A Two Phase Verification Algorithm for Cyclic Workflow Graphs

Yongsun Choi

Dept. of Systems Management & Engineering, Inje University, Kimhae, 621-749, Korea
yschoi@inje.ac.kr

ABSTRACT

The widespread automation of e-business processes has made workflow analysis and design an integral part of information management. Graph-based workflow models enables depicting complex processes in a fairly compact form. This free form, on the other hand, can yield models that may fail depending on the judgment of the modeler and create modeling situations that cannot be executed or will behave in a manner not expected by the modeler. Further, cycles in workflow models needed for purposes of rework and information feedback increase the complexity of workflow analysis. This paper presents a novel method of partitioning a cyclic workflow process, represented in a directed graph, into a set of acyclic subgraphs. This allows a cyclic workflow model to be analyzed further with several smaller sub-flows, which are all acyclic. As a convincing example, we present two-phased verification of structural conflicts in workflow models for those incurred from the inappropriate composition of partitioned flows and the others within each acyclic subgraph, which is much easier to comprehend and verify individually than the whole workflow model, in general.

***Keywords*:** workflow model, directed graph, feedback, partition, verification

1. INTRODUCTION

The recent surge of e-business research and development has resulted in the automation of thousands of business processes by means of workflow management systems (WfMS), both within and across corporate boundaries ([5], [7], [9], [16], [18], , [30], [32]). WfMS enable the design, analysis, optimization, and execution of business processes. The basic WfMS functions include the separation between the business process logic and business applications, management of relationships among process participants, integration of internal and external process resources, and monitoring and control of process performances ([24], [34], [35]). There are typically two stages of workflow management, workflow specification and workflow execution. The former defines a workflow model, and the latter generates workflow instances guided by the workflow model.

A workflow model (also called workflow definition) consists of a set of activities and their execution sequences, the start and termination of the process, and the resources needed for executing individual activities, such as participants, associated IT applications, and data ([34], [35]). The objective of workflow modeling is to provide high-level specification of processes that are independent of the implementation intricacies of the target workflow management system ([8], [13], [18], [21], [22], [35]). Of all workflow perspectives, e.g., control-flow, data, organization, task, and operation, the control-flow perspective is the most prominent one because it defines the backbone of the workflow on which other perspectives can be specified [1].

Workflow models must be correctly defined before being deployed in a workflow management system to avoid any costly maintenance delays due to runtime errors in the process model [29]. Therefore, it is essential to verify the workflow model before its deployment. Despite the importance of workflow verification, few commercial workflow systems provide formal verification tools. This lack of verification support can be attributed to the fact that most of the more than 250 commercially available WfMS use a vendor-specific ad-hoc modeling techniques [5] without theoretical framework for the representation, analysis, and manipulation of workflow systems [8]. Although these vendor-provided tools facilitate workflow specification and visualization, and some support simulation of processes under various conditions, they usually do not support formal workflow analyses. While process simulation can provide useful insight into process behavior, it does not address questions about the interrelationships among process components [8]. Consequently, few workflows are thoroughly checked before they are deployed in practice, often resulting in errors that need to be corrected in an ad hoc fashion at prohibitive costs.

There are two research approaches to ensure control flow correctness in workflow models – *build it correctly*, or *check it completely* [31]. The former ([10], [17], structured model in [21]), relying on strict rules in composing the model, may not model certain processes due to syntactical restrictions [21] and may not be very suitable for practical implementation in industry [31]. The latter, like Petri nets ([1], [4], [6], [26], [31], [33]) or the graph-theoretic techniques ([8], [14], [25], [29]), on the contrary, appeals more by allowing the user tremendous flexibility in expressing process requirements offering interesting analysis challenges to the modeler [31]. The Petri-nets-based workflow verification depends on the formalism of Petri nets, which is rarely adopted by existing WfMS, and makes it hard to be embedded into most of WfMS. Therefore, for non-Petri-nets workflow models, it is difficult to apply Petri-nets-based verification techniques as it requires the

transformation from each different modeling paradigm into the Petri nets formalism plus an effort of determining the corresponding structural conflicts in the original model. The majority of business processes have been analyzed using acyclic (i.e. loop-free) free-choice nets, a special class of Petri nets that enjoys the added advantage that *soundness* can be verified in polynomial time [1]. Additionally, establishing soundness in a free-choice net implies that the net is free from deadlock, and is also alive, i.e., no dead tasks [2]. However, the modeling of exceptions or precedence partially destroys the free-choiceness of the equivalent Petri net mapping. Moreover, modeling iteration necessitates the presence of loops in the control flow model, a problem that is yet to be satisfactorily addressed [31].

Graph-based workflow models have been widely used in industry to specify control flows of a business process. Although graphs provide a great flexibility for depicting complex processes in a fairly compact form, their free-form nature also needs to resolve models that may fail due to a lack of support for the modeler to avoid situations that cannot be executed or will behave in a manner not expected by the modeler [12]. Graph reduction ([25], [29]) and block-wise abstraction [14] have been proposed to identify structural conflicts in workflow graphs, but both approaches are limited to acyclic models [3].

