicates no more updates are allowed. For more details about the other transformations we refer the reader to [2].

Finally, note that there is a small difference between this mapping and the one presented in [2] because in this paper we consider no data objects are supposed to exist before the execution of a BP in our BP2OLC procedure, whereas [2] considers data objects have an initial state when instantiating a BP. This difference causes the transformation in [2] referring to the writing of the data object has to be slightly changed for the first writing of the object in our BP2OLC procedure, in order to comply with our assumption 2. It means the first time the data object is written, the responsible transition of the Petri net does not have any input data places.

3.2 Step 2. Reachability Graph from Petri Net

Definition 2. *A* reachability graph related to a Petri net *is a 3-tuple* $RG_{PN} = (N, M, T_{RG})$*, where:*

- $N = \{n_1, n_2, ..., n_n\}$ *is the set of nodes of the reachability graph.* $\forall n_i \in N, \bullet n_i$ *and* $n_i \bullet$ *represent immediately previous and next nodes of* n_i*, respectively.*
- $M : P \times N \rightarrow \mathbb{N}$ *represents the* markup *of the net.*
- $T_{RG} \subseteq (N \times N)$ *are the transitions of the reachability graph.*

The reachability graph is obtained by analysing the Petri net by means of well-known algorithms. Each node of the reachability graph represents a reachable marking state of the net and each arc a possible change of state, i.e. the firing of a transition. However, due to the characteristics of our semantic mapping between BPMN and Petri net, in the reachability graph resulting from such Petri nets it holds that $M(p, n) \in [0, 1], \forall n \in N, \forall p \in P$. In addition, the information about the markup of the net contained in every node always corresponds with both a sequence flow of the BP model and a state of the data object, as illustrated in Figure 4. It means there is always one token in a control place of the Petri net and one in a data place, except in the beginning (until an activity writes the data object for the first time) and in the final nodes of the reachability graph (in which, on the contrary, all the tokens in control places have been consumed).

Given the previous definitions, the following functions can be defined:

- Function *map* : $T_{RG} \rightarrow T_{PN}$ is defined to map the transitions of a reachability graph into the transitions of a Petri net.
- Function *state* : $N \rightarrow P_D$ returns the state of the data object of the business process model contained in the current node of the reachability graph. *state*$(n) = \{p_d \in P_D : M(p_d, n) = 1\}$.

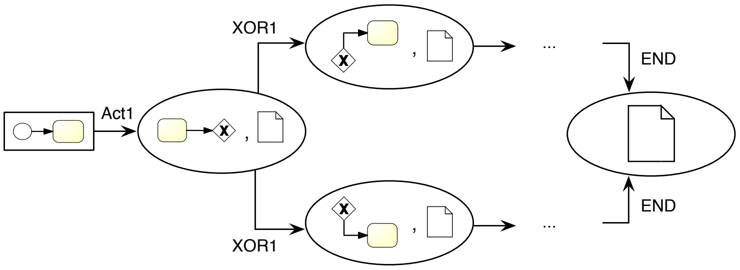


Fig. 4. Content of the arcs and nodes of a reachability graph

- Function $flow : \mathcal{P}(N) \rightarrow \mathcal{P}(P_C)$ returns the set of sequence flow elements of the business process model contained in a set of nodes of the reachability graph. $flow(N') = \{p_c \in P_c : \exists n \in N'(M(p_c, n) = 1))\}$.
- Function $activity : N \rightarrow E_{BP}$ returns the flow node of the business process model contained in the input arc of the current node of the reachability graph. $activity(n) = \{e_i \in E_{BP} : p_c = (e_i, e_o) \wedge M(p_c, n) = 1\}$.

The node of the reachability graph with no input arrows is called $firstNode \in N : \nexists \bullet firstNode$ and it is the start node of a reachability graph. The nodes of the reachability graph with no output arrows, whose input is called END and with no tokens in a control place are *normal* final nodes of the reachability graph. We will describe *abnormal* final nodes in Section 3.3.

