TOWARD A HYBRID ALGORITHM FOR WORKFLOW GRAPH STRUCTURAL VERIFICATION

Fodé Touré[1], Karim Baïna[1] and Walid Gaaloul[2]
[1] *ENSIAS, Université Mohammed V-Souissi, BP 713 Agdal, Rabat, Morocco*
[2] *DERI-NUIG, IDA Business Park, Galway, Ireland*

Keywords: Business processes, business process validation, workflow structural checking, graph reduction, graph traversal.

Abstract: Appropriate definition, analysis, checking and improvement of business process models are indispensable before their deployment within workflow management systems. In this paper, we focus on business process model verification that insures business process structural correctness. Our proposal consist in a new hybrid algorithm of workflow graph structural validation combining graph reduction and traversal mechanisms. Our algorithm will be discussed and compared to existing workflow structural checking approaches.

1 INTRODUCTION

A process model is the formal definition of a business process. The objective of a model is to produce high-level specifications of workflows, independently of workflow management system. Consequently, the processes must be correctly modeled before they are implemented as workflows.The invalid processes deployment can lead processes-based applications to states of incoherence and can even provoke very critical breakdowns without the slightest possibility of resumption from where the interest of the validation of the models of processes

There are several aspects in a process model including the structure, the dataflow, the roles, the application interface, the temporal constraints and others. In a practical way and taking into account the life cycle of a process (Sadiq et al., 2004), the process model validation can be divided into three steps: structural validation, contextual validation and pragmatic validation (Figure 1).

The structural validation (aka verification) consists first in syntactic checking of the model by taking into account the modeling language then in the basic models various combinations validation and possibly to correct the model structure. The contextual validation is not only interested in the internal structure of nodes but also in their production influence on the neighboring nodes behavior. The pragmatic validation is interested in the checking of the re-usability of a valid model in new run-time contexts.

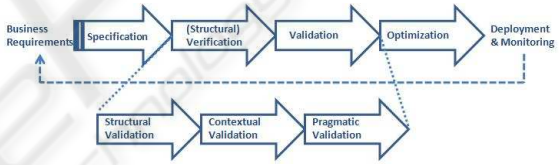

Figure 1: Process life cycle and process of validation steps.

Process structure is the most important and primary aspect of a process model. It builds the base to capture other aspects of the workflow needs. It is why, in the suite of this paper, we will essentially be interested in the structural validation.

Our paper is structured as follows: section 2 introduces concept definitions, section 3 details our workflow verification algorithm, section 4 studies related works, section 5 brings discussion elements, and section 6 concludes.

2 CONCEPT DEFINITIONS

Workflow specifications use graphic objects. Most of workflow management systems use a proprietary language. However, Workflow Management Coalition $(WfMC)$ has developed a standard process definition language and an interface specification which could be used to transfer process models between products. In this language, the processes are modeled by using two object types: node and flow.

Touré F., Bäına K. and Gaaloul W. (2008).
TOWARD A HYBRID ALGORITHM FOR WORKFLOW GRAPH STRUCTURAL VERIFICATION.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - ISAS*, pages 442-447
DOI: 10.5220/0001722404420447

The node is classified in two categories: task and condition. A task, graphically represented by a rectangle, represents the work to be made to realize some objectives. A condition, graphically represented by a circle, is used to build choice structures. A flow links two nodes in graph and is graphically represented by an arrow. Syntactically and in a basic way, objects model have to verify following rules (Figure 2):

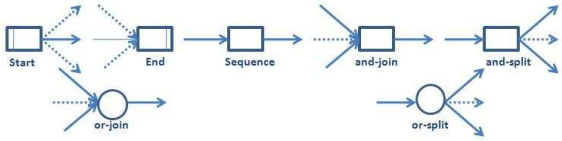

Figure 2: Process model objects syntactic rule.