This paper presents a novel method of partitioning into a set of acyclic subgraphs and decomposed verification of structural conflicts for a cyclic workflow process, even for nested structures, represented in a directed graph. Our approach of feedback partitioning and decomposed verification is unique and has several distinctive features useful for workflow analysis and design. First, we present two-phased verification of structural conflicts in workflow models. Our method detects certain types of structural conflicts, in the early stage, incurred from the inappropriate composition of feedback flows. And then, our method applies verification to each acyclic subgraph, which is much easier to comprehend and verify individually than the whole workflow model, in general. This decomposition feature has not been seen in any existing workflow verification method. Second, each acyclic flow can be utilized in hierarchically decomposing the given model and is a candidate for a sub-workflow, the knowledge of which is helpful for managing large-scale workflows. Finally, our method is based on a directed graph representation, more generic and adopted widely by the WfMS vendors, and utilizes well-known algorithms of graph-theoretic techniques, which makes it easy to embed our approach into most WfMS.

The rest of the paper is organized is as follows. In Section 2, we present the preliminary concepts such as directed graph representation and types of structural conflicts of a workflow model. In Section 3, we present the way of partitioning a cyclic workflow graph into the subgraphs of acyclic flows with illustrative examples. Section 4 delineates the two-phased verification of structural conflicts and presents the associated theorems. Section 5 discusses related work, summarizes our contributions, and outlines future research directions.

2. WORKFLOW MODELS IN DIRECTED GRAPHS

A workflow graph is a directed graph $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{T})$ with a set of nodes $\boldsymbol{N}$ and a set of edges $(i, j) \in \boldsymbol{T}$, where $i, j \in \boldsymbol{N}$. Each edge, called a *transition*, links two nodes and represents the execution order of nodes. A *node* is classified into two types, task and coordinator. A *task*, represented by a rectangle, stands for a unit of work to be done. A *coordinator*, represented by a circle, stands for a point where only one of succeeding paths is selected or several paths are merged. Depending on the types of nodes and the number of incoming and outgoing transitions, nodes can be classified into five categories, i.e., *sequence*, *AND-split*, *AND-join*, *XOR-split*, and *OR-join*, as shown in Fig. 1. *Start* and *End* nodes are used to indicate the beginning and the end of the given workflow process, respectively ([25], [29]).

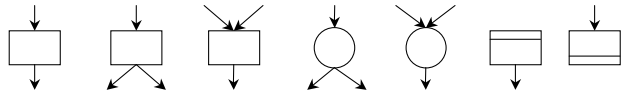


Fig. 1. Classification of nodes in workflow graphs

Fig. 2(a) illustrates an example cyclic workflow, extending an acyclic example in [29]. We make a simplifying assumption on the workflow graph that a node cannot be a join and split at the same time. Any node that joins and concurrently splits can be converted into a set of a join and a split, separate and connected with a transition.

Structural conflicts in workflow models

In this paper, we focus on the verification of three types of structural conflicts, deadlock, lack of synchronization, and livelock as illustrated in Fig. 3 ([1], [20], [29]). *Deadlock* refers to a situation in which a workflow instance gets into a stalemate such that no further activity can be executed. This happens when only some partial subset of the join paths to an AND-Join is executed, the AND-Join node *k* will wait forever and block the continuation of the process (Fig. 3(a)). *Lack of synchronization* refers to a situation in which the concurrent flows are joined by an OR-Join, resulting in unintentional multiple executions of activities following the node *k* (Fig. 3(b)). *Livelock* refers to a situation in which certain loop(s) of tasks are continuously performed, and there is no execution path leading to termination or cannot terminate properly (Fig. 3(c)). Note that livelock occurs only where a cyclic structure exists.

Other types of structural conflicts such as *dangled* nodes are relatively easier to detect than those described above, from the graph-theoretic viewpoint. A dangled

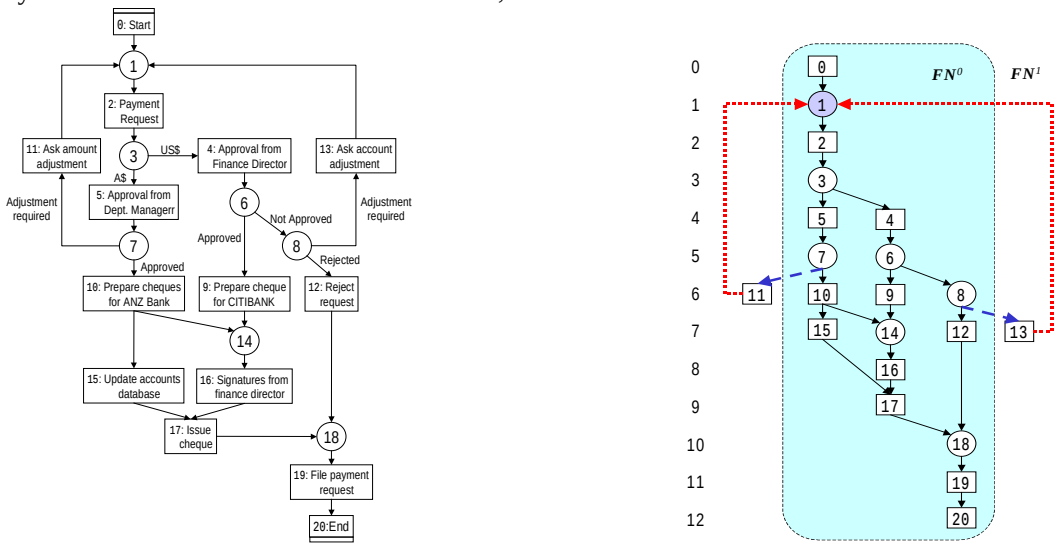


Fig. 2. (a) An example cyclic workflow graph; and (b) the normalized graph

node can be found simply by checking the reachability of each node from the Start node and to the End node.