3.3 Step 3. Object Life Cycle from Reachability Graph

Definition 3. *An* object life cycle of a data object of a business process *is a 2-tuple* $OLC = (S_{OLC}, T_{OLC})$*, where:*

- $S_{OLC} = \{s_1, s_2, ..., s_n\}$ *is the set of states in which the data object can be.* $\forall s_i \in S_{OLC}, \bullet s_i$ *and* $s_i \bullet$ *represent immediately previous and next states of state* s_i*, respectively. Let* $start \in S$ *and* $end \in S$ *be the start and the final states of the OLC, respectively. Then,* $S_{OLC} \setminus (start \bigcup end) = P_D = D_{BP}$
- $T_{OLC} \subseteq S_{OLC} \times S_{OLC} \times \mathcal{P}(N)$ *is the set of transitions that appear in the object life cycle. Each transition contains a set of nodes of the reachability graph from which it has been generated. Function replace* : $T_{OLC} \times N \times \mathcal{P}(N) \rightarrow T_{OLC}$ *replaces the set of nodes before node* N *in the path of a transition for a specific set of nodes.*

We have defined Algorithms 1 and 2 to obtain an OLC from a reachability graph. Algorithm 1 receives the reachability graph resulting from the previous step and the list of activities of the BP that write the data object. Its output is the OLC together with a set of data anomalies found while creating it.

Algorithm 1. Algorithm to initialize an object life cycle, call Algorithm 2 from a reachability graph and post-process nodes already processed in Algorithm 2 (RG2OLC)

```
1: **IN:** $RG_{DPN} = (N, M, T_{RG})$; $WRITERS_{BP}$
2: **OUT:** $S_{OLC}$; $T_{OLC}$; $WARN \subseteq N$
3: $S_{OLC} \leftarrow \{START\_STATE\}$; $T_{OLC} \leftarrow \emptyset$
4: $INPUT \leftarrow (WRITERS, firstNode, START\_STATE, \emptyset, \emptyset, \emptyset, \emptyset, S_{OLC}, T_{OLC})$
5: $(S_{OLC}, T_{OLC}, PNODES, PP, WARN) \leftarrow RG2OLC(INPUT)$
6: $found \leftarrow 1$ // Post-processing of nodes in PP
7: while $found \neq 0$ do
8:    $found \leftarrow 0$
9:    for all $(node, assocPath) \in PP$ do
10:      for all $(s_i, s_o, path) \in T_{OLC}$ do
11:         if $node \in path$ then
12:            $found \leftarrow found + 1$; $newT \leftarrow (s_i, s_o, path)$
13:            $T_{OLC} \leftarrow T_{OLC} \bigcup replace(newT, node, assocPath)$
14:         end if
15:      end for
16:   end for
17: end while
18: return $(S_{OLC}, T_{OLC}, WARN)$
```

Its behaviour consists of calling Algorithm 2 with the appropriate parameters and post-processing the resulting reachability graph. Algorithm 2 is a recursive algorithm that builds an OLC by processing a reachability graph node by node from its start node. Its input set and steps are described below.

Input of Algorithm 2.
- $WRIT \subseteq E$ is the set of activities that write the data object.
- $cNode \in N$ is the node being processed.
- $cState \in D$ is the current state of the data object.
- $PNODES \subseteq N$ is the set of already processed nodes.
- $PATH \subseteq N$ contains a set of nodes of the reachability graph, which is the information required in the transitions of the object life cycle.
- $PP = \{pair_1, pair_2, ..., pair_n\}$, where $pair_i = (node, assocPath), node_i \in N, assocPath_i \subseteq N$ is a set of pairs containing a node of the reachability graph and a set of nodes associated to that node, which conceptually corresponds to the path contained in variable PATH when processing that node.
- $WARN \subseteq N$ is a set of nodes related to deadlocks in the Petri net.
- $S'_{OLC} \subseteq S_{OLC}$ is the set of states of the resulting object life cycle.
- $T'_{OLC} \subseteq T_{OLC}$ is the set of transitions of the resulting object life cycle.