Definition 2.1 (Structural validation)**.** *Structural validation is all techniques used to identify objects incorrect combinations, means allowing to avoid them or possibly to correct them to increase the pre-execution reliability of a process.*

A Directed Acyclic Graph (DAG) can contain two types of structural conflict: deadlock (definition 2.2) and lack of synchronization (definition 2.3)(Sadiq and Orlowska, 1999; van der Aalst et al., 2002; Lin et al., 2002)

Definition 2.2 (Process graph with deadlock)**.** *A process graph contains a deadlock if it produces an instance subgraph which contains on the same path the following regular pattern (or-split)t^*(and-join) where t is a task and $t*$ a path containing a sequence (eventually empty) of tasks.*

Definition 2.3 (Process graph with lack of synchronization)**.** *A process graph contains a lack of synchronization if it produces an instance subgraph which contains on the same path the following regular pattern (and-split)t^*(or-join) where t is a task.*

As reduction goal is to delete valid structures to verify graph correctness, the graph transformation by introducing new objects or exchanging nodes order, does not constitute a reduction rule because graph integrity is not respected.

The figure 7 is a graph which cannot be reduced by the four first rules $R_1 \ldots R_4$ of (Sadiq and Orlowska, 2000) thus, it is a split-join graph.

3 OUR WORKFLOW VERIFICATION ALGORITHM

3.1 Algorithm Principle

Studying all publications around around structural conflicts in workflow graph, we easily realize that in

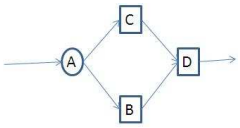

Figure 3: Graph with deadlock.

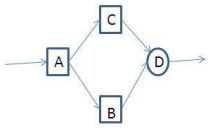

Figure 4: Graph with lack of synchronization.

Definition 2.4 (Workflow graph reduction)**.** *Workflow graph reduction consists in deleting correct structures in graph by respecting nodes Scheduling and making sure that this deletion does not introduce new conflict or does not delete existing conflict.*

a process graph, any structural conflict is concerned by a join node. Therefore, to detect a conflict in a process graph, it is sufficient to localize split-join incorrect combinations. However, this localization can be very complex when we pass in the scale (graph is too big), so it is important to reduce the size of the graph according to the definition 2.4, while preserving nodes order.

Our approach consists in graph reducing at first then in localizing incorrect combinations in graph by graph traversal. For that purpose, we use seven rules ($R_1 \ldots R_7$) of which four ($R_1 \ldots R_4$) of Sadiq-Orlowska algorithm (Sadiq and Orlowska, 2000). As Sadiq-Orlowska algorithm fifth rule applies to a split-join graph particular case on four level and as the identification of the whole structure is very complicated, we replaced it by the rules $R_5 \ldots R_7$ which insure the split-join graph correctness in a general way.

R_1 - Terminal Reduction Rule (Sadiq and Orlowska, 2000): the terminal reduction rule deletes process graph beginning task and end task if the number of transitions attached to them is equal to one. The beginning (respectively end) task deletion condition implies that it is neither and-split nor or-split (is neither or-join nor and-join).

R_2 - Sequential Reduction Rule (Sadiq and Orlowska, 2000): If the current node forms a sequential structure, that means, if it has exactly one incoming flow and one outgoing flow, the sequential rule deletes the current node of the graph and deletes its outgoing transition (the deleted node incoming transition will point to this last one outgoing node. G

Definition 2.5 (split-join graph)**.** *A split-join is graph that contains only combinations of split-join with split (respectively join) adjacent of different type. That kind of graphs is irreducible by the four first rules of (Sadiq and Orlowska, 2000).*

Definition 2.6 (Correct join node)**.** *In a split-join graph, a join node which is not involved in a structural conflict is correct.*

Definition 2.7 (First level join Node)**.** *In a split-join graph, a first level join node is a join node which has no other join node as predecessor.*

is a graph where $f_1(n_1,n), f_2(n,n_2) \in F^2 \Rightarrow_{R_2} G ::= G'(N \setminus \{n\}, F \cup \{f(n_1,n_2)\} \setminus \{f_1, f_2\}))$.

R_3 - Adjacent Reduction Rule (Sadiq and Orlowska, 2000): The adjacent reduction rule targets two components types . If the current node is not deleted neither according to the terminal reduction rule nor the sequential reduction rule, it means that it forms a structure (i) split or (ii) join.