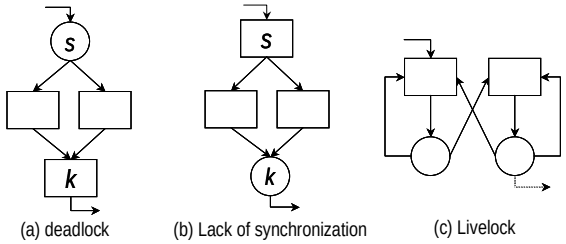


Fig. 3. Types of structural conflicts

3. PARTITIONING A CYCLIC WORKFLOW INTO ACYCLIC FLOWS

Cycles in workflow models are needed for purposes of rework and information feedback. For instance, the workflow in Fig. 2 contains two simple cycles, i.e., a cycle that does not contain any other cycles [19]. Our method identifies and partitions off each acyclic flow from the given cyclic workflow model. At the first iteration of partition, our method partitions off a maximally connected subgraph from the *Start* to the *End*, spanned by all nodes reachable to the *End* without multiple visits to any nodes contained, called *main flow*. If the original workflow model has nested feedback structures, our method proceeds on partitioning further to the remaining subgraph(s) that contain(s) cyclic structures. Otherwise, our method identifies and partition off each additional acyclic *feedback flow* from the remaining subgraph(s). More formal definitions and detailed descriptions of our method are given next.

The algorithm of *Depth-First Search* can be applied to identify the "back edges" where each of those completes a cycle, with the complexity of $O(|\boldsymbol{N}|+|\boldsymbol{T}|)$ [15]. An edge (u, v) is classified as a back edge if v is already discovered but any of its adjacent nodes are not discovered yet, when to explore edge (u, v) in the course of *Depth-First Search*. With all *back edges* removed, the rank of each node can be computed from the resulting acyclic workflow graph with the complexity of $O(|\boldsymbol{T}|)$ [19]. Fig. 2(b) shows the normalized graph of the workflow process in Fig. 2(a), with nodes rearranged by its rank $r(i)$, $i \in \boldsymbol{N}$, computed from the resulting acyclic workflow graph and shown to the left in Fig. 2(b). Fig. 4(b) shows another example of normalized graph for a workflow of Fig. 4(a) with nested cycles.

A normalized graph is helpful for identifying feedback structures of a cyclic workflow process. By examining the normalized graphs in Fig. 2(b) and Fig. 4(b), we find some interesting phenomena. First, every dotted upstream transition (i, j), with $r(i) > r(j)$, merges a feedback flow, which is a back edge as defined above. Second, each dashed transition initiates a new feedback flow. Transition (13, 7) in Fig. 4(b) has both characteristics. These transitions can be used as the cut sets to group the nodes in $\boldsymbol{N}$. Next, we describe an iterative way of identifying those transitions initiating or merging each feedback flow and partitioning the graph of a cyclic workflow into subgraphs of acyclic flows.

Steps of a partitioning

The back edges of a cyclic workflow graph found during normalization are called as *Feedback Join Transitions* (***FJT***). Each transition (i, j) of ***FJT*** has the property of $r(i) > r(j)$ and corresponds to a distinct simple cycle of a cyclic workflow graph. ***FJT*** = {(11, 1), (13, 1)} and ***FJT*** = {(11, 7), (13, 7), (23, 19), (24, 1)} for the examples shown in Fig. 2 and 4, respectively. Nodes that merge feedback flows are called as *Feedback Joins*, denoted by ***FJ***, where

$$\boldsymbol{FJ} = \{\, j \mid (i, j) \in \boldsymbol{FJT} \,\}, \qquad (1)$$

FJ = { 1 } and ***FJ*** = { 1, 7, 19 } for the examples shown in Fig. 2 and 4, respectively.

Definition 1. For any $i \in \boldsymbol{N}$, let the *Order of Feedback of i*, called by *OF(i)*, denote the minimum number of transitions of ***FJT*** to pass to the *End* node. The subset of nodes with *OF(i)* = *n* will be called as n^{th}*-order Feedback Nodes* and denoted by $\boldsymbol{FN}^n$.

Note that $\boldsymbol{N} = \cup_n \boldsymbol{FN}^n$. Nodes of $\boldsymbol{FN}^0$ do not need any transitions in ***FJT*** to reach the *End* node, and $\boldsymbol{FN}^0$ can be identified as follows,

$$\boldsymbol{FN}^0 = \{\, i \mid R_{FJT}(i, End) = 1 \,\}, \quad (2)$$

where $R_{FJT}(i, End)$, which can be computed with complexity of $O(|\boldsymbol{T}|)$, denote the *reachability* of node *i* to the *End* node without the transitions of ***FJT***. That is, $R_{FJT}(i, End) = 1$ when node *i* can reach the *End* node without passing through any transitions in ***FJT***, or $R_{FJT}(i, End) = 0$ otherwise. $R_{FJT}(End, End)$ is defined to be 1. Table 1 shows $R_{FJT}(i, End)$ for each node *i*, resulting $\boldsymbol{N} - \boldsymbol{FN}^0 = \{ 11, 13 \}$ for the workflow in Fig. 2. It can be shown that $\boldsymbol{N} - \boldsymbol{FN}^0 = \{ 11, 16, 17, 18, 19, 20, 21, 22, 23, 24 \}$ for the workflow in Fig. 4. The subgraph spanned by $\boldsymbol{FN}^0$ will be called as the *main flow* and will be denoted as ***MF****(Start, End)*.