Check for and add new transitions (lines 3-7). A new transition of one of the types shown in Figures 5a and 5b must be added to the OLC in case that a new state of the data object is found in the reachability graph. If, on the contrary, the node shows that the data object is still in the current state but

Algorithm 2. Algorithm to generate the life cycle of a data object from a reachability graph (RG2OLC)

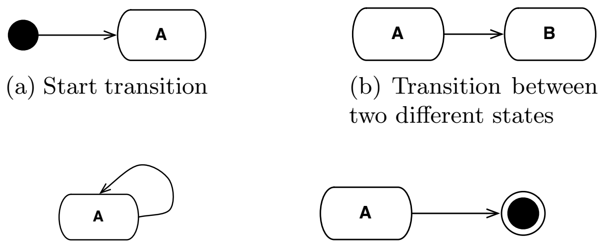
```
1: IN: $WRIT, cNode, cState, PNODES, PATH, PP, WARN, S'_{OLC}, T'_{OLC}$
2: OUT: $S'_{OLC}, T'_{OLC}, PNODES, PP, WARN$
3: if $state(cNode) \neq \emptyset \wedge (cState \neq state(cNode) \vee activity(cNode) \in WRIT)$ then
4:    $S'_{OLC} \leftarrow S'_{OLC} \bigcup state(cNode)$
5:    $T'_{OLC} \leftarrow T'_{OLC} \bigcup (cState, state(cNode), PATH)$
6:    $cState \leftarrow state(cNode)$; $PATH \leftarrow cNode$
7: end if
8: $PATH \leftarrow PATH \bigcup cNode$
9: if $cNode \notin PNODES$ then
10:    $PNODES \leftarrow PNODES \bigcup cNode$
11:    if $cNode\bullet = \emptyset$ then
12:       if $activity(cNode) = END$ then
13:          $S'_{OLC} \leftarrow S'_{OLC} \bigcup FINAL\_STATE$
14:          $T'_{OLC} \leftarrow T'_{OLC} \bigcup (cState, FINAL\_STATE, PATH)$
15:       else
16:          $WARN \leftarrow WARN \bigcup cNode$ // Deadlock detected
17:       end if
18:    else
19:       for all $next \in cNode\bullet$ do
20:          $IN \leftarrow (WRIT, next, cState, PNODES, PATH, PP, WARN, S'_{OLC}, T'_{OLC})$

21:          $(S''_{OLC}, T''_{OLC}, PNODES', PP', WARN') \leftarrow RG2OLC(IN)$
22:          $S'_{OLC} \leftarrow S'_{OLC} \bigcup S''_{OLC}$; $T'_{OLC} \leftarrow T'_{OLC} \bigcup T''_{OLC}$; $PP \leftarrow PP \bigcup PP'$
23:          $PNODES \leftarrow PNODES \bigcup PNODES'$; $WARN \leftarrow WARN \bigcup WARN'$
24:       end for
25:    end if
26: else
27:    if $activity(cNode) \notin WRIT$ then
28:       $PP \leftarrow PP \bigcup (cNode, PATH)$ // Save for post-processing
29:    end if
30: end if
31: return $(S'_{OLC}, T'_{OLC}, PNODES, PP, WARN)$
```

we find that the activity of the node is one of those that write the data object in the BP model, a self-transition will be added (Figure 5c). For instance: (i) in loops in a BP a data object may be written by an activity consecutively twice, giving rise to a self-transition (e.g. data object *Candidates* of Figure 1 has a self-transition in state *updated* due to a loop); (ii) loop activities also cause self-transitions, as can be inferred from Table 1.

Update variable $PATH$ (line 8). New nodes will be added to the path in order to collect the information contained in the transitions of the life cycle[3].

[3] Note that operator *union* ($\bigcup$) neither inserts duplicates nor null or empty values.