(i) split structures fusion : If the current node forms a split structure and has a single incoming transition and if the current node is of the same type as its predecessors node, the rule moves the current node outgoing transitions to the predecessors node and deletes the current node (and-split associativity and or-split associativity): $\forall n_1,n_2,n_3 \in N$, (or-split(or-split($n_1,n_2$),$n_3$)$\Rightarrow_{R_3}$ or-split(n_1,n_2,n_3)) et (and-split(and-split(n_1,n_2),n_3)$\Rightarrow_{R_3}$ and-split(n_1,n_2,n_3)) ;

(ii) join structures fusion : Otherwise if the current node forms a join structure and has a single outgoing transition and if the current node is of the same type as its successors node, the rule moves the current node incoming transitions to the successors node and deletes the current node (and-join associativity and or-join associativity) : $\forall n_1,n_2,n_3 \in N$, (or-join(or-join($n_1,n_2$),$n_3$)$\Rightarrow_{R_3}$ or-join(n_1,n_2,n_3)) et (and-join(and-join(n_1,n_2),n_3)$\Rightarrow_{R_3}$ and-join(n_1,n_2,n_3));

R_4 - Closed Reduction Rule (Sadiq and Orlowska, 2000): Sequential and adjacent reduction rules application introduce generally process graph deformations. Nodes of the same type can have more than one transition between them (process graph becomes thus not elementary). The closed reduction rule deletes all transitions between such nodes except only one. G is a graph where $f_1(n_1,n_2), f_2(n_1,n_2) \in F^2 \Rightarrow_{R_4} G ::= G'(N, F \setminus \{f_2\})$

After application of previous rules, we obtain either (1) an empty graph, or (2) a split-join graph. In the first case, the original graph is without structural conflict. In the second case (figure 7), next rules consist in traversing the $R_1 \ldots R_4$ resulting reduced process graph seeking for not correct join nodes (as stated in definition 2.6).

R_5 - Combination (split-join) Valid Rule:

1 For every join node, find the first split S_0 common to all paths leading to the join node considered;

2 If current join node is of or-join type, to state that the split-join combination is valid, it is necessary to make sure that, according to S_0 outgoing transitions activation, only one of the join node incoming transitions is activated. Otherwise there is lack of synchronization conflict.

3 If current join node is of and-join type, to state that the split-join combination is valid, it is necessary to make sure that, according to S_0 outgoing transitions activation, all incoming transitions of join node are activated. Otherwise there is deadlock conflict.

R_6 - Validation by Extension Rule: In a split-join graph, the correctness of a join node, which has as predecessors only correct join nodes, depends on the activation of these last ones. Any and-join node, which has as predecessors only correct join nodes, is correct if these nodes are simultaneously active. On the other hand, any or-join node, which has as predecessors only correct join nodes, is correct if it is always only one of these nodes that is active. In the figure 5

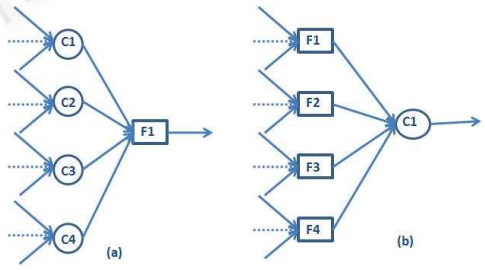

Figure 5: Validation by extension.

(a) $F1$ is correct if $C1$, $C2$, $C3$ and $C4$ are correct and simultaneously active. On the other hand, in the figure 5 (b), if $F1$, $F2$, $F3$ and $F4$ are correct and active simultaneously, $C1$ is not correct.

R_7 - Semi-validation by Extension Rule: In a split-join graph, the correctness of a join node, which has as predecessors correct join nodes and split nodes, depends on the considered join node nature and the activation or no of incoming flow from split nodes in relation to the correct join nodes. In a model looking like the figure 6 if the node B is correct, according to the nature of the join node C, the correctness of C

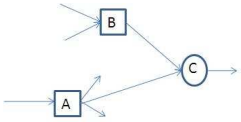

Figure 6: Validation by semi-extension.

depends on the way of incoming transition activating from the node *A* in relation to the node *B*. For the figure 6 particular case, node *C* is correct if nodes *A* and *B* are not activated simultaneously. To verify the correctness in similar case, we are not obliged to make a graph traversal, we use only the relation enter *A*, the split S_0 of *B* and join node *C* nature.