Table 1. $R_{FJT}(i, End)$ of each node *i* in Figure 2

i	0	1	2	3	4	5	6	7	8	9	16	11	12	13	14	15	16	17	18	19	20
$\boldsymbol{R}_{FJT}(i, End)$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1

Definition 2. The n^{th}*-order Feedback Joins*, denoted by $\boldsymbol{FJ}^n$, is the subset of feedback joins in $\boldsymbol{FN}^n$. That is $\boldsymbol{FJ}^n = \boldsymbol{FJ} \cap \boldsymbol{FN}^n$. The n^{th}*-order Feedback Splits*, denoted by $\boldsymbol{FS}^n$, is the subset of nodes that split the subgraph spanned by $\boldsymbol{FN}^{n+1}$ from the subgraph spanned by $\boldsymbol{FN}^n$, identified as $\boldsymbol{FS}^n = \{\, i \mid (i, j) \in \boldsymbol{T}, i \in \boldsymbol{FN}^n, j \in \boldsymbol{FN}^{n+1} \cup \boldsymbol{FJ}^n \}$.

Note that $\boldsymbol{FJ} = \cup_n \boldsymbol{FJ}^n$. By definition 2, we get $\boldsymbol{FJ}^0$ as follows,

$$\boldsymbol{FJ}^0 = \boldsymbol{FJ} \cap \boldsymbol{FN}^0. \quad (3)$$

$\boldsymbol{FJ}^0 = \{ 1 \}$ and $\boldsymbol{FJ}^0 = \{ 1, 7 \}$ for the examples of Fig. 2 and 4, respectively.

The set of nodes in any of $\boldsymbol{FS}^n$ will be called as *Feedback Splits*, denoted by ***FS***, where $\boldsymbol{FS} = \cup_n \boldsymbol{FS}^n$. Finally, $\boldsymbol{FS}^0$ will be identified. Since $\boldsymbol{FN}^1$ is not available yet and there will be no transition from a node in $\boldsymbol{FN}^0$ to any node in $\cup_{n\geq 2} \boldsymbol{FN}^n$, we identify $\boldsymbol{FS}^0$ as follows,

$$\boldsymbol{FS}^0 = \{\, i \mid (i, j) \in \boldsymbol{T}, i \in \boldsymbol{FN}^0, j \in (\boldsymbol{N} - \boldsymbol{FN}^0) \cup \boldsymbol{FJ}^0 \} \quad (4)$$

$\boldsymbol{FS}^0 = \{ 7, 8 \}$ and $\boldsymbol{FS}^0 = \{ 10, 13, 15 \}$ for the workflow in Fig. 2 and 4, respectively.

At each iteration, our method identifies $\boldsymbol{FN}^n$, $\boldsymbol{FJ}^n$ and $\boldsymbol{FS}^n$. $\boldsymbol{FN}^n$ and $\boldsymbol{FJ}^n$ is identified by classifying ***N*** and ***FJ***, respectively. On the other hand, $\boldsymbol{FS}^n$ is newly discovered from $\boldsymbol{FN}^n$, at each iteration. When all the nodes in ***FJ*** are classified into one of $\boldsymbol{FJ}^n$, we can conclude that $\boldsymbol{FN}^{n+1} = \boldsymbol{N} - \cup_{m\leq n} \boldsymbol{FN}^m$ and no further classification of nodes is required. For instance, $\boldsymbol{FJ}^0 = \boldsymbol{FJ}$ for the workflow in Fig. 2, and we can conclude with $\boldsymbol{FN}^1 = \boldsymbol{N} - \boldsymbol{FN}^0 = \{ 11, 13 \}$. Otherwise, there exist more feedback structures in the subgraph(s) spanned by $\boldsymbol{N} - \cup_{m\leq n} \boldsymbol{FN}^m$ and further classification of nodes is required.

Recursive partition for nested feedback structures

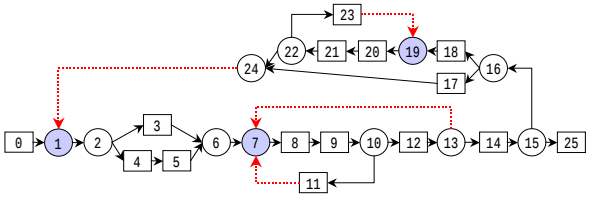
Definition 3. For any $fs \in \boldsymbol{FS}^{n-1}$, let $\boldsymbol{Desc}^n(fs)$ denote the set of nodes in $\boldsymbol{FN}^n$ and $\boldsymbol{Desc}^{n+}(fs)$ denote the set of nodes in $\cup_{m\geq n} \boldsymbol{FN}^m$, respectively, that can be reached from *fs* by the transitions in ***T***. For any $fj \in \boldsymbol{FJ}^{n-1}$, let $\boldsymbol{FAncs}^n(fj)$ denote the set of nodes in $\boldsymbol{FN}^n$ and $\boldsymbol{FAncs}^{n+}(fj)$ denote the set of nodes in $\cup_{m\geq n} \boldsymbol{FN}^m$, respectively, that can reach *fj* by the transitions in ***T***.