(a) Start transition (b) Transition between two different states (c) Self-transition (d) Final transition

Fig. 5. Kinds of transitions of an object life cycle

Revise the last activity executed and act consistently (lines 9-31). Algorithm 2 is returned either when a *normal* final node of the reachability graph is reached, when an *abnormal* final node[4] is found, or when the current node is in the list of processed nodes. In the last case, if, furthermore, the activity represented in the node writes the data object, the node has been properly processed in lines 3-7 and the rest of the reachability graph does not have to be re-processed. Otherwise, the already processed node is saved in a list of nodes that must be properly post-processed later. This way we avoid processing nodes of the reachability graph more than once and we ensure that Algorithm 2 always terminates. If none of the previous situations appears, we must go on processing the reachability graph and update the variables whose values must be propagated.

Post-process the necessary nodes (lines 6-17 of Algorithm 1). Some scenarios represented in a BP model can give rise to the appearance of transitions between the same two states, which differ from each other in their contents. This situation is detected in the reachability graph when reaching a node that has already been processed and its corresponding activity of the BP does not modify the data object being examined. In Algorithm 2 only one of the transitions is added to the OLC. To add the new transition with the right content, we must find the transition to which the node refers, add a duplicate transition to the OLC and set its content to the proper value.

4 Detecting and Showing Data Anomalies

As aforementioned in this paper, we assume soundness (also called correctness) in the control flow of BPs, but the process can be *unsound* regarding data perspective. The data-related deadlocks that appear in a reachability graph (i.e. *abnormal* final nodes) indicate data-related anomalous situations (known as *data*

[4] *Abnormal* final nodes are those with no output transitions and with no input transitions called END. They indicate there is a deadlock in the Petri net that stops the execution. We collect them in a list of warnings that will have to be addressed later.

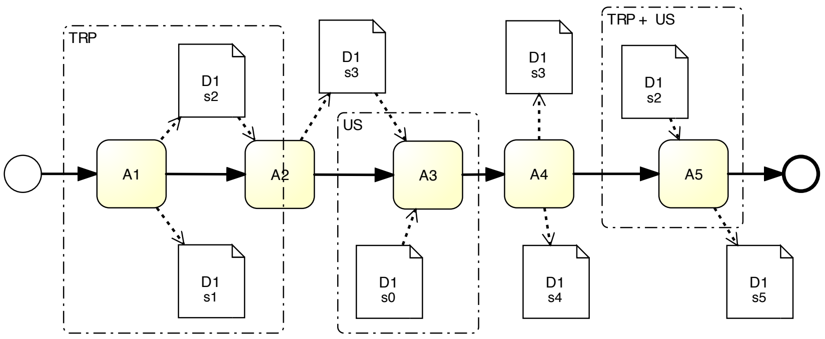


Fig. 6. Business process model with data anomalies

anomalies) in the represented BP model. All these data anomalies cause deadlocks in the Petri net, so the OLC generated is not complete. For example, in Figure 7 only the states and transitions outlined with black solid lines are generated when processing the actual reachability graph that represents the BP model in Figure 6[5]. States such as *s0* and *s5* are not detected without managing some data anomalies present in the BP model previously. There are two different groups of anomalous situations, which can be mapped into data-related problems defined in [2,3]. The resolution of the data anomalies in the BP is out of the scope of this paper. We explain how to modify the Petri net to solve the deadlocks and be able to simulate the whole execution of the BP modelled, with the aim of generating all the states and transitions there represented.

Too restrictive preconditions (TRP). This kind of problems appear when the data object specified as input of an activity can be in a different state at that moment, and so the activity may stay waiting indefinitely. For instance, in the BP model of Figure 6, activity *A2* will get blocked if *A1* writes data object *D1* in state *s1*, and *A5* will get blocked if *D1* is in either *s3* or *s4*. To fix this kind of problems we relax the precondition, assuming that the state of the data object that caused the deadlock is also a precondition of the blocked activity.

Unreachable states (US). In this case, the state specified for the input data object of an activity is unreachable (either because it does not exist yet or because the object is in another state at that moment), blocking the execution of the activity. It appears in activities *A3* and *A5* of the BP model in Figure 6. To fix it, we assume the data object can be at that state at that moment and so we continue processing the BP model from the blocked activity with that "unreachable" state.