3.2 Our Algorithm

Require: *G* respecting the preconditions (hypotheses)
Ensure: $Reduce(G)$ =true iff *G* is valid ; false else
$G' \longleftarrow G$
$lastsize \longleftarrow size[G'] + 1$
while $(lastsize > size[G'])$ **do**
$lastsize \longleftarrow size[G']$
Pass 1 : apply to G' terminal reduction rule (R_1), then sequential reduction rule (R_2), then adjacent reduction rule (R_3).
if $(lastsize < size[G'])$ **then**
Pass 2 : apply to G' closed reduction rule (R_4).
end if
end while
if (graph G' is empty) **then**
% the original graph *G* is valid
return true □
else
% the original graph G' is a split-join graph
G" $\longleftarrow G'$
% rules R_5, R_6, R_7 application
Seek all join nodes Nj of the graph G"
for all Nj **do**
if first level(Nj) **then**
Pass 3 : apply to G" the rule(R_5).
Find the first Split S_0 common to all paths leading to the considered join node Nj;
According to the activation of S_0 outgoing flows;
if correct(Nj) **then**
% correct split-join combination
else
% The graph *G* contains a structural conflict
return false □
end if
else
apply to G" the rule(R_6).
if correct(Nj) **then**
% correct split-join combination
else
% The graph *G* contains a structural conflict
return false □
end if
else
apply to G" the rule(R_7).
if correct(Nj) **then**
% correct split-join combination
else
% The graph *G* contains a structural conflict
return false □
end if
else
Find the first Split S_0 common to all paths leading to the considered join node Nj;
According to the activation of S_0 outgoing flows;
if correct(Nj) **then**
% correct split-join combination
else
% The graph *G* contains a structural conflict
return false □
end if
end if
end for
% the original graph *G* is valid
return true □
end if

3.3 Application Example

In this section, we are going to apply our algorithm to a process model. For that purpose, we choose a model proposed by (Lin et al., 2002) as counter example to prove that the Sadiq-Orlowska algorithm is not complete.

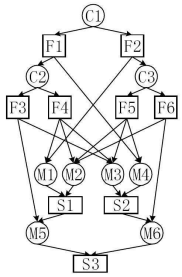

Figure 7: split-join graph (Lin et al., 2002).

The figure 7 (Lin et al., 2002) shows an irreducible model by the four first rules of Sadiq-Orlowska algorithm (Sadiq and Orlowska, 2000) thus, it is a split-join graph. Our algorithm rules R_5, R_6, R_7 application on this model is presented in the table 1 below.

Our algorithm application on the figure 7 model shows that this last one is without structural conflict because all join nodes are correct (definition 2.6). It is important to note that in our algorithm, the rule (R_5) is effectively applied only for first level join nodes. Thus, for a higher level join node, the verification is insured by rules R_6, R_7. In other words, we make the (split-join) graph partial traversal only to verify the correctness of first level join nodes. For the remainder, we study lower level join nodes behavior.

Table 1: Our algorithm application on figure7 graph.

Join Node	Split S_0	Traversal	Stat	Obs
M1 (or)	C1 (or)	C1→F1→C2→F4→M1 / C1→F2→M1	one active flow on two	correct R_5
M2 (or)	C1 (or)	C1→F1→C2→F4→M2 / C1→F2→C3→F5→M2 / C1→F2→C3→F6→M2	one active flow on three	correct R_5
M3 (or)	C1 (or)	C1→F1→C2→F3→M3 / C1→F1→C2→F4→M3 / C1→F2→C3→F5→M3	one active flow on three	correct R_5
M4 (or)	C1 (or)	C1→F1→M4 / C1→F2→C3→F5→M4	one active flow on two	correct R_5
S1 (and)		correct(M1)=true ; correct(M2)=true	Two active flows on two	correct R_6
S2 (and)		correct(M3)=true ; correct(M4)=true	Two active flows on two	correct R_6
M5 (or)	C1 (or)	correct(S1)=true / C1→F1→**C2**→F3→M5	one active flow on two	correct R_7
M6 (or)	C1 (or)	correct(S2)=true / C1→F2→**C3**→F6→M6	one active flow on two	correct R_7
S3 (and)		correct(M5)=true ; correct(M6)=true	Two active flows on two	correct R_6