Definition 4. Let $fs \in \boldsymbol{FS}^{n-1}$ *and* $fj \in \boldsymbol{FJ}^{n-1}$. The n^{th}*-order Feedback Flow* $\boldsymbol{FF}^n(fs, fj)$ denotes the subgraph spanned by the set of nodes $\{ fs, fj \} \cup (\boldsymbol{Desc}^n(fs) \cap \boldsymbol{FAncs}^n(fj))$. The $(n+)^{th}$*-order Feedback Flow* $\boldsymbol{FF}^{n+}(fs, fj)$ denotes the subgraph spanned by the set of nodes $\{ fs, fj \} \cup (\boldsymbol{Desc}^{n+}(fs) \cap \boldsymbol{FAncs}^{n+}(fj))$.

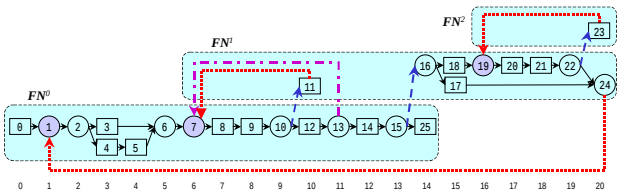
Note that we can let the main flow ***MF***(*Start*, *End*) equal to $\boldsymbol{FF}^0(Start, End)$, assuming that $Start \in \boldsymbol{FS}^{-1}$ and $End \in \boldsymbol{FJ}^{-1}$.

When ***FJ*** is not fully classified yet, there exist at least one $\boldsymbol{FF}^{n+}(fs, fj)$ that contains node(s) of ***FJ***. The corresponding subgraph contains some feedback structures and requires further partitioning. Our method recursively applies the steps of partition to each of these subgraphs. Otherwise, $\boldsymbol{FF}^{n+}(fs, fj)$ does not require further partitioning and equals to $\boldsymbol{FF}^n(fs, fj)$, by definition.

The workflow of Fig. 2, with $\boldsymbol{FN}^0$, $\boldsymbol{FJ}^0 = \boldsymbol{FJ}$, $\boldsymbol{FS}^0$, and $\boldsymbol{FN}^1$, can be partitioned into the main flow ***MF*** (0, 20), $\boldsymbol{FF}^1(7, 1)$ spanned by {7, 11, 1}, and $\boldsymbol{FF}^1(8, 1)$ spanned by {8, 13, 1}, requiring no further partitioning. For the workflow of Fig. 4, we have ***MF*** (0, 25), $\boldsymbol{FF}^1(10, 7)$, $\boldsymbol{FF}^1(13, 7)$, and $\boldsymbol{FF}^{1+}(15, 1)$. Since $\boldsymbol{FF}^{1+}(15, 1)$ contains a node of ***FJ*** not classified yet, i.e., node 19 $\notin \boldsymbol{FJ}^0$, it contains some cyclic structures and requires further partitioning. In a similar manner, we can get $\boldsymbol{FN}^1 = \{ 11, 16, 17, 18, 19, 20, 21, 22, 24 \}$, $\boldsymbol{FJ}^1 = \{ 19 \}$, and $\boldsymbol{FS}^1 = \{ 2 \}$ from $\boldsymbol{FF}^{1+}(15, 1)$. Since nodes in ***FJ*** are fully classified, we can conclude that $\boldsymbol{FN}^2 = \boldsymbol{N} - \boldsymbol{FN}^0 - \boldsymbol{FN}^1 = \{ 23 \}$ and derive additional acyclic subgraphs of $\boldsymbol{FF}^1(15, 1)$ and $\boldsymbol{FF}^2(22, 19)$. Table 2 summarizes the results of the analyses for the workflow graphs of Fig. 2 and 4. Note that each finally derived subgraph of a feedback flow, except main flow, matches for distinct simple cycle in the given workflow graph.



(a) A workflow with nested cycles



(b) Normalized graph

Fig. 4. Normalization of a workflow graph with nested cycles

Table 2. Summary of partitioning

Case	Target graph	Classified nodes	Derived subgraphs
Fig. 2	$G(N,T)$	FJ = {1}; V–FN^0 = {11,13}, FJ^0 = {1}, FS^0= {7,8}.	MF (0, 20) FF^1 (7, 1) FF^1 (8, 1)
Fig. 4	$G(N,T)$	FJ = {1,7,19}; V–FN^0 = {11,16,17,18,19, 20,21,22,23,24}, FJ^0 = {1,7}, FS^0 = {10,13,15}.	MF (0, 25) FF^1 (10, 7) FF^1 (13, 7) FF^{1+} (15, 1)
	FF^{1+} (15, 1)	FN^1 = {11,16,17,18,19,20, 21,22,24}, FJ^1 = {19}, FS^1 = {22}; FN^1= {23}.	FF^1 (15, 1) FF^2 (22, 19)

The proposed method allows a cyclic workflow model to be analyzed further, if necessary, with several smaller subflows, which are all acyclic. The complexity of this phase can be estimated as follows. Let q be the maximum degree of feedback and r be the average number of subgraphs in FN^n that need to be further partitioned, in the given workflow graph. The identification of FN^n (with FS^{n-1} *and* FJ^{n-1}) from $FF^{n+}(fs, fj)$, where $fs \in FS^{n-1}$ and $fj \in FJ^{n-1}$, is rather straightforward with R_{FJT} (i, fj), where $i \in FF^{n+}(fs, fj)$, that has complexity $O(|T\uparrow|)$, where $|T\uparrow|$ is the number of transitions in the cyclic subgraph $FF^{n+}(fs, fj)$. Therefore, the complexity of this step will be $O(qr|T|) < O(|N| \cdot |T|)$.