There may be other alternatives for dealing with data anomalies, but their study is out of the scope of this paper. To apply the solutions described above, the Petri net has to be "re-constructed" in order to obtain a new reachability graph

[5] For the sake of clarity we are using a simplified BP to illustrate the data anomalies.

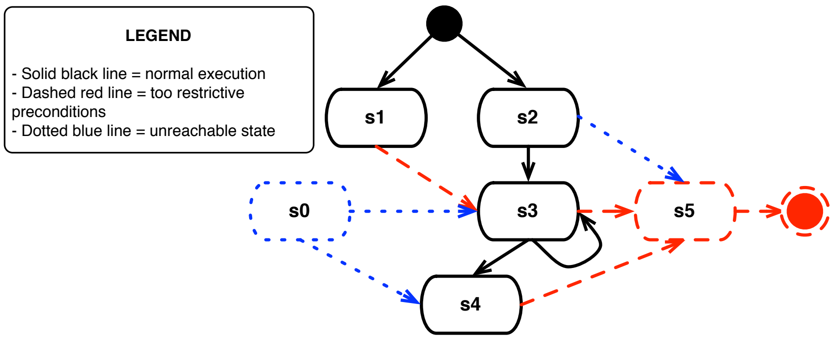

LEGEND
- Solid black line = normal execution
- Dashed red line = too restrictive preconditions
- Dotted blue line = unreachable state
s0
s1
s2
s3
s4
s5

Fig. 7. Object life cycle of data object *D1* of the business process in Fig. 6

and then process it. For too restrictive preconditions, this re-construction has 4 steps. The resulting Petri net is kept while processing the rest of warnings.

1. Find the transition at which the deadlock takes place.
2. Identify the data place that is input of the blocked transition.
3. Identify the data place at which there is a token.
4. As we now know the transition that could not be triggered (step 1) and the actual current state of the data object (step 3), to re-construct the Petri net we have to duplicate the blocked transition and replace the arrow from the data place without token (step 2) by an arrow from the data place with token (step 3), leaving the rest of inputs and outputs like they are.

Unreachable states are reflected in a Petri net in the form of unfired transitions. To detect them and fix them, we have to find the transitions that were never triggered and set the markup of the net with a token in each of their input places. Then a new reachability graph with this configuration is obtained and we can examine the rest of the net from that point by processing it.

In order to warn the modeller/analyst about the presence of data anomalies in the BP model, new elements emerged from dealing with them are marked in a different way in the OLC. In Figure 7, transitions and states generated from too restrictive preconditions are shown with dashed red lines, and those corresponding to unreachable states have dotted blue lines.

5 Related Work

The importance of complementing the activity-centered view of BP models with an object-oriented view has been described by Snoeck et al. [9]. We are generating such a view from the data objects that appear in a BP model.

The work most related to our approach is the one of Ryndina et al. In [10] they present an ad-hoc approach for the automatic generation of OLCs from a BP model and propose some techniques to analyse the consistency of BPs and OLCs on the basis of the concepts of *conformance* and *coverage* between OLCs.

Their procedure is based on transformation rules that are applied directly to a BP model. However, no data anomalies are described nor detected in their approach. Besides, our use of well-known Petri nets algorithms makes it more unlikely to introduce errors while implementing the procedure. In [11] the opposite procedure is introduced, i.e. an approach for generating a BP model from OLCs of different data objects is described.

Data anomalies in BP models have been addressed by several researchers. Sadiq et al. [3] explain the importance of managing the data requirements in BPs and introduce some ideas related to the modelling and validation of data, such as the importance of considering the type of data and their structure. They also state some data anomalies that may appear in a BP model, which in turn are referenced by the authors in [4]. In that work, Sun et al. divide the same problems into three main groups with one or more scenarios, and then they explain the matching of every scenario with the data anomalies in [3]. Awad et al. describe three kinds of data anomalies that can also be mapped to anomalies defined in the previously mentioned work [2]. They have developed an approach for diagnosing and automatically repairing these three kinds of problems on the basis of Petri nets. A prototype has been implemented in Oryx [12]. Besides, they propose some validation algorithms targeted at fixing these data anomalies, which are being implemented to correct BPMN models. In our BP2OLC procedure, we use the transformations described in that work to carry out step 1 of the BP2OLC procedure. However, the mentioned work on data anomalies does not consider the generation of OLCs from a BP model.