4 STATE OF THE ART

As we exposed it in section 2, a process graph contains two structural conflict: deadlock and lack of synchronization (Sadiq and Orlowska, 2000). In the goal to verify or to assure the correctness of a process model, several propositions were made among others: reduction-based algorithms (Sadiq and Orlowska, 2000; Lin et al., 2002), graph-traversal-based algorithm (Perumal and Mahanti, 2005; Perumal and Mahanti, 2007), approach of transformation of not valid model in valid model (Liu and Kumar, 2005) and approach of model conversion in Petri net (van der Aalst et al., 2002). As our approach does not concern this last point, we are not going to present it in this paper.

Sadiq-Orlowska Algorithm. This algorithm consists in deleting of all certainly correct structures in a workflow graph. The process of reduction reduces finally a structurally correct workflow graph in an empty graph. On the other hand, a workflow graph with structural conflict is not completely reduced. The process of reduction uses five reduction rules - **terminal, sequential, adjacent, closed and overlapped** - as long as they are capable to reduce the graph. These reduction rules are applied by visiting all graph nodes and verifying if a reduction rule can be applied.
The complexity of the algorithm worst cases is $O((size(G))^2)$ where $size(G) = |N| + |F|$ (Sadiq and Orlowska, 1999).

Lin-Zhao-Li-Chen Algorithm. By using figure 7 model, Lin-Zhao-Li-Chen demonstrated that Sadiq-Orlowska algorithm is limited because he cannot reduce, in empty graph, models which are nevertheless correct (what we called split-join graph). So, Lin-Zhao-Li-Chen algorithm is an improvement of Sadiq-Orlowska. This algorithm is based on seven reduction rules of which the four first of Sadiq-Orlowska algorithm: **terminal, sequential, adjacent, closed, choice-convergence, synchronizer-convergence, merge-fork** (Lin et al., 2002). Its theoretical complexity is $O(|N| + |F|)^2.|N|^2)$ where $|N|$ is the number of nodes and $|F|$ is the number of transitions (flow).

Mahanti-Sinnakkrishnan Algorithm. This algorithm uses properties and methods of the algorithms "Traversal in depth" (Depth-First Search: DFS) and $AO*$ to create and verify a workflow graph various instance subgraphs. The properties of the algorithm $AO*$ are used to choose only select instance subgraphs in a way that verifying this subset of instance subgraphs is equivalent to the verification of the complete workflow graph. In this algorithm, the structural conflicts are identified after two Traversals in depth of every instance subgraph by using the operations: Create-Instance-Subgraph (CIS) and Verify-Instance-Subgraph (VIS). Its complexity, according to them, is $O(|F|^2)$ where $|F|$ is the number of transitions (flow) (Perumal and Mahanti, 2005).

5 DISCUSSION

Our algorithm interest compared to the reduction explains itself by the fact that it is complete and simpler. It verifies any acyclic workflow graph without transforming it. Compared to our algorithm, Mahanti-Sinnakkrishnan algorithm is less efficient because it makes useless traversals. In our approach, the reduction is applied when it is necessary "to simplify" the graph and the search is applied in a partial and localized way. Thus, the hybridization gathers the best both graph traversal and reduction. In the aim of assuring a good understanding of the of this paper, it is necessary to define certain properties.

Property 5.1 (Termination)**.** *an algorithm terminates if it doesn't lead to an abnormal break during its execution.*

Property 5.2 (Correctness)**.** *a correct algorithm must answer, after termination, by the affirmative or the negative according to the result of the execution.*

Property 5.3 (Completeness)**.** *an algorithm is complete if it verifies the structural correctness of any graph.*

Property 5.4 (Transformation)**.** *an algorithm transforms a graph if, during verification, it changes nodes order or introduces new node into the graph.*

Table 2: Workflow verification algorithms Comparison.