4. TWO-PHASED VERIFICATION OF CONTROL FLOWS

This section presents two-phased verification of structural conflicts in workflow models. First, it detects certain types of structural conflicts, in the early stage, incurred from the inappropriate composition of feedback flows. And then, it applies verification to each acyclic subgraph, which is much easier to comprehend and verify individually than the whole workflow model, in general.

4.1 Early stage verification on combining partitioned flows

Lemma 1. A workflow with an AND- feedback join node *fj* will *deadlock* at *fj*. A workflow with an AND-feedback split node *fs* will potentially result in an *infinite loop (i.e., livelock)* or *multiple instances.*

Any $FF^n(fs, fj)$ split from a XOR-split *fs* and merged into a OR-join *fj*, which is exclusive with the originating subflow in FN^n, will not introduce additional deadlock and lack of synchronization conflict to the combined flow. On the other hand, feedback joins or feedback splits of AND-type incur structural conflicts to the combined flow. Fig. 5 shows that conflicts are caused by combining a feedback flow with AND-controls. An AND-join at *fj* can never be started, thus causing a deadlock (Fig. 5(a)). An AND-split at *fs* may never end since *fs* always triggers a feedback, thus leading to an infinite loop without any type of proper exit from the loop, or may result in multiple instances of subflow sf_1 before the synchronization, if exist, with the subflow sf_2 exited from the loop (Fig. 5(b)).

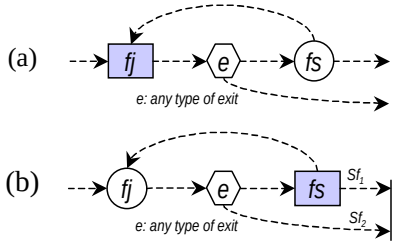
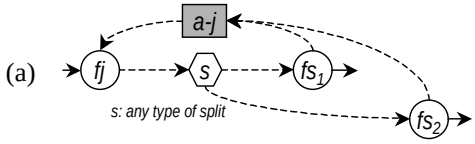


Fig. 5. Conflicts caused by AND-controls as a feedback join or a feedback split

Lemma 2. An AND-join *a-j* that joins $FF^n(fs_1, fj)$ and $FF^n(fs_2, fj)$, with $fs_1 \neq fs_2$, incurs a potential deadlock at *a-j* as illustrated in Fig. 6(a). In addition, an AND-split *a-s* that splits $FF^n(fs, fj_1)$ and $FF^n(fs, fj_2)$, with $fj_1 \neq fj_2$, may result in multiple instances of subflow sf_3 or deadlock at *j* as illustrated in Fig. 6 (b).

In Fig. 6 (a), when there exists some precedence relationship between fs_1 and fs_2, it is clear that a deadlock conflict will occur at the AND-join node *a-j*. The only way to prevent this deadlock is when split *s* is of AND-type and both fs_1 and fs_2 always lead to the AND-join *a-j* altogether, if any one of them does. But, this cannot be guaranteed and may lead to potential deadlock. In Fig. 6 (b), when there exists some precedence relationship between fj_1 and fj_2, it is clear that lack of synchronization conflict will occur at one of them visited later. Note that subflow sf_1 or sf_2 may have some XOR-exit, not represented in the figure for simplification, which makes the analysis even more complex. The only way to prevent this lack of synchronization conflict is when the join node *j* that subflow sf_1 and sf_2 merge should be of AND-join and both sf_1 and sf_2 should always lead to the AND-join *j* altogether, if any one of them does so. But, this cannot be guaranteed either and may lead to potential deadlock. Fig. 6 (c) shows the control-flow-equivalent workflow transformed from the workflow of Fig. 6 (b), by duplicating certain nodes [28] where these additional constraints are satisfied.



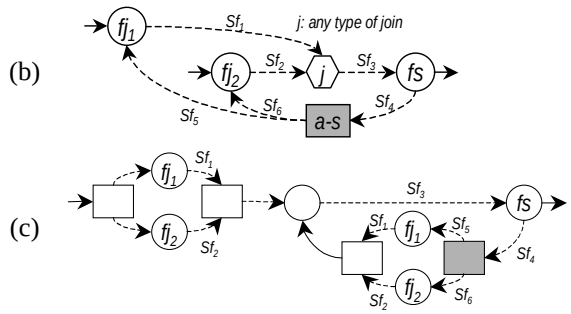


Fig. 6. Inadequate usages of (a) AND-Join and (b) AND-Split on feedback paths; (c) equivalent control flow transformed from (b) of a special case

Lemmas 1 and 2 can be used to detect certain types of structural conflict related to feedback flows. Our method identifies those ill-structured control flows based on Lemmas 1 or 2 in a workflow model. The decomposed verification phase described next, will assume that the workflow models contain only XOR- feedback split and join nodes.

4.2 Decomposed verification for each acyclic flow

We conjecture that the verification of a given workflow can be done by verifying the *main flow* and each partitioned *feedback flow* of the given workflow, where all of which are acyclic. Theorem 1 below argues the correctness of this approach formally.