Finally, Sakr et al. have developed a framework for querying both control flow and data flow perspectives of BPs [13]. Data perspective can be queried from OLCs. However, no automatic generation of OLCs is included and the framework is not targeted at the detection and management of data anomalies.

6 Conclusions and Future Work

In this paper we introduce a model-driven approach for the automatic generation of a data-centered view of a BP composed of the life cycles of the data objects the BP model has. It consists of mapping a BPMN model into a target semantic domain, Petri nets, which allows us to use techniques specific to that domain, in particular obtaining its reachability graph, for analysing the source model. Then, the reachability graph is mapped into an OLC model. An advantage of our procedure is that the resulting OLCs include information about the activities that are executed in each transition and, hence, it provides the same full information required to understand BP execution as activity-centered process diagrams.

Besides, our procedure is robust in the sense that it provides an accurate result despite having a BP with data anomalies as input. Furthermore, we detail how this procedure can be used to detect two kinds of data anomalies present in a BP model. For each group of data anomalies identified, the following questions have been answered: (i) what does the anomalous situation mean in terms of the

BP model and the resulting OLC?; (ii) how can it be detected in the reachability graph?; and (iii) how can the Petri net be re-constructed to fix the deadlock?

A prototype of the BP2OLC procedure has been implemented reusing the code of ProM[6], an open-source platform for process mining that counts on a number of plugins and components to work with Petri nets. The developed prototype corresponds to steps 2 and 3 of the BP2OLC procedure and also contains the detection and handling of the data anomalies described above. It receives a Petri net in format PNML 1.3.2. as input and returns the life cycle of the data object represented in that net. The software is available upon request.

As future work, we plan to extend the kinds of BP structures considered, to take data objects from repositories into account, and to study alternatives to manage and repair the detected data anomalies in the source BP model.

References

1. Bpmn 2.0, recommendation, OMG (2011)
2. Awad, A., Decker, G., Lohmann, N.: Diagnosing and repairing data anomalies in process models. In: BPM Workshops, pp. 5–16 (2009)
3. Sadiq, S., Orlowska, M.E., Sadiq, W., Foulger, C.: Data flow and validation in workflow modelling. In: Fifteenth Australasian Database Conference (ADC). CRPIT, vol. 27, pp. 207–214. ACS (2004)
4. 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(4), 374–391 (2006)
5. Awad, A., Weidlich, M., Weske, M.: Specification, verification and explanation of violation for data aware compliance rules. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 500–515. Springer, Heidelberg (2009)
6. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
7. Harel, D., Rumpe, B.: "Meaningful modeling: what's the semantics of "semantics"? Computer 37(10), 64–72 (2004)
8. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
9. Snoeck, M., Poelmans, S., Dedene, G.: An architecture for bridging oo and business process modelling. In: Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages, TOOLS 33, pp. 132–143 (2000)
10. Ryndina, K., Kuster, J., Gall, H.: Consistency of business process models and object life cycles. Models in Software Engineering, 80–90 (2007)
11. Kuster, J., Ryndina, K., Gall, H.: Generation of business process models for object life cycle compliance. In: Business Process Management, pp. 165–181 (2007)
12. Decker, G., Overdick, H., Weske, M.: Oryx - an open modeling platform for the BPM community. In: Proceedings of the 6th International Conference on Business Process Management, pp. 382–385. Springer, Heidelberg (2008)
13. Sakr, S., Awad, A.: A framework for querying graph-based business process models. In: WWW, pp. 1297–1300 (2010)

[6] http://prom.win.tue.nl/tools/prom/