	Completeness	Transformation	Complexity
Sadiq-Orlowska	No	No	$O((\|N\|+\|F\|)^2)$
Lin-Zhao-Li-Chen	Yes	Yes	$O((\|N\|+\|F\|)^2.\|N\|^2)$
Mahanti-Sinnakkrishnan	Yes	No	$O(\|F\|^2)$
Touré-Baïna	Yes	No	$O(\|K\|^2+\|F'\|^2)$

Taking into account properties above and what is exposed higher in this paper, we can affirm that (1) for **termination**, our algorithm finishes by convergence of the reduction and the traversal; (2) for **correctness**, our algorithm answers YES when it is correct and NO otherwise; (3) for **completeness**, our algorithm verifies the correctness of any graph, same complexes; and (4) for **transformation**, our algorithm does not transform the graph because it does not change nodes order and does not introduce new node. Our algorithm begins with the application of the four first rules $R_1 \ldots R_4$ reduce graph size. This choice takes into account the fact that if the workflow graph G_i is the graph obtained after application of i iterations of the rules R_1 to R_4 on G, then: reduction rules do not produce structural conflicts, that means, if G_i is correct then G_{i+1} is correct; reduction rules do not delete structural conflicts, that means, if G_i is incorrect then G_{i+1} is incorrect; and, If the application of rules R_1 to R_4 cannot reduce G to an empty graph then G_i has to be a graph split-join otherwise G will always be reduced to an empty graph. In case where graph is not completely reducible, the rule R_5 allows to verify the correctness of first level join nodes by making a partial and localized traversal of the graph. For others join nodes, we apply rules R_6, R_7 or a simple deduction by using predecessors correct join nodes behavior.
Algorithm complexity is $O((|N|+|F|)^2)$ where $|N|+|F|$ represents the number of nodes and transition in the workflow graph (Sadiq and Orlowska, 2000), in case the graph is completely reducible by rules R_1 to R_4. The average case complexity is much lower in $O((|N|+|F|)^2)$, since the first iterations reduce radically workflow graph size.
In case where the graph is not reducible (split-join graph), the worse case is that R_5 is the only applicable rule to verify join nodes correctness. In that case, we would be obliged to traverse the entire new graph. But, as this case does not exist in a split-join graph then our algorithm complexity is theoretically lower than that of the graph traversal in depth algorithm. Thus more efficient with a complexity $O(|F'|^2)$ where $|F'|$ is the number of transition between join first level nodes and their split S_0. In the final, in the worst of the cases, reduction $O(|K|^2)$ + traversal $O(|F'|^2)$ gives an algorithm in $O(|K|^2+|F'|^2)$ where $|K|$ is the number of reducible node and transition and $|F'|$ is the number of transition between first level join nodes and their split S_0. Table 2 compares verification algorithms according to defined properties.

6 CONCLUSIONS

Through this paper you discovered a new effective and complete algorithm that verifies the structural correctness of any acyclic workflow graph by hybridizing the graph reduction and traversal. In our perspective, we work on our algorithm application for cyclic workflow graphs structural verification through a workflow graph structural verification tool.

REFERENCES

Lin, H., Zhao, Z., Li, H., and Chen, Z. (2002). A novel graph reduction algorithm to identify structural conflicts. In *HICSS*, page 289.

Liu, R. and Kumar, A. (2005). An analysis and taxonomy of unstructured workflows. In *Business Process Management*, pages 268–284.

Perumal, S. and Mahanti, A. (2005). A graph-search based algorithm for verifying workflow graphs. In *DEXA Workshops*, pages 992–996. IEEE Computer Society.

Perumal, S. and Mahanti, A. (2007). Applying graph search techniques for workflow verification. In *HICSS*, page 48.

Sadiq, S., Orlowska, M., Sadiq, W., and Foulger, C. (2004). Data flow and validation in workflow modelling. In *ADC '04: Proceedings of the 15th Australasian database conference*, pages 207–214, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Sadiq, W. and Orlowska, M. E. (1999). Applying graph reduction techniques for identifying structural conflicts in process models. In *CAiSE*, pages 195–209.

Sadiq, W. and Orlowska, M. E. (2000). Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134.

van der Aalst, W. M. P., Hirnschall, A., and Verbeek, H. M. W. E. (2002). An alternative way to analyze workflow graphs. In *CAiSE*, pages 535–552.