Theorem 1. *If there exist no structural conflicts in* ***MF****(Start, End) and each* $\boldsymbol{FF}^n$*(fs, fj), then there exist no structural conflict in the integrated model.*

Proof: Any possible additional structural conflicts in the integrated model caused by combining each $\boldsymbol{FF}^{1+}$*(fs, fj)*, originated *from and merged into* ***MF****(Start, End)*, are of *lack of synchronization.* After early stage verification with Lemma 1 and 2, each $\boldsymbol{FF}^{1+}$*(fs, fj)* splits from an XOR feedback split *fs*, merges at an OR feedback join *fj*, and does not cross another feedback flow by an AND-split or an AND-join. Therefore, there is no chance that any additional Lack of Synchronization conflict would be caused in the combined model. The fact that each $\boldsymbol{FF}^{1+}$*(fs, fj)* can be composed in similar manner completes the proof. ///

By Theorem 1, verification of structural conflicts in a cyclic workflow model can be done by verifying structural conflicts of ***MF****(Start, End)* and each $\boldsymbol{FF}^n$*(fs, fj)*, which are all acyclic. That is, in decomposed verification approach, we only need to verify *acyclic* flows. This is a very important feature of our approach and can lead to simplified verification and significant computational efficiency compared to those approaches that verify a cyclic graph as a whole. For instance, the graph reduction technique [25] can be applied to a cyclic workflow graph after the original graph is partitioned into *s* subflow graphs. As a result of partitioning, the computation effort can be even improved from $O((|\boldsymbol{N}|+|\boldsymbol{T}|)^2 \cdot |\boldsymbol{N}|^2)$ ([3], [25]) to $O((|\boldsymbol{N}|+|\boldsymbol{T}|)^2 \cdot |\boldsymbol{N}|^2 / s^3)$ by dividing $|\boldsymbol{N}|$ and $|\boldsymbol{T}|$ by *s*, respectively.

5. CONCLUDING REMARKS

Workflow analysis and design is a fundamental task in business process management because of the recent surge of e-business process automation efforts in the corporations worldwide. Although hierarchical decomposition of a complex workflow process is an useful step in workflow analysis and design, until now there has been no reported work of its automated support in the literature. When the process model is complex, it is quite perplexing for a human to recognize substructures of potential subprocesses [2].

In this paper, we proposed a unified approach of feedback partition and decomposed verification of structural conflicts that can handle cyclic workflows efficiently. We showed how to identify the main flow and each feedback flow that completes each simple cycle, which are all acyclic. Hierarchical decomposition of a workflow model by feedback structures can make workflow analysis and design more accurate and efficient and can lead to a better design. We also showed that verification of structural conflicts in a cyclic workflow model could be done by independently verifying structural conflicts in main flow and each feedback flow, which are all acyclic. That is, in decomposed verification approach, we only need to verify *acyclic* flows.

Our work is related to three main studies in workflow verification, namely the graph reduction approach ([25], [29]), the Petri nets approach ([1], [4], [6], [33]), and the logic-based approach ([11], [27]). The graph reduction approach detects structural errors by trying to reduce the original workflow graph into an empty one. If this attempt fails, the workflow is determined to contain errors. One main drawback is that the graph reduction technique cannot handle cyclic workflows and overlapping patterns algorithmically. The Petri-nets-based workflow verification depends on the formalism of Petri nets. Therefore, for non-Petri-nets workflow models, it is difficult to apply their verification technique. The logic-based approach depends on the expressive power of propositional logic and does not take advantage of instance flows in its verification method. Furthermore, none of these approaches use the concept of feedback partitioning, which has been shown to provide advantageous features in workflow analysis and design. Our work unifies process decomposition and verification while other verification techniques focus mainly on process verification. In sum, our approach provides a useful, if not better, alternative to existing workflow verification methods.

The issues that merit future research are: to establish results for verifying correctness of a workflow model including other perspectives, for instance the data-flow;

transformation of a free-form graph-structured workflow model into a structured model, like in BPEL4WS [10]; to extend results for further workflow analysis of process reengineering.

ACKNOWLEDGEMENT

This work was supported by the 2003 Inje University research grant.

REFERENCES

[1] Aalst, W. M. P. van der, "The Application of Petri Nets to Workflow Management", *The Journal of Circuits, Systems and Computers*, vol. 8, no. 1, pp. 21-66, 1998.

[2] Aalst, W. M. P. van der, "Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques", In W.M.P. van der Aalst, J. Desel, and A. Oberweis, Editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pp. 161-183, Springer-Verlag, Berlin, 2000.

[3] Aalst, W. M. P. van der, "An alternative way to analyze workflow graphs," *14th Int. Conf. On Adv. Info. Sys. Eng.*, pp. 535-552, 2002.

[4] Aalst, W. M. P. van der and A. H. M. ter Hofstede, "Verification of workflow task structures: A Petri-Net-based Approach," *Information Systems*, vol. 25, no. 1, pp. 43-69, 2000.

[5] Aalst, W.M.P. van der, A.H.M. ter Hofstede, and M. Weske, "Business Process Management: A Survey", In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, Editors, *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pp. 1-12. Springer-Verlag, Berlin, 2003.

[6] Adam, N. R., V. Atluri, and W. Huang. "Modeling and Analysis of Workflows using Petri Nets," *Journal of Intelligent Information Systems*, vol. 10, pp. 131-158, 1998.

[7] Aissi, S., P. Malu, and K. Srinivasan. "E-business process modeling: the next big step," *IEEE Computer*, vol. 35, no. 5, pp. 55-62, 2002.

[8] Basu, A. and R. W. Blanning, "A formal approach to workflow analysis," *Information Systems Research*, vol. 11, no. 1, pp. 17-36, 2000.

[9] Basu, A. and A. Kumar, "Research commentary: Workflow management issues in e-Business," *Information Systems Research*, vol. 13, no. 1, pp. 1-14, 2002.

[10] BEA Systems, IBM Corporation, & Microsoft Corporation, Inc., *Business Process Execution Language for Web Services*, Version 1.1, 2003, available at http://www.ibm.com/developerworks/library/ws-bpel/.

[11] Bi, H. H. and J. L. Zhao, "Mending the Lag between Commerce and Research: A Logic-based Workflow Verification Approach," *Computational Modeling and Problem Solving in the Networked World*, Kluwer Academic Publishers, pp. 191-212, 2003.

[12] Business Process Management Initiative, *Business Process Modeling Notation*, Working Draft 1.0, August 2003, available at http://www.bpmi.org/

[13] Casati, F., S. Ceri, B. Pernici, G. Pozzi, "Conceptual Modeling of Workflows", In M.P. Papazoglou, Editor, *Proceedings of the 14th International Object-Oriented and Entity-Relationship Modeling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 341–354. Springer-Verlag, Berlin, 1998.

[14] Choi, Y. and J. L. Zhao, "Matrix-based abstraction and verification of e-business processes," In *Proc. the 1st Workshop on e-Business*, pp. 154-165, 2002.

[15] Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms* (2nd Ed.), MIT Press, 2001.

[16] Delphi Group, *BPM2002: Market Milestone Report*, available at http://www.delphigroup.com/.

[17] Fan, W. and S. Weinstein, "Specifying and reasoning about workflows with path constraints", In *Proceedings of the 5th International Computer Science Conference (ICSC'99)*, HongKong, China, volume 1749 of *Lecture Notes in Computer Science*, Springer, pp. 13-15, 1999.

[18] Georgakopoulos, D., M. Hornick, and A. Sheth, "An overview of workflow management: from process modeling to workflow automation infrastructure", *Distributed and Parallel Databases*, vol. 3, pp.119-153, 1995.

[19] Gondran, M. and M. Minoux, *Graphs and Algorithms*, John Wiley & Sons Ltd., 1984.

[20] Hofstede, A. H. M. ter, M. E. Orlowska, and J. Rajapakse, "Verification Problems in Conceptual Workflow Specifications," *Data and Knowledge Engineering*, vol. 24, no. 3, pp. 239-256, 1998.

[21] Kiepuszewski, B., *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*, PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002.

[22] Kumar, A., and Zhao, J. L., "Dynamic Routing and Operational Controls in Workflow Management Systems," *Management Science*, vol. 45, no. 2, pp. 253-272, 1999.

[23] Kwan, M.M. and P.R. Balasubramanian, "Adding workflow analysis techniques to the IS development toolkit," *Proc. 31st Int'l Conf. on System Sciences*, vol. 4, pp. 312-321, 1998.

[24] Leymann, F., D. Roller, and A. Reuter, *Production Workflow: Concepts and Techniques*, Prentice-Hall, Englewood Cliffs, NJ, 2000.

[25] Lin, H., Z. Zhao, H. Li, and Z. Chen, "A novel graph reduction algorithm to identify structural conflicts," *Proc. of the 35th Hawaii Int. Conf. On Sys. Sci. (HICSS'02)*, IEEE Computer Society Press, 2002.

[26] Murata, T., "Petri nets: Properties, analysis, and applications", *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, 1989.

[27] Mukherjee S., H. Davulcu, M. Kifer, P. Senkul, and G. Yang, "Logic Based Approaches to Workflow Modeling and Verification", In J. Chomicki, R. Meyden, G. Saake, Edtiors, *Logics for emerging applications of databases*, Springer-Verlag, pp. 167-202, 2004

[28] Oulsnam, G., "Unraveling unstructured programs", *Computer Journal*, vol. 25, no. 3, pp. 379-387, 1982.

[29] Sadiq, W. and M. E. Orlowska, "Analyzing process models using graph reduction techniques," *Information Systems*, vol. 25, no. 2, pp.117-134, 2000.

[30] Sheth, A. P., W. M. P. van der Aalst, and I. B. Arpinar, "Processes driving the networked economy," *IEEE Concurrency*, vol. 7, no. 3, pp. 18–31, 1999.

[31] Sivaraman, E. and M. Kamath, "On the use of Petri nets for business process modeling", *11th Annual Industrial Engineering Research Conference*, Orlando, Florida. 2002

[32] Stohr, E. A. and J. L. Zhao, "Workflow automation: Overview and research issues," *Information Systems Frontiers*, vol. 3, no. 3, pp. 281-296, 2001.

[33] Verbeek, H. M. W., T. Basten, and W. M. P. van der Aalst, "Diagnosing workflow processes using Woflan," *Computer Journal*, vol. 44, no. 4, pp. 246-279, 2001.

[34] Workflow Management Coalition, *Glossary. Document Number WfMC-TC-1011*, 1999.

[35] Workflow Management Coalition, Interface 1: Process Definition Interchange Process Model. Document Number WfMC TC-1016-P, 1